Università degli Studi di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Master Thesis

Improvement and Analysis of behavioural models with variability

Candidate
Christian Grioli

Supervisor
Dr. Stefania Gnesi

Referee
Prof. Pierpaolo Degano

Academic Year 2012-2013

To my family and my friends.

Abstract

Product Lines or Families represent a new paradigm widely used to describe company products with similar functionality and requirements in order to improve efficiency and productivity of a company. In this context many studies are focused on the research of the best behavioural model useful to describe a product family and to reason about properties of the family itself. In addition the model must allow to describe in a simple way different types of variability, needed to characterize several products of the family.

One of the most important of these models is the Modal Transition System (MTS), an extension of a Labelled Transition System (LTS), which introduces two types of transitions useful to describe the *necessary* and *allowed* requirements. These models have been broadly studied and several its extensions have been described. These extensions follow different approaches which entail the introduction of more and more complex and expressive requirements. Furthermore MTS and its extensions define a concept of refinement which represents a step of design process, namely a step where some allowed requirements are discarded and other ones become necessary.

In this thesis we introduce a new model, the *Constrained Modal Transition System (CMTS)*, which is a particular and more expressive extension of MTS. Moreover we study different and useful properties correlated to the CMTS. Also, we use CMTS as an useful tool to determine and to define a a hierarchy of expressivity of the known extensions with variability of LTSs and MTSs. In order to check different properties of a product family, we introduce a new deontic-temporal logic based on CTL* interpreted over CMTSs able to express classical safety and liveness properties as well as concepts like obligatory, permission and prohibition. Finally some useful optimizations are introduced to guarantee a less expensive verification from complexity point of view.

Contents

List of Figures

Chapter 1

Introduction

Today for many large and medium size companies the market is become very competitive because of the globalization, the quick development of technologies which requires updates over and over again and the customers who want new reliable products developed in short time. For companies, every aspect of technical production must be improved in order to reach low costs and high productivity and so high profits. Nevertheless in the present day the improvement of the company technical production is not enough to survive in the market, it becomes fundamental the management aspects of the company too. Indeed, now companies must be ready to follow the market and its changes, to change the initial target and to be more flexible than the past. For these reasons good management choices allow the product victory or defeat in the market and so the company victory or defeat.

In this context the **Software Engineering** helps us to study the problem and to find the best suitable solution to improve efficiency and productivity and, in effect, many studies of Software Engineering focus on the development and the studying of efficient decision-support softwares, specification languages (visual or not) useful to allow the communication and the idea exchange among several company stakeholders, approaches, strategies and architectures needed to maximize the profit and so on.

One of these approaches exploits a typical computer science technique: the reuse. In Computer Science the idea is to exploit the code, the interface or the program structure developed, checked and verified for other projects in order to produce new code for the new project with less possible effort. This technique has been applied in a systematic way in the software production until now but recently it is applied in a more general way, not only in the technical production but in the development of business strategy and in the decision phase of manager too. Instead of designing, realizing the necessary and possible requirements, coding and verifying a single product, we can generalize the entire production line to the family of products where every product has some common features of the family and own different features.

In this case we use the term **Software Product Line** [19] [31] to indicate a set of

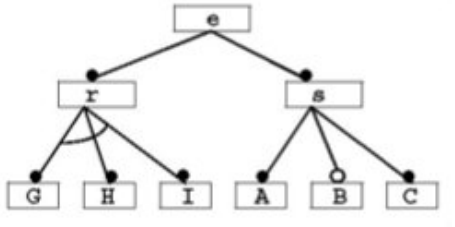
software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission, and that are developed from a common set of core assets in a prescribed way. This approach allows us to exploit the benefits of the reuse in every phase of our production line and these benefits are converted in profit for companies. Of course this technique is easily extendible to other production fields and it is called **Product Line** where this term means a family of a generic kind of products with some common features that satisfies the specific needs of a particular market segment and that is developed exploiting the reuse technique.

An example of model to define features and their usage constraints in product-lines is a **Feature Model** [7]: the features are organized into a tree, called a *feature diagram*, which is used to declaratively specify product line members. Every node of this tree represents a feature, which can be primitive (leaves) or compound (inner node) and the arrows define the relationship between children features and the parent node.
These relationships can be:

- *and*: all children must be selected
- *alternative*: only one child can be selected
- *or*: one or more can be selected
- *mandatory*: features that required
- *optional*: features that are optional

These models are visual and the Figure 1.1(a) describes the several types of relationship, whereas the Figure 1.1(b) shows us an example of this model.

(a) Relationship Types



(b) Feature Diagram example

Figure 1.1: Feature models

In addition to a specification language, we would like to have a technique or a methodology to check if our specification is correct with respect to our requirements and this problem is a typical one of verification and validation. The research in the area of verification has given rise to several different methods: more “classic” ones

are usually based on some *testing* activities, for example we may test our system with different inputs and then evaluating outputs but this approach is much expensive from point of view of the computational cost and the execution time, moreover the increase of complexity of systems introduces unpredictable errors which are complicated to find.

For these reasons but not only the techniques based on formal methods are preferred. The idea of formal methods is simple: on the one hand we have a formal description of our system and on the other hand we have a formal description of properties that the system must satisfy and, exploiting some types of algorithms, this method can check if the properties are satisfied by our system. In literature these methods can be roughly divided into two types: the *behavioural* one, in which a property is described by a set of "right" behaviours of our system and the *logical* one, in which a property is described by a logic. The latter approach is typical in model checking techniques, whereas the former is based on the checking equivalence (or some kind of similarity) between behaviours of our model and our property. Typically these techniques are applied to a model which describes a single computer system, namely a single product with a well-defined behaviour. We want to highlight that this behaviour may also be very complex and/or have non-deterministic choices, but anyway it is always and only related to a single product.

So for our needs we would like to have a formalism useful to describe a product family (a set of possible products with possibly different behaviours) and in literature it is possible to find some useful formalisms. For our purpose we take into account the *modal transition system* (MTS), introduced by Larsen and Thomsen in [36]. The MTS is a particular extension of a *labelled transition system* (LTS), which is a standard formalism to describe processes and it can be seen as a graph with nodes as states and arrows as labelled transitions. On the contrary of LTSs, MTSs have two kinds of transitions: *must* transitions representing required behaviours in all products and *may* transitions representing allowed behaviours, which can exist or not in every correct product.

For example in the Figure 1.2 we can see a MTS which represents a specification of a simple component that, received a **request**, in some way it provides **response**. The handling of the request is underspecified: the component may make a query A or a query B. If it makes a query A then it may make a query B or it may send the answer directly, whereas if it makes a query B then it must answer. In the figure we can also see a possible implementation described by a LTS.

Whereas LTSs have a some kind of behavioural equivalence called *bisimilarity*, MTSs have a generalization of the bisimilarity which allows us to understand if a MTS is a refinement of an other one, that is if the set of implementations derived by a MTS is a subset of the set of implementations derived by the other one. In this way when we say "*MTS N is a refinement of a MTS M*", we mean that N is a MTS derived by M removing some allowed features and changing some allowed features into required ones.

Of course this formalism has been studied broadly and several extensions were

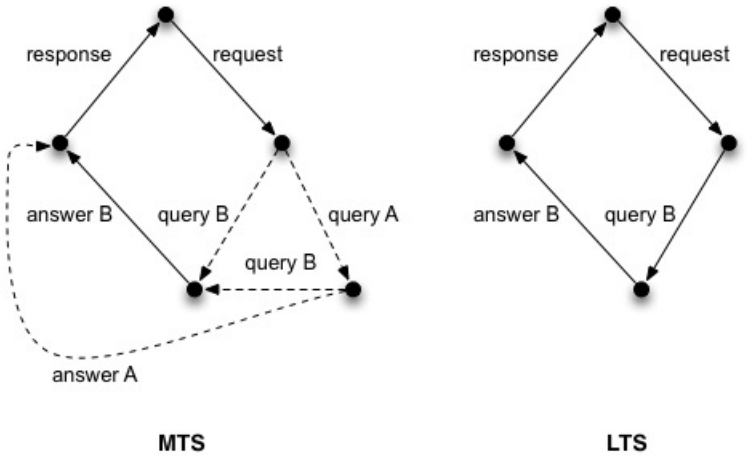


Figure 1.2: An example of a MTS and an its implementation

defined such as *disjunctive modal transition system* (DMTS) in [37] which transforms the must transition in a must hypertransition, that is a set of transitions and its semantics requires that at least one of these transitions has to be present in the final product. Another extension is *1-selecting modal transition system* defined in [27] which modifies the semantics of must hypertransition transforming the disjunctive choice in an exclusive choice, now for every set of transitions related to a hypertransition one and only one transition has to be present in the product. In [26] a generalization of DMTS and 1MTS was defined and it called *Generalized Extension Modal Transition System* GEMTS, which introduces two new types of hypertransitions, described by means of $\Diamond$ and $\Box$, where their meaning is "at most k of n", for $\Diamond$ and "at least k of n", for $\Box$.

Finally in [12] MTSs were extended exploiting a new approach, the may and must transitions are unified in a single type of transition like in LTS and an obligation formula, which represents required features, is connected to the states. This model is called *Obligation Transition System* (OTS), moreover OTSs were further extended in [13], obtaining *Parametric Modal Transition System* (PMTS) which adds conditional choices and persistence to the expressivity of model.

1.1 Contribution of the Thesis

In this thesis we introduce a new type of formalism in order to model specifications and it is called *Constrained Modal Transition System* (CTMS). In practice a CMTS has a typical transition system structure but the word "Modal" is introduced for several reasons:

1. to differentiate this formalism with *Constrained Transition System* which is another type of transition system where every transition can have a set of

constraints

2. to relate this formalism to the other ones derived by MTSs

3. to understand which kinds of constraints we can express on transitions. In effect these constraints describe us if a transition may or must exist in every good product or how many transitions (minimum and maximum) of a specific set may exist, so these constraints describe "modal" situations.

Then we study some problems related to CMTSs as the existence of a minimal number of constraints to describe a CMTS, the consistency problem and if it is possible to resolve it easily, the compositionality between two CMTSs, the refinement relation of CMTSs. Unfortunately some problems are still open, for example the complexity class of the refinement, the relation between the minimal number of constraints and CMTSs.

The second step is the usage of CMTSs to find a expressivity hierarchy of all models known until now, introducing some extensions of CMTSs, in particular introducing some kinds of guards both for transitions and for constraints. Finally we study if it is possible to find a "good" and interesting logic to use the model checking technique over CMTSs, obtaining a new type of *deontic-temporal logic* developed by means of a new different approach which, to the best of our knowledge, has never been used. Moreover we identify some useful kind of CMTSs, derivable from a generic CMTSs easily, which can give semantics to logic formulae in a simple way.

1.2 Outline of the Thesis

The thesis is organized as follows:

- **Chapter 2** describes the background of this thesis, in particular we describe several extensions of MTS in a more detailed way and some logics used as starting point to develop our logic. Moreover we introduce some concepts and definitions used in following chapters.

- **Chapter 3** introduces the new formalism, describing some its features. In particular we study the refinement relation, a possible way to compose different CMTSs and if it is possible to resolve the minimalization problem, that is if we have a CMTS M then we can determine a CMTS N, semantically equivalent to M, with a minimal number of constraints.

- **Chapter 4** extends the CMTS formalism further, introducing some kind of guard in both transitions and constraint definitions. So these extensions are studied in a deeply way and some problems, caused from how these extensions are described, is presented and analysed.

- **Chapter 5** exploits CMTS and its extensions defined in Chapter 3 and Chapter 4 to present an expressivity hierarchy of all models described in Chapter 2. We show some theorems and some examples to describe the structure of the hierarchy, to understand the expressivity of each model and the relationship between them.

- **Chapter 6** presents some logics to describe properties over CMTS and its extensions in order to use the several kinds of CMTS and these logics in a model checking technique. Moreover we describe a particular restriction of CMTS suitable to reason about the logic in a simple way, where "simple" is meant from pointview of computational costs, explaining both algorithmically and conceptually how we can transform a generic CMTS in a CMTS with a more useful structure.

- **Chapter 7** concludes the thesis and discusses about future and ongoing work for this formalism.

Chapter 2

Background and Preliminaries

In this chapter we describe the MTS model and its extensions in a deeper way, in particular, for every model, we present its formal definition and the one of refinement related to the model itself. All these models have been studied extensively, considering different points of view and characteristics, in effect in literature it is possible to find many works related to them. In this section, of course, we only describe the useful characteristics for our purposes, ignoring everything else.

Moreover we would like to use the formal methods with the logical approach, so our specifications should be described by a particular logic. For this reason we also present some different logics, which are typical in the model checking context and, as we will see in the **Chapter 6**, they are our starting point to develop the new logic too.

However, first of all, we introduce an example of a possible "concrete" specification which might be encountered in our everyday life. In following sections and chapters we exploit this example continuously to show the characteristics of each model and the expressivity differences among them.

2.1 Example

Suppose we have to describe a family of vending machines which are very simplified machines and related to a particular type of vending market segment: the drinking one. In this case we might have several requirements for our machines, in particular:

1. A vending machine is activated by a coin. The only accepted coins are the one euro coin for European products and the one dollar coin for US products. Only one kind of coins is accepted.

2. After inserting a coin, the user has to choose whether he wants sugar or not, by pressing one of two buttons. Then, the user may select the drink.

3. The choice of drinks (coffee, tea, cappuccino) varies between products. However, every product of the family delivers coffee, and every product of the

family delivers at least two different drinks.

4. After delivering the drink, a done message is displayed, and, optionally, an alert tone is rung.

5. The machine goes back in the idle state when the cup is taken by the user.

As we can see, these requirements describe a set of possible, partially different products and we can divide them in two large categories: US products and European products. Moreover each category has several possible products.

For a better understanding of product-lines concept, we suppose to link every possible choice of any feature, previously described in the example, to a label in the following way:

- feature **coins type**: **US coin** for US coins and **EU coin** for European coins.
- feature **sugar choice**: **Sugar** to describe the feature which allows to choose a beverage with or without sugar.
- feature **drink type**: **Coffee**, **Tea**, **Cappuccino** for the type of chosen drink.
- feature **message type**: **Answer** to describe that our product shows a message after the delivering of chosen drink.
- feature **ring type**: **Ring** if our product has alert tone feature, otherwise **NoRing**.

Exploiting these labels we can describe all products derivable from our specification:

- the US category has:
  1. **US coin**, **Sugar**, **Coffee**, **Tea**, **Answer**, **NoRing**
  2. **US coin**, **Sugar**, **Coffee**, **Tea**, **Answer**, **Ring**
  3. **US coin**, **Sugar**, **Coffee**, **Cappuccino**, **Answer**, **NoRing**
  4. **US coin**, **Sugar**, **Coffee**, **Cappuccino**, **Answer**, **Ring**
  5. **US coin**, **Sugar**, **Coffee**, **Tea**, **Cappuccino**, **Answer**, **NoRing**
  6. **US coin**, **Sugar**, **Coffee**, **Tea**, **Cappuccino, Answer**, **Ring**
- the European category has:
  1. **EU coin**, **Sugar**, **Coffee**, **Tea**, **Answer**, **NoRing**
  2. **EU coin**, **Sugar**, **Coffee**, **Tea**, **Answer**, **Ring**
  3. **EU coin**, **Sugar**, **Coffee**, **Cappuccino**, **Answer**, **NoRing**
  4. **EU coin**, **Sugar**, **Coffee**, **Cappuccino**, **Answer**, **Ring**

5. **EU coin**, **Sugar**, **Coffee**, **Tea**, **Cappuccino**, **Answer**, **NoRing**
6. **EU coin**, **Sugar**, **Coffee**, **Tea**, **Cappuccino, Answer**, **Ring**

We highlight as a single, simple specification, such as one described in the example, allow us to describe a large number of different products which have similar features at the same time.

2.2 Models

In this section we introduce several models developed to describe a specification, every one has some advantages and some disadvantages for example the more expressive model is, the higher computational cost is. Even though a categorization of these models has never been made, we can see easily some common characteristic in their definition:

1. every model is a particular type or an extension of transition systems

2. some models introduce the modal operators $\Box$ and $\Diamond$ in their definition and we could call the set of these models like **Modal Family**.

3. other models introduce the obligation concept, using logic formulae related to states. Every formula represents features requested and we could call these models like **Obligation Family**

4. some models in the Modal Family introduce the hypertransition concept, that is a transition described by a pair (s, T) where s is a source state and T is a set of pairs (l, s') where l is a label and s' is a possible target state. We could call these models **Modal HyperTransition Family**. In addition the set of models which use only the transition concept could be called **Modal Transition Family**.

5. some models in the Modal HyperTransition Family introduce the modal operators with them classical meaning, $\Box$ "necessity" and $\Diamond$ "possibility". We could call the set of these models **Alethic Modal Family**. On the other hand some models use the modal operators but their meaning is modified. We could call these models **Extended Modal Family**.

The Figure 2.1 summarizes these families, describing the history of all these models and the different families just presented. Moreover, as we can see, the set of these models is very heterogeneous because of different approaches used in their development.

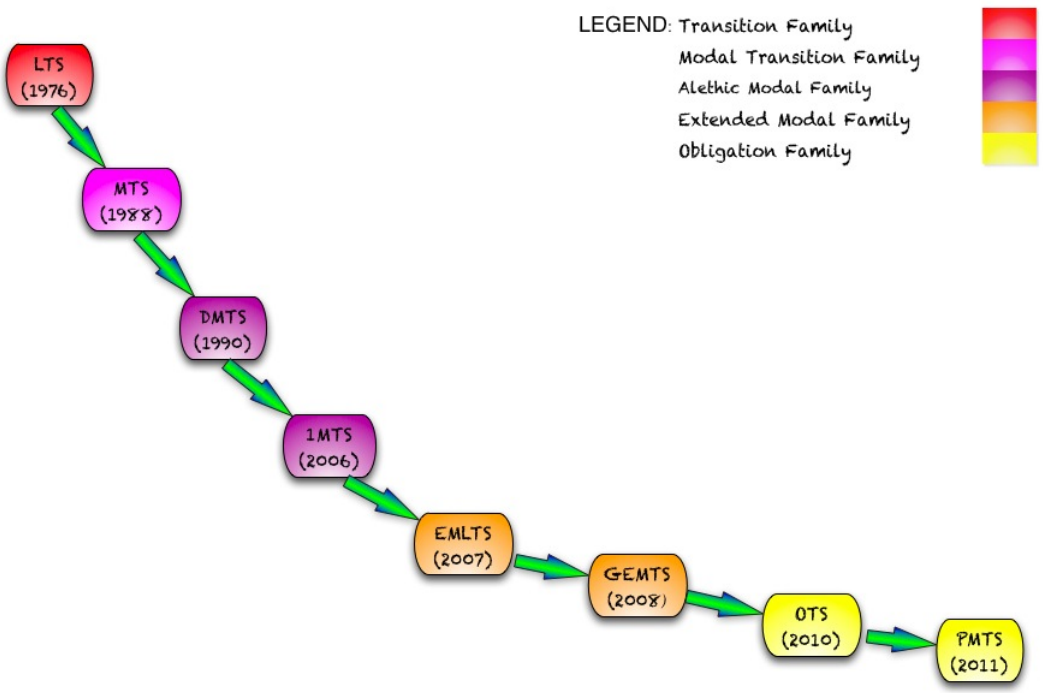


Figure 2.1: The history of models

2.2.1 LTS

In [32], for the first time, Keller introduced one of most common, used formalism: the **Labelled Transition System** which is an extension of the **Transition System**.

Definition 2.1 (**Transition System (TS)**)**:**
A *Transition System (TS)* is a tuple $T = (\mathcal{S}, \longrightarrow, \mathcal{S}_0)$ where:

- $\mathcal{S}$ is a set of states
- $\longrightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is a transition relation
- $\mathcal{S}_0$ is a set of initial states

Sometimes, when the information of initial states is irrelevant, $\mathcal{S}_0$ is not presented. ■

Definition 2.2 (**Labelled Transition System (LTS)**)**:**
A *Labelled Transition System (LTS)* over a set of actions Σ is a tuple $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of possible actions

- $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathcal{S}_0$ is a set of initial states

We denote the set of all possible LTSs by $\mathbb{LTS}$.

■

Let $(s, \alpha, s') \in \longrightarrow$ for some $s, s' \in S$ and $\alpha \in \Sigma$ then we call **source state** the state s, **target state** the state s' and **label** the action α. Moreover a state without outgoing transitions is called **terminal state**. For convenience $s \xrightarrow{\alpha} s'$ and $(s, \alpha, s') \in \longrightarrow$ describe the same thing.

LTS has been introduced to describe parallel and concurrent programs and communicating systems, that is systems where their behaviour is defined by executions of actions. In some context, TSs is called *unlabelled TSs* to highlight the difference with LTSs.

The intuitive behaviour of a LTS can be described as follows. The LTS starts in some initial state $s_0 \in \mathcal{S}_0$, chosen in a non deterministic way, and evolves using the transition relation. Each step in the LTS follows this rule: if s is the current state, then a transition $s \xrightarrow{\alpha} s'$, originating from s, is selected *non-deterministically* and the action α is performed and the LTS evolves from the state s to the state s'.

From this point of view it is possible define a subset of LTSs, which only describes deterministic systems.

Definition 2.3 (**Action-Deterministic LTS**)**:**
A Labelled Transition System (LTS) $L = (\mathcal{S}, \Sigma, \rightarrow, \mathcal{S}_0)$ is *action-deterministic* if and only if:

1. $\forall s \in S, \alpha \in \Sigma.\ (s, \alpha, s') \in \longrightarrow\ \wedge\ (s, \alpha, s'') \in \longrightarrow \Rightarrow s' = s''$
2. $|\mathcal{S}_0| = 1$

■

The condition 1) requests a restriction of the relation transition: the target state of any transition is univocally determined by its source state and its label. The condition 2) requests a LTS which has got one and only one initial state, removing the non-deterministic choice in initial states.

Moreover we add some useful definitions to handle the LTS behaviour, presented like in [6].

Definition 2.4:
Let $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ be a LTS. For $s \in S$ and $\alpha \in \Sigma$, the set of direct *α-successors* of s is defined as:

$$Post(s, \alpha) = \{s' \in \mathcal{S}. \mid s \xrightarrow{\alpha} s'\},\ Post(s) = \bigcup_{\alpha \in \Sigma} Post(s, \alpha)$$

The set of direct α-*predecessors* of s is defined as:

$$Pre(s,\alpha) = \{s' \in \mathcal{S}. \mid s' \xrightarrow{\alpha} s\},\ Pre(s) = \bigcup_{\alpha \in \Sigma} Pre(s,\alpha)$$

■

Definition 2.5 (**Path Fragment**)**:**
Let $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ be a LTS. A *finite path* π is a state sequence $s_0, s_1, \ldots, s_n$ such that $s_i \in Post(s_{i-1})$ for all $0 < i \leq n$, where $n \geq 0$. An *infinite path* π is a state sequence $s_0, s_1, \ldots$ such that $s_i \in Post(s_{i-1})$ for all $i > 0$

■

Definition 2.6 (**Maximal and Initial Path**)**:**
A *maximal* path fragment is either a finite path fragment that ends in a terminal state, or an infinite path fragment. An *initial* path fragment is a path which starts in an initial state, that is $s_0 \in \mathcal{S}_0$

■

Definition 2.7 (**Path**)**:**
A *path* of a LTS is an initial, maximal path fragment.

■

Definition 2.8 (**Run Fragment**)**:**
Let $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ be a LTS. A *finite run* ρ is a sequence $s_0\alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n$ such that $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $0 < i \leq n$, where $n \geq 0$. An *infinite run* ρ is a sequence $s_0\alpha_1 s_1 \alpha_2 \ldots$ such that $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $i > 0$.

■

The definitions of maximal and initial run is the same of maximal and initial path, but in this case we consider a run fragment and not a path fragment.

Definition 2.9 (**Run**)**:**
A *run* of a LTS is an initial, maximal run fragment.

■

Another problem studied in the LTS world is the following: "*taken two different LTSs L and L_1, is it possible to know if L and L_1 are behaviourally equivalent?*", that is "*is it possible to say that L and L_1 have the same behaviour?*". The relation which resolves this problem is the **bisimulation**. It was introduced by Park in [41] and in literature it is possible to find several kinds of bisimulations like weak [39], dynamic [16] and so on.

Now we see the definition of bisimulation:

Definition 2.10 (**Bisimulation**)**:**
Let $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ be a LTS and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ be a binary relation over $\mathcal{S}$. Then $\mathcal{R}$ is called *bisimulation* over L if, whenever sRt:

- if $s \xrightarrow{\alpha} s'$ then $\exists t' \in \mathcal{S}$ such that $t \xrightarrow{\alpha} t'$ and $s' \mathcal{R} t'$
- if $t \xrightarrow{\alpha} t'$ then $\exists s' \in \mathcal{S}$ such that $s \xrightarrow{\alpha} s'$ and $s' \mathcal{R} t'$

■

In some context this bisimulation is also called **strong** bisimulation.

Definition 2.11 (Bisimilarity):
Let $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{S}_0)$ be a LTS and $s, t \in \mathcal{S}$ be two states. Then we say that s and t are bisimilar, written $s \sim t$, if it exists a bisimulation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$. The relation $\sim = \bigcup_{\mathcal{R} \text{ is a bisimulation}} \mathcal{R}$ is called *bisimilarity*.

■

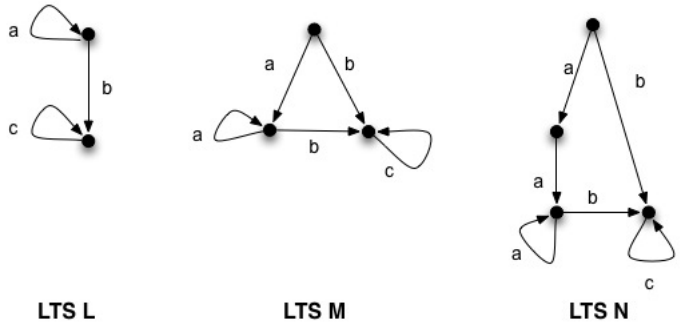


Figure 2.2: Examples of LTSs

In Figure 2.2 we can see some different LTSs. Using the bisimulation concept, it is possible to find out that LTS L and LTS M are bisimilar, whereas LTS N has a different behaviour compared with the other LTSs. For example this simple action sequence: $\{a, b\}$ is possible in L, M, but not in N.

2.2.2 MTS

In [36] Larsen and Thomsen introduced a new formalism: the *modal transition system*. They noted that the LTS formalism is expressively too poor in order to provide a convenient specification. In effect any specification defined through a LTS will limit the possible implementations to a single (behavioural) equivalence class and the reason is simple: taken a LTS and using the bisimulation we will be able to derive other semantically equivalent LTSs, so every specification (LTS) will describe all and only different products but with the same behaviour.

On the other hand we would like to describe by a specification a wide collection of (possibly inequivalent) implementations and, exploiting some technique, this collection should be constantly reduced during the design process in order to determine a single implementation eventually. It becomes clear that LTSs are not enough expressive for this task.

The basic initial idea is simple: a specification makes requirements to implementations through their operational behaviour and there are two types of requirements: the **necessary** one (**must**) and the **possibility** one (**may**). In order to achieve more expressive specifications Larsen and Thomson add modalities to the transitions of a specification:

- **necessary transition**, written $s \xrightarrow{\alpha}_{\Box} s'$, which means that in every right implementation the ability of performing α-action must exist.
- **admissible transition**, written $s \xrightarrow{\alpha}_{\Diamond} s'$, which means that in every right implementation the ability of performing α-action may exist, that is this ability is allowed but not required.

Definition 2.12 (**Modal Transition System (MTS)**)**:**
A *Modal Transition System (MTS)* is a tuple $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ where:

- S is a set of states
- Σ is a set of actions
- $\longrightarrow_{\Box} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation which describes necessary requirements of process behaviours (must transitions)
- $\longrightarrow_{\Diamond} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation which describes admissible requirements of process behaviours (may transitions)

Moreover $\longrightarrow_{\Box} \subseteq \longrightarrow_{\Diamond}$ and this property is called **consistency requirement**.

We denote the set of all possible MTSs by $\mathbb{MTS}$.

■

In other context the MTS without the last property, that is $\longrightarrow_{\Box} \subseteq \longrightarrow_{\Diamond}$, is called **Mixed Transition System** [20].

In Figure 1.2 we can see a typical MTS, where we draw the must transitions as **solid arrows** and the may transitions as **dashed arrows**.

Now in the MTS world a specification is modelled by a MTS, whereas every product or implementation is represented by a LTS as we can see in Figure 1.2. Moreover, in this way, every specification can represent a set of possible LTSs with different behaviours. We may view implementations as specifications where all requirements are necessary ones.

Definition 2.13:
A LTS $L = (\mathcal{S}, \Sigma, \longrightarrow)$ is a MTS where $\longrightarrow_{\Box} = \longrightarrow_{\Diamond} = \longrightarrow$.

■

For example in Figure 2.3 we can see an implementation derived by the MTS in Figure 1.2, but it has a different behaviour compared with the implementation in Figure 1.2.

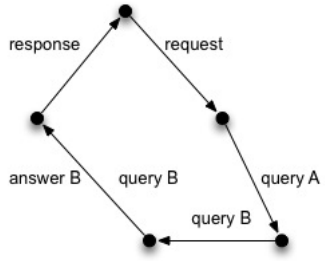


Figure 2.3: An implementation of the MTS in Figure 1.2

The next step is to understand when an implementation is "good", taken a specification, or when a particular specification is derived by another one, that is a relation which describes us the design process. Intuitively, let s, t be two specifications then we can say that s is derived by t if any behavioural aspect allowed by s is also allowed by t and, dually, any behavioural aspect requested by t must be also requested by s. These concepts are formalized by the notion of refinement:

Definition 2.14 (Refinement):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ be a MTS. A binary relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is called *refinement* if and only if $(s,t) \in \mathcal{R}$ implies:

1. $s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow t \xrightarrow{\alpha}_{\Diamond} t' \wedge (s', t') \in \mathcal{R}$

2. $t \xrightarrow{\alpha}_{\Box} t' \Rightarrow s \xrightarrow{\alpha}_{\Box} s' \wedge (s', t') \in \mathcal{R}$

s is said to be a refinement of t ($s \trianglelefteq t$) if some refinement relation $\mathcal{R}$ exists and $(s,t) \in \mathcal{R}$.

■

Of course a straightforward generalization allows us to compare states from different MTSs. Moreover if $s_0 \trianglelefteq t_0$, where s_0 and t_0 are the initial states of L and M, respectively and L is a LTS and M is a MTS then we will say that L is an **implementation** of M.

In addition the refinement relation enjoys many pleasant properties:

1. $\trianglelefteq$ is itself a refinement, in particular the maximal one

2. $\trianglelefteq$ is a preorder, that is it enjoys the reflexive and transitive property

3. the refinement is a generalization of the bisimulation, in effect if $\longrightarrow_{\Diamond} = \longrightarrow_{\Box}$ the notions of refinement and bisimulation coincide, and $\trianglelefteq$ becomes the bisimilarity $\sim$.

2.2.3 DMTS

Sometimes in the modelling of a product-line we would like to say "taken a set of possible features, at **least one** of them **must** be present in our products". Unfortunately, we cannot handle this situation using MTSs but the DMTS formalism resolves this problem.

The DMTS, introduced in [37] [9], extends the MTS formalism: the type of must transition is modified from transition to hypertransition, whereas the may transitions are unchanged.

Definition 2.15 (**Hypertransition**)**:**
Let $\mathcal{S}$ be a set of states, Σ be a set of actions and $s \in \mathcal{S}$ be a state. Then a *hypertransition* is a tuple (s, T), where $T \subseteq \Sigma \times \mathcal{S}$.
A transition (s, α, s') is a particular hypertransition (s, T) where:

1. $|T| = 1$, that is T is a singleton
2. $T = \{(\alpha, s')\}$

■

Now it is possible to define a DMTS:

Definition 2.16 (**Disjunctive Modal Transition System (DMTS)**)**:**
A *Disjunctive Modal Transition System (DMTS)* is a tuple $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of actions
- $\longrightarrow_{\Box} \subseteq \mathcal{S} \times \mathcal{P}(\Sigma \times \mathcal{S})$ is the must transition relation
- $\longrightarrow_{\Diamond} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the may transition relation

We denote the set of all possible DMTSs by $\mathbb{DMTS}$.

■

Intuitively $s \longrightarrow_{\Box} T$ may be understood as $\bigvee_{(\alpha,s')\in T} s \longrightarrow_{\Box} \{(\alpha, s')\}$. On the contrary to MTS, DMTS allows us to define an inconsistent specification by the expression $s \longrightarrow_{\Box} \emptyset$. Another difference is that in the DMTS definition no-one consistency requirement exists.

Definition 2.17 (**Syntactic consistency**)**:**
A DMTS is called *syntactically consistent* if $s \longrightarrow_{\Box} T$ implies:

1. $T \neq \emptyset$
2. $\forall(\alpha, s') \in T.\ (s, \alpha, s') \in \longrightarrow_{\Diamond}$

■

The syntactic consistency requires that every hypertransition (s, T) has the set $T \neq \emptyset$ and every transition, required through some hypertransition, must be also allowed, conceptually this consistency looks like the MTS one.

Definition 2.18:
A MTS is a particular DMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ where:

1. M is syntactically consistent
2. $\forall s \in \mathcal{S}.\ (s, T) \in \longrightarrow_{\Box} \Rightarrow |T| = 1$, that is T is singleton

■

The next step is to introduce the refinement relation for DMTS:

Definition 2.19 (Refinement):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ be a DMTS. A binary relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is called *refinement* if and only if $(s, t) \in \mathcal{R}$ implies:

1. $s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow t \xrightarrow{\alpha}_{\Diamond} t' \wedge (s', t') \in \mathcal{R}$
2. $t \rightarrow_{\Box} V \Rightarrow s \rightarrow_{\Box} U$ such that $\forall (\alpha, s') \in U.\ \exists (\alpha, t') \in V \wedge (s', t') \in \mathcal{R}$

s is said to be a refinement of t ($s \trianglelefteq t$) if some refinement relation $\mathcal{R}$ exists and $(s, t) \in \mathcal{R}$.

■

As in the MTS case, a straightforward generalization allows us to compare states from different DMTSs, moreover if $s_0 \trianglelefteq t_0$, where s_0 is the initial state of a LTS L and t_0 is the initial state of a DMTS M, then we may say that L is an implementation of M. In addition this refinement relation has the same properties of the MTS relation. Note that the refinement of a DMTS coincides with the refinement as defined on MTS and bisimulation as defined on LTS.

Example 2.1. *Suppose that our vending machine has this requirement: "The choice of drinks (coffee, tea, cappuccino) varies between the products. However, every product of the family delivers at least one different drink". Then we can model it using the DMTS as described in the Figure 2.4. Note that, for convenience, may transitions are not described. This is not an error since must hypertransitions guarantee us the presence of may transitions implicitly, if and only if the DMTS is a syntactically consistent.*

Moreover, as we can see, the LTSs L, M and N are some of the possible implementations of our DMTS.

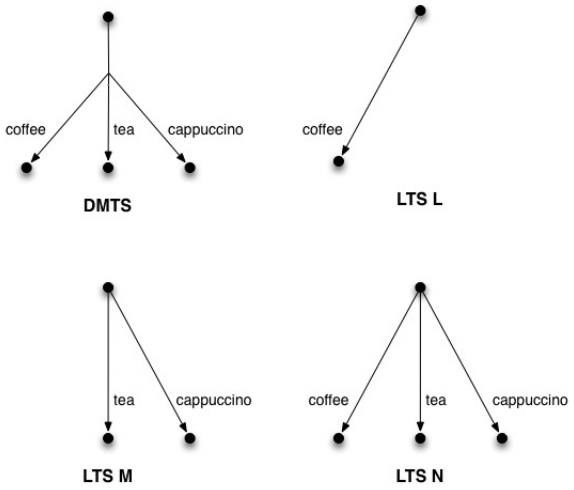


Figure 2.4: An example of DMTS and its implementations

2.2.4 1MTS

As we said in the DMTS section, taken a set of possible features, the DMTS allows us to choose among these features in a disjunctive way, that is for every set we can make a disjunctive choice. A simple extension may be the change of the choice type. Using 1MTS, introduced in [27], we can choose in an exclusive way and, from the modelling pointview, the exclusive choice is equivalent to say "taken a set of possible features, **one and only one** of them **must** be present in our products".

The 1MTS takes advantage of the hypertransition concept and in addition it introduces a new concept: the **choice function**.

Definition 2.20 (**Choice function**)**:**
Let A be a set, $\mathcal{P}_A \subseteq \mathcal{P}(A)$ and $\gamma : \mathcal{P}_A \longrightarrow A$. Then γ is a *choice function* if $\forall B \in \mathcal{P}_A.\ \gamma(B) \in B$. We denote the set of all choice functions on $\mathcal{P}_A$ by $\mathbf{choice}(\mathcal{P}_A)$. ■

In our context A will be the set of all possible transitions of the entire specification, $\mathcal{P}(A)$ will be the set of all possible hypertransitions, $\mathcal{P}_A$ will be the set of all possible hypertransitions of our specification, B will be a particular hypertransition and a function γ, taken a hypertransition B, will return one and only one element of B, that is a transition.

Moreover we introduce a new definition in order to handle the hypertransition in a more simple way.

Definition 2.21:
Let $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ be a generic relation and $s \in \mathcal{S}$ be a state. Then we define $(s \xrightarrow{\alpha}) = \{t \in \mathcal{S} \mid (s, \alpha, t) \in \longrightarrow\}$ and $(s \longrightarrow) = \{(\alpha, t) \in \Sigma \times \mathcal{S} \mid (s, \alpha, t) \in \longrightarrow\}$ ■

Definition 2.22 (1-selecting Modal Transition System (1MTS)):
A *1-selecting Modal Transition System (1MTS)* is a tuple $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of actions
- $\longrightarrow_{\Box} \subseteq \mathcal{S} \times (\mathcal{P}(\Sigma \times \mathcal{S}) \setminus \emptyset)$ is the must transition relation
- $\longrightarrow_{\Diamond} \subseteq \mathcal{S} \times (\mathcal{P}(\Sigma \times \mathcal{S}) \setminus \emptyset)$ is the may transition relation

Moreover $\longrightarrow_{\Box} \subseteq \longrightarrow_{\Diamond}$ (consistency requirement).

We denote the set of all possible 1MTSs by $1\mathbb{MTS}$.

■

Note the two little changes: first of all, the may transition relation also uses the hypertransition and both may relation and must relation cannot consider the "inconsistent" hypertransition, that is the hypertransition (s, T) where $T = \emptyset$.

The reason of the introduction of may hypertransition is simple. Consider the system in Figure 2.5 (a). It may be either interpreted as DMTS (Figure 2.5 (b)) or as 1MTS(Figure 2.5 (c)) and for a better understanding we draw the must hypertransition and the may transitions explicitly. Now we take the DMTS and try to reason about its implementations.

As we can see in the Figure 2.6 LTSs L and I are two possible implementations of M, furthermore the DMTS N is a refinement of M too. If we consider the system (b) in Figure 2.5 with the exclusive interpretation of must hypertransitions, we can easily note that this system fails, in effect the LTS I in Figure 2.6 is an implementation of this system but it is not satisfied the exclusive interpretation. The 1MTS described in Figure 2.5 (c) solves this problem.

Definition 2.23:
A MTS is a particular 1MTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ where:

1. $\forall s \in \mathcal{S}.\ (s, T) \in \longrightarrow_{\Diamond} \Rightarrow |T| = 1$, that is T is singleton
2. $\forall s \in \mathcal{S}.\ (s, T) \in \longrightarrow_{\Box} \Rightarrow |T| = 1$, that is T is singleton

■

Note that the set $(s \longrightarrow_{\Diamond})$ (or equivalently $(s \longrightarrow_{\Box})$) in a 1MTS has elements like $\Theta = (T)$, where $T \subseteq \Sigma \times \mathcal{S}$, that is its elements are hypertransitions. Moreover the refinement definition needs a new concept:

Definition 2.24:
Let $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ be a generic relation between states. Then the extension of $\mathcal{R}$ to $(\Sigma \times \mathcal{S}) \times (\Sigma \times \mathcal{S})$ is: for $\vartheta = (\alpha, s') \in \Sigma \times \mathcal{S}$ and $\vartheta_1 = (\alpha_1, s'_1) \in \Sigma \times \mathcal{S}$, we define $(\vartheta, \vartheta_1) \in \mathcal{R} \Leftrightarrow \alpha = \alpha_1 \ \wedge (s', s'_1) \in \mathcal{R}$.

■

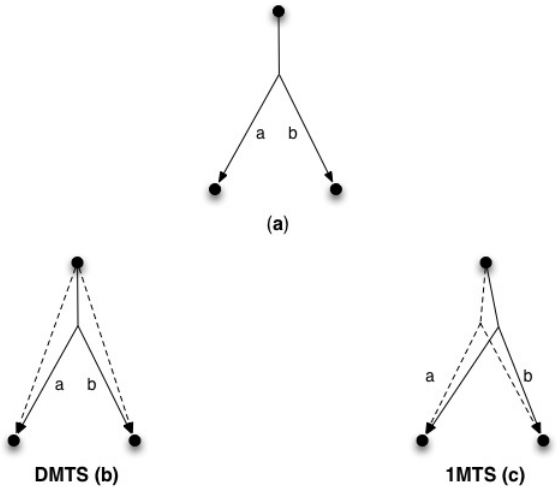


Figure 2.5: An example of DMTS and 1MTS

Now we can describe the refinement relation for 1MTS:

Definition 2.25 (Refinement):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ be a 1MTS. A binary relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is called *refinement* if and only if $\forall (s,t) \in \mathcal{R}$ and $\forall \gamma \in \textbf{choice}(s \longrightarrow_{\Diamond})$. $\exists \widehat{\gamma} \in \textbf{choice}(t \longrightarrow_{\Diamond})$ such that the following holds:

1. $\forall \Theta_s \in (s \longrightarrow_{\Diamond})$. $\exists \Theta_t \in (t \longrightarrow_{\Diamond})$. $(\gamma(\Theta_s), \widehat{\gamma}(\Theta_t)) \in \mathcal{R}$

2. $\forall \Theta_t \in (t \longrightarrow_{\Box})$. $\exists \Theta_s \in (s \longrightarrow_{\Box})$. $(\gamma(\Theta_s), \widehat{\gamma}(\Theta_t)) \in \mathcal{R}$

s is said to be a refinement of t ($s \trianglelefteq t$) if some refinement relation $\mathcal{R}$ exists and $(s,t) \in \mathcal{R}$.

■

As in other cases, a straightforward generalization allows us to compare states from different 1MTSs and, of course, this refinement relation has the same properties of the MTS and DMTS refinement relation. Note that the refinement of a 1MTS coincides with the refinement as defined on MTS and bisimulation as defined on LTS.

Example 2.2. *Suppose that our vending machine has this requirement: "The choice of drinks (coffee, tea, cappuccino) varies between the products. However, every product of the family delivers one and only one different drink". Then we can model this request using the 1MTS as described in the Figure 2.7.*
As we can see the LTSs L, M and N are all possible implementations of our 1MTS.

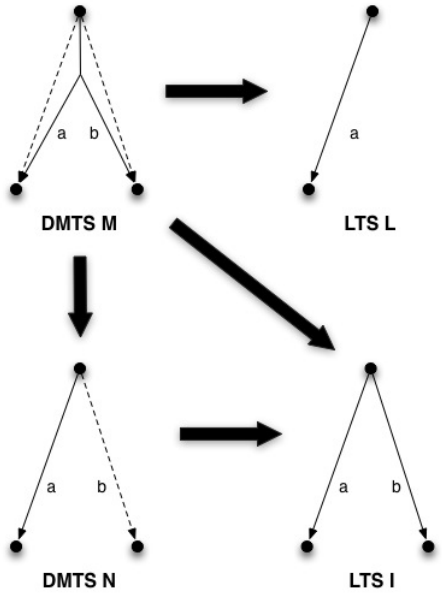


Figure 2.6: An example of DMTS and its problem with exclusive choices

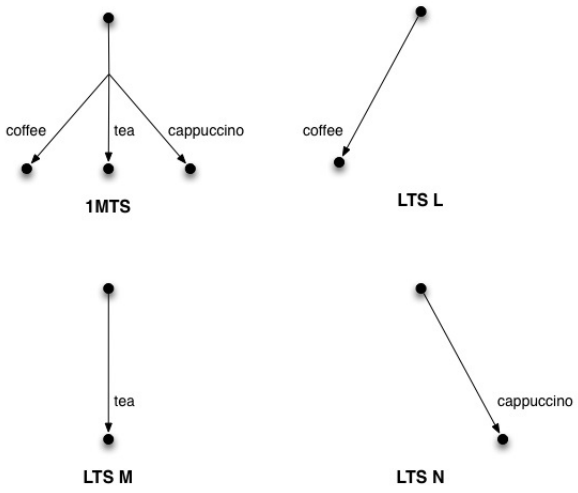


Figure 2.7: An example of 1MTS and its implementations

2.2.5 GEMTS

A generalization of DMTS and 1MTS was introduced in [26]: we can only handle requirements as "at least one feature of a some set is required", using DMTS, and "exactly one feature of a some set is required", using 1MTS, so Fantechi and Gnesi in [26] added the possibility to model the requirement "at most one feature of a some set is required", allowing to describe a bigger number of specifications than DMTS or 1MTS models.

This extension is clearly conceptual and, as we will see, hypertransition concept is still used.

Definition 2.26 (Generalized Extended Modal Transition System (GEMTS)): A *Generalized Extended Modal Transition System (GEMTS)* is a tuple $(S, \Sigma, \Box, \Diamond, s_0)$ where:

- S is a set of states
- Σ is a set of actions
- $\Box \subseteq S \times (\mathcal{P}(\Sigma \times S) \times \mathcal{N})$ is the "at least k of n" transition relation
- $\Diamond \subseteq S \times (\mathcal{P}(\Sigma \times S) \times \mathcal{N})$ is the "at most k of n" transition relation
- $s_0 \in S$ is the initial state

We denote the set of all possible GEMTSs by $\mathbb{GEMTS}$.

■

The transition relations $\Box$ and $\Diamond$ describe two types of hypertransitions but if in DMTS and 1MTS $\Box$ means "necessary" and $\Diamond$ "possible", now their meaning changes.

We write respectively:

- $s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Box_k} s_1, s_2 \ldots s_n$ to denote elements of the relation $\Box$ and its meaning is any product of the family should have at least k of n transitions $s \xrightarrow{\alpha_i} s_i$
- $s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Diamond_k} s_1, s_2 \ldots s_n$ to denote elements of the relation $\Diamond$ and its meaning is any product of the family should have at most k of the n transitions $s \xrightarrow{\alpha_i} s_i$

In addition it is implicitly assumed that the number of actions on arrows must coincide with that of target states and the order is important, finally the property $0 < k \leq n$ should always hold. Moreover these relations have some other properties:

1. $s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Box_k} s_1, s_2 \ldots s_n \Rightarrow s \xrightarrow{\alpha_2,\ldots,\alpha_n}_{\Box_{k-1}} s_2 \ldots s_n$, that is if at least k transitions must be taken from a set S, then we can deduce that at least k-1 transitions must be taken from a set $S \setminus t$, where t is a some simple transition.
2. $s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Box_k} s_1, s_2 \ldots s_n \wedge s \xrightarrow{\alpha_2,\ldots,\alpha_n}_{\Diamond_k} s_2 \ldots s_n$ means any product of the family should have *exactly* k of the n transitions $s \xrightarrow{\alpha_i} s_i$, this defines the relation $\Box \cap \Diamond$ as the relation *exactly k of n.*
3. if $k = n$ then $s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Box_n} s_1, s_2 \ldots s_n \Rightarrow s \xrightarrow{\alpha_1,\alpha_2,\ldots,\alpha_n}_{\Diamond_n} s_1, s_2 \ldots s_n$. In effect products which satisfy the property "at least n of n" for some set of transitions are products which have got exactly n transitions, whereas products which satisfy the property "at most n of n" for some set of transitions are all possible products.

Finally, note that every GEMTS has one single initial state. In the definition of MTS or DMTS or 1MTS initial states are not present because we implicitly assume them. In the GEMTS case, on the contrary, we require a restriction on initial states: we may not have a generic set of initial states, we must have a single initial state.

Definition 2.27:
A MTS is a particular GEMTS $M = (\mathcal{S}, \Sigma, \Box, \Diamond, s_0)$ where:

1. $\forall s \in \mathcal{S}.\ (s, T, k) \in \Diamond \Rightarrow |T| = 1 \wedge k = 1$, note that T is singleton
2. $\forall s \in \mathcal{S}.\ (s, T, k) \in \Box \Rightarrow |T| = 1 \wedge k = 1$, note that T is singleton

■

Note that the property (3), which we have just described for a GEMTS, in a MTS is equivalent to the consistency requirement.
In the Figure 2.8 we can see the relation between GEMTS and one of the other models.

modality	MTS	DMTS	1-MTS	EMTS	GEMTS
$\diamond$ (may)	at most 1-of-1	at most n-of-n	at most 1-of-n	at most 1-of-n	at most k-of-n
$\Box$ (must)	at least 1-of-1	at least 1-of-n	exactly 1-of-n	at least 1-of-n	at least k-of-n

Figure 2.8: Expressivity relationship between GEMTS and other models

In [25] Fantechi and Gnesi defined another model, the **Extended Modal Transition System (EMTS)** which is a special case of GEMTS:

Definition 2.28 (Extended Modal Transition System (EMTS)):
A *Extended Modal Transition System (EMTS)* is a particular GEMTS $M = (\mathcal{S}, \Sigma, \Box, \Diamond, s_0)$ where:

1. $\forall s \in \mathcal{S}.\ (s, T, k) \in \Diamond \Rightarrow k = 1$
2. $\forall s \in \mathcal{S}.\ (s, T, k) \in \Box \Rightarrow k = 1$

■

The refinement relation defined in [26] is a restriction of a generic refinement relation, in effect it describes only the connection between a product (or LTS) and a specification (or GEMTS), all intermediate steps of the refinement process are ignored.

Definition 2.29:
Let $P = (S_P, \Sigma, \longrightarrow_P, s_{P_0})$ be a LTS. We say P belongs to the family $(GEMTS)$ $F = (S_F, \Sigma, \Box, \Diamond, s_{F_0})$ if and only if $(s_{P_0}, s_{F_0}) \in \mathcal{R}$ where $(s, t) \in \mathcal{R}$ if and only if:

1. $t \xrightarrow{\alpha_1,\ldots\alpha_n}_{\Box_k} t_1 \ldots t_n \Rightarrow \exists I \subseteq \{1 \ldots n\}.\ k \leq |I| \leq n \ \wedge\ \forall i \in I.\ s \xrightarrow{\alpha_i} s_i \wedge (s_i, t_i) \in \mathcal{R}$
2. $t \xrightarrow{\alpha_1,\ldots\alpha_n}_{\Diamond_k} t_1 \ldots t_n \Rightarrow \nexists I \subseteq \{1 \ldots n\}.\ k < |I| \leq n \wedge \forall i \in I.\ s \xrightarrow{\alpha_i} s_i \ \wedge\ (s_i, t_i) \in \mathcal{R}$
3. $s \xrightarrow{\alpha} s' \Rightarrow \exists k, U \subseteq \Sigma \times S_F, t' \in S_F.\ (\alpha, t') \in U \wedge ((t, U, k) \in \Box \vee (t, U, k) \in \Diamond) \wedge (s', t') \in \mathcal{R}$

■

We can also say P is a product of F or P conforms to F.

Example 2.3. *Consider the requirement of our vending machine: "The choice of drinks (coffee, tea, cappuccino) varies between the products. However, every product of the family delivers at least two different drinks". Then we can model the requirement using the GEMTS as showed in Figure 2.9.*

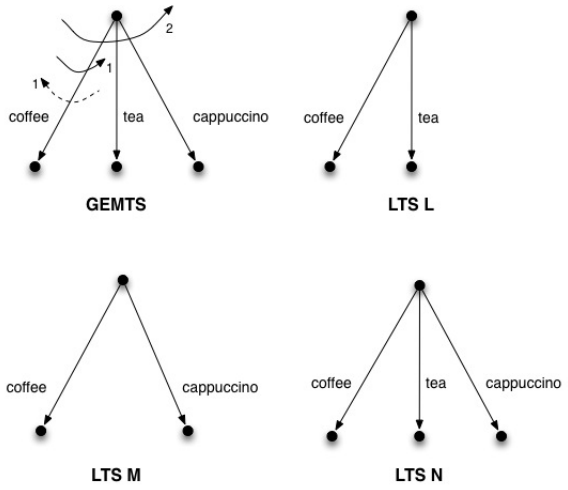


Figure 2.9: An example of GEMTS and its implementations

As we can see the LTSs L, M and N are all possible implementations of our GEMTS.

2.2.6 OTS

Until now we have been seeing all models which use and generalize the concepts of may (or possible) and must (or necessary) transition. As we will see, the last two models take advantage of a new approach: the **obligation formula**. The main idea is to describe only the necessary requirements, using a logic formula. The models which we can derive are called OTS [12], formalized by Beneš and Křetínký.

The first important concept is the obligation formula:

Definition 2.30 (**Obligation formula syntax**)**:**
A *positive boolean formula* over set X of atomic propositions is given by the following syntax:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \tag{2.1}$$

where $x \in X$. The set of all positive boolean formulae over X is denoted as $\mathcal{B}^+(X)$. ■

Definition 2.31 (**Obligation formula semantics**)**:**
The semantics of φ, denoted like $[\![\varphi]\!]$, is a subset of subsets of X satisfying φ and it is inductively defined:

- $[\![x]\!] = \{Y \subseteq X \mid x \in Y\}$
- $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$
- $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$
- $[\![tt]\!] = \mathcal{P}(X)$
- $[\![ff]\!] = \emptyset$

■

We can deduce, therefore, that the obligation formula is a typical logic formula and its semantics is a set of possible subsets of X and each element of this set satisfies the formula.

Definition 2.32 (**Transition System with Obligations (OTS)**)**:**
A *Transition System with Obligations (OTS)* is a tuple $(\mathcal{S}, \Sigma, \dashrightarrow, \mathbf{\Omega})$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of actions
- $\dashrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the may transition relation
- $\mathbf{\Omega} : \mathcal{S} \longrightarrow \mathcal{B}^+(\Sigma \times \mathcal{S})$ is the set of obligations

We denote the set of all possible OTSs by $\mathbb{OTS}$.

■

Note that the formulae **tt** and **ff** are never needed as proper subformulae of any other formula, they are used to specify, respectively, all possible implementations, useful to describe a may transition, and the lack of possible implementations, useful to describe a inconsistent specification.

Moreover, we can impose a *consistency requirement*:

$$\mathbf{\Omega}(s) \neq \mathbf{ff} \text{ and if } \mathbf{\Omega}(s) \text{ contains } (\alpha, t) \text{ then } s \overset{\alpha}{\dashrightarrow} t \tag{2.2}$$

which guarantees that all required behaviours are also allowed.

Using OTS it is possible to describe some of other models:

1. a DMTS is an OTS where the must obligations are in CNF, so an arbitrary OTS can be expressed as DMTS because any logic formula can be translated into a CNF
2. a consistent DMTS is a DMTS which satisfies the consistency requirement
3. a MTS is an OTS where the must obligations are just conjunctions of atomic predicates and it satisfies the consistency requirement
4. a LTS is a MTS such that whenever $s \overset{\alpha}{\dashrightarrow} t$ then $\mathbf{\Omega}(s) = (\alpha, t) \wedge \varphi$ for some φ, that is all behaviours are both allowed and required

The refinement concept is slightly more complicated than the classical definition because we must handle the logic formulae. First of all we define a relation which allows us to relate two formulae.

Definition 2.33:
Let $\mathcal{R} \subseteq X \times X$, let $\varphi, \psi \in \mathcal{B}^+(X)$. We write $\varphi \sqsubseteq_{\mathcal{R}} \psi$ to denote:

$$\forall M \in [\![\varphi]\!].\ \exists N \in [\![\psi]\!].\ \forall n \in N.\ \exists m \in M.\ (m, n) \in \mathcal{R} \tag{2.3}$$

■

Now we can define the refinement of an OTS.

Definition 2.34 (Refinement):
Let $P = (\mathcal{S}, \Sigma, \dashrightarrow, \mathbf{\Omega})$ be an OTS. We say that $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a *refinement relation* if $(s, t) \in \mathcal{R}$ implies:

1. $s \overset{\alpha}{\dashrightarrow} s' \Rightarrow t \overset{\alpha}{\dashrightarrow} t' \wedge (s', t') \in \mathcal{R}$
2. $\mathbf{\Omega}(s) \sqsubseteq_{\Sigma\mathcal{R}} \mathbf{\Omega}(t)$ where $\Sigma\mathcal{R} = \{((\alpha, s), (\alpha, t)) \mid \alpha \in \Sigma, (s, t) \in \mathcal{R}\}$

We say s refines t ($s \trianglelefteq t$) if there is a refinement relation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$

■

Of course a straightforward generalization is possible and in [12] the refinement definition is directly presented in the generalized way.

We say that a process I is an implementation of a specification S if I is a LTS and $(s_0 \trianglelefteq t_0)$ where s_0 is the initial state of I and t_0 is the initial state of S. We denote the set of all implementations of S by $[\![S]\!] = \{I \mid I \text{ is an implementation of } S\}$. Note that this refinement definition coincides with the refinement one on all other models.

Example 2.4. *Consider the requirement of our vending machine: "The choice of drinks (coffee, tea, cappuccino) varies between the products. However, every product of the family delivers at least two different drinks". Then we can model the requirement using the OTS as described in Figure 2.10. For convenience, in the figure the formulae ignore the states, moreover LTS L is described by an OTS. As we can see the LTSs L, M and N are all possible implementations of our OTS.*

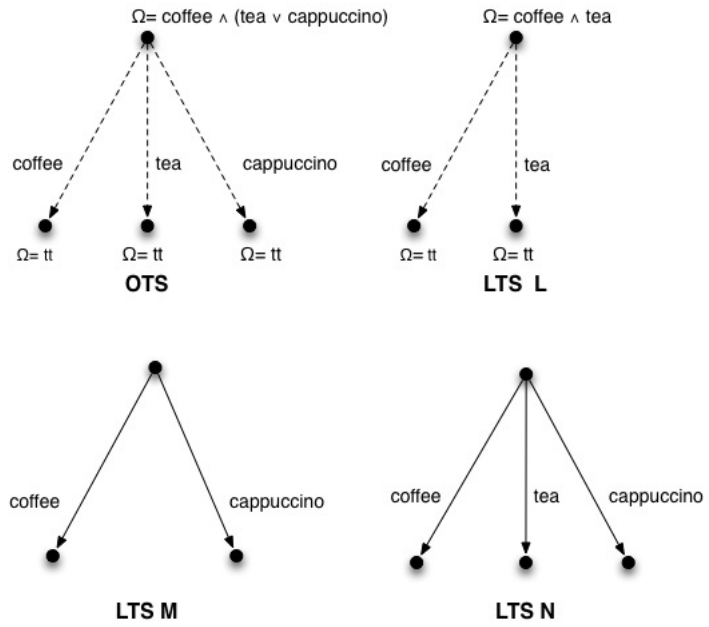


Figure 2.10: An example of OTS and its implementations

2.2.7 PMTS

The OTS formalism is very interesting but several useful requirements cannot be expressed, for example the exclusive requirement. In [13] Beneš, Křetínký, Larsen Møller and Srba introduce a new type of formalism which extends OTS, allowing to model exclusive, conditional and persistent choices. Note that it is the first attempt to describe conditional and persistent requirements.

First of all the logic formula syntax is extended:

Definition 2.35 (**Obligation formula syntax**)**:**
A *boolean formula* over set X of atomic propositions is given by the following syntax:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \tag{2.4}$$

where $x \in X$. The set of all boolean formulae over X is denoted as $\mathcal{B}(X)$.

■

The semantics is modified and extended by the assignment:

Definition 2.36 (**Satisfaction relation**)**:**
Let $\nu \subseteq X$ be a truth assignment, that is a set of elements with value true, and let $\varphi \in \mathcal{B}(X)$ be a logic formula over X. Then the *satisfaction relation* $\models \subseteq \mathcal{P}(X) \times \mathcal{B}(X)$ is defined in the following way:

- $\nu \models \mathbf{tt}$

- $\nu \not\models \mathbf{ff}$
- $\nu \models x \Leftrightarrow x \in \nu$
- $\nu \models \neg\varphi \Leftrightarrow \nu \not\models \varphi$
- $\nu \models \varphi \wedge \varphi_1 \Leftrightarrow \nu \models \varphi$ and $\nu \models \varphi_1$
- $\nu \models \varphi \vee \varphi_1 \Leftrightarrow \nu \models \varphi$ or $\nu \models \varphi_1$

■

Now we can define the semantics of an obligation formula

Definition 2.37 (**Obligation formula semantics**)**:**
The semantics of φ is a subset of subsets of X satisfying φ:

$$[\![\varphi]\!] = \{Y \subseteq X \mid Y \models \varphi\} \tag{2.5}$$

■

Definition 2.38 (**Parametric Modal Transition System (PMTS)**)**:**
A *Parametric Modal Transition System (PMTS)* is a tuple $(\mathcal{S}, \Sigma, \dashrightarrow, \mathcal{P}, \mathbf{\Omega})$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of actions
- $\dashrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the may transition relation
- $\mathcal{P}$ is a finite set of parameters
- $\mathbf{\Omega} : \mathcal{S} \longrightarrow \mathcal{B}((\Sigma \times \mathcal{S}) \cup \mathcal{P})$ is the set of obligations over atomic propositions containing outgoing transitions and parameters

We implicitly assume that whenever $(\alpha, t) \in \mathbf{\Omega}(s)$ then $(s, \alpha, t) \in \dashrightarrow$.

We denote the set of all possible PMTSs by $\mathbb{PMTS}$.

■

Moreover we call PMTS *positive* if, for all $s \in \mathcal{S}$, any negation occurring in $\mathbf{\Omega}(s)$ is applied only to parameters. A PMTS is called *parameter-free* if $\mathcal{P} = \emptyset$. Finally we assume that $T(s) = \{(\alpha, s') \mid (s, \alpha, s') \in \dashrightarrow\}$, $\bigwedge \emptyset = \mathbf{tt}$ and if the obligation function for some state is not listed in the system description then it is implicitly understood as $\mathbf{\Omega}(s) = \bigwedge T(s)$.

As we have seen with OTS, we can use PMTS to describe other models:

1. an OTS is a PMTS parameter-free and positive
2. a DMTS is an OTS where the must obligations are in CNF, for all $s \in \mathcal{S}$

3. a MTS is a DMTS where the must obligations are just conjunctions of atomic predicates, for all $s \in \mathcal{S}$

4. a LTS is a MTS such that for all $s \in \mathcal{S}$ $\mathbf{\Omega}(s) = \bigwedge T(s)$

Now we want to describe the refinement but first we need of a simple definition:

Definition 2.39:
Let $M = (\mathcal{S}, \Sigma, \dashrightarrow, \mathcal{P}, \mathbf{\Omega})$ be a PMTS and $\nu \subseteq \mathcal{P}$ be a truth assignment of parameters. Then for all $s \in \mathcal{S}$, we denote by $Tran_\nu(s) = \{E \subseteq T(s) \mid E \cup \nu \models \mathbf{\Omega}(s)\}$, that is the set of all admissible sets of transitions from s under the fixed truth values of the parameters.

■

We can now define the notion of refinement between PMTSs and we define directly the generalized version.

Definition 2.40 (Refinement):
Let $P_1 = (\mathcal{S}_1, \Sigma, \dashrightarrow_1, \mathcal{P}_1, \mathbf{\Omega}_1)$ and $P_2 = (\mathcal{S}_2, \Sigma, \dashrightarrow_2, \mathcal{P}_2, \mathbf{\Omega}_2)$ be two PMTSs. We say that $\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a *refinement relation* if for each $\nu_1 \subseteq \mathcal{P}_1$ there exists $\nu_2 \subseteq \mathcal{P}_2$ such that for every $(s,t) \in \mathcal{R}$ holds:

$$\forall M \in Tran_{\nu_1}(s).\ \exists N \in Tran_{\nu_2}(t).\ \forall(\alpha, s') \in M.\ \exists(\alpha, t') \in N.\ (s', t') \in \mathcal{R} \wedge \forall(\alpha, t') \in N.\ \exists(\alpha, s') \in M.\ (s', t') \in \mathcal{R}$$

We say s refines t ($s \trianglelefteq t$) if there is a refinement relation $\mathcal{R}$ such that $(s,t) \in \mathcal{R}$

■

Of course the refinement as defined on PMTS coincides with the standard modal refinement notions on MTS, DMTS and OTS. On LTS it coincides with bisimulation.

In Figure 2.11 we can see an example of this model and some steps of refinement: the initial PMTS model has two parameters *reqYfromR* and *reqYfromG* and it is the model on the right. By means of the refinement relation, described by $\leq_m$, we can derive another, more refined PMTS or some LTSs such as ones on the left in the Figure 2.11. Note that the obligation function related to the state *green* requires an exclusive choice between transitions labelled with *stop* and *ready*. In the model it is possible to find a cycle such that, starting our execution from the state *green*, we can return back to the *green* after some steps. When we reach the state *green* we can choose *stop* or *ready* and this choice is non-deterministic and different every time. To guarantee a persistent choice between these two transitions, namely every time we stay in the state *green*, we must always make the same choice, we add the parameter *reqYfromG*. Once we define the value of this parameter then, by means of the obligation function, we have the guarantee that only one transition is taken and it is always the same for every cycle. The same reasoning holds for the state *red*.

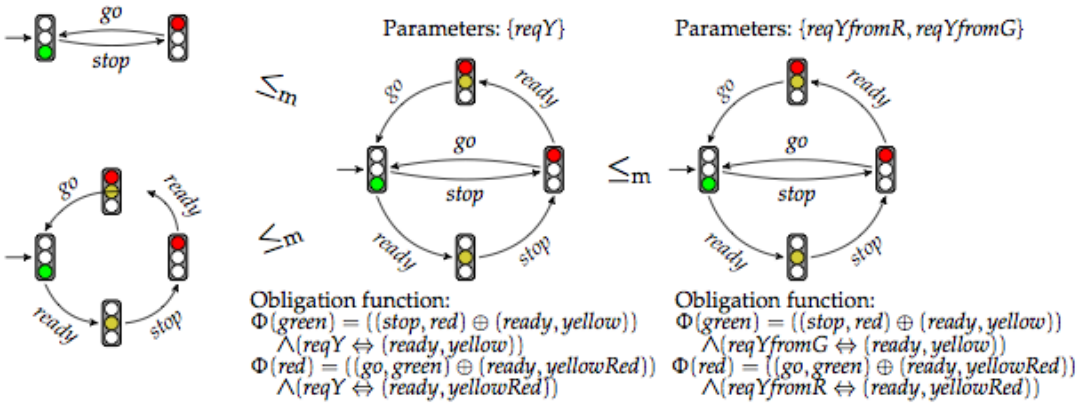


Figure 2.11: An example of PMTS and its implementations

A possible refinement is a PMTS where parameters *reqYfromG* and *reqYfromR* are unified in a single parameter *reqY*. Note that in this way we lose all models derived by the choices *reqYfromG*=**tt**, *reqYfromR*=**ff** and *reqYfromG*=**ff**, *reqYfromR*=**tt**.

Anyway in this example it is possible to understand the importance of persistent choice and when we need it, namely when we have two or more possible outgoing transitions, a cycle and the need to guarantee the same choice among several outgoing transitions in a state every time we reach it.

2.3 Logics

As we said in the introduction of this chapter, we would like to use formal methods with a logical approach, where properties to be checked are described by logic formulae. In the previous section we described some possible models useful to represent product specifications. Now in this section we see some logics known in literature and commonly used to formalize properties to be checked over the model.

Depending on the type of property which we would like to check, we have different types of logics. Note that the choice of properties to be checked influences the type of model used to describe the system too, in effect some characteristics are better emphasized by certain models rather than other ones. For example, in some cases, our concern is the behaviour of the system, which is described by labelled transitions, and the best model is obviously LTS, seeing that the LTS describes as the behaviour of a system evolves by means of states and labelled transitions. An example of these logics is the Hennessy-Milner Logic. In other context, instead, we want to reason only about properties of reached states, whereas we are not interested absolutely to know what sequences of actions are necessary to reach the state. In general this is typical of modal logics.

These logics extend the propositional logic (or first-order logic) by some new

operators expressing modality. For example in the alethic logic, the operators $\Box$ and $\Diamond$ describe the necessity and possibility, respectively. Other types of modality that can be used are temporal modalities, deontic modalities (which consider the obligation, the prohibition and the permission), epistemic modalities (which consider the knowledge) and doxastic modalities (which consider the belief). The semantics of all these logics is usually formalized by the Kripke Semantics or Possible World Semantics [14] [43].

In Computer Science these logics are widely used, in particular the Temporal Logic, but recently the other logics have been studied in some different context, for example the Deontic Logic in a Product-Line context, the Epistemic Logic in the security context and in distributed systems. Seeing that our interest is in reached states and not transitions, often the LTS model is a poor model for our purposes, in effect in these cases it is used another type of model, called **Kripke Structure**, which was introduced in [18].

Definition 2.41 (**Kripke Structure (KS)**)**:**
Let AP be a set of atomic propositions. A *Kripke Structure (KS)* M over AP is a tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{L})$ where:

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{S}_0$ is a set of initial states
3. $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a transition relation which must be total, that is, for every $s \in \mathcal{S}$ there is a state $s' \in \mathcal{S}$ such that $(s, s') \in \mathcal{R}$
4. $\mathcal{L} : \mathcal{S} \longrightarrow \mathcal{P}(AP)$ is a function that labels each state with the set of atomic propositions true in that state

■

As we can see a KS is a simple extension of a Transition System (TS), obtained by adding a labelling function to the Transition System. This labelling function is very important because it connects each state to some set of properties described by atomic propositions.

When we want to model a behaviour of a system, we do not handle a TS but we use a LTS. In some cases we would like to handle both behaviour aspects of the model and properties of reached states, therefore we must extend the LTS model in a similar way to what we have done in the extension of a TS in a KS, the derived model is called *Doubly Labelled Transition System* (L^2TS) [22].

Definition 2.42 (**Doubly Labelled Transition System**($\mathbf{L^2TS}$))**:**
Let AP be a set of atomic propositions. A *Doubly Labelled Transition System (L^2TS)* M over AP is a tuple $(\mathcal{S}, \Sigma, \longrightarrow, \mathcal{L})$ where:

1. $\mathcal{S}$ is a finite set of states

2. Σ is a set of actions

3. $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation

4. $\mathcal{L} : \mathcal{S} \longrightarrow \mathcal{P}(AP)$ is a labelling function that associates a set of atomic propositions to each state.

■

Note that this formalism is so really important in Model Checking techniques that sometimes it is the only formalism described, like in [6] where L^2TS is directly called Transition System.

Also we introduce a new concept: the trace. We have already seen the concept of path, that is a sequence of states, and execution, that is an alternating sequence of states and actions. Actions are mainly used to model the behaviour of models, but this is not our unique interest in KS and L^2TS, in effect we would want to focus on the states that are visited during executions too. Note that our concern is not the state itself but properties related to the state, which are modelled by atomic propositions. It is simple to understand that in these models it is interesting to see all possible, reached atomic propositions, therefore we must also consider a sequence of the form $\mathcal{L}(s_0)\mathcal{L}(s_1)\mathcal{L}(s_2)\ldots$, namely the sequence which describes the set of atomic propositions that are valid along the execution and we may call such sequences **traces**. Moreover we can deduce that a trace is a word over the alphabet $\mathcal{P}(AP)$.

Definition 2.43 (Trace):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathcal{L})$ be a L^2TS over the set AP, that is the set of atomic propositions and let $\pi = s_0 s_1 \ldots s_n$ be a finite path fragment then the *trace* of π, called $trace(\pi)$, is $trace(\pi) = \mathcal{L}(s_0)\mathcal{L}(s_1)\ldots\mathcal{L}(s_n)$.
Let $\pi = s_0 s_1 \ldots$ be an infinite path fragment then the *trace* of π, called $trace(\pi)$, is $trace(\pi) = \mathcal{L}(s_0)\mathcal{L}(s_1)\ldots$.

■

As we can see the concept of trace is very simple and it is easy to be calculated when we know the path of our L^2TS, the definition of trace for a KS is easily derived by the L^2TS one.

Finally, we may say our models are grouped, conceptually, in several categories:

1. **unlabelled system**: systems which have got no labelled transitions and no atomic propositions related to states.

2. **action-based system**: systems which have got labelled transitions and no atomic propositions related to states. Labels describe actions to be executed by our system

3. **state-based system**: systems which have got no labelled transitions but they have got atomic propositions related to states. Every state has a set of atomic propositions which are true in the state itself.

4. **action and state-based system**: systems which have got labelled transitions and atomic proposition related to states. Labels describe executable actions, instead atomic propositions related to a state represent the properties true of such state.

For example TS belongs to the first category, LTS to the second, KS to the third and L^2TS to the last one. In Figure 2.12 we summarize the categories and transfor-

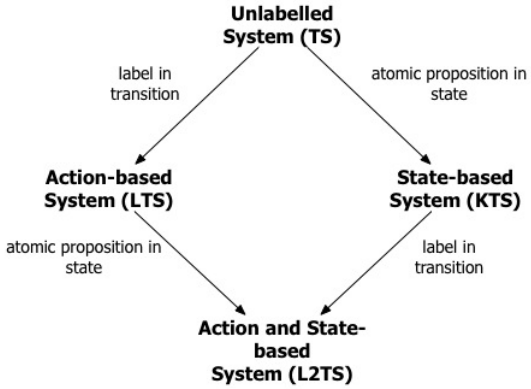


Figure 2.12: The categories of Transition System

mations needed to change a category in another one, also it is possible to define a transformation from KS to LTS and vice versa in a simple way [21].

Of course in literature many different types of logics exist but we will only present the ones useful to understand the logic which we will describe in the Chapter 6. Depending on what we will consider important, that is the behaviour or the current state, we will utilize different models.

Before to introduce these logics, we describe some details about the Deontic Logic. The Deontic Logic is a Modal Logic which extends the propositional logic with the modalities **O** "it is obligatory that", **F** "it is forbidden that" and **P** "it is permitted that". From an axiomatic pointview these operators are related in the following way:

1. $\mathbf{F}\varphi = \mathbf{O}\neg\varphi$, that is "it is forbidden that φ holds" is equivalent to "it is obligatory that φ does not hold"

2. $\mathbf{P}\varphi = \neg\mathbf{O}\neg\varphi$, that is "it is permitted that φ holds" is equivalent to "it is not obligatory that φ does not hold"

3. $\mathbf{O}\varphi \Rightarrow \mathbf{P}\varphi$, that is "it is obligatory that φ holds" implies "it is permitted that φ holds"

From a semantics pointview, specifically the possible-world semantics, these operators are equivalent to:

1. $w \models \mathbf{O}\varphi \Leftrightarrow \forall w'.(w, w') \in \mathcal{R} \Rightarrow w' \models \varphi$, namely given a current world w, w satisfies $\mathbf{O}\varphi$ if and only if all possible world w', which can be reached from w, satisfies φ

2. $w \models \mathbf{F}\varphi \Leftrightarrow \forall w'.(w, w') \in \mathcal{R} \Rightarrow w' \not\models \varphi$, namely given a current world w, w satisfies $\mathbf{F}\varphi$ if and only if all possible world w', which can be reached from w, does not satisfy φ

3. $w \models \mathbf{P}\varphi \Leftrightarrow \exists w'.(w, w') \in \mathcal{R} \wedge w' \models \varphi$, namely given a current world w, w satisfies $\mathbf{P}\varphi$ if and only if it exists one possible world w' which can be reached from w and satisfies φ

For more details it is possible read [2], [30], [38] and [40].

2.3.1 Hennessy-Milner Logic

Hennessy and Milner in [29] described the first attempt of a logical characterization of the behaviour, which allows us to describe properties of systems in a simple way and to understand if two systems are behaviourally equivalent. In this context we are concerned to reason about properties of the behaviour of systems, where the behaviour can easily be seen as a set of executed actions by the system itself. In order to verify these properties, we can describe a system by means of some kind of model where actions are represented by labelled transitions, in addition note that the possible reached states are not interesting for our purposes. The syntax of Hennessy Milner Logic (HML) is the following:

Definition 2.44 (**Hennessy Milner Logic (HML) Syntax**)**:**

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi \ \wedge \ \varphi_1 \mid \langle\alpha\rangle\varphi$$

where $\alpha \in \Sigma$, that is α is an action.

■

Some other useful syntax notations are:

- $\mathbf{ff} = \neg\mathbf{tt}$
- $\varphi \vee \varphi_1 = \neg(\neg\varphi \wedge \neg\varphi_1)$
- all expressions derived by the classical propositional logic hold
- $[\alpha]\varphi = \neg\langle\alpha\rangle\neg\varphi$

As we can see this logic is simple and it is derived by the classic propositional logic by way of adding the modal operators $\langle . \rangle$ and $[.]$ with the meaning of "*possibility*" and "*necessary*", respectively.

The semantics is described by the satisfaction relation $\models$ that defines if a state of a LTS satisfies a some property.

Definition 2.45 (**Hennessy-Milner Logic (HML) Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be a LTS and ϕ, ϕ_1 be two logic formulae. Then the satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ holds the following:

- $s \models \mathbf{tt}$
- $s \models \neg\phi \Leftrightarrow s \not\models \phi$
- $s \models \phi \wedge \phi_1 \Leftrightarrow s \models \phi$ and $s \models \phi_1$
- $s \models \langle\alpha\rangle\phi \Leftrightarrow \exists s' \in \mathcal{S}.\ s \xrightarrow{\alpha} s' \wedge\ \ s' \models \phi$

■

Depending on these definitions, it holds also:

- $s \models \mathbf{ff} \Leftrightarrow s \not\models \mathbf{tt}$
- $s \models \phi \vee \phi_1 \Leftrightarrow s \models \phi$ or $s \models \phi_1$
- $s \models [\alpha]\phi \Leftrightarrow \forall s' \in \mathcal{S}. s \xrightarrow{\alpha} s' \Rightarrow s' \models \phi$

Moreover in [29] Hennessy and Milner defined a relation between HML formulae and the bisimilarity relation over LTS. In order to describe it we introduce a new concept:

Definition 2.46:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be a LTS and s be a state. Then we denote by $\mathcal{F}(s)$ the set of properties (or formulae) satisfied by the state s, that is $\mathcal{F}(s) = \{\varphi \mid s \models \varphi\}$

■

Definition 2.47 (**Image-Finite LTS**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be a LTS . We call it *image-finite* if and only if $\forall s \in \mathcal{S}, \alpha \in \Sigma$ the set $\{s' \in \mathcal{S} \mid s \xrightarrow{\alpha} s'\}$ is finite.

■

Theorem 2.1 (**Hennessy-Milner Theorem**)**.** *Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be an image-finite LTS and s, t be two states. Then*

$$s \sim t \Leftrightarrow \mathcal{F}(s) = \mathcal{F}(t)$$

Example 2.5. *Consider the LTSs in Figure 2.2. As we could see the LTSs (a) and (b) are bisimilar, whereas (c) is not. Suppose that our HML formula is* $\varphi = \langle a\rangle\langle a\rangle\langle b\rangle[c]\mathbf{tt}$*. Of course all LTSs in the figure satisfy* φ*.*

Now we take the HML $\varphi_1 = \langle a\rangle[b]\mathbf{tt}$*, we can see that in this case all LTSs satisfy* φ_1 *too. Finally, we make a slight change on* φ_1*,* $\varphi_2 = \langle a\rangle\langle b\rangle\mathbf{tt}$*, this time LTSs (a) and (b) satisfy* φ_2*, but (c) does not satisfy* φ_2*, therefore we can conclude that (a) and (c) are not bisimilar surely and the same holds for (b) and (c). To understand if (a) and (b) are bisimilar, we should determine if it exists a HML formula* ϕ *such that (a) satisfy* ϕ *and (b) does not satisfy it (or vice versa), in effect if this formula exists then we can conclude that (a) and (b) are not bisimilar, otherwise the theorem guarantees us that (a) and (b) are bisimilar.*

2.3.2 Computation Tree Logic

The second type of logic which we see is a typical temporal logic widely used in the Model Checking world. Initially Pnueli in [42] introduced a temporal logic for the specification and verification of reactive systems and this logic is called **Linear Temporal Logic (LTL)**. We use the term "*linear*", because the notion of time is path-based and viewed to be linear, that is at each moment of time only one possible successor state exists, so each time moment has a unique possible future. It is simple to understand that the interpretation of LTL formulae is defined in terms of paths. Paths, of course, are derived from a transition system where each state might have several, distinct direct successor states, and thus several computations (path) may derive by a state. The reason about this situation is that our system may be branching. Unfortunately, a LTL formula φ is satisfied in a state s if and only if all possible computations that start in s satisfy φ, that is the LTL implicitly assumes the universal quantification over all possible computations. Obviously, this requirement is very strong because sometimes we want to know if some property is verified for only some possible computations (existential interpretation).

In [17], [23] Clarke and Emerson introduced a new type of logic which allows us to handle formulae with an existential interpretation too. To handle this situation the interpretation of time must be changed, now time is not linear but branching, that is the time is not an infinite sequence of states anymore but it is an infinite tree of states. The main idea of the branching time is that at each moment there may be several different possible futures, each moment of time has several choice and so several possible futures. For this reason the logic is known as **branching temporal logic**. As we will see, the semantics of a branching temporal logic is based on the usage of an infinite, directed tree of states rather than an infinite sequence. Each state has different possible (infinite) computations and the tree itself represents all possible computations. Of course we focus on computations with the initial state as the root of tree, in fact each traversal of the tree starting in its root represents a single possible path. We may see this tree as the tree which we can directly obtain from a transition system by "unfolding" of some state. We call this logic

Computation Tree Logic.

Note that in this section we do not present the LTL logic and the reason will be explained later in this section, anyway a good presentation about LTL may be found in [6].

First step of our presentation is the syntax of CTL

Definition 2.48 (**Computation Tree Logic Syntax**)**:**
Let AP be a set of atomic propositions then the syntax of *Computation Tree Logic* formulae is:

$$\varphi ::= \mathbf{tt} \mid p \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \exists\pi \mid \forall\pi$$
$$\pi ::= \mathbf{X}\varphi \mid \varphi \ \mathbf{U} \ \varphi_1$$

where $p \in AP$. We call φ formulae *state formulae* and π formulae *path formulae.*

■

It is possible to derive other formulae such as ff, $\varphi \vee \varphi$ using the negation and the typical equivalences of propositional logic.

The meaning of operators $\exists$ and $\forall$ is the obvious one, on the other hand in path-formulae we introduce two new types of operators:

- **X** is called "*next*" , it is a unary operator and its meaning is that a formula $\mathbf{X}\varphi$ holds in a current state if and only if φ holds in the next state
- **U** is called "*until*", it is a binary operator and its meaning is that a formula $\varphi\mathbf{U}\phi$ holds in a current state if and only if in some future moment ϕ holds and all states until that moment hold φ formula

Furthermore it is possible to add two new types of formulae:

1. $\Diamond\varphi = \mathbf{tt}\ \mathbf{U}\varphi$, also called "*eventually*" and its meaning is that in some moment in the future φ will hold
2. $\Box\varphi = \neg\Diamond\neg\varphi$, also called "*always*" and its meaning is that φ is true from now forever.

Before to see the semantics we introduce some new concepts:

Definition 2.49 (**Path(s)**)**:**
Let s be a generic state of a some kind of transition system. We denote by $Path(s)$ the set of all possible paths with s as initial state. We denote the set of all possible paths by $Path$.

■

Definition 2.50 (**Suffix**)**:**
Let $\sigma = s_0, s_1 \ldots$ be a generic path and let $i \in \mathcal{N}$ be an index. We denote by $\sigma[i] = s_i$ the i-th state of σ, whereas we denote by $suffix(\sigma, i) = \varsigma$ the suffix of the path σ from the i-th state, that is if $\varsigma = t_0, t_1 \ldots$ then $\forall j \in \mathcal{N}.\ \varsigma[j] = t_j = s_{i+j} = \sigma[i+j]$.

■

Since CTL has two types of formulae, path and state-formulae then the semantics of CTL formulae is defined by two satisfaction relations: one for the state-formulae and one for the path-formulae. The semantics can be defined over a KS or a L^2TS, indifferently.

Definition 2.51 (**Computation Tree Logic Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, AP, \mathcal{L})$ be a L^2TS, $s \in \mathcal{S}$ be a state, ϕ, ψ be state formulae and ρ be a path formula. Then the satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ is defined for state-formulae by:

- $s \models tt$
- $s \models p \Leftrightarrow p \in L(s)$
- $s \models \neg\phi \Leftrightarrow s \not\models \phi$
- $s \models \phi \wedge \psi \Leftrightarrow s \models \phi$ and $s \models \psi$
- $s \models \exists\rho \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \rho$
- $s \models \forall\rho \Leftrightarrow \forall\sigma \in Path(s).\ \sigma \models \rho$

For a path σ, the satisfaction relation $\models \subseteq Path \times \pi$ is defined for path-formulae by:

- $\sigma \models \mathbf{X}\phi \Leftrightarrow \sigma[1] \models \phi$
- $\sigma \models \phi\ \mathbf{U}\ \psi \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \psi\ \wedge \forall 0 \leq i < j.\ \sigma[i] \models \phi$

■

Unfortunately, some useful properties, easily described by a LTL formula, cannot be expressed by CTL and it holds the vice versa too. In order to solve this lack Emerson and Halpern in [24] proposed a new, more general logic called CTL*, which combines the features of LTL and CTL.

CTL* allows us the usage of path quantifiers $\exists$ and $\forall$ arbitrarily nested with typical LTL-formulae, in particular with linear temporal operators such as **X** and **U**. For this reason CTL* is an extension of CTL: in CTL, indeed, each linear temporal operator must be immediately preceded by a path quantifier. Finally, as in CTL, the syntax of CTL* distinguishes between state and path formulae: the first ones are the same of CTL, whereas the latter ones are defined as LTL formulae, where CTL* state formulae can be used as atoms too.

Definition 2.52 (**CTL* Syntax**)**:**
Let AP be a set of atomic propositions then the syntax of *CTL** formulae is:

$$\varphi ::= \mathbf{tt} \mid p \mid \neg\varphi \mid \varphi\ \wedge\ \varphi \mid \exists\pi$$
$$\pi ::= \varphi \mid \neg\pi \mid \pi \wedge \pi \mid \mathbf{X}\pi \mid \pi\ \mathbf{U}\ \pi$$

where $p \in AP$.

■

As for CTL we can derive some operators like $\vee$ and other typical logic operators, in addition $\Diamond\pi = \mathbf{tt}\ \mathbf{U}\ \pi$, $\Box\pi = \neg\Diamond\neg\pi$ and $\forall\pi = \neg\exists\neg\pi$ (note that this is not possible in CTL).

Definition 2.53 (**CTL* Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, AP, \mathcal{L})$ be a L^2TS , $s \in \mathcal{S}$ be a state, ϕ, ψ be state-formulae and ρ, ρ_1 be two path-formulae. Then the satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ is defined for state-formulae by:

- $s \models tt$
- $s \models p \Leftrightarrow p \in L(s)$
- $s \models \neg\phi \Leftrightarrow s \not\models \phi$
- $s \models \phi \wedge \psi \Leftrightarrow s \models \phi$ and $s \models \psi$
- $s \models \exists\rho \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \rho$

For a path σ, the satisfaction relation $\models \subseteq Path \times \pi$ is defined for path-formulae by:

- $\sigma \models \phi \Leftrightarrow \sigma[0] \models \phi$
- $\sigma \models \neg\rho \Leftrightarrow \sigma \not\models \rho$
- $\sigma \models \rho \wedge \rho_1 \Leftrightarrow \sigma \models \rho$ and $\sigma \models \rho_1$
- $\sigma \models \mathbf{X}\rho \Leftrightarrow suffix(\sigma, 1) \models \rho$
- $\sigma \models \rho\ \mathbf{U}\ \rho_1 \Leftrightarrow \exists j \geq 0.\ suffix(\sigma, j) \models \rho_1\ \wedge \forall 0 \leq i < j.\ suffix(\sigma, i) \models \rho$

■

Example 2.6. *Consider the L^2TSs in Figure 2.13. As we may see these L^2TSs are derived by LTSs in Figure 2.2, adding to each state a set of labels which describes actions of outgoing transitions.*
For convenience we implicitly assume that a fairness condition holds, namely for each state it is guaranteed that for any possible transition eventually it is enabled, without this condition we may be deadlocked in the state with a self-loop labelled by a.

Consider the CTL formula $\varphi = \forall\Diamond C$, that is for all possible paths eventually C holds, obviously all L^2TSs in Figure 2.13 satisfy φ. Now we suppose that $\varphi = \exists\Diamond(A \wedge \neg B \wedge \neg C)$, that is it exists a possible path that eventually only A holds, in this case L^2TSs M and N do not satisfy φ because all their states can be satisfy C or $A \wedge B$, whereas O satisfies φ because it exists a path where eventually only A holds.*

Another possible formula is $\varphi = \forall\Box((A \wedge B) \vee C)$, namely for all possible paths it always holds $A \wedge B$ or C. If we see the Figure 2.13, we realize that L^2TSs M and

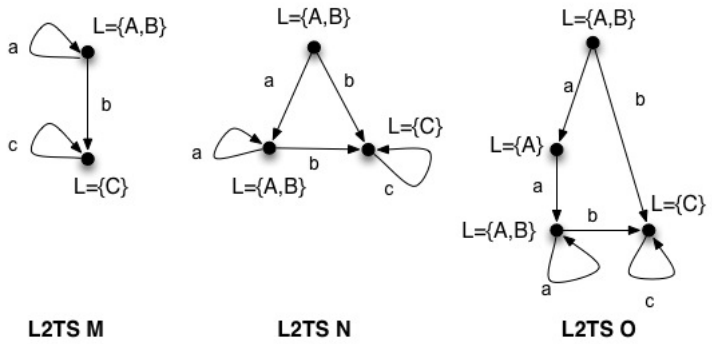


Figure 2.13: Examples of L^{2}TSs

N satisfy φ because all their states can be satisfy C or $A \wedge B$, on the other hand O does not satisfy φ because it exists a path where only A holds.

Finally we consider a formula $\varphi = \forall\mathbf{X}((A \wedge B) \vee C)$, that is it for all possible paths the next state holds $A \wedge B$ or C. This time, L^2 TSs M and N satisfy φ because for all possible paths the states reached satisfy C or $A \wedge B$, instead O does not satisfy φ because it exists a path where the next state reached by this path does not hold C.

2.3.3 ACTL*

De Nicola and Vaandrager in [21] introduced a new type of logic: an action-version based of CTL* interpreted over LTSs. They note that LTSs have been widely used to interpret process algebra and to handle communicating systems, on the other hand KS is the common model for handling many modal logics such as the temporal ones. Also it is clear that KS and LTS are two partially different models, so it becomes interesting to know if it is possible to find a connection between the logic interpreted over KS and the one interpreted over LTS. Finally in [21] De Nicola and Vaandrager showed two transformations: one from CTL* to ACTL* and the other one from ACTL* to CTL*, proving that ACTL* is expressively equivalent to CTL*. This property holds for their restrictions ACTL and CTL too.

ACTL* is derived by CTL*, simply removing atomic propositions on state-formulae and adding a new operator to path-formulae: $\mathbf{X}_\alpha$.

Definition 2.54 (ACTL* Syntax):
Let Σ be a set of actions then the syntax of *ACTL** formulae is:

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \exists\pi$$
$$\pi ::= \varphi \mid \neg\pi \mid \pi \wedge \pi \mid \mathbf{X}\pi \mid \mathbf{X}_\alpha\pi \mid \pi \ \mathbf{U} \ \pi$$

where $\alpha \in \Sigma$.

■

Definition 2.55 (**ACTL* Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be a LTS, $s \in \mathcal{S}$ be a state, ϕ, ψ be two state-formulae and ρ, ρ_1 be two path-formulae. Then the satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ is defined for state formulae by:

- $s \models \mathbf{tt}$
- $s \models \neg\phi \Leftrightarrow s \not\models \phi$
- $s \models \phi \wedge \psi \Leftrightarrow s \models \phi$ and $s \models \psi$
- $s \models \exists\rho \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \rho$

For a path σ, the satisfaction relation $\models \subseteq Path \times \pi$ is defined for path formulae by:

- $\sigma \models \phi \Leftrightarrow \sigma[0] \models \phi$
- $\sigma \models \neg\rho \Leftrightarrow \sigma \not\models \rho$
- $\sigma \models \rho \wedge \rho_1 \Leftrightarrow \sigma \models \rho$ and $\sigma \models \rho_1$
- $\sigma \models \mathbf{X}\rho \Leftrightarrow suffix(\sigma, 1) \models \rho$
- $\sigma \models \mathbf{X}_\alpha\rho \Leftrightarrow (\sigma[0], \alpha, \sigma[1]) \in \longrightarrow \wedge\ suffix(\sigma, 1) \models \rho$
- $\sigma \models \rho\ \mathbf{U}\ \rho_1 \Leftrightarrow \exists j \geq 0.\ suffix(\sigma, j) \models \rho_1\ \wedge \forall 0 \leq i < j.\ suffix(\sigma, i) \models \rho$

■

As we can see, the new operator $\mathbf{X}_\alpha$ is similar to $\mathbf{X}$ but in addition it requires that the transition, executed by our path, has the label equals to α.

From this syntax is possible to derive some other operators like $\mathbf{ff}, \vee, \Diamond, \Box$ and so on. Also now we can define the operator $\mathbf{X}_\tau\varphi = \mathbf{X}\varphi \wedge \neg \bigvee\{\mathbf{X}_\alpha\varphi \mid \alpha \in \Sigma\}$, where τ describes the "invisible" action and $\bigvee\{\varphi_i \mid i \in [1 \ldots n]\} = \varphi_1 \vee \ldots \vee \varphi_n$.

In order to define more powerful modalities which will significantly shorten the notation, a simple auxiliary logic of actions is introduced:

Definition 2.56 (**Action formulae**)**:**
Let Σ the set of actions then an action formula γ is generated by:

$$\gamma ::= \alpha \mid \neg\gamma \mid \gamma \wedge \gamma$$

where $\alpha \in \Sigma$. We write $\mathbf{tt}$ for $\neg(\alpha_0 \wedge \neg\alpha_0)$ where α_0 is a generic action. All other operators like $\mathbf{ff}, \vee$ and so on can be introduced in the classic way.

■

Definition 2.57 (**Satisfaction relation**)**:**
Let Σ be the set of actions, λ, λ_1 be two action formulae and β be an action then the *satisfaction relation* $\models \subseteq \Sigma \times \gamma$ is inductively defined as:

- $\beta \models \alpha \Leftrightarrow \beta = \alpha$
- $\beta \models \neg\lambda \Leftrightarrow \beta \not\models \lambda$
- $\beta \models \lambda \wedge \lambda_1 \Leftrightarrow \beta \models \lambda$ and $\beta \models \lambda_1$

■

Definition 2.58 (**Derived modalities**)**:**
Let λ, λ_1 be two action formulae, φ, φ_1 be two ACTL* formulae and α be an action then we can introduce some useful modalities by using the action formulae:

- $\mathbf{X}_\tau\varphi = \mathbf{X}\varphi \wedge \neg \bigvee\{\mathbf{X}_\alpha\varphi \mid \alpha \in Act\}$
- $\mathbf{X}_\lambda\varphi = \bigvee\{\mathbf{X}_\alpha\varphi \mid \alpha \in \Sigma \wedge \alpha \models \lambda\}$
- $\varphi_\lambda \mathbf{U}_{\lambda_1} \varphi_1 = (\varphi \wedge (\mathbf{X}_\tau \mathbf{tt} \vee \mathbf{X}_\lambda \mathbf{tt}))\ \mathbf{U}\ (\varphi \wedge \mathbf{X}_{\lambda_1}\varphi_1)$
- $\varphi_\lambda \mathbf{U} \varphi_1 = (\varphi \wedge (\mathbf{X}_\tau \mathbf{tt} \vee \mathbf{X}_\lambda \mathbf{tt}))\ \mathbf{U}\ \varphi_1$
- $\varphi\langle\alpha\rangle\varphi_1 = \exists(\varphi_{\mathbf{ff}}\ \mathbf{U}_\alpha \varphi_1)$
- $\varphi\langle\epsilon\rangle\varphi_1 = \exists(\varphi_{\mathbf{ff}}\ \mathbf{U}\ \varphi_1)$
- $\langle\alpha\rangle\varphi = \mathbf{tt}\langle\alpha\rangle\varphi$
- $[\alpha]\varphi = \neg\langle\alpha\rangle\neg\varphi$

where ϵ is the empty string. ■

The restriction ACTL, instead, has the following syntax:

Definition 2.59 (**ACTL Syntax**)**:**
Let Σ be a set of actions, γ, γ_1 be two action formulae then the syntax of *ACTL* formulae is:

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi\ \wedge\ \varphi \mid \exists\pi \mid \forall\pi$$
$$\pi ::=\ \mathbf{X}_\gamma\varphi \mid \mathbf{X}_\tau\varphi \mid \varphi_\gamma \mathbf{U}_{\gamma_1}\ \varphi \mid \varphi_\gamma \mathbf{U}\ \varphi$$

■

2.3.4 Hennessy Milner Logic over MTS

In preceding sections we saw some logics interpreted over a LTS or a KS and, even though they are partially different like formalisms, anyway they have got a common structure: a set of states and only one transition relation, that is the type of transitions may be different between several formalisms, for example in KS $\longrightarrow \subseteq \mathcal{S} \times \mathcal{S}$ and in LTS $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$, but each model has a unique type of transition relation.

In [15], [34] Larsen defined a restriction of HML, that is a HML without negation and described a semantics over MTS.

Definition 2.60 (**Hennessy-Milner Logic (HML) Syntax**)**:**

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid \varphi \wedge \varphi_1 \mid \varphi \ \vee \ \varphi_1 \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi$$

where $\alpha \in \Sigma$. ■

The semantics is described by the satisfaction relation $\models$ that defines if a state of a MTS satisfies a some property.

Definition 2.61 (**Hennessy Milner Logic (HML) Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ be a MTS and φ, φ be two logic formulae. Then the satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ holds the following:

- $s \models \mathbf{tt}$
- $s \not\models \mathbf{ff}$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \varphi \vee \varphi_1 \Leftrightarrow s \models \varphi$ or $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Box} s' \wedge \ s' \models \varphi$
- $s \models [\alpha]\varphi \Leftrightarrow \forall s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow \ s' \models \varphi$

■

Note that the equivalence $\langle\alpha\rangle\varphi = \neg[\alpha]\neg\varphi$ is no more true, whereas if M is a LTS then this satisfaction relation is the same of HML over LTS, described previously.

As for the HML over LTS, we can find a relation between the HML logic over MTS and the refinement relation:

Definition 2.62:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ be a MTS and s be a state. Then we denote by $\mathcal{F}(s)$ the set of formulae satisfied by the state s, that is $\mathcal{F}(s) = \{\varphi \mid s \models \varphi\}$

■

Theorem 2.2. *Let s, t be two states of a MTS M. Then $s \trianglelefteq t \Leftrightarrow \mathcal{F}(s) \subseteq \mathcal{F}(t)$*

Unfortunately, we cannot add the classical negation formula and the reason is simple, since any $\mathcal{F}(s)$ of any specification is *complete* with respect to the negation, that is for any formula φ either $\varphi \in \mathcal{F}(s)$ or $\varphi \notin \mathcal{F}(s)$. Therefore $\mathcal{F}(s) \subseteq \mathcal{F}(t) \Leftrightarrow \mathcal{F}(s) = \mathcal{F}(t)$.

2.3.5 MHML

Another type of logic interpreted over MTS is MHML, a particular action-based and branching-time temporal logic based on the "Hennessy-Milner logic with Until" [22] [33], defined by Asirelli, ter Beek, Fantechi and Gnesi in [3] [4]. In practice the MHML extends the HML by adding of the until operator and existential and universal state operators (quantifying over paths) like ones described in CTL.

Definition 2.63 (**MHML Syntax**)**:**
Let Σ be a set of actions then the syntax of *MHML* formulae is:

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid E\pi \mid A\pi$$
$$\pi ::= \varphi \ \mathbf{U}\varphi \mid \varphi \ \mathbf{U}^{\Box}\varphi$$

■

The informal meaning of the new non-standard operators of MHML is as follows:

- $\langle\alpha\rangle\varphi$: a next state exists, reachable by a must transition executing action α, in which φ holds
- $[\alpha]\varphi$: in all next states, reachable by any must and may transition executing action α, φ holds
- $\varphi \ \mathbf{U}\varphi_1$: in the current state, or in a future state of a path, φ_1 holds, while φ holds in all preceding states of the path
- $\varphi \ \mathbf{U}^{\Box}\varphi_1$: in the current state, or in a future state of a path, φ_1 holds, while φ holds in all preceding states of the path and the path leading to that state is a must path

Definition 2.64 (**Must Path**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ be a MTS and $\sigma = s_0 s_1 \ldots$ be a path. Then σ is a must path if and only if $\forall i \geq 0$. $s_i \xrightarrow{\alpha_i}_{\Box} s_{i+1}$ and it is denoted by $\sigma^{\Box}$.

■

Definition 2.65 (**MHML Semantics**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ be a MTS , σ be a path, π be a path-formula and φ, φ_1 be two state-formulae. Then the satisfaction relation for state-formulae $\models \subseteq \mathcal{S} \times \varphi$ holds the following:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Box} s' \wedge \ s' \models \varphi$

- $s \models [\alpha]\varphi \Leftrightarrow \forall s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow\ s' \models \varphi$
- $s \models E\pi \Leftrightarrow \exists \rho \in Path(s).\ \rho \models \pi$
- $s \models A\pi \Leftrightarrow \forall \rho \in Path(s).\ \rho \models \pi$

The satisfaction relation for path-formulae $\models \subseteq Path \times \pi$ holds the following:

- $\sigma \models \varphi\ \mathbf{U}\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \varphi_1$ and $\forall 0 \leq i < j . \sigma[i] \models \varphi$
- $\sigma \models \varphi\ \mathbf{U}^{\Box}\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma^{\Box}[j] \models \varphi_1$ and $\forall 0 \leq i < j . \sigma^{\Box}[i] \models \varphi$

■

In this case the equation $\langle\alpha\rangle\varphi = \neg[\alpha]\neg\varphi$ does not hold, also these operators can be interpreted in a deontic way, in particular $\langle\alpha\rangle\varphi$ is equivalent to $\mathbf{O}(\alpha)\varphi$ and $\neg[\alpha]\neg\varphi$ is equivalent to $\mathbf{P}(\alpha)\varphi$.

2.3.6 vaCTL

vaCTL, introduced in [5] is an extension of MHML: it adds action formulae and implicitly the deontic operators **O** "*it is obligatory that*" and **P** "*it is permitted that*".

Definition 2.66 (**vaCTL Syntax**)**:**
Let Σ be a set of actions, γ, γ_1 be two action formulae then the syntax of *v-ACTL* formulae is:

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi\ \wedge\ \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \langle\alpha\rangle^{\Box}\varphi \mid [\alpha]^{\Box}\varphi \mid E\pi \mid A\pi$$
$$\pi ::= \varphi\{\gamma\}\mathbf{U}\{\gamma_1\}\ \varphi \mid \varphi\{\gamma\}\mathbf{U}^{\Box}\{\gamma_1\}\ \varphi$$

■

The informal meaning of the new operators of vaCTL is as follows:

- $\langle\alpha\rangle^{\Box}\varphi$: a next state exists, reachable by a must transition executing action α, in which φ holds
- $[\alpha]^{\Box}\varphi$: in all next states, reachable by must transitions executing action α, φ holds
- $\varphi\{\gamma\}\mathbf{U}\{\gamma_1\}\varphi_1$: in a state of a path reached by an action satisfying γ_1, φ_1 holds, whereas φ holds in all preceding states and all actions executed meanwhile along the path satisfy γ
- $\varphi\{\gamma\}\mathbf{U}^{\Box}\{\gamma_1\}\varphi_1$: in a state of a path reached by an action satisfying γ_1, φ_1 holds, whereas φ holds in all preceding states and the path leading to that state is a must path along which all actions executed meanwhile satisfy γ

Note that the operators $\langle\alpha\rangle^{\Box}\varphi$ and $[\alpha]^{\Box}\varphi$ represent the classical deontic operators **O** and **P**, respectively.

Before to introduce the semantics we define a new concept:

Definition 2.67:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow)$ be a LTS and $\sigma = s_0 s_1 \ldots$ be a path. For every $i \geq 0$ we denote the i-th action of σ by $\sigma\{i\} = \alpha_i$ such that $(\sigma[i], \alpha_i, \sigma[i+1]) \in \longrightarrow$.

■

Of course this concept is easily extended to MTS.

Definition 2.68 (vaCTL Semantics):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ be a MTS , σ be a path, π be a path formula and φ, φ_1 be two state formulae. Then the satisfaction relation for state-formulae $\models \subseteq \mathcal{S} \times \varphi$ holds the following:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Diamond} s' \wedge\ s' \models \varphi$
- $s \models [\alpha]\varphi \Leftrightarrow \forall s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow\ s' \models \varphi$
- $s \models \langle\alpha\rangle^{\Box}\varphi \Leftrightarrow \exists s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Box} s' \wedge\ s' \models \varphi$
- $s \models [\alpha]^{\Box}\varphi \Leftrightarrow \forall s' \in \mathcal{S}.\ s \xrightarrow{\alpha}_{\Box} s' \Rightarrow\ s' \models \varphi$
- $s \models E\pi \Leftrightarrow \exists \rho \in Path(s).\ \rho \models \pi$
- $s \models A\pi \Leftrightarrow \forall \rho \in Path(s).\ \rho \models \pi$

The satisfaction relation for path formulae $\models \subseteq Path \times \pi$ holds the following:

- $\sigma \models \varphi\{\gamma\}\mathbf{U}\{\gamma_1\}\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \varphi, \sigma\{j\} \models \gamma_1, \sigma[j+1] \models \varphi_1$ and $\forall 0 \leq i < j.\ \sigma[i] \models \varphi, \sigma\{i\} \models \gamma$
- $\sigma \models \varphi\{\gamma\}\mathbf{U}^{\Box}\{\gamma_1\}\varphi_1 \Leftrightarrow \sigma$ is a must path and $\sigma \models \varphi\{\gamma\}\mathbf{U}\{\gamma_1\}\varphi_1$

■

Chapter 3

Constrained Modal Transition System

In this chapter we will introduce a new formalism: **Constrained Modal Transition System**. It is one of main contributions of this Thesis and it is an extension of work and ideas described in [26], where Fantechi and Gnesi introduced the GEMTS formalism.

Reasons about the need of developing of a new formalism are multiple. On the hand we would want to study and understand very well the properties and potentialities of GEMTSs, which could be difficult to see and to realize due to the formalism itself, namely the mathematical description of several concepts. In addition we would like to understand if it is possible to extend GEMTSs, that is if it makes sense adding features to GEMTSs, and how to make it: we should so reason about what features can be added, where these features should be included in the formalism and what consequences are derived by the chosen features, both from a conceptual pointview, namely the ideas represented by the formalism itself, and from a practical pointview, namely how these ideas are described mathematically in the formalism.

On the other hand, whereas GEMTSs follow the classic approach, namely the formalism is defined like a transition system enhanced with two different kinds of transitions (may and must), OTS and PMTS, the last developed models, introduce a new approach, namely an approach based on logic formulae, called *obligation formulae*, which describe needed requirements. Moreover in [12] Beneš and Krětínský describe a process algebra for OTS, enriching the ways to reason about all these systems, obtaining an alternative characterization of them and, at the same time, providing a more compact description language for them. These concepts are very important in the study of specification models from a theoretical pointview, but in the product lines the modeller must often handle requirements as "*exactly two features are requested*" or "*at most four features are requested*" or "*at least two features and at most three features are requested*", therefore the description of these requirements by means of a logic formula, in some cases, could be little intuitive or

the formula itself could hide some requirements. In effect we must reason about the semantics of logic operators, which is described by sets of transitions satisfying the formula hence, in this way, we might see all satisfying sets of transitions but not the property described through these sets.

Example 3.1. *Suppose we have four different features* $F = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$ *and suppose we want to model this requirement* $R =$ *"at least two features of* F *must be present". A first attempt to represent* R *is an obligation formula with all possible implementations described directly, but in this way, of course, we have a very large formula. Another possibility is a "smart" formula like* $\varphi = (\mathcal{A} \wedge (\mathcal{B} \vee \mathcal{C} \vee \mathcal{D})) \vee (\mathcal{B} \wedge (\mathcal{C} \vee \mathcal{D})) \vee (\mathcal{C} \wedge \mathcal{D})$*, effectively it is described the situation where if we choose* $\mathcal{A}$ *then at least one among* $\mathcal{B}, \mathcal{C}, \mathcal{D}$ *must be taken, or if we choose* $\mathcal{B}$ *(and implicitly assume that* $\mathcal{A}$ *is not taken) then at least one between* $\mathcal{C}, \mathcal{D}$ *must be considered or we must take both* $\mathcal{C}$ *and* $\mathcal{D}$*, assuming implicitly that* $\mathcal{A}$ *and* $\mathcal{B}$ *are not present. As we can see, our requirement is correctly described by* φ *but suppose that we only know* φ *then our initial requirement will not be easily deducible from* φ*.*

Now we reason about the requirement $R_1 =$ *"at most three features of* F *must be requested". In this case we could define our formula by means of all possible implementations described directly, as before we have a very large formula again. Anyway, this time, we have a simple and clever formula, which represents the requirement,* $\varphi = (\neg\mathcal{A} \vee \neg\mathcal{B} \vee \neg\mathcal{C} \vee \neg\mathcal{D})$*. In effect any implementation which satisfies* φ *must have one or more missing features. Again our requirement is correctly described by* φ *but it is not easily deducible by* φ*.*

So we would want a formalism more useful than the OTS one from modeler pointview and, at the same time, expressively equivalent to the OTS one.

3.1 CMTS definition

The starting point to develop our new formalism is to find out if a more useful way to represent relations $\Diamond$ and $\Box$ of GEMTSs exists. For this purpose we can note that $\Box$ means "at least k of n" and $\Diamond$ means "at most k of n", so it is simple to realize that $\Box$ describes the minimum number of features required and $\Diamond$ the maximum one.

Therefore starting from this remark, we can introduce some new concepts:

Definition 3.1 (**Choice Set**)**:**
Let $\mathcal{E}$ be a set of elements. Then we call *Choice Set* a set $CS \subseteq \mathcal{E}$ of elements of $\mathcal{E}$ that *may* be chosen.

We denote the set of all possible choice sets of $\mathcal{E}$ by $\mathcal{CS}(\mathcal{E})$.

■

Note that if $\mathcal{E}$ is clear from the context then it is omitted, moreover $\mathcal{CS}(\mathcal{E}) = \mathcal{P}(\mathcal{E})$, namely the powerset of $\mathcal{E}$.

Definition 3.2 (**Constraint**):
Let $\mathcal{E}$ be a set of elements and $\mathcal{N}$ be the set of natural numbers. Then we call *Constraint* a tuple $< CS, [min, max] >$ where:

- $CS \in \mathcal{CS}(\mathcal{E})$ describes a choice set, namely a set of elements which can be chosen
- $[min, max] \in \mathcal{N} \times \mathcal{N}$ is an interval where min describes the minimum number of required elements of CS and max represents the maximum number of required elements of CS

We denote the set of all possible constraints of $\mathcal{E}$ by $Constraints(E)$.

■

Again in this case, if $\mathcal{E}$ is clear from the context then it is omitted, moreover the type of $Constraints(E)$ is $\mathcal{CS}(\mathcal{E}) \times (\mathcal{N} \times \mathcal{N})$.

The meaning of a constraint $< CS, [min, max] >$ related to a set $\mathcal{E}$ of elements is very simple: it requires that any possible subset of $\mathcal{E}$ to be correct **must** have K elements of CS such that $min \leq K \leq max$. Note that we do not require:

- to choose always and only a unique possible subset of CS, we can choose any possible subset of CS provided that its size is included between min and max
- to choose always and only all possible subsets of CS with size $min \leq H \leq max$, we can choose any possible subset of CS with any possible size provided that this size is included between min and max

This idea can be described formally as follows:

Definition 3.3 (**Constraint Satisfaction**):
Let $\mathcal{E}$ be a set of elements, $c =< CS, [min, max] >$ be a constraint and $I \subseteq \mathcal{E}$ a possible set of elements of $\mathcal{E}$. Then we define a *satisfaction relation* $\models \subseteq \mathcal{P}(\mathcal{E}) \times Constraints(\mathcal{E})$ as follows:

$$I \models c \Leftrightarrow min \leq |I \cap CS| \leq max$$

where the operator $|.| : \mathcal{P}(\mathcal{E}) \longrightarrow \mathcal{N}$ is the classic cardinality operator, namely $|S|$ describes the cardinality or the size of a set S.

■

Definition 3.4 (**Constraint Semantics**):
Let $\mathcal{E}$ be a set of elements and $c =< CS, [min, max] >$ be a constraint. Then we denote the semantics of c by $[\![c]\!]$ where:

$$[\![c]\!] = \{I \subseteq \mathcal{E} \mid I \models c\}$$

■

Note that in this case $\emptyset$ and $\{\emptyset\}$ are two different elements: the first one describes a constraint with an empty semantics, the second one represents the semantics with only one set of transitions which satisfies the constraint, that is the empty set of transitions.

Definition 3.5 (Solution):
Let $\mathcal{E}$ be a set of elements, $c =< CS, [min, max] >$ be a constraint and $I \subseteq \mathcal{E}$ be a possible subset of $\mathcal{E}$. We call I *solution* of c if and only if $I \in [\![c]\!]$.

■

First of all, to simplify the notation in the following sections, we introduce some utility functions:

1. $\forall c =< CS, [min, max] >\in Constraints(\mathcal{E})$ we denote the choice set of c by $Choice(c)$, namely $Choice(c) = CS$
2. $\forall c =< CS, [min, max] >\in Constraints(\mathcal{E})$ we denote the interval of c by $Card(c)$, namely $Card(c) = [min, max]$
3. $\forall c =< CS, [min, max] >\in Constraints(\mathcal{E})$ we denote the minimum value of the interval of c by $Card_{min}(c)$, namely $Card_{min}(c) = min$
4. $\forall c =< CS, [min, max] >\in Constraints(\mathcal{E})$ we denote the maximum value of the interval of c by $Card_{max}(c)$, namely $Card_{max}(c) = max$
5. $\forall c \in Constraints(\mathcal{E})$ and $\forall I \subseteq \mathcal{E}$ we denote the number of elements of $Choice(c)$ in I by $\#_c I$, namely $\#_c I = |I \cap Choice(c)|$

Trivially, let $\mathcal{E}$ be a set of elements then the following two properties always hold:

1. if $c \in Constraints(\mathcal{E})$ and $Card(c) = [min, max]$ such that $max < min$ then $[\![c]\!] = \emptyset$, in effect $\forall I \subseteq \mathcal{E}.\ I \not\models c$ because if $max < min$ then an any I cannot hold, at the same time, $min \leq \#_c I$ and $\#_c I \leq max$.
2. if $c, c_1 \in Constraints(\mathcal{E})$ such that
   - $c =< CS, [min, max] >$
   - $c_1 =< CS, [min, max_1] >$
   - $|CS| < max$
   - $max_1 = |CS|$

   then $[\![c]\!] = [\![c_1]\!]$.

   The reason is simple: from set theory we know that let A, B be two sets then it holds:

   - $\emptyset \subseteq A \cap B \subseteq A$

- $\emptyset \subseteq A \cap B \subseteq B$

Therefore, in our case, for any $I \subseteq \mathcal{E}$ it always holds

$$0 \leq \#_c I \leq |CS| = |Choice(c)| = |Choice(c_1)|$$

Note that $\forall I \subseteq \mathcal{E}.\#_c I = \#_{c_1} I$ because $Choice(c) = Choice(c_1)$.

Trivially, for hypothesis,

$$I \in [\![c_1]\!] \Leftrightarrow min \leq \#_{c_1} I \leq max_1 \Rightarrow min \leq \#_c I \leq max \Leftrightarrow I \in [\![c]\!]$$

If $I \in [\![c]\!]$ then surely $\#_c I \leq |CS| = max_1$ because this property always holds for every I, so we can say:

$$I \in [\![c]\!] \Leftrightarrow min \leq \#_c I \leq max \Rightarrow min \leq \#_{c_1} I \leq max_1 \Leftrightarrow I \in [\![c]\!]$$

The two properties are very important because they allow to reduce the number of useful constraints which we can use to model requirements.

Definition 3.6 (**Correct Constraint**)**:**
Let $c =< CS, [min, max] >$ be a possible constraint. Then we say that c is a *correct constraint* if and only if holds:

$$0 \leq min \leq max$$

■

From now on, we will assume implicitly that our constraints are correct. The next step is to understand how to extend the semantics concept to the set of constraints: the simple idea is that when we have a set S of constraints, we want every constraint in S is satisfied.

Definition 3.7 (**Semantics of a Set of Constraint**)**:**
Let $\mathcal{E}$ be a set of elements and S be a set of constraints. Then we denote the semantics of S by $[\![S]\!]$ where:

$$[\![S]\!] = \{I \subseteq \mathcal{E} \mid \forall c \in S.\ I \models c\}$$

■

Trivially, the definition of solution can be extended to a set of constraints:

Definition 3.8 (**Solution**)**:**
Let $\mathcal{E}$ be a set of elements, S be a set of constraints and $I \subseteq \mathcal{E}$ be a possible subset of $\mathcal{E}$. We call I *solution* of S if and only if $I \in [\![S]\!]$.

■

Theorem 3.1. *Let $\mathcal{E}$ be a set of elements and S be a set of constraints, such that $S \subseteq Constraints(\mathcal{E})$. Then:*

$$[\![S]\!] = \bigcap_{c \in S} [\![c]\!]$$

Proof.
From Definition 3.7 we know that

$$I \in [\![S]\!] \Leftrightarrow \forall c \in S.\ I \in [\![c]\!] \Leftrightarrow \bigwedge_{c \in S} I \in [\![c]\!]$$

for classic semantics of $\forall$. So we can easily conclude that

$$\bigwedge_{c \in S} I \in [\![c]\!] \Leftrightarrow I \in \bigcap_{c \in S} [\![c]\!]$$

for typical definition of intersection of sets. □

***Corollary* 3.1:** *The semantics of the union of constraints is equivalent to the intersection of the semantics of single constraints. Formally,*

$$I \in [\![\bigcup_i c_i]\!] \Leftrightarrow I \in \bigcap_i [\![c_i]\!]$$

Let S be a set of constraints and c be a constraint then, as we have just said, $[\![S \cup c]\!] = [\![S]\!] \cap [\![c]\!]$ and from the set theory we can say $[\![S]\!] \cap [\![c]\!] \subseteq [\![S]\!]$. In effect the addition of a new constraint to a set S entails a restriction of the semantics of S and the more restrictive the added constraint is, the larger the number of solutions of S to be deleted is.

Of course, taken a set S of constraints, the most restrictive constraint is a constraint c such that $[\![S]\!] \cap [\![c]\!] = \emptyset$. Trivially constraints with semantics equal to $\emptyset$ are the most restrictive but, depending on the set S, we can also find out many other constraints which are the most restrictive, for example taken S we can consider a constraint c such that $[\![c]\!] = \mathcal{P}(\mathcal{E}) \setminus [\![S]\!]$. In this case obviously $[\![S]\!] \cap [\![c]\!] = \emptyset$.

On the other hand, the less restrictive constraint is a constraint c such that $[\![S]\!] \cap [\![c]\!] = [\![S]\!]$. Trivially constraints with semantics equal to $\mathcal{P}(\mathcal{E})$, where $\mathcal{E}$ is the set of elements which can be considered, are the less restrictive. Again, we can also find out several other constraints which are the less restrictive, for example, taken S, any constraint c such that $[\![S]\!] \subseteq [\![c]\!]$ is the less restrictive, indeed $[\![S]\!] \cap [\![c]\!] = [\![S]\!]$.

Definition 3.9 (**More-restrictive and non-restrictive**)**:**
Let S be a set of constraints, such that $[\![S]\!] \neq \emptyset$, and c be a constraint. Then we call c:

- *more-restrictive* for S if and only if $[\![S]\!] \cap [\![c]\!] = \emptyset$

- *non-restrictive* for S if and only if $[\![S]\!] \cap [\![c]\!] = [\![S]\!]$

We call c:

- *general more-restrictive* if and only if $\forall S \subseteq Constraints(\mathcal{E}).\ [\![S]\!] \cap [\![c]\!] = \emptyset$
- *general non-restrictive* if and only if $\forall S \subseteq Constraints(\mathcal{E}).\ [\![S]\!] \cap [\![c]\!] = [\![S]\!]$

■

Note that for set theory, taken a set T:

- $\forall S \subseteq X.\ S \cap T = \emptyset \Leftrightarrow T = \emptyset$
- $\forall S \subseteq X.\ S \cap T = S \Leftrightarrow T = \mathcal{P}(X)$

For now we focus on general non-restrictive constraints and we try to understand what characteristics these constraints must have.

Theorem 3.2. *Let $c =< CS, [min, max] >$ be a constraint. Then c is the general non-restrictive constraint $\Leftrightarrow min = 0 \ \wedge \ max = |CS|$.*

Proof.
Case $\Rightarrow$): c is the general non-restrictive constraint if and only if $[\![c]\!] = \mathcal{P}(\mathcal{E})$. Suppose for absurdum that $min \neq 0 \ \vee max \ \neq |CS|$. Suppose $min \neq 0$ but if we take $I \subseteq \mathcal{E}$, such that $|I \cap CS| = 0$, then $I \not\models c$ because $|I \cap CS| < min$, but this is impossible as $I \in [\![c]\!]$.

Suppose $max \neq |CS|$ but if we take $I \subseteq \mathcal{E}$, such that $|I \cap CS| = |CS|$, then $I \not\models c$ because $|I \cap CS| > max$, but also this is impossible as $I \in [\![c]\!]$. So we can deduce $min = 0 \wedge max = |CS|$.
Case $\Leftarrow$): from set theory we know that $\forall I \subseteq \mathcal{E}.\ 0 \leq |I \cap CS| \leq |CS|$, so $\forall I \subseteq \mathcal{E}.\ I \models c$, deducing $[\![c]\!] = \mathcal{P}(\mathcal{E})$. □

So it is clear that a general non-restrictive constraint deletes nothing from semantics pointview, therefore if a general non-restrictive constraint is considered or not it is not important. For these reasons in the following sections we implicitly assume that if a subset T of outgoing transitions has not a constraint explicitly related to it, then it has a implicitly constraint $c_T =< T, [0, |T|] >$.

In addition let S be a set of constraints, we denote the semantics of an empty set of constraints $[\![\emptyset]\!] = \mathcal{P}(\mathcal{E})$, namely it is equivalent to $[\![\{< \emptyset, [0, 0] >\}]\!]$.

Definition 3.10 (Consistency):
Let $\mathcal{E}$ be a set of elements, $c \in Constraints(\mathcal{E})$ a constraint and S be a set of constraints. Then we call:

- *c consistent* if and only if $[\![c]\!] \neq \emptyset$, namely it exists at least one combination of elements such that the constraint is satisfied. This type of consistency is call *Local Consistency*.

- S *consistent* if and only if $[\![S]\!] \neq \emptyset$, namely it exists at least one combination of elements such that every constraint is satisfied. This type of consistency is call *Global Consistency*

. ■

Note that we can derive two properties from these definitions:

Theorem 3.3 (**Global Consistency implies Local Consistency**). *Let S be a set of constraints then it holds:*

$$S \text{ is consistent} \Rightarrow \forall c \in S.\ c \text{ is consistent}$$

Proof.
If S is consistent then:

$$[\![S]\!] \neq \emptyset \Leftrightarrow [\![\bigcup_{c \in S} c]\!] \neq \emptyset$$

For Theorem 3.1 we know that:

$$[\![\bigcup_{c \in S} c]\!] \neq \emptyset \Leftrightarrow \bigcap_{c \in S} [\![c]\!] \neq \emptyset \Rightarrow \forall c \in S.\ [\![c]\!] \neq \emptyset$$

□

The vice versa is not true as we can see in the following example:

Example 3.2. *Suppose that our set $\mathcal{E} = \{a, b, c\}$ and it exists three constraints:*

1. $c_1 =< \{a, b\}, [1, 1] >$
2. $c_1 =< \{b, c\}, [1, 1] >$
3. $c_1 =< \{a, c\}, [1, 1] >$

It is simple to demonstrate that each constraint is consistent:

1. $[\![c_1]\!] = \{\{a\}, \{b\}, \{a, c\}, \{b, c\}\}$
2. $[\![c_2]\!] = \{\{b\}, \{c\}, \{a, c\}, \{a, b\}\}$
3. $[\![c_3]\!] = \{\{a\}, \{c\}, \{a, b\}, \{b, c\}\}$

But the set $S = \bigcup_{i \in [1,3]} c_i$ is not consistent. In effect we can note that $[\![S]\!] = \bigcap_{i \in [1,3]} [\![c_i]\!] = \emptyset$

Furthermore we add a function to handle and describe the outgoing transitions of a state:

Definition 3.11:
Let $s \in \mathcal{S}$ be a state, Σ be a set of actions and $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ be a transition relation. Then we denote the outgoing transitions of s by $Trans(s)$ where:

$$Trans(s) = \{(\alpha, s') \mid (s, \alpha, s') \in \longrightarrow\}$$

We denote the set of all possible outgoing transitions by $\mathfrak{Trans}$.

■

For convenience we represent the constraint c of a state s with $Choice(c) = \emptyset$ like a constraint $c' =< Trans(s), [0, |Trans(s)|] >$, in this way $[\![c]\!] = \mathcal{P}(Trans(s))$.

Now we have enough information to present the new formalism: **Constrained Modal Transition System**.

Definition 3.12 (**Constrained Modal Transition System**)**:**
A *Constrained Modal Transition System* is a tuple $(\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathfrak{C} : \mathcal{S} \longrightarrow \mathcal{P}(Constraints(\mathfrak{Trans}))$ is a function which taken a state s as input returns a set of possible constraints where constraints are defined over outgoing transitions of s

Moreover it holds that:

1. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ c$ is a correct constraint.
2. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \emptyset$.
3. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \subseteq Trans(s)$.
4. $\forall s \in \mathcal{S}.\ \forall c, c_1 \in \mathfrak{C}(s).\ Choice(c) \neq Choice(c_1)$.

We denote the set of all possible CMTS by $\mathbb{CMTS}$.

■

Note that the further conditions mean:

1. "every constraint must be correct"
2. "every constraint must not have an empty choice set"
3. "for every state, any possible constraint must have only outgoing transitions of the state itself as choice set"

4. "for every state, any possible choice set derivable by outgoing transitions must be correlated to a only one constraint"

We call this last property: **uniqueness of choice sets**. These conditions make sense because:

1. if a CMTS have an incorrect constraint then it is surely an inconsistent CMTS
2. if a constraint c has an empty choice set then this means we want to reason about no outgoing transitions and, of course, this makes little sense.
3. if we have a constraint of a certain state s with a choice set such that some transitions are not outgoing transitions of s then it is impossible to satisfy the constraint. From more conceptual pointview, it makes less sense to define a constraint which introduces some restrictions over not-present transitions.
4. if we have several constraints with the same choice set CS then this means we must consider several limitations over transitions of CS then it make more sense if we have only one constraint which describes all these limitations in an equivalent way.

In addition, note that, for every state s, the semantics of s, $[\![s]\!]$, has as type $\mathcal{P}(\mathcal{P}(Trans(s)))$.

Example 3.3. *Suppose we want to describe the requirement: " The choice of drinks (coffee, tea, cappuccino) varies between the products. However, every product of the family delivers coffee, and every product of the family delivers at least two different drinks" of our vending machines. A possible CMTS which describes this requirement could be like one in the Figure 3.1*

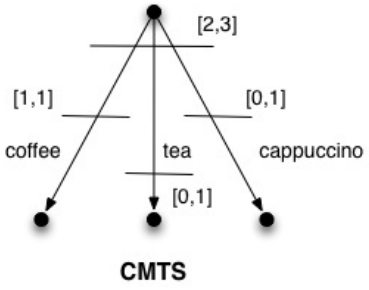


Figure 3.1: An example of CMTS

As we can see, we have a constraint related to all features and it requires that at least two of them and at most three of them must be considered. Moreover we have further singleton constraints: the one related to coffee feature which requires implicitly that the feature must be always considered. The other ones, indeed, require implicitly that the features associated may be considered.

In addition we can easily extend the semantics concept from constraints to states:

Definition 3.13 (**Semantics of a CMTS state**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ be a CMTS and s be a possible state in $\mathcal{S}$. Then we denote the semantics of s by $[\![s]\!]$ where:

$$[\![s]\!] = [\![\mathfrak{C}(s)]\!]$$

■

It is clear that the semantics of a single state s is univocally determined by the semantics of constraints related to s.

Definition 3.14 (**Global and Local Consistency in CMTS**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ be a CMTS and let $s \in \mathcal{S}$ be a state. Then we say that:

- s is *consistent* if and only if $[\![s]\!] \neq \emptyset$. We call this type of consistency *local CMTS consistency.*
- M is *consistent* if and only if $\forall s \in \mathcal{S}.\ [\![s]\!] \neq \emptyset$. We call this type of consistency *global CMTS consistency.*

■

Corollary* 3.2** (**Global CMTS Consistency implies Local CMTS Consistency**):*** *Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ be a CMTS. If M is consistent then $\forall s \in \mathcal{S}$. s is consistent*

Example 3.4. *In Figure 3.2 there are four different CMTSs:*

M) is an inconsistent CMTS even though every constraint is consistent. In this case the set of constraints has not a possible solution.

N) is a consistent CMTS, in effect a possible solution of these constraints is the set $I = \{a, b\}$. Note that for convenience I is only a set of actions, from theoretical pointview I should be a set of pairs (α_i, s_i), where $\alpha_i \in \Sigma$ is action and $s_i \in \mathcal{S}$ is a target state.

O) is an inconsistent CMTS because it exists an inconsistent constraint: in this case the constraint is $< \{b, c\}, [3, 3] >$.

P) is an inconsistent CMTS because it exists an incorrect constraint: in this case the constraint is $< \{a, b, c\}, [2, 1] >$.

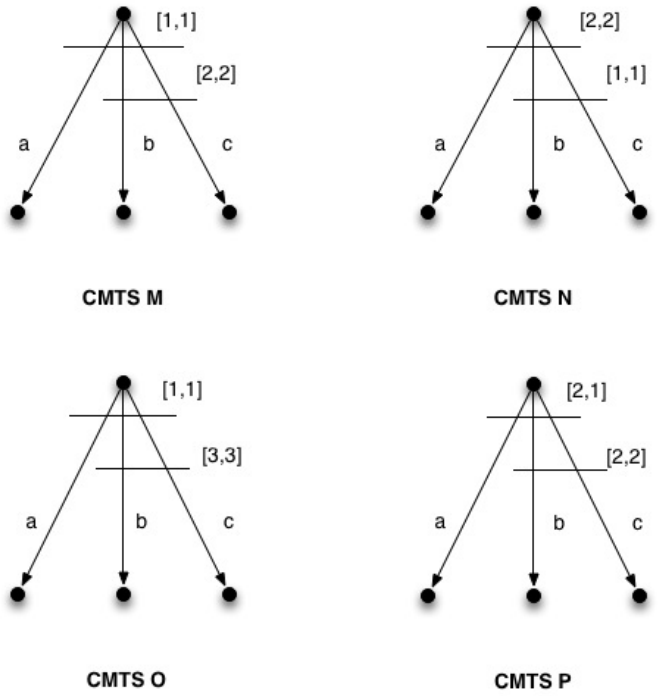


Figure 3.2: Consistency of CMTSs

For our dissertation we will restrict CMTSs to a particular class: the action-deterministic systems. This choice is motivated by the observation that in a real context and in particular in the Product-Lines one, for any state of a possible specification, each feature, which describe a particular characteristic of the product, must be unique. Of course each state has some different features but it is never possible to have two features which described the same characteristic and, at the same time, are different.

Definition 3.15 (**Action-Deterministic CMTS**)**:**
An *Action-Deterministic CMTS* is a tuple $(\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ where:

- $(\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ is a CMTS
- s_0 is the unique initial state

Moreover it holds another property :

$$\forall s \in \mathcal{S}, \alpha \in \Sigma.\ (s, \alpha, s') \in \longrightarrow \wedge\ (s, \alpha, s'') \in \longrightarrow \implies s' = s''$$

■

In the following chapters we will implicitly assume that a CMTS is an action-deterministic CMTS, note that almost all properties which we will describe for

action-deterministic CMTS are easily extendible to the non-deterministic case. In every case where this extension is not so simple or it can create some troubles, we will shortly explain the problem and the possible solutions.

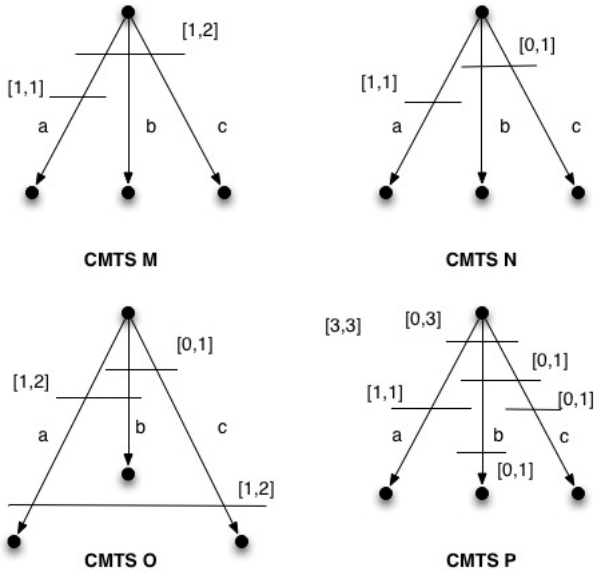


Figure 3.3: Several syntactically different CMTSs but semantically equivalent

Finally, we want to observe a property of CMTSs: taken a CMTS M, it is possible to find some other CMTSs such that they are syntactically different but semantically equivalent to M, namely all CMTSs model the same specification but they describe this specification in a different way. An example of this property can be seen in Figure 3.3.

3.1.1 Constraints study

The new concept of constraint is very simple but it hides several useful properties which could be used to a better understanding of properties of the entire CMTS model. In this subsection we will describe some theorems and properties directly related to the constraint idea.

In the preceding section we introduced the concept of semantics of a constraint but if we want to calculate the semantics of a certain constraint c, we must generate all possible combinations of elements and determine which combinations satisfy c and this method clearly requires a very hard work. Unfortunately, if we will want the exact set of all combinations of elements which satisfy c, this method is the only possible one.

On the other hand, for some problems, we can avoid to calculate the semantics in this way by means of a "smart" use of information described by the constraint as

we will see in next theorems. Note that these theorems are general and independent by the set $\mathcal{E}$ of elements to be considered.

Theorem 3.4 (**Local Consistency**). *Let $c =< CS, [min, max] >$ be a constraint in Constraints($\mathcal{E}$). Then it holds:*

$$[\![c]\!] \neq \emptyset \Leftrightarrow 0 \leq min \leq max \wedge min \leq |CS|$$

Proof.
Case $\Rightarrow$): $[\![c]\!] \neq \emptyset \Leftrightarrow \exists I \subseteq \mathcal{E}.\ I \models c$, namely $min \leq \#_c I \leq max$. Moreover we know from the set theory $\forall J.\ 0 \leq |J \cap CS| \leq |CS|$. Since $min \leq \#_c I$ and $\#_c I \leq |CS|$, for transitivity we can conclude $min \leq |CS|$.
Case $\Leftarrow$): if $min \leq max$ and $min \leq |CS|$ then we can consider a set $I \subseteq \mathcal{E}$ such that $\#_c I = min$ and this is possible because $min \leq |CS|$. In effect $0 \leq \#_c I \leq |CS|$, so if $min > |CS|$ then $\#_c I = min$ is impossible for any possible I. Seeing that $min \leq \#_c I \leq max$, we can deduce $I \in [\![c]\!]$. □

We can deduce a very important corollary:

***Corollary* 3.3:** *If a constraint c is local consistent then it is correct.*

The vice versa is not true: for example we could have a constraint $c =< CS, [min, max] >$ such that:

1. $0 \leq min \leq max \wedge |CS| < min$, so $[\![c]\!]\emptyset$ and we can deduce that c is not local consistent

2. seeing that $0 \leq min \leq max$, then c is correct

Theorem 3.5 (**Constraints Inclusion**). *Let $c =< CS_c, [min_c, max_c] >, c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >$ be two consistent constraints in Constraints($\mathcal{E}$). If $CS_c = CS_{c_1}$ then it holds:*

$$[\![c]\!] \subseteq [\![c_1]\!] \Leftrightarrow (min_{c_1} \leq min_c \leq max_c \leq max_{c_1})$$

Proof.
We know that $I \in [\![c]\!] \Leftrightarrow I \models c \Leftrightarrow min_c \leq \#_c I \leq max_c$ and the same holds for c_1. Moreover $CS_c = CS_{c_1}$, for hypothesis, so we have that $\forall I \subseteq \mathcal{E}.\ \#_c I = \#_{c_1} I$ because $\#_c I = |I \cap CS_c| = |I \cap CS_{c_1}| = \#_{c_1} I$.
Case $\Rightarrow$): Suppose true $[\![c]\!] \subseteq [\![c_1]\!]$ then we know that $\forall I \subseteq \mathcal{E}.\ I \models c \Rightarrow I \models c_1$. Consider $\#_c I = min_c$ then $I \models c$ and, for hypothesis, $I \models c_1$. Since we know that $\#_c I = \#_{c_1} I$ we have that $I \models c_1 \Leftrightarrow min_{c_1} \leq \#_{c_1} I = \#_c I = min_c \leq max_{c_1}$. So we can conclude that $min_{c_1} \leq min_c \leq max_{c_1}$. By using the same reasoning with max_c we can deduce that $min_{c_1} \leq max_c \leq_{max_{c_1}}$. In addition, for correctness of constraints, $min_c \leq max_c$, concluding $min_{c_1} \leq min_c \leq max_c \leq max_{c_1}$ is true.
Case $\Leftarrow$): Suppose that $min_{c_1} \leq min_c \leq max_c \leq max_{c_1})$ and $I \in [\![c]\!]$. We know that this is equivalent to $min_c \leq \#_c I \leq max_c$ and, using the hypotheses, we can deduce that $min_{c_1} \leq min_c \leq \#_c I = \#_{c_1} I \leq max_c \leq max_{c_1}$, therefore $I \models c_1$. □

Theorem 3.6 (**Global consistency implies Local consistency**). *Let S be a set of constraints. Then it holds:*

$$[\![S]\!] \neq \emptyset \Rightarrow \forall c \in S.\ [\![c]\!] \neq \emptyset$$

Proof.
We know that $[\![S]\!] = [\![\bigcup_{c\in S} c]\!] = \bigcap_{c\in S} [\![c]\!]$. It is simple to understand $[\![\bigcup_{c\in S} c]\!] \neq \emptyset \Leftrightarrow \bigcap_{c\in S} [\![c]\!] \neq \emptyset$.
Obviously if $\exists c \in S.\ [\![c]\!] = \emptyset$ then $\bigcap_{c\in S} [\![c]\!] = \emptyset$, so we can deduce that $\forall c \in S.\ [\![c]\!] \neq \emptyset$ □

The vice versa is not true, in Figure 3.2 the CMTS (a) has two consistent constraints but their union is not consistent.

***Corollary* 3.4:** *The Local inconsistency implies the Global inconsistency.*

Theorem 3.7 (**Constraints strict inclusion**). *Let $c =< CS_c, [min_c, max_c] >$ and $c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >$ be two consistent constraints in Constraints($\mathcal{E}$). If $CS_c = CS_{c_1}$ then it holds:*

$$\begin{aligned}[\![c]\!] \subset [\![c_1]\!] \Leftrightarrow\ & (min_{c_1} < min_c \leq max_c \leq max_{c_1}) \vee \\ & (min_{c_1} \leq min_c \leq max_c < max_{c_1})\end{aligned}$$

Proof.
For hypothesis, we have $\forall I \subseteq \mathcal{E}.\ \#_c I = \#_{c_1} I$.
Case $\Rightarrow$): In this case we have two properties:

1. $\forall I \subseteq \mathcal{E}.\ I \in [\![c]\!] \Rightarrow I \in [\![c_1]\!]$
2. $\exists I \subseteq \mathcal{E}.\ I \in [\![c_1]\!] \wedge I \notin [\![c]\!]$

The first property describes the constraints inclusion and so, for Theorem 3.5, we know that $[\![c]\!] \subseteq [\![c_1]\!] \Leftrightarrow (min_{c_1} \leq min_c \leq max_c \leq max_{c_1})$.
The second property says us that $\exists I \subseteq \mathcal{E}.\ (min_{c_1} \leq \#_{c_1} I = \#_c I \leq max_{c_1}) \wedge ((\#_c I < min_c) \vee (max_c < \#_c I))$.
Suppose $\#_c I < min_c$, we know that $min_{c_1} \leq \#_{c_1} I = \#_c I \leq max_{c_1}$ so we can deduce $min_{c_1} \leq \#_c I < min_c \leq max_c \leq max_{c_1}$.
On the other hand, if we suppose true $max_c < \#_c I$, then we can deduce $min_{c_1} \leq min_c \leq max_c < \#_c I \leq max_{c_1}$.
Case $\Rightarrow$): Suppose true $((min_{c_1} < min_c \leq max_c \leq max_{c_1}) \vee (min_{c_1} \leq min_c \leq max_c < max_{c_1}))$. If $(min_{c_1} < min_c \leq max_c \leq max_{c_1})$ then $\forall I \subseteq \mathcal{E}.\ min_c \leq \#_c I \leq max_c \Rightarrow min_{c_1} \leq \#_{c_1} I = \#_c I \leq max_{c_1}$. In addition taken $J \subseteq \mathcal{E}$, such that $\#_c J = \#_{c_1} J = min_{c_1}$, then $min_{c_1} \leq \#_c J = \#_{c_1} J = min_{c_1} \leq max_{c_1}$ and $min_{c_1} = \#_c J < min_c$, so we can conclude that $\exists J \subseteq \mathcal{E}.\ J \in [\![c_1]\!] \wedge J \notin [\![c]\!]$.
By using a similar reasoning we can demonstrate the theorem for $(min_{c_1} \leq min_c \leq max_c < max_{c_1})$. □

Theorem 3.8. *Let* $c =< CS_c, [min_c, max_c] >, c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >$ *be two consistent constraints. If* $CS_c = CS_{c_1}$ *then:*

$$[\![c]\!] \cap [\![c_1]\!] = \emptyset \Leftrightarrow (min_{c_1} > max_c) \vee (min_c > max_{c_1})$$

Proof.
Note that if at least one of two constraints is not consistent, then it is simple to verify $[\![c]\!] \cap [\![c_1]\!] = \emptyset$. Moreover the two constraints have the same choice set, so $\forall I \subseteq \mathcal{E}.\ \#_c I = \#_{c_1} I$.
Case $\Rightarrow$): if $[\![c]\!] \cap [\![c_1]\!] = \emptyset$ is true then we may say $\forall I \in [\![c]\!].\ I \notin [\![c_1]\!]$. We know that $I \notin [\![c_1]\!] \Leftrightarrow (\#_{c_1} I < min_{c_1}) \vee (max_{c_1} < \#_{c_1} I)$. Since $\#_c I = \#_{c_1} I$ for any I, we can say that $\forall I.\ min_c \leq \#_c I \leq max_c \Rightarrow (\#_{c_1} I = \#_c I < min_{c_1}) \vee (max_{c_1} < \#_{c_1} I = \#_c I)$.
From this observation it becomes simple to deduce $(min_c \leq max_c < min_{c_1}) \vee (max_{c_1} < min_c \leq max_c)$, obtaining our theorem. Note that the same reasoning holds if we consider $\forall I \in [\![c_1]\!].\ I \notin [\![c]\!]$.
Case $\Leftarrow$): Suppose $(min_{c_1} > max_c) \vee (min_c > max_{c_1})$ is true.
If $(min_{c_1} > max_c)$ is true, since we have consistent constraints, we can deduce $min_c \leq max_c < min_{c_1}$, so $\forall I.\ min_c \leq \#_c I \leq max_c \Rightarrow \#_c I = \#_{c_1} I < min_{c_1}$, therefore $\forall I.\ I \in [\![c]\!] \Rightarrow I \notin [\![c_1]\!]$.
On the other hand, if $(min_c > max_{c_1})$ is true, since we have consistent constraints, we can deduce $max_{c_1} < min_c \leq max_c$, so $\forall I.\ min_c \leq \#_c I \leq max_c \Rightarrow max_{c_1} < \#_c I = \#_{c_1} I$. Again $\forall I.\ I \in [\![c]\!] \Rightarrow I \notin [\![c_1]\!]$. □

Unfortunately, the reasoning about the global consistency is a very complicated work and it is impossible to find out a simple property which characterizes it, so we can only reason by means of the semantics of sets of constraints. Anyway, sometimes we may resolve the problem to compare two different set of constraints in a simple way, indeed we may define a over-approximation for each set of constraints and then compare these over-approximations.

Definition 3.16 (**Over-approximation**)**:**
Let $S \subseteq Constraints(\mathcal{E})$ be a set of constraints, such that S is consistent. We denote the over-approximation of S by $\hat{S} =< CS_{\hat{S}}, [min_{\hat{S}}, max_{\hat{S}}] >$ where:

- $CS_{\hat{S}} = \bigcup_{c \in S} Choice(c)$
- $min_{\hat{S}} = min\{|I \cap CS_{\hat{S}}| \mid I \in [\![S]\!]\}$
- $max_{\hat{S}} = max\{|I \cap CS_{\hat{S}}| \mid I \in [\![S]\!]\}$

■

Note that if $S = \{c\}$, where c is a single constraint, then $\hat{S} =< CS_{\hat{S}}, [min_{\hat{S}}, max_{\hat{S}}] >$ where:

- $CS_{\hat{S}} = Choice(c)$
- $min_{\hat{S}} = min$ and $max_{\hat{S}} = max$, where $c =< CS, [min, max] >$

Theorem 3.9. *Every over-approximation constraint is a consistent constraint.*

Proof.
Let $\hat{S} =< CS_{\hat{S}}, [min_{\hat{S}}, max_{\hat{S}}] >$ be a over-approximation of S. For construction, surely $0 \leq min_{\hat{S}} \leq max_{\hat{S}}$, in effect we use the operator min and max over the same set $\{|I \cap CS_{\hat{S}}| \mid I \in [\![S]\!]\}$. In addition, from the set theory, we know that $\forall I \subseteq \mathcal{E}.\ |I \cap CS_{\hat{S}}| \leq |CS_{\hat{S}}|$, so $min_{\hat{S}} \leq max_{\hat{S}} \leq |CS_{\hat{S}}|$.

The only particular case is when $[\![S]\!] = \emptyset$ but in this case S is inconsistent, therefore we do not consider it. □

It is simple to understand why $\hat{S}$ is a over-approximation of S, in effect $[\![S]\!] \subseteq [\![\hat{S}]\!]$ by means of $\hat{S}$ definition. Now we will see some properties about over-approximation.

Theorem 3.10 (**Over-approximations maintain inclusion**). *Let S, S_1 be two sets of constraints of $Constraints(\mathcal{E})$ and $\hat{S}, \hat{S}_1$ the over-approximations of S and S_1, respectively. Then if $Choice(\hat{S}) = Choice(\hat{S}_1)$ it holds:*

$$[\![S]\!] \subseteq [\![S_1]\!] \Rightarrow [\![\hat{S}]\!] \subseteq [\![\hat{S}_1]\!]$$

Proof.
If $[\![S]\!] \subseteq [\![S_1]\!]$ is true then we can deduce that $\forall I \subseteq \mathcal{E}.\ I \in [\![S]\!] \Rightarrow I \in [\![S_1]\!]$.
From over-approximation definition, we know that $\exists J \in [\![S]\!].\ |J \cap CS_{\hat{S}}| = min_{\hat{S}}$ and $\exists K \in [\![S]\!].\ |K \cap CS_{\hat{S}}| = max_{\hat{S}}$.
Moreover, for hypothesis, we can say $J \in [\![S_1]\!]$ and $K \in [\![S_1]\!]$. We do not know the values of $min_{\hat{S}_1}$ and $max_{\hat{S}_1}$, but we note that $Choice(\hat{S}) = Choice(\hat{S}_1)$. Therefore we can surely say that $min_{\hat{S}_1} \leq |J \cap CS_{\hat{S}_1}| = |J \cap CS_{\hat{S}}|$, in addition we can say the same for $max_{\hat{S}_1}$, namely $|K \cap CS_{\hat{S}}| = |K \cap CS_{\hat{S}_1}| \leq max_{\hat{S}_1}$.
It is simple to conclude that $min_{\hat{S}_1} \leq \min_{\hat{S}} \leq max_{\hat{S}} \leq max_{\hat{S}_1}$ and for Theorem 3.5 the theorem is true. □

Unfortunately, the vice versa is not true and the next example show us a negative case:

Example 3.5. *Suppose we have the set of elements $\mathcal{E} = \{a, b, c\}$ and two sets of constraints, S and S_1. S has the following constraints:*

1. $c_1 =< \{a, b\}, [0, 1] >$
2. $c_2 =< \{b, c\}, [0, 1] >$
3. $c_3 =< \{a, c\}, [0, 2] >$

On the other hand, S_1 has the following constraints:

1. $c_4 =< \{a, b\}, [0, 2] >$

2. $c_5 =< \{b, c\}, [0, 1] >$

3. $c_6 =< \{a, c\}, [0, 1] >$

Their semantics are:

- $[\![S]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, c\}\}$
- $[\![S_1]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}\}$

It is clear that $[\![S]\!] \not\subseteq [\![S_1]\!]$ *and* $[\![S_1]\!] \not\subseteq [\![S]\!]$.

Now we calculate the two over-approximations, using the definition:

- $\hat{S} =< \{a, b, c\}, [0, 2] >$
- $\hat{S_1} =< \{a, b, c\}, [0, 2] >$

In this case $\hat{S}$ *is the same constraint* $\hat{S_1}$ *so, obviously,* $[\![\hat{S}]\!] = [\![\hat{S_1}]\!]$. *Finally we can deduce* $[\![\hat{S}]\!] \subseteq [\![\hat{S_1}]\!]$ *but* $[\![S]\!] \not\subseteq [\![S_1]\!]$.

Theorem 3.11. *Let* $S \subseteq Constraints(\mathcal{E})$ *be a set of constraints,* $\hat{S}$ *be a over-approximation of* S *and* $c =< CS, [min_c, max_c] > \in Constraints(\mathcal{E})$ *be a constraint. Then if* $Choice(\hat{S}) = Choice(c)$ *it holds:*

$$[\![S]\!] \subseteq [\![c]\!] \Leftrightarrow [\![\hat{S}]\!] \subseteq [\![c]\!]$$

Proof. Case $\Rightarrow$): if $[\![S]\!] \subseteq [\![c]\!]$ then $\forall I \in [\![S]\!]$. $min_c \leq |I \cap CS| \leq max_c$. Now we define $\hat{S} =< CS_{\hat{S}}, [min_{\hat{S}}, max_{\hat{S}}] >$ as we have just described in Definition 3.16. For construction we know that $\exists J \subseteq \mathcal{E}$. $|J \cap CS_{\hat{S}}| = min_{\hat{S}}$ and $\exists K \subseteq \mathcal{E}$. $|K \cap CS_{\hat{S}}| = max_{\hat{S}}$. Note that $CS_c = CS_{\hat{S}}$, therefore we can deduce:

1. $min_c \leq |J \cap CS_c| = |J \cap CS_{\hat{S}}| = min_{\hat{S}} \leq max_c$
2. $min_c \leq |K \cap CS_c| = |K \cap CS_{\hat{S}}| = max_{\hat{S}} \leq max_c$

Since $\hat{S}$ is a single constraint, $Choice(\hat{S}) = Choice(c)$ and $min_c \leq min_{\hat{S}} \leq max_{\hat{S}} \leq max_c$, by means of Theorem 3.5 we can conclude the theorem.
Case $\Leftarrow$): if $[\![\hat{S}]\!] \subseteq [\![c]\!]$ then, since $\hat{S}$ is a over-approximation of S, we have $[\![S]\!] \subseteq [\![\hat{S}]\!] \subseteq [\![c]\!]$. □

3.1.2 Refinement

In this subsection we focus on the concept of refinement of a CMTS. Recall the conceptual idea of refinement: let M and M_1 be two specifications, then we can say M is a refinement of M_1 if and only if the set of implementations satisfying M is a subset of the set of implementations satisfying M_1. Moreover note that, from modelling pointview, a generic specification is modelled by MTS, DMTS, GEMTS and so on, whereas the implementation is always represented by LTS.

The refinement relation has deeply been studied and in literature it is possible to find two different types of refinements: **modal** and **thorough** [1], [35]. In Section 2.2 we saw several models which describe specifications and for each model we described a particular refinement relation $\mathcal{R}$ and we denoted the maximal refinement relation by $\trianglelefteq$. Moreover we can note that each presented relation defines the conceptual idea of refinement by means of properties over syntactic components, for example in MTS the refinement relation is based on two conditions over may and must transitions, which are syntactic components of MTS together with states and label. This type of refinement relation is called **modal refinement relation**.

Unfortunately, this relation is not complete. For example, suppose to have two MTSs M and N like ones in Figure 3.4. Trivially, the set of implementations (or LTSs) which satisfy the specification modelled by M is the same of the one of N, but $M \not\trianglelefteq N$.

To solve this problem is introduced the concept of **thorough refinement relation** which exploits the concept of semantics of a model. The idea is simple: taken a model M which describes a specification like MTS, DMTS and so on, we define the semantics of M, denoted by $[\![M]\!]$ such that $[\![M]\!] = \{I \mid I \trianglelefteq M \wedge I \text{ is a LTS}\}$, namely the semantics describes the set of all LTSs which derived from the specification M. Note that the semantics of M is defined by means of $\trianglelefteq$, therefore the definition of semantics depends by the typical modal refinement relation of a MTS, DMTS and so on.

Now it is possible to define a new type of refinement: let M and M_1 be two models which represent a specification then we say a **thorough refinement relation** between M and M_1 exists if and only if $[\![M]\!] \subseteq [\![M_1]\!]$, that is if every LTS L, which satisfies the specification M, also satisfies M_1. This refinement describes exactly the conceptual idea of refinement. In addition an inconsistent specification S is a wrong specification, hence any possible LTS cannot satisfy S and we describe this concept by $[\![S]\!] = \emptyset$.

We want to highlight that the non completeness of modal refinement relations is a clear disadvantage, on the other hand the modal refinement has an important property: it is a "syntactic" relation, namely it considers only syntactic aspects of models, so from computational pointview the modal refinement is less expensive than the thorough refinement, in effect the thorough refinement requires the computation of all LTSs which satisfy the specification.

For avoid confusion in the meaning of $[\![]\!]$, we introduce a convention:

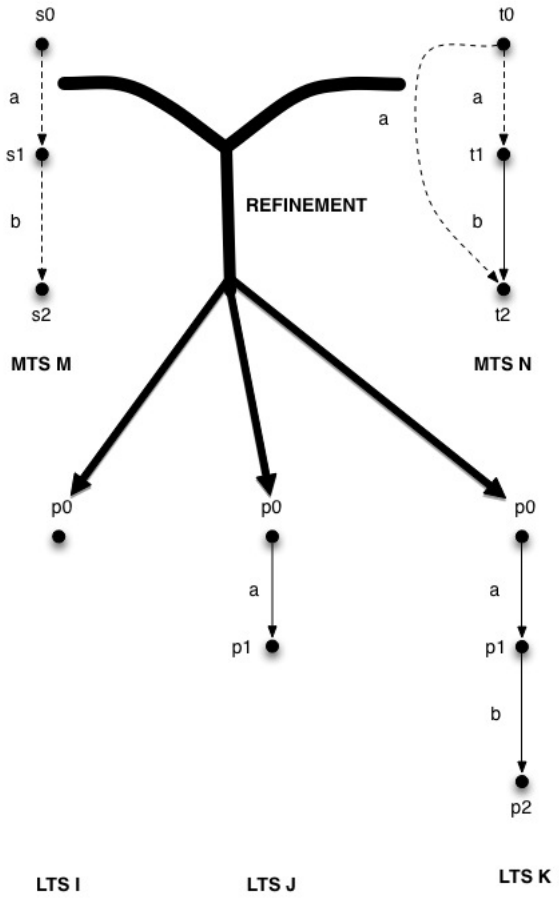


Figure 3.4: An example of two MTSs not modal refinable but semantically equivalent

- if the operator $[\![.]\!]$ is applied to constraints or single states, then the meaning of $[\![.]\!]$ is one defined in Definition 3.4, Definition 3.7 and Definition 3.13.
- if the operator $[\![[\![.]\!]]\!]$ is applied to states or the entire model, namely to a CMTS, then the meaning of $[\![[\![.]\!]]\!]$ is one defined by thorough modal refinement, previously described. In effect, taken a CMTS M, the semantics of M is the set of all implementations which satisfy the specification M and this idea is exactly captured by thorough refinement relation.

Now our task is to find a some formalization to describe the refinement idea in the CMTS world. Initially, we focus on the modal refinement relation and the first idea is to exploit the semantics of constraints: taken two states s and t we can determine the set $[\![s]\!]$ and $[\![t]\!]$, namely two sets of sets of outgoing transitions which satisfy constraints of s and t, respectively, then we compare these two sets.

In this context we can observe as $[\![\mathfrak{C}(s)]\!]$ for a some state s, namely the semantics of constraints related to s described in previous sections, and $[\![\Omega(s)]\!]$, defined in [12] for OTS models, represent the same thing, namely a set of sets of outgoing

transitions. The difference, of course, is how this set is determined, in first case we use the constraint concept, whereas in the second we use logic formulae. Anyway we could define a refinement relation in the same way of OTSs. First we introduce a relation to compare two sets of sets of transitions by means of a some relation $\mathcal{R}$:

Definition 3.17:
Let $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ be a relation and $S, T \in \mathcal{P}(\mathcal{P}(\Sigma \times \mathcal{S}))$ be two sets of sets of outgoing transitions. We write $S \sqsubseteq_{\mathcal{R}} T$ to denote:

$$\forall I \in S.\ \exists J \in T.\ \forall(\alpha, s) \in I.\ \exists(\alpha, t) \in J.\ (s,t) \in \mathcal{R} \wedge \forall(\alpha, t) \in J.\ \exists(\alpha, s) \in I.\ (s,t) \in \mathcal{R}$$

■

Of course this definition can be extended to a more general case, where we have $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}_1$. Note that it describes the bisimulation relation over a set of sets of outgoing transitions.

The refinement of CMTSs is defined consequently:

Definition 3.18 (Refinement):
Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs. We say that $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is a *refinement relation* if $(s,t) \in \mathcal{R}$ implies:

1. $s \xrightarrow{\alpha} s' \Rightarrow t \xrightarrow{\alpha} t' \wedge (s', t') \in \mathcal{R}$
2. $[\![\mathfrak{C}(s)]\!] \sqsubseteq_{\mathcal{R}} [\![\mathfrak{C}(t)]\!]$

We say s refines t ($s \trianglelefteq t$) if there is a refinement relation $\mathcal{R}$ such that $(s,t) \in \mathcal{R}$. We say M refines N if and only if $s_{M_0} \trianglelefteq s_{N_0}$.

■

This formalization is simple and clear but it has a problem: we use the semantics of constraints by means of the second condition, so it requires the computation of all possible sets of transitions which satisfy every constraint in $\mathfrak{C}(s)$, for some s. From computational pointview, this choice is very expensive so we would like to understand if it is possible to define another refinement relation more "syntactic", namely a relation which does not exploit the semantics of constraints. For convenience we call the refinement relation, just described, **semantic modal refinement relation** to explain the nature of concepts used to formalize the modal refinement relation. In the remaining section we will describe the **syntactic modal refinement relation**. In addition the relation $\trianglelefteq$ is characterized in two different ways:

- $\trianglelefteq^{Sem}$: it is used in the context of semantic modal refinement relation, namely $s \trianglelefteq^{Sem} t$ if there is a semantic modal refinement relation $\mathcal{R}$ such that $(s,t) \in \mathcal{R}$
- $\trianglelefteq^{Syn}$: it is used, instead, in the context of syntactic modal refinement relation, that is $s \trianglelefteq^{Syn} t$ if there is a syntactic modal refinement relation $\mathcal{R}$ such that $(s,t) \in \mathcal{R}$

Our idea of refinement is simple: for every state, at each refinement step, we must reduce the sets of outgoing transitions and this is possible if we delete some outgoing transitions and/or the constraints reduce the possible valid sets. Both operations are very delicate. The deleting of transitions must be executed in a correct way: suppose we must remove the transition $t = (\alpha, s')$ but, unfortunately, this transition t might be in some choice set of some constraints, therefore we must also remove the transition t from every choice set where it is present.

Instead, the refinement of a single constraint can be divided in two different aspects:

1. we reduce the choice set, eliminating some transitions
2. we reduce the interval $[min, max]$ in some suitable way

The first operation is simple but it has a side effect: if we delete a transition t from a choice set of a constraint c, then c does not handle t any longer. If t is not deleted as outgoing transition then the restriction over t which has been imposed by c does not exist any longer. In some case this situation implies the introduction of a new possible solution for the constraint which was not allowed before of the refinement step. It is clear that if this happens, it is wrong. We can conclude that every transition which is deleted from a choice set must be also deleted as outgoing transitions.

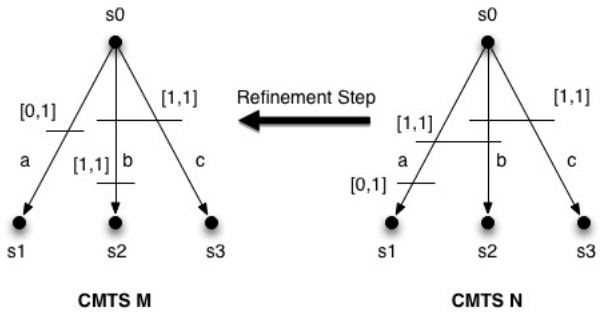


Figure 3.5: A possible step of wrong refinement

Example 3.6. *Suppose we have the CMTS N of Figure 3.5. The possible solutions which satisfy all conditions are:* $[\![N]\!] = \{\{b\}, \{a, c\}\}$.

Now we see a possible refinement step where the transition (a, s_1) *is deleted by the choice set of the constraint* $< \{(a, s_1), (b, s_2)\}, [1, 1] >$ *but this transition* (a, s_1) *is not deleted as outgoing transition, for example we consider the CMTS M of Figure 3.5. In this case the requirement* $< \{(a, s_1), (b, s_2)\}, [1, 1] >$*, which means that transitions* (a, s_1) *and* (b, s_2) *can be exclusively chosen, misses. This situation is wrong, indeed, the possible semantics of CMTS M is:* $[\![M]\!] = \{\{b\}, \{a, b\}\}$. *Therefore we can deduce* $[\![M]\!] \not\subseteq [\![N]\!]$.

Now we try to reason about the second operation: in this case we want to reduce the semantics of a constraint c without change the choice set, but from Theorem 3.5 we know that this is possible if the new constraint $c' =< CS, [min', max'], refined by $c =< CS, [min, max] >$, holds the property $min \leq min' \leq max' \leq max$. Of course we reason about only correct constraints. Therefore, taken two states s and t, s refines t if every constraint c of t is reduced in some constraint c' of s.

Moreover, for the refinement concept, the local and global inconsistency must be maintained in each refinement step, in effect taken two CMTSs M and M_1 if $[\![M_1]\!] = \emptyset$, namely M_1 is inconsistent then M is a refinement of M_1 if and only if $[\![M]\!] = \emptyset$ and the reason is simple: we say M is a refinement of M_1 if and only if $[\![M]\!] \subseteq [\![M_1]\!]$. On the other hand, for the same reason, the local and global consistency need not be maintained in each refinement step.

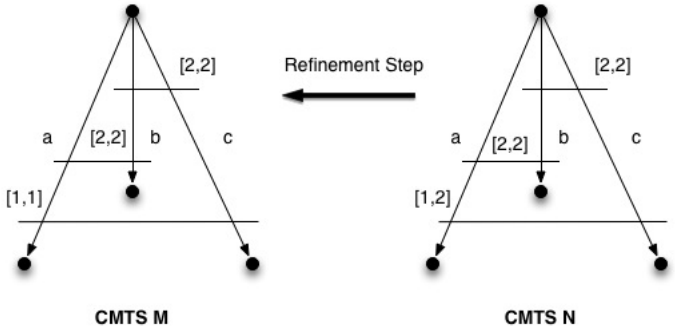


Figure 3.6: A possible refinement step from a consistent CMTS to an inconsistent CMTS

Example 3.7. *Suppose we have the CMTS N of Figure 3.6. The possible sets of outgoing transition which satisfy all conditions are:* $[\![N]\!] = \{\{a, b, c\}\}$. *By means of a refinement step we can derive the CMTS M of Figure 3.6. This refinement step is correct even though the CMTS M is inconsistent, indeed,* $[\![M]\!] = \emptyset \subseteq [\![N]\!]$.

Note that, taken two CMTSs like M and N in Figure 3.7, we can have some "strange cases". In effect it is simple to see that M and N are inconsistent, but for refinement concept M is a refinement of N and N is a refinement of M, even though they are very different. Note that their semantics $[\![M]\!] = \emptyset$ and $[\![N]\!] = \emptyset$, so $[\![M]\!] \subseteq [\![N]\!]$ and $[\![N]\!] \subseteq [\![M]\!]$. Anyway, in general we are not interested in inconsistent specifications, so we will avoid inconsistent CMTSs.

In conclusion our refinement relation must guarantee us:

1. in each step a generic constraint can lose some transitions in its choice set or its cardinality can be reduced

2. new transitions cannot be added in a generic state

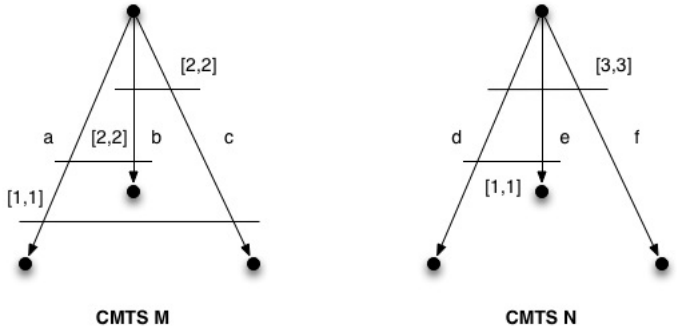


Figure 3.7: A strange case of refinement

3. local and global inconsistency must always hold

Before to see the syntactic modal refinement relation, we introduce a new concept to reason about the deleted transitions of a state. Since CMTSs are action-deterministic, we can take account of labels directly, seeing as each label identifies univocally a transition.

Definition 3.19:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS and s be a state. We denote the set of all labels related to outgoing transitions by $Label(s) = \{\alpha \in \Sigma \mid \exists s' \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow\}$.

■

We can extend this definition to constraints too.

Definition 3.20:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS and $c \in \mathfrak{C}(s)$ be a constraint of a some state s. We denote the set of all labels related to choice set of c by $Label(c) = \{\alpha \in \Sigma \mid \exists s' \in \mathcal{S}.\ (\alpha, s') \in Choice(c)\}$.

■

Now we have enough information to describe the refinement between two constraints:

Definition 3.21 (Refinement between two constraints):
Let c, c_1 be two constraints. We say c is a refinement of c_1 regarding a relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, denoted by $c \trianglelefteq_{\mathcal{R}} c_1$ if and only if:

- $\forall(\alpha, s') \in Choice(c).\ \exists(\alpha, t') \in Choice(c_1).\ (s', t') \in \mathcal{R}$
- c and c_1 are correct constraints
- $Card_{min}(c_1) \leq Card_{min}(c)$

- $Card_{max}(c) \leq Card_{max}(c_1)$

■

Note that this refinement definition of constraints is not correct, for example all constraints of CMTS (a) in Figure 3.5 are correctly refined for constraints of CMTS (b) but, as we said previously, that refinement is wrong. The definition of refinement of constraints is correct under a further hypothesis:

Theorem 3.12 (**Correctness of refinement of consistent constraints**). *Let* $c =< CS_c, [min_c, max_c] >\in \mathfrak{C}(s), c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >\in \mathfrak{C}_1(s_1)$ *be two consistent constraints for certain states* s, s_1 *of some CMTSs and* $\mathcal{R}$ *be a relation.*

If $Label(s) \subseteq Label(s_1) \setminus (Label(c_1) \setminus Label(c))$ *then it holds:*

$$c \trianglelefteq_{\mathcal{R}} c_1 \Rightarrow [\![c]\!] \sqsubseteq_{\mathcal{R}} [\![c_1]\!]$$

Before to see the proof, we try to reason about the theorem: it says that taken two constraints of certain states of CMTSs then they are a correct refinement if $Label(s) \subseteq Label(s_1) \setminus (Label(c_1) \setminus Label(c))$, namely if the refined state s loses some transitions of s_1 or more precisely if s loses at least all and only the transitions which are present in c_1 but not in c, described by $(Label(c_1) \setminus Label(c))$. We want to highlight that in the models being CMTSs, transitions are univocally determined by labels, hence we have so many transitions as labels and, implicitly, $Label(s) = Label(s_1) \setminus (Label(c_1) \setminus Label(c))$ requires that s has a number of transitions less or at most equal to s_1.

Proof.
We suppose $c \trianglelefteq_{\mathcal{R}} c_1$ and we take a set of transitions $I \subseteq Label(s)$ such that $I \in [\![c]\!]$, namely $min_c \leq \#_c I \leq max_c$. So we define $J = \{(\alpha, t') \mid (\alpha, s') \in I \ \wedge \ (s', t') \in \mathcal{R}\}$, trivially, for construction $I \sqsubseteq_{\mathcal{R}} J$.

Now we must understand if $J \in [\![c_1]\!]$. Since $c \trianglelefteq_{\mathcal{R}} c_1$, we know that $Label(c) \subseteq Label(c_1)$ and the labels determine univocally transitions. Moreover, seeing that we have the guarantee $I \sqsubseteq_{\mathcal{R}} J$ is true, we can reason directly over labels.

For construction of J, we have that $|J \cap Label(c_1) \setminus Label(c)| = |J \cap (Label(c_1)| - |J \cap Label(c))| = 0$. In effect all labels of J are the same of I, for construction, and $I \subseteq Label(s)$, where the labels in $Label(c_1) \setminus Label(c)$ does not exist.

Therefore we can deduce that, $\#_c I = \#_{c_1} J$, even though I and J have different elements and c and c_1 have different choice sets. In conclusion, since $c \trianglelefteq_{\mathcal{R}} c_1$, we have also $min_{c_1} \leq min_c \leq \#_c I = \#_{c_1} J \leq max_c \leq max_{c_1}$, deducing $J \in [\![c_1]\!]$. □

Another observation is necessary: suppose we have two state s and t and moreover a constraint $c \in \mathfrak{C}(t)$ exists but no constraint in s derives from a reduction of c. We have previously said that this situation is wrong and in general this sentence is true, except for a particular case. If constraint $c =< CS, [min, max] >$ with $min = 0$ exists then a set of transitions I such that $|I \cap CS| = 0$ is valid for c

because $min = 0 \leq |I \cap CS| = 0$. So if the state s has no transitions of CS and it has the same constraints (modified conveniently) of t, except the constraint c, it is simple to understand that $[\![s]\!] \subseteq [\![t]\!]$. The Example 3.8 explains the situation.

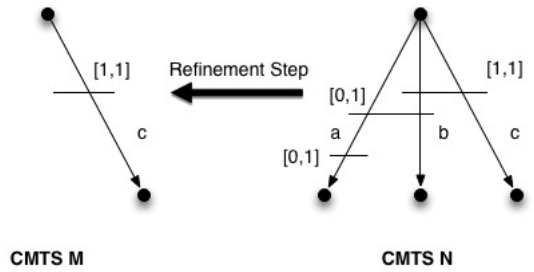


Figure 3.8: A special case of the refinement

Example 3.8. *Suppose we have the CMTS N in Figure 3.8 and we compute its semantics* $[\![N]\!] = \{\{b\}, \{c\}, \{a, c\}\}$. *Now we consider the CMTS M in Figure 3.8. As we can see some constraints being in N are lost in M, therefore inasmuch as we said previously M should be not a refinement of N. Anyway we try to compute the semantics of M* $[\![M]\!] = \{\{c\}\}$, *then we can deduce* $[\![M]\!] \subseteq [\![N]\!]$. *From theoretical pointview M is a refinement of N. The reason is simple: we consider the constraint* $cons =< \{a, b\}, [0, 1] >$, *in this case it is possible to have a set of outgoing transitions I which have not transitions a and b. For cons is correct the choice of not taking any transitions in its choice set hence a state, where each transition of the choice set of cons is not present, is a possible valid state, under the condition that other constraints are modified in a suitable way.*

Another way to see this situation is to interpret the absence of a set of transitions T as a constraint $cons_1 =< T, [0, 0] >$, *in effect its semantics is equal to* $\{I \mid 0 \leq |I \cap T| \leq 0\}$.

In conclusion we can interpret the absence of $\{a, b\}$ *as* $< \{a, b\}, [0, 0] >$ *and, trivially, this constraint is a valid reduction of cons.*

Definition 3.22 (**Syntactic modal refinement**):
Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0}), N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs. A binary relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is called *syntactic modal refinement* if and only if $(s, t) \in \mathcal{R}$ implies:

- $\forall (s, \alpha, s') \in \longrightarrow_M \,.\, \exists (t, \alpha, t') \in \longrightarrow_N \,.\, (s', t') \in \mathcal{R}$
- $\forall c_t \in \mathfrak{C}_N(t)$ exactly one of two conditions must hold:
  - $\exists c_s \in \mathfrak{C}_M(s)$ such that:
    1. $c_s \trianglelefteq_{\mathcal{R}} c_t$
    2. $(Label(c_t) \setminus Label(c_s)) \cap Label(s) = \emptyset$

– $c_t =< CS, [min, max] >$ such that $min = 0$ and $Label(CS) \cap Label(s) = \emptyset$

We may say s is a *refinement* of t ($s \trianglelefteq^{Syn} t$) if a syntactic refinement relation $\mathcal{R}$ exists such that $(s,t) \in \mathcal{R}$. We say M is a *refinement* of N if and only if $s_{M_0} \trianglelefteq^{Syn} s_{N_0}$. ■

Now we can try to explain this definition:

1. the first condition $\forall(s, \alpha, s') \in \longrightarrow_M \, . \, \exists(t, \alpha, t') \in \longrightarrow_N \, . \, (s', t') \in \mathcal{R}$ requests that no new transitions can exist in the state s.

2. the second condition is more complex and it tries to reason about the constraints. s can be considered a refinement of t if and only if s reduces or at most satisfies exactly each constraint defined in t. The first sub-condition requires that, taken a constraint c_t of t, must exist a constraint c_s which refines c_t, as we have previously explained. The second sub-condition handles the special case, namely if c_t allows allows to have sets of transitions without transitions of $Choice(c_t)$ and if in s all transitions of $Choice(c_t)$ are deleted then we have a possible refinement of t and constraints of s, related to c_t, are rightly absent.

Note that the global and local consistency do not hold in a refinement step, an example is described in Figure 3.9.

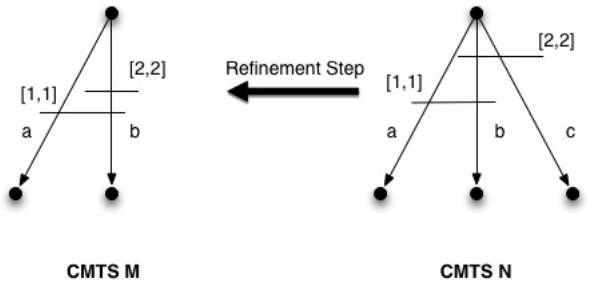


Figure 3.9: An example of not maintaining of the consistency in a refinement step

Theorem 3.13 (**Local inconsistency is maintained**). *Let $s \in \mathcal{S}_M, t \in \mathcal{S}_N$ be two states of two CMTSs such that $(s,t) \in \mathcal{R}$, for some syntactic refinement relation $\mathcal{R}$. Let c_t be a constraint of t and c_s be a constraint of s such that $c_s \trianglelefteq_{\mathcal{R}} c_t$. Then it holds:*

$$c_t \text{ is inconsistent} \Rightarrow c_s \text{ is inconsistent}$$

Proof.
We know from Theorem 3.4 that $c_t =< CS_t, [min_t, max_t] >$ is inconsistent if and

only if $(min_t > max_t) \vee (min_t > |CS_t|)$. Since c_t is a constraint of a CMTS state we have the guarantee $min_t \leq max_t$ is always true, so we can deduce $min_t > |CS_t|$.

Moreover, taken $c_s =< CS_s, [min_s, max_s] >$ we know that $c_s \trianglelefteq_{\mathcal{R}} c_t$ if and only if $(min_t \leq min_s) \wedge (max_s \leq max_t)$ and all transitions in CS_s are also present in CS_t such that the reached target states are in the relation $\mathcal{R}$. Since our CMTSs are action-deterministic, we can guarantee that every transition is univocally determined by labels, so in practice $\trianglelefteq_{\mathcal{R}}$ requires $|Label(CS)_s| \leq |Label(CS_t)|$.

Now we can deduce that if $c_s \trianglelefteq_{\mathcal{R}} c_t$ then $|CS_s| \leq |CS_t| < min_t \leq min_s$, so also c_s is inconsistent.

Finally note that if $min_t = 0$, trivially, c_t will never be able to be inconsistent. □

Theorem 3.14 (**Global inconsistency is maintained**). *Let $s \in \mathcal{S}_M, t \in \mathcal{S}_N$ be two states of two CMTSs such that $(s,t) \in \mathcal{R}$, for some syntactic refinement relation $\mathcal{R}$. Then it holds:*

$$t \text{ is inconsistent} \Rightarrow s \text{ is inconsistent}$$

Proof.
Trivially if an inconsistent constraint exists in t then t is inconsistent and so also s is inconsistent, for Theorem 3.13.

The only particular case is when t has all consistent constraints but their union is inconsistent, namely $\bigcap_{c \in \mathfrak{C}_N(t)} [\![c]\!] = \emptyset$.

As we have said, if a constraint $c_t =< CS_t, [min_t, max_t] >$ and $min_t = 0$ and $Label(s) \cap Label(cs_t) = \emptyset$ then s might not have a constraint related to c_t, but this equivalent to have a constraint $c_s =< CS_s, [0,0] >$ where $c_s \trianglelefteq_{\mathcal{R}} c_t$. Therefore for convenience we can suppose that every constraint c_t has a corresponding constraint c_s.

In addition we know that $[\![c_s]\!] \subseteq [\![c_t]\!]$, seeing that $c_s \trianglelefteq_{\mathcal{R}} c_t$, hence trivially we can say $\bigcap_{c \in \mathfrak{C}_M(s)} [\![c]\!] \subseteq \bigcap_{c \in \mathfrak{C}_N(t)} [\![c]\!] = \emptyset$, deducing the global inconsistency for s too. □

Now we demonstrate the correctness of syntactic modal refinement.

Theorem 3.15 (**Correctness of syntactic modal refinement**). *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0}), N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs. Let $s \in \mathcal{S}_M, t \in \mathcal{S}_N$ be two states such that $(s,t) \in \mathcal{R}$ for some syntactic refinement relation $\mathcal{R}$. Then it holds:*

$$[\![s]\!] \sqsubseteq_{\mathcal{R}} [\![t]\!]$$

Proof.
First of all, for Theorem 3.14 if t is inconsistent then s is inconsistent too. Trivially, if s is inconsistent and t is not the theorem is true. Note that this is a possible situation for our syntactic modal refinement and it happens, for example, when we delete too many transitions from a choice set of a certain constraint.

Now we restrict to the situation where $[\![s]\!] \neq \emptyset$ and $[\![t]\!] \neq \emptyset$.

As we have said, if a constraint $c_t =< CS_t, [min_t, max_t] >$ and $min_t = 0$ and $Label(s) \cap Label(cs_t) = \emptyset$ then s might not have a constraint related to c_t, but this equivalent to have a constraint $c_s =< CS_s, [0, 0] >$ where $c_s \trianglelefteq_{\mathcal{R}} c_t$. Therefore for convenience we can also suppose that every constraint c_t has a corresponding constraint c_s, such that $[\![c_s]\!] \sqsubseteq_{\mathcal{R}} [\![c_t]\!]$.

It is simple to understand that $[\![s]\!] = \bigcap_{c \in \mathfrak{C}_N(s)} [\![c]\!] \sqsubseteq_{\mathcal{R}} \bigcap_{c \in \mathfrak{C}_M(t)} [\![c]\!] = [\![t]\!]$

□

The last topic about refinement for CMTSs is the thorough refinement relation. This definition is based on concept of semantics of MTSs and states:

Definition 3.23:
Let $M = (\mathcal{S}_M, \Sigma, \rightarrow_M, \dashrightarrow_M)$ be a MTS and $\trianglelefteq$ be a modal refinement relation. Then the semantics of a state $s \in \mathcal{S}$ is :

$$[\![[\![s]\!]]\!] = \{I \mid I \trianglelefteq s \wedge I \text{ is a LTS}\}$$

The semantics of M is equal to $\bigcup_{s_i \text{ is an initial state of } M} [\![[\![s_i]\!]]\!]$

■

The idea of the semantics of a state s is to describe all possible LTSs which are derived by s exploiting a refinement relation, in effect the semantics of a specification can be described by the set of all implementations (or LTS) which satisfy the specification itself. In addition, note that the refinement relation stipulates which specifications refine which specifications, but since our implementations are just special specifications the refinement serves as an implementation relation at the same time.

In order to simplify the notation taken two sets of LTSs S and T, we say $S \subseteq T$ to denote that $\forall I \in S.\ \exists J \in T.\ I \sim J$.

Now we recall the definition of thorough refinement for MTS:

Definition 3.24 (Thorough refinement relation for MTS):
Let $M = (\mathcal{S}_M, \Sigma, \rightarrow_M, \dashrightarrow_M), N = (\mathcal{S}_N, \Sigma, \rightarrow_N, \dashrightarrow_N)$ be two MTSs. Then a relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is called *thorough refinement relation* if and only if $\forall (s, t) \in \mathcal{R}$ it holds $[\![[\![s]\!]]\!] \subseteq [\![[\![t]\!]]\!]$.

We may say s is a *refinement* of t ($s \trianglelefteq^{Thorough} t$) if a thorough refinement relation R exists such that $(s, t) \in R$.

■

In the CMTS context we have some problems to be solved. First of all, we do not know how the implementation concept must be represented, but following the other works in literature, we can assume that an implementation is described by a LTS. In effect a LTS can be seen as a specification which cannot be modified anymore.

The second problem is how to correlate a LTS to a CMTS by means of a refinement relation, in effect our refinement relations take into account two initial CMTSs hence the solution is to find a way to describe a LTS as a CMTS. Of course, this is possible seeing that a LTS can be conceived as a CMTS where all constraints are satisfied and the set of constraints has only one possible valid solution. Only in Chapter 5 we will have enough information to formalize the concept of LTS described by CMTS and hence, the concept of thorough refinement too. Finally we make an observation as simple as important: from theoretic pointview a LTS L is an implementation of a specification M if and only if L satisfies the requirements of M. In the CMTS context the requirements of M are described by constraints, so we can deduce the following property:

Definition 3.25 (**LTS is a solution**)**:**
Let L be a LTS, M be a CMTS. Then L is an implementation of M if and only if for each state s_L of L then the set of outgoing transitions of s_L is a possible solution of the set of constraints $\mathfrak{C}(s_M)$ such that s_M is the related state of s_L in M.

■

Anyway this result will be more clear in Chapter 5, when we introduce how a LTS can be described by means of CMTS. It is clear that, seeing that thorough refinement depends on the modal refinement then in the CMTS case we can have two types of thorough refinement: **syntactic**, if we consider the $\trianglelefteq^{Syn}$ relation, or **semantic**, if we consider the $\trianglelefteq^{Sem}$ relation.

Definition 3.26 (**Thorough semantic refinement relation for CMTS**)**:**
Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{0_M}), N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{0_N})$ be two CMTSs. Then a relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is called *thorough semantic refinement relation* if and only if $\forall (s,t) \in \mathcal{R}$ it holds $[\![[\![s]\!]]\!]^{Sem} \subseteq [\![[\![t]\!]]\!]^{Sem}$.

■

Definition 3.27 (**Thorough syntactic refinement relation for CMTS**)**:**
Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{0_M}), N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{0_N})$ be two CMTSs. Then a relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is called *thorough syntactic refinement relation* if and only if $\forall (s,t) \in \mathcal{R}$ it holds $[\![[\![s]\!]]\!]^{Syn} \subseteq [\![[\![t]\!]]\!]^{Syn}$.

■

As we can see, the concept of thorough refinement relation can be easily described in the same manner done for the MTS. However we want to highlight that these definitions are based on the concept of semantics of a CMTS but, unfortunately, in this context we does not give a formal definition of semantics because we does not know how to describe LTS in a correct way. Only in Chapter 5 we describe the concept of semantics in a formal way, in this section we must be satisfied only of an informal description: *the semantics of a CMTS M is the set of all LTSs which satisfy the requirements of M*. Unfortunately, we find that our two modal refinement relations are not complete: the syntactic modal refinement relation is not complete

both for consistent CMTSs and for inconsistent CMTSs. For example in Figure 3.10 CMTSs M and N are inconsistent, so it should be true $M \trianglelefteq^{Syn} N$, but it is simple to see $M \not\trianglelefteq^{Syn} N$, because M has a further transition labelled with d which is not present in N. For consistent CMTSs, we can see the example in Figure 3.11, in

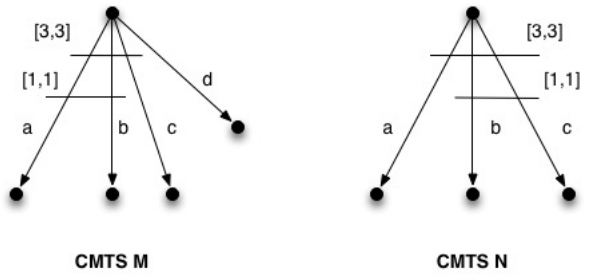


Figure 3.10: An example of non completeness of syntactic and semantic refinement between two inconsistent CMTSs

this case CMTSs M and N have the same semantic but $M \not\trianglelefteq^{Syn} N$. Finally in the

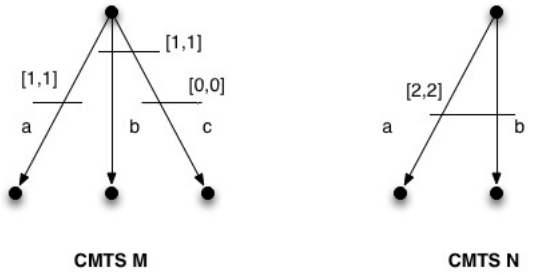


Figure 3.11: An example of non completeness of syntactic and semantic refinement between two consistent CMTSs

Figure 3.12 we have two CMTSs M and N such that the semantics of M is equal to $\{\{a,c\},\{b,c\}\}$, whereas the semantics of N is equal to $\{\{a\},\{b\},\{a,c\},\{b,c\}\}$, so it is simple to understand that M is a refinement of N but again $M \not\trianglelefteq^{Syn} N$.

Also the semantic modal refinement relation is not complete for inconsistent CMTSs and the example is the one in Figure 3.10. In this case, indeed, $[\![[\![M]\!]]\!] = [\![[\![N]\!]]\!] = \emptyset$ but $M \not\trianglelefteq^{Sem} N$. The incompleteness is also maintained for consistent CMTS as we can see in Figure 3.11, M and N have the same semantic but $M \not\trianglelefteq^{Sem} N$.

Another example of non-semantic modal completeness over consistent CMTSs is the one described in Figure 3.13. In this context $[\![[\![M]\!]]\!] = [\![[\![N]\!]]\!] = \{(a,b)\}$ but $M \not\trianglelefteq^{Sem} N$.

Theorem 3.16 (**Completeness of the Semantic modal refinement**). *Let* $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{0_M})$ *and* $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{0_N})$ *be two CMTSs. If both*

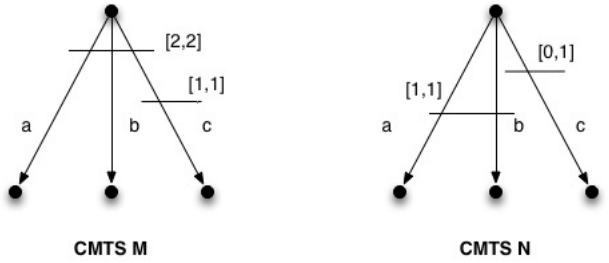


Figure 3.12: An other example of non completeness of syntactic refinement between two consistent CMTSs

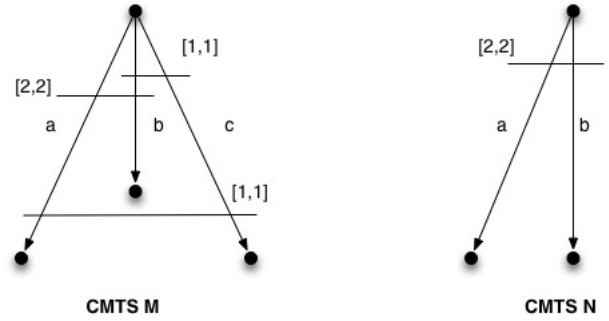


Figure 3.13: An example of non completeness of the semantic refinement between two consistent CMTSs

CMTSs satisfy the following property:

$$\forall s.\ \forall(\alpha, s') \in Trans(s).\ \exists I \in [\![s]\!].\ (\alpha, s') \in I$$

then $[\![[\![M]\!]]\!]^{Sem} \subseteq [\![[\![N]\!]]\!]^{Sem} \Rightarrow M \trianglelefteq^{Sem} N$.

Note that the property implicitly discards inconsistent CMTSs and CMTSs with an outgoing transition t in some state s, which is never considered in some solution, because constraints prevent the presence of t. For the property, each transition must be present in some solution.

Proof.
We consider the relation:

$$\mathcal{R} = \{(s_M, s_N) \mid \forall\alpha.\ (s_M, \alpha, s'_M) \in \longrightarrow_M \Rightarrow (s_N, \alpha, s'_N) \in \longrightarrow_N \wedge (s'_M, s'_N) \in \mathcal{R}\}$$

and we prove that it is a semantic modal refinement relation.

Trivially, taken s_M and s_N the first condition of semantic modal refinement holds, in effect if $(s_M, \alpha, s'_M) \in \longrightarrow_M$ then $(s_N, \alpha, s'_N) \in \longrightarrow_N$, for definition of $\mathcal{R}$.

Now suppose $I_M \in [\![\mathfrak{C}(s_M)]\!]$ then surely a LTS L_M such that $Trans(s_{L_M}) = I_M$ exists, since a LTS L derived from a CMTS K is a set of solutions of constraints

related to states of K. For hypotheses, exists a LTS L_N derived from N such that $L_N \sim L_M$, but then a set $I_N \in [\![\mathfrak{C}(s_N)]\!]$ exists, such that $\forall(\alpha, s') \in I_M.\ \exists(\alpha, t') \in I_N \wedge (s', t') \in \mathcal{R}$ and $\forall(\alpha, t') \in I_N.\ \exists(\alpha, s') \in I_M \wedge (s', t') \in \mathcal{R}$. Trivially, $[\![\mathfrak{C}(s_M)]\!] \sqsubseteq_{\mathcal{R}} [\![\mathfrak{C}(s_N)]\!]$. □

Both syntactic modal refinement and semantic modal refinement are preorders, in Appendix A we show in a detail way that both refinements hold the reflexive and transitive property.

3.1.3 Non Determinism

Now we imagine that our hypothesis of action-determinism in the CMTS definition is deleted, it is interesting to understand what happens to our syntactic refinement relation. Surely it does not work any longer, because in its definition we use the function *Label* which exploits the action-determinism. It is simple to extend this function in order to handle the non-determinism: a transition is not uniquely determined any longer by means of a label, in this case it is determined by a label and a target state.

Anyway, also changing the function *Label*, our refinement relation does not work. The problem is in the constraint concept together with the non-determinism. Previously we have implicitly assumed that a constraint is refined if and only if its cardinality is reduced or some transition is deleted, seeing that the CMTS is action-deterministic then, in each refinement step, the transitions of constraints can only be reduced. In a non-deterministic context this is not true any longer, in effect we can add further transitions provided they are equivalent to some existent transitions. For example in Figure 3.14 the two CMTSs M and N have the same semantics:

- $[\![M]\!] = \{\{(a, s_1), (a, s_2), (b, s_3)\}, \{(a, s_1), (a, s_2), (c, s_4)\}\}$
- $[\![N]\!] = \{\{(a, t_1), (b, t_2)\}, \{(a, t_1), (c, t_3)\}\}$

Moreover $\forall I \in [\![M]\!].\ \exists J \in [\![N]\!].\ I \sim J$ and also the vice versa holds. So our syntactic modal refinement should be able to handle these situations. Unfortunately, adding transitions make trouble because some requirements can be lost, in effect *min* and *max* of each constraint count the number of transitions without considering if and what transitions are "equivalent", for some relation $\mathcal{R}$. For example in Figure 3.15 we have two CMTSs M and N. As we can see, M and N have only one constraint and using the definition of syntactic refinement of constraints we can find out that the constraint in M is a refinement of the constraint in N, in effect the cardinality does not change and we add only a transition labelled with a, equivalent to another, already existing transition. If we allow this situation then we are in error, in effect the LTSs I and J in Figure 3.15 are bisimilar and they are derivable from M, but at the same time they are wrong for N. In conclusion we have $M \trianglelefteq N$ but $[\![M]\!] \not\subseteq [\![N]\!]$.

The solution could be modified in a "smart" way: for example we can require that all equivalent transitions are connected to a single constraints which has the

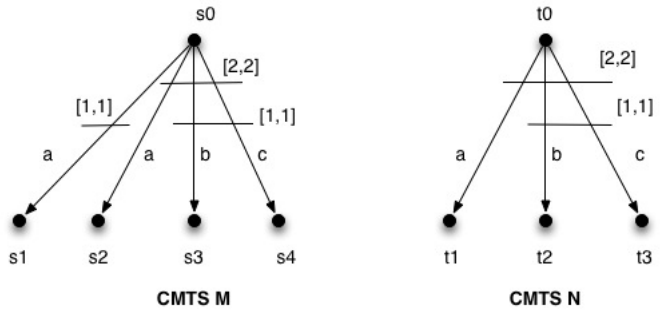


Figure 3.14: An example of two non-deterministic CMTSs

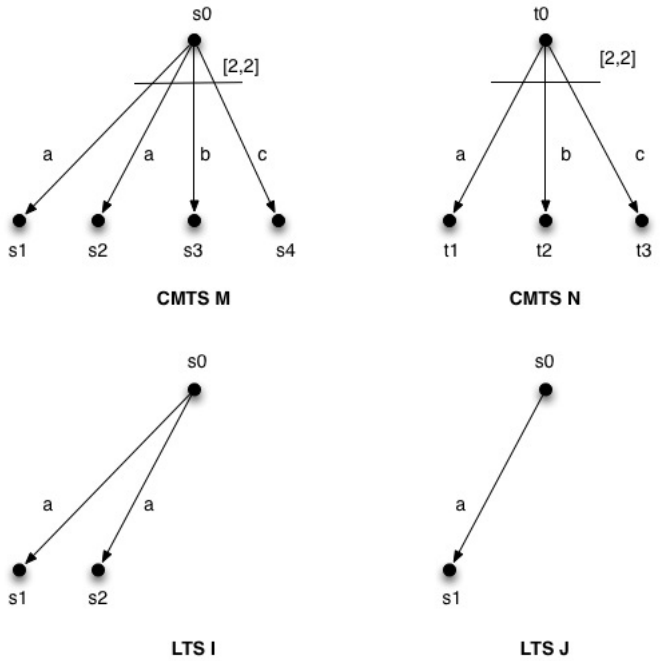


Figure 3.15: An example of problem in the refinement of non-deterministic CMTSs

set of all equivalent transitions as choice set and $[1, 1]$ as cardinality. In our case, in Figure 3.16 M has a constraint $< \{(b, s_2), (b, s_3)\}, [1, 1] >$ and this solve our problem. Unfortunately, this solution is not correct and also in this case we could lose some requirements. In effect the semantics of N is $[\![N]\!] = \{\{(a, t_1), (c, t_3)\}, \{(b, t_2)\}\}$, whereas the semantics of M is $[\![M]\!] = \{\{(a, s_1), (b, s_3)\}, \{(b, s_2), (c, s_4)\}\}$, so for our syntactic refinement $M \trianglelefteq N$, but from semantic pointview M is not a refinement of N.

A solution does not likely exist because we would like to reason about refinement in a syntactic way, as we have previously described, by means of using of semantic concepts like equivalence between transitions. From another pointview, we would want to find out a way to count only transitions which are different according to a

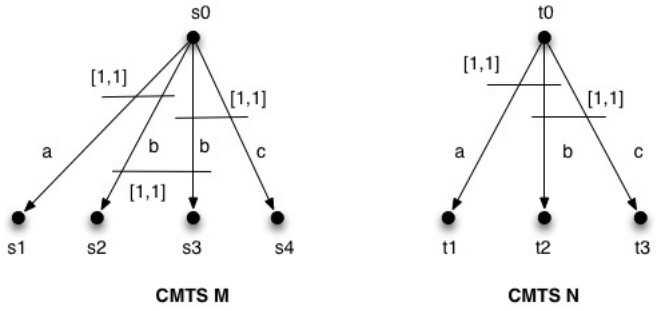


Figure 3.16: Another example of problem in the refinement of non-deterministic CMTSs

some relation $\mathcal{R}$, namely a semantic concept, by exploiting constraints and transitions which are typical syntactic concepts and they know nothing about equivalence or refinement.

In this case the only possible useful refinement is the semantic modal refinement which handles directly the semantics of constraint.

3.2 Minimalization problem

In this section we would like to understand if, taken a CMTS M, it is possible to find a CMTS M' such that $[\![M]\!] = [\![M']\!]$, namely they are semantically equivalent but M' has less constraints than M. Of course if it is possible then we would like to reason about the CMTS with the minimal number of possible constraints, in effect our refinements are computed by means of constraints, hence from computational pointview it is useful to determine the semantics using less constraints as possible.

First of all, we define a particular type of constraint which is a *witness* of the semantics of a set of constraints S and a constraint c.

Definition 3.28:
Let S be a set of constraints of a CMTS. Then we extend the function $Choice$ to a set of constraints in the following way:

$$Choice(S) = \bigcup_{c \in S} Choice(c)$$

■

Definition 3.29 (Witness of the semantics of a set of constraints):
Let c be a consistent constraint of a CMTS and S be a consistent set of constraints of a CMTS, such that $Choice(c) = Choice(S)$. If $[\![S]\!] \cap [\![c]\!] \neq \emptyset$ we define a *witness* for $[\![S]\!] \cap [\![c]\!]$, $w_{c,S} =< CS_w, [min_w, max_w] >$ where:

- $CS_w = Choice(c) = Choice(S)$

- $min_w = min\{|I \cap CS_w| \mid I \in [\![S]\!] \cap [\![c]\!]\}$
- $ma_w = max\{|I \cap CS_w| \mid I \in [\![S]\!] \cap [\![c]\!]\}$

■

When S and c are clear from the context they are omitted. Anyway it is simple to understand that min_w describes the minimal number of transitions in $Choice(c)$ for a generic set of transitions satisfying both S and c, whereas max_w describes the maximal number. In addition, note that the constraint $w_{c,S}$ is an over-approximation of the set of constraints $(S \cup c)$. From this last observation we can deduce the following corollary:

***Corollary* 3.5:** *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that $Choice(c) = Choise(S)$ and $w_{c,S}$ be the constraint defined as described in Definition 3.29. Then it holds:*

$$[\![c]\!] \cap [\![S]\!] \subseteq [\![w_{c,S}]\!]$$

Trivially, being $w_{c,S}$ a over-approximation, the vice versa is not always true. In addition we can see an important property between the two single constraints c and $w_{c,S}$:

Theorem 3.17. *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that $Choice(c) = Choise(S)$ and $w_{c,S}$ be the constraint defined as described in Definition 3.29. Then it holds:*

$$[\![w_{c,S}]\!] \subseteq [\![c]\!]$$

Proof.
Initially, taken $w_{c,S} =< CS_w, [min_w, max_w] >$ and $c =< CS_c, [min_c, max_c] >$, note that $CS_c = CS_w$ for construction. Moreover for construction, we know that :

- $\exists J \in [\![S]\!] \cap [\![c]\!].\ |J \cap CS_c| = min_w$
- $\nexists K \in [\![S]\!] \cap [\![c]\!].\ |K \cap CS_c| < min_w$

Therefore we can deduce that surely $min_c \leq |J \cap CS_c|$ is true, seeing that $J \in [\![c]\!]$, deducing $min_c \leq min_w$

By using the same reason we can demonstrate that $max_w \leq max_c$. For Theorem 3.5, we can conclude the theorem. □

The vice versa is not true, trivially. In effect if $\forall I \in [\![S]\!].\ |I \cap Choice(c)| > min_c$ then we have $min_w > min_c$ hence, taken a set of transitions J such that $|J \cap Choice(c)| = min_c$, we can conclude that $J \in [\![c]\!]$ but $J \notin [\![w_{c,S}]\!]$.

Now we introduce an important concept about constraints:

Definition 3.30 (**Useless constraint**)**:**
Let S be a set of constraints and $c \in S$ be a constraint. We call c *useless constraint* regarding the semantics of S if and only if it holds:

$$[\![S]\!] = [\![S \setminus c]\!]$$

■

The meaning of useless constraint regarding a set of constraints S is simple: if we delete the useless constraint from S then the semantics of S does not change, that is the useless constraint does not add useful information about the semantics.

Theorem 3.18. *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that $Choice(c) = Choise(S)$ and $w_{c,S}$ be the constraint defined as described in Definition 3.29. Then it holds:*

$$[\![w_{c,S}]\!] \cap [\![S]\!] = [\![S]\!] \cap [\![c]\!]$$

Proof.
We know that $[\![w_{c,S}]\!] \subseteq [\![c]\!]$, so it is simple to understand $[\![w_{c,S}]\!] \cap [\![S]\!] \subseteq [\![S]\!] \cap [\![c]\!]$.

We must only demonstrate the vice versa. We know that $[\![S]\!] \cap [\![c]\!] \subseteq [\![w_{c,S}]\!]$, hence it is simple to deduce $[\![S]\!] \cap [\![c]\!] \subseteq [\![w_{c,S}]\!] \cap [\![S]\!]$. □

In the following theorems of this section we assume implicitly some important hypotheses. Our starting point is to consider a set of constraints S and a constraint c such that:

1. $Choice(c) = Choice(S)$ and $|Choice(c)| > 0$
2. $[\![S]\!] \cap [\![c]\!] \neq \emptyset$
3. let $|Choice(c)| = n$ then $S = \{s_i =< CS_i, [min_i, max_i] >\}$ holds:
   - $1 \leq i \leq n$
   - $\forall i, j.\ Choice(s_i) = Choice(s_j) \Leftrightarrow i = j$, that is the constraints in S are all different
   - $\forall i.\ |Choice(s_i)| = n - 1$, that is every constraint has a choice set equals to the choice set of constraint c minus a transition

For convenience we call them the **key hypotheses**.

Theorem 3.19. *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that they satisfy the key hypotheses. Let w be the constraint defined as described in Definition 3.29 using S and c.*

If $[\![S]\!] \subseteq [\![c]\!]$ then it exists a useless constraint regarding the semantics of S and c in the set $S \cup w$.

Proof.
We know that $[\![S]\!] \subseteq [\![c]\!]$, hence we can say $[\![S]\!] \cap [\![c]\!] = [\![S]\!]$. For construction of w we know that $[\![S]\!] \cap [\![c]\!] = [\![S]\!] \subseteq [\![w]\!]$.

It is simple to understand $[\![S]\!] \cap [\![w]\!] = [\![S]\!]$ and to conclude that the constraint w is useless for the semantics of S and c. Note that the semantics of S and c, that is $[\![S]\!] \cap [\![c]\!]$ is the same of S and w, that is $[\![S]\!] \cap [\![w]\!]$. □

Another interesting observation is that also c is a useless constraint, trivially $[\![S]\!] \cap [\![c]\!] = [\![S]\!]$, seeing that $[\![S]\!] \subseteq [\![c]\!]$.

Theorem 3.20. *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that they satisfy the key hypotheses. Let w be the constraint defined as described in Definition 3.29 using S and c.*

If $[\![c]\!] \subseteq [\![S]\!]$ then it exists a useless constraint regarding the semantics of S and c in the set $S \cup w$.

Proof.
Trivially, $[\![S]\!] \cap [\![c]\!] = [\![c]\!]$, so the constraint w derivable from S and c is equivalent to c. We can so deduce that $[\![S]\!] \cap [\![w]\!] = [\![w]\!]$, concluding all constraints in S are useless. □

The most complicated case is when $[\![S]\!] \not\subseteq [\![c]\!]$ and $[\![c]\!] \not\subseteq [\![S]\!]$.

Theorem 3.21. *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that they satisfy the key hypotheses. Let w be the constraint defined as described in Definition 3.29 using S and c.*

If $[\![S]\!] \not\subseteq [\![c]\!]$ and $[\![c]\!] \not\subseteq [\![S]\!]$ then it exists a useless constraint regarding the semantics of S and c in the set $S \cup w$.

Proof.
Unfortunately, we do not know which constraint in $S \cup w$ is useless.

Case 1) suppose for absurdum that w is useless: if it is true then $[\![S]\!] \cap [\![w]\!] = [\![S]\!]$. Moreover we know that $[\![S]\!] \cap [\![w]\!] = [\![S]\!] \cap [\![c]\!]$, so we can conclude that $[\![S]\!] \subseteq [\![c]\!]$, but this result is opposite of our hypotheses, so it is impossible. We can deduce w is useful.

Case 2) suppose for absurdum that no constraint in S is useless. Moreover we suppose that w is useful. Seeing that $\forall s_i \in S.\ s_i$ is useful then we can say that $\forall s_i \in S.\ \exists I \in [\![(S \setminus s_i)]\!] \cap [\![w]\!].\ I \notin [\![s_i]\!]$, that is $\forall s_i \in S.\ \exists I \in [\![w]\!].\ I \notin [\![s_i]\!]$. Note that $\exists J \in [\![S]\!]$ such that $|J \cap Choice(w)| = min_w$ and $\exists K \in [\![S]\!]$ such that $|K \cap Choice(w)| = max_w$, for definition.

In addition we know that $I \in [\![w]\!]$, so surely $min_w \leq |I \cap Choice(w)| = |I \cap Choice(s_i)| + |I \cap r_i|$ for any s_i. Therefore $min_w - 1 \leq |I \cap Choice(s_i)|$, if $|Icapr_i| = 1$ otherwise $min_w \leq |I \cap Choice(s_i)|$, for any s_i. In the same way $|I \cap Choice(w)| = |I \cap Choice(s_i)| + |I \cap r_i| \leq max_w$, so $|I \cap Choice(s_i)| \leq max_w - 1$, if $|Icapr_i| = 1$ otherwise $|I \cap Choice(s_i)| \leq max_w$, for any s_i.

From all this information we can say $I \notin [\![s_i]\!] \Leftrightarrow (|I \cap Choice(s_i)| < min_i) \vee (|I \cap Choice(s_i)| > max_i)$ and so we can deduce that $\forall s_i.\ min_w < min_i \vee max_w - 1 > max_i$.

If $\forall s_i.\ min_w < min_i$, we have three possibilities:

1. $min_w = |Choice(w)|$, namely J has all transitions in $Choice(w)$ but in this case $\forall i.\ min_i = min_w - 1$ and it is impossible for our hypothesis

2. $0 < min_w < |Choice(w)|$, in this case exists a set of transitions J such that $|J \cap Choice(w)| = min_w$. For any constraint s_i we have $|J \cap Choice(s_i)| + |J \cap r_i| = min_w$ and, in addition, it holds $|J \cap Choice(s_i)| \geq min_i > min_w$. Seeing that $0 \leq |J \cap r_i| \leq 1$ we can deduce that $J \notin [\![S]\!]$. This is an absurdum.

3. $min_w = 0$ we know that $|J \cap Choice(w)| = 0$ is possible and $J \in [\![S]\!]$, therefore we can deduce $\forall i.\ min_i = 0$. Again it is an absurdum

In the similar way, if $\forall s_i.\ max_w - 1 > max_i$, we have three possibilities:

1. $max_w = 0$ it is simple to understand the absurdum.

2. $0 < max_w < |Choice(w)|$, in this case exists a set of transitions J such that $|J \cap Choice(w)| = max_w$. For any constraint s_i we have $|J \cap Choice(s_i)| + |J \cap r_i| = max_w$ and, in addition, it holds $|J \cap Choice(s_i)| \leq max_i < max_w - 1$. Seeing that $0 \leq |J \cap r_i| \leq 1$ we can deduce that $J \notin [\![S]\!]$. This is an absurdum.

3. $max_w = |Choice(w)|$ then $|J \cap Choice(w)| = |Choice(w)|$ is possible. In this case, for any s_i, $|J \cap Choice(s_i)| = max_w - 1$ and at the same time $|J \cap Choice(s_i)| \leq max_i < max_w - 1$, reaching an absurdum.

The last case is when we have three sets of constraints:

- $S_1 = \{s_i \mid min_w < min_i \wedge max_w - 1 \leq max_i\}$
- $S_2 = \{s_i \mid min_w < min_i \wedge max_w - 1 > max_i\}$
- $S_3 = \{s_i \mid min_i \leq min_w \wedge max_w - 1 > max_i\}$

Of course these sets are a partition of S. Now we must understand if it is possible to find:

1. $J \in [\![S]\!].\ |J \cap Choice(w)| < min_w$
2. $K \in [\![S]\!].\ |K \cap Choice(w)| > max_w$

Case 1) Of course $min_w > 0$. If $min_w = |Choice(w)|$ then some constraint s_i exists such that $min_i > |Choice(w)|$ and surely $J \notin [\![S]\!]$. In addition, seeing that $S_1 \neq \emptyset$ and $S_2 \neq \emptyset$ then taken a constraint $s_1 \in S_1$ we can say $\forall J \in [\![s_1]\!] \Leftrightarrow$

$|J \cap Choice(s_1)| = min_1 > min_w$, so surely $|J \cap Choice(s_1)| + |J \cap r_1| > min_w$, concluding that for every $J \in [\![S]\!]$. $J \cap Choice(w)| \not< min_w$

Case 2) Of course $max_w < Choice(w)$. If $max_w = 0$ then surely $K \notin [\![S]\!]$ as some constraints s_i with $max_i < -1$ should be exists and this is impossible. In addition, seeing that $S_2 \neq \emptyset$ and $S_3 \neq \emptyset$ then taken a constraint $s_2 \in S_2$ we can say $\forall K \in [\![s_2]\!] \Leftrightarrow |K \cap Choice(s_2)| \leq max_2 < max_w - 1$, so surely $|K \cap Choice(s_2)| + |K \cap r_2| < max_w$, concluding that for every $K \in [\![S]\!]$. $K \cap Choice(w)| \not> max_w$.

Seeing that min_w and max_w are computed using $[\![S]\!]$ too, then we have that semantics of S has enough information, in effect at least one set of transitions H such that $|H \cap Choice(w)| = min_w$ and I such that $|I \cap Choice(w)| = max_w$ exist, whereas no set of transitions J such that $|J \cap Choice(w)| < min_w$ and K such that $|K \cap Choice(w)| > max_w$ exist. Hence it is possible to deduce that S could be a restriction of semantics of w, then we can conclude that w is useless, obtaining an absurd. □

Theorem 3.22 (**Minimality theorem**). *Let c be a consistent constraint of a CMTS , S be a consistent set of constraints of a CMTS such that they satisfy the key hypotheses. Let w be the constraint defined as described in Definition 3.29.*

Then it exists a useless constraint regarding the semantics of S and c in the set $S \cup w$

Proof.
The proof follows from Theorem 3.19, Theorem 3.20 and Theorem 3.21. □

Now suppose to delete one condition from the minimality theorem, in the specific the condition $[\![S]\!] \cap [\![c]\!] \neq \emptyset$ then it exists some cases where all constraints are needed. The reason about that is we do not compute the constraint w because the intersection is empty.

Example 3.9. *Suppose we have $S = \{s_1, s_2, s_3\}$ such that:*

- $s_1 =< \{a, b\}, [0, 1] >$
- $s_2 =< \{a, c\}, [0, 1] >$
- $s_3 =< \{b, c\}, [0, 1] >$

Their semantics is simple:

- $[\![s_1]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, c\}, \{b, c\}\}$
- $[\![s_2]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}\}$
- $[\![s_3]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}\}$

Moreover we suppose $c =< \{a, b, c\}, [2, 2] >$ where its semantics is $[\![c]\!] = \{\{a, b\}, \{b, c\}, \{a, c\}\}$.

Trivially, $[\![S]\!] \cap [\![c]\!] = \emptyset$. In addition we have not got useless constraints:

- $[\![S]\!] = \{\emptyset, \{a\}, \{b\}, \{c\}\}$
- $[\![s_1]\!] \cap [\![c]\!] = \{\{a, c\}, \{b, c\}\}$
- $[\![s_2]\!] \cap [\![c]\!] = \{\{a, b\}, \{b, c\}\}$
- $[\![s_3]\!] \cap [\![c]\!] = \{\{a, b\}, \{a, c\}\}$
- $[\![s_1 \cup s_2]\!] \cap [\![c]\!] = \{\{b, c\}\}$
- $[\![s_1 \cup s_3]\!] \cap [\![c]\!] = \{\{a, c\}\}$
- $[\![s_2 \cup s_3]\!] \cap [\![c]\!] = \{\{a, b\}\}$

In our dissertation we have introduced key hypotheses that are very restrictive. For each state s we could define a constraint tree, where the root is the constraint with choice set $Trans(s)$ and leaves is singleton constraints. Each internal node c is connected to its child node c_1, if $Choice(c_1) = Choice(c) \setminus t$ where t is a particular transition, namely each child node has a choice set equivalent to father one minus a transition. By means of our minimality theorem we can reason about only one level. We would like to understand if the minimality is maintained between different levels. For example in Figure 3.17 we describe the situation of our constraint tree when we handle only one level of constraints. In this case we have a constraint with a choice set of size N and N constraints related to it such that their choice set has size $N - 1$. Instead, in Figure 3.18 we describe the situation of our constraint tree

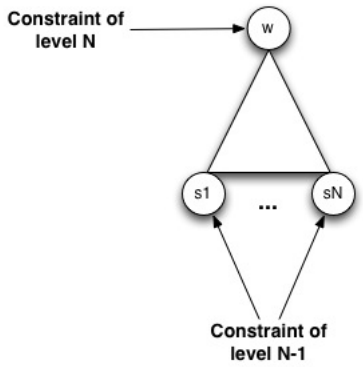


Figure 3.17: A graphical idea of the single level of a constraint tree

when we handle two levels of constraints. In this case we describe the situation which we consider in the following theorem: we handle a constraint w with a choice set of size N, a set S of constraints related to w such that the size of S is N and for each constraint $s_i \in S$ the choice set of s_i has size $N - 1$. Finally, taken a specific constraint $s_j \in S$ we consider a set T of constraints related to s_j such that T has size $N - 1$ and each constraint $t_k \in T$ has a choice set with size $N - 2$.

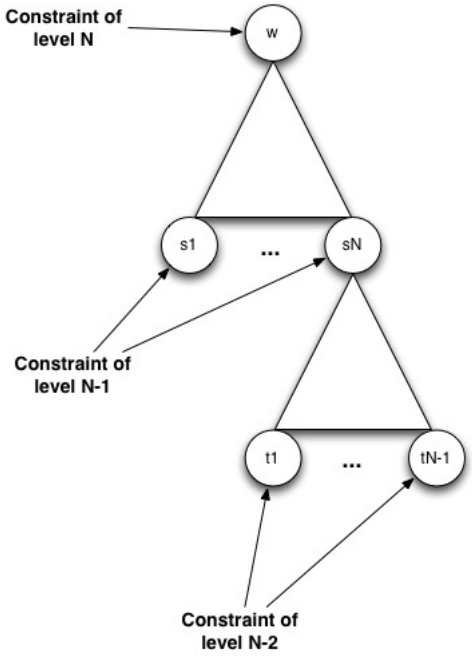


Figure 3.18: A graphical idea of more levels of a constraint tree

Theorem 3.23. *Let c be a constraint such that $|Choice(c)| = n$, S be a set of n distinct constraints such that $\forall 1 \leq i \leq n.\ |Choice(s_i)| = n-1$ and T be a set of $n-1$ distinct constraints such that $\forall 1 \leq i \leq n.\ |Choice(t_i)| = n-2$.*

If the following properties hold:

1. $Choice(c) = Choice(S)$
2. $\exists 1 \leq k \leq n.\ Choice(s_k) = Choice(T)$
3. $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] \neq \emptyset$

Then at least one useless constraint exists in the set $c \cup S \cup T$

Proof.
From Theorem 3.22 we know that at least one useless constraint between w and S exists, where w is derived by c and S. In addition, at least one useless constraint between w_k and T exists, where w_k is derived by s_k and T. Some different cases can happen:

1. w is a useless constraint and all constraints in S are useful:
   (a) $\exists 1 \leq l \leq n-1.\ t_l \in T$ is useless
   (b) w_k is useless
2. w is a useful constraint and a constraint $s_j \in S$ is useless:

(a) the useless constraint s_j is the constraint s_k, namely $j = k$ and we have two possibilities:
   (i) w_k is the useless constraint and T has only useful constraints
   (i) w_k is the useful constraint and $\exists t_h \in T$ which is the useless constraint

(b) the useless constraint s_j is not the constraint s_k, namely $j \neq k$ and we have two possibilities:
   (i) w_k is the useless constraint and T has only useful constraints
   (i) w_k is the useful constraint and $\exists t_h \in T$ which is the useless constraint

Case 1) Seeing that w is useless we have $[\![S]\!] \cap [\![w]\!] = [\![S]\!]$. We have two possibilities:

(a) w_k is useful, namely $[\![s_k]\!] \cap [\![T]\!] = [\![w_k]\!] \cap [\![T]\!] = [\![w_k]\!] \cap [\![T \setminus t_l]\!]$. Since w is useless then c is a useless constraint too. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![S]\!] \cap [\![T]\!] = [\![S \setminus s_k]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![S \setminus s_k]\!] \cap [\![w_k]\!] \cap [\![T]\!] = [\![S \setminus s_k]\!] \cap [\![w_k]\!] \cap [\![T \setminus t_l]\!]$. In this context two constraints are useless: w and t_l.

(b) w_k is useless, namely $[\![s_k]\!] \cap [\![T]\!] = [\![w_k]\!] \cap [\![T]\!] = [\![T]\!]$, deducing s_k is useless too. Since w is useless then c is a useless constraint too. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![S]\!] \cap [\![T]\!] = [\![S \setminus s_k]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![S \setminus s_k]\!] \cap [\![T]\!]$. In this context two constraints are useless: w and s_k.

Case 2) This case is slightly more complicated because we have several possible situations.

Case 2.a) This time w is useful and s_k is the useless constraint, namely $[\![c]\!] \cap [\![S]\!] = [\![w]\!] \cap [\![S]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!]$.

Case 2.a.i) w_k is useless and so $[\![T]\!] \cap [\![s_k]\!] = [\![T]\!] \cap [\![w_k]\!] = [\![T]\!]$. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!] \cap [\![T]\!]$. We have only one useless constraint: s_k.

Case 2.a.ii) w_k is useful and so $[\![T]\!] \cap [\![s_k]\!] = [\![T]\!] \cap [\![w_k]\!] = [\![w_k]\!] \cap [\![T \setminus t_l]\!]$. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!] \cap [\![s_k]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!] \cap [\![w_k]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!] \cap [\![w_k]\!] \cap [\![T \setminus t_l]\!]$.

We know that s_k is useless so $[\![w]\!] \cap [\![S]\!] = [\![w]\!] \cap [\![S \setminus s_k]\!]$ and this is true if and only if $[\![w]\!] \cap [\![S \setminus s_k]\!] \subseteq [\![s_k]\!]$. In addition $[\![w_k]\!] \subseteq [\![s_k]\!]$. Unfortunately we have not a relation between $[\![w_k]\!]$ and $[\![w]\!] \cap [\![S \setminus s_k]\!]$, hence w_k is useful as it contains the missing information of t_l.

Case 2.b) w is useful and $s_j \neq s_k$, so we have $[\![c]\!] \cap [\![S]\!] = [\![w]\!] \cap [\![S]\!] = [\![w]\!] \cap [\![S \setminus s_j]\!]$.

As before we can have two possibilities:

Case 2.b.i) w_k is useless and so $[\![T]\!] \cap [\![s_k]\!] = [\![T]\!] \cap [\![w_k]\!] = [\![T]\!]$, so s_k is useless too. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_j]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus (s_j \cup s_k)]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus (s_j \cup s_k)]\!] \cap [\![T]\!]$. We have two useless constraint: s_k and s_j.

Case 2.b.ii) w_k is useful and so $[\![T]\!] \cap [\![s_k]\!] = [\![T]\!] \cap [\![w_k]\!] = [\![w_k]\!] \cap [\![T \setminus t_l]\!]$. Therefore $[\![c]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus s_j]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus (s_j \cup s_k)]\!] \cap [\![s_k]\!] \cap [\![T]\!] = [\![w]\!] \cap [\![S \setminus (s_j \cup s_k)]\!] \cap [\![w_k]\!] \cap [\![T \setminus t_l]\!]$.

For each situation at least one constraint is useless and our theorem is true. □

Definition 3.31:
Let s be a state. Then we say that s has a *minimal set of constraints* if and only if it holds:

$$\forall c \in \mathfrak{C}(s).\ [\![\mathfrak{C}(s)]\!] \neq [\![\mathfrak{C}(s) \setminus c]\!]$$

■

Definition 3.32:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ be a CMTS. Then we say that M is *minimal* if and only if it holds:

$$\forall s \in \mathcal{S}\ s \text{ has a minimal set of constraints}$$

■

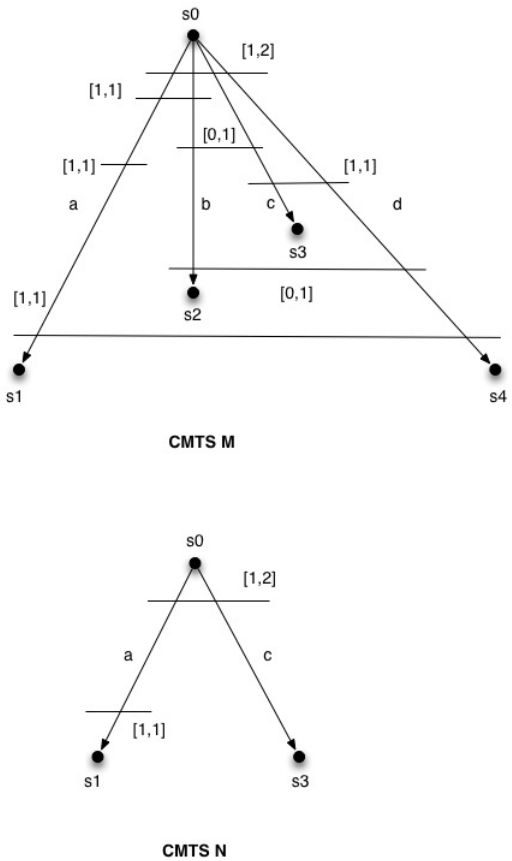


Figure 3.19: Two semantically equivalent CMTSs with a different number of constraints

This theorem extends the minimality from local context to global one. Unfortunately, these theorems are only existence theorems and it is an open problem if, taken a CMTS M, we can derive the exact minimal number of possible constraints.

Moreover it is simple to define some algorithms which allow to compute, taken a CMTS M, the minimal CMTS semantically equivalent to M. In addition we can develop in a simple way an algorithm to determine if a CMTS M is consistent or not. For more detail it is possible to see the Appendix.

Last observation: in a few contexts we may have a CMTS with an high number of constraints like one in Figure 3.19. As we have seen, each new constraint reduces the semantics of a CMTS, hence in general increasing the number of constraints, we decrease valid sets of transitions for constraints themself.

In Figure 3.19 CMTS M has many constraints but the only possible set of correct transitions are $\{\{(a, s_1)\}, \{(a, s_1), (c, s_3)\}\}$, for example the CMTS N is semantically equivalent to M and it has few constraints. Of course each modeller can describe the family in some different ways, but in general a very high number of constraints is symptom of a wrong modelling.

3.3 No-Choice CMTS

The CMTS is a peculiar formalism because it uses the concept of a strange constraint. When we consider a constraint $c =< CS, [min, max] >$, if we want to generate a set of transitions which satisfies c, we have two types of choices:

1. the number of transitions in CS which we want to take
2. the exact transitions of CS to be taken

We can observe that a special class of CMTS, where we have no choice, exists.

Definition 3.33 (**No-Choice CMTS**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS. We say that M is a No-Choice CMTS if and only if holds:

1. $\forall s \in \mathcal{S}.\ \forall c =< CS, [min, max] > \in \mathfrak{C}(s).\ |CS| = min = max$
2. $\forall s \in \mathcal{S}.\ \forall t \in Trans(s).\ \exists c \in \mathfrak{C}(s).\ t \in Choice(c)$

■

Consider the situation where we have a constraint $c =< CS, [min, max] >$ where $min = max$. In this case we do not choose the number of transitions to be taken because each possible valid solution I must hold $min = |I \cap CS| = max$, therefore we can only choose which transitions in CS must be taken. If we add the condition $|CS| = min = max$ then we cannot even choose which transitions may be taken, because we have only one possibility: we must consider all transitions of CS.

The second property guarantees us every possible transition is connected to some constraint, so we have the guarantee that each possible transition must be always taken. We have previously said if a transition t is not related to any constraint, then we can implicitly assume that exists a constraint $c =< \{t\}, [0, 1] >$, so if we would have only first property, then we might have a transition $t \in Trans(s)$, which is not related to a constraint. We can so deduce that exists an implicit constraint c which does not satisfy property (1), therefore our CMTS is not no-choice. Moreover, we

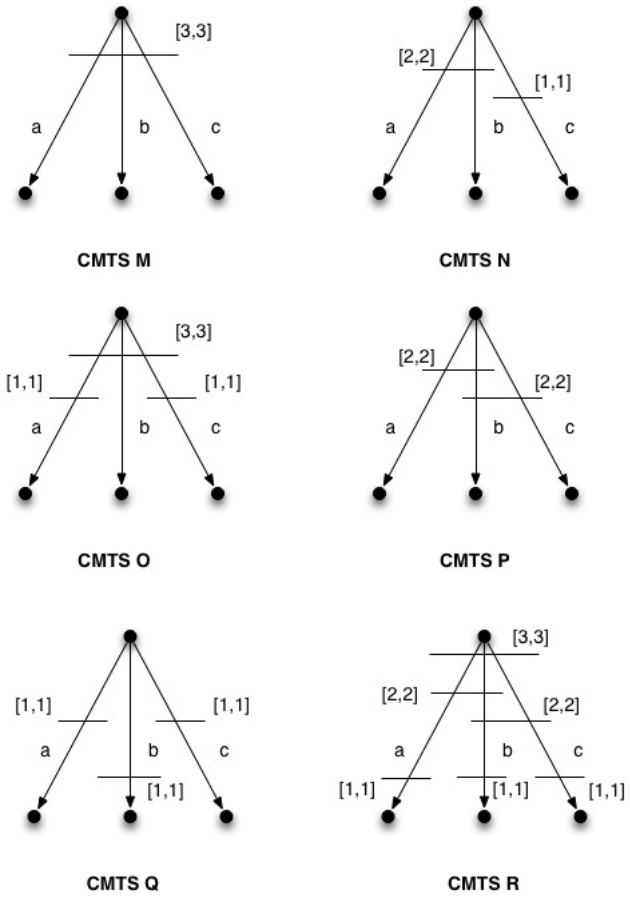


Figure 3.20: Some examples of No-Choice CMTSs

know that, taken a generic CMTS M, it is possible to define some different CMTSs N such that M and N are semantically equivalent. Of course, in the context of CMTS No-Choice this property is still held but, due to the particular structure of a CMTS No-Choice, it implies some further characteristics. As we can see in the Figure 3.20, taken a No-Choice CMTS, we can find out some syntactic different No-Choice CMTSs which are all semantically equivalent. Therefore we could want to a "witness" or a "descriptor" which represents the class of these all semantically equivalent CMTSs.

Definition 3.34 (**No-Choice CMTS Witness**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS. We say that M is a *Witness* of a set of semantically equivalent No-Choice CMTSs if and only if it holds:

1. M is a No-Choice CMTS
2. $\forall s \in \mathcal{S}.\ \forall c =< CS, [min, max] >\in \mathfrak{C}(s).\ |CS| = min = max = 1$

We call **Normal Form** this CMTS and we denote it by $NF(M)$.

■

A Normal Form of a CMTS is a CMTS where each transition is related to a constraint with a choice set singleton and cardinality equals to $[1, 1]$. Moreover note that a Normal Form is not the minimal CMTS, in effect the minimal No-Choice CMTS is a CMTS which satisfies the following property:

$$\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \{c\}$$

where:

1. $c =< CS, [min, max] >$
2. $CS = Trans(s) \wedge min = max = |CS|$

In addition, note that for each couple of CMTS No-Choice (C, C') of Figure 3.20 it always holds $C \trianglelefteq^{Sem} C'$, whereas sometimes $C \trianglelefteq^{Syn} C'$ does not hold. The reason is simple $\trianglelefteq^{Sem}$ take into account the semantics derived by constraints, therefore the relation abstracts from the real syntactic structure of the CMTS and it considers only the global meaning of all constraints. On the other hand, $\trianglelefteq^{Syn}$ considers the syntactic description of constraints and, hence, some problem might exist, for example $M \not\trianglelefteq N$, even if they describe the same set of solutions.

Definition 3.35 (**Construction of Normal Form**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a No-Choice CMTS. We can easily derive $NF(M)$ in the following way:

$$NF(M) = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}_{NF(M)}, s_0)$$

where:

1. $\forall s \in \mathcal{S}.\ \forall c =< CS, [min, max] >\in \mathfrak{C}_{NF(M)}(s).\ |CS| = min = max = 1$, namely every constraint has a choice set singleton and cardinality $[1, 1]$
2. $\forall s \in \mathcal{S}.\ \forall t \in Trans(s).\ \exists c \in \mathfrak{C}_{NF(M)}(s).\ Choice(c) = \{t\}$, namely every transition is related to a constraint

■

For our purpose a product of a product family is a No-Choice CMTS. In the following chapters we will show that each No-Choice CMTS is equivalent to a LTS, therefore we will use indistinctly LTS, product and implementation to denote the same thing.

Theorem 3.24 (**Uniqueness of Normal Form**). *Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a No-Choice CMTS, then its Normal Form is unique.*

Proof.
Suppose for absurdum that we have two different normal forms N and N'. Since N and N' are normal forms of M, namely they have only constraints with choice set as singleton and cardinality equals to $[1, 1]$, and at the same time N and N' are different, then one of the following properties holds:

1. $\exists s \in \mathcal{S}.\ \exists \alpha \in \Sigma, s' \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow_N \wedge\ (s, \alpha, s') \in \longrightarrow'_N$
2. $\exists s \in \mathcal{S}.\ \exists c =< CS_c, [min_c, max_c] > \in \mathfrak{C}_N(s)$. such that:
   $\forall c' =< CS_{c'}, [min_{c'}, max_{c'}] > \in \mathfrak{C}_{N'}(s).\ (CS_c \neq CS_{c'}) \vee (min_c \neq min_{c'}) \vee (max_c \neq max_{c'})$

The first case holds when $\longrightarrow_N \neq \longrightarrow'_N$, that is some outgoing transitions of some state s is not present in N', but we know $\longrightarrow_M = \longrightarrow_N = \longrightarrow_{N'}$, for construction of Normal Form, so this case is impossible.

The second case holds if N has some constraint c which does not exists in N'. As before, for construction of Normal Form, we know that for every state we have as many constraints as transitions, each constraint has a singleton choice set and cardinality equals to $[1, 1]$. Therefore we can deduce that every constraint in N and N' have cardinality equals to $[1, 1]$, so the second case is possible if and only if a constraint in N with a different choice set regarding all constraints in N' exists, but also this is impossible because each constraint in N and N' has a singleton choice set related to a particular transition and $\longrightarrow_N = \longrightarrow_{N'}$, deducing all constraints in N are equivalent to constraints in N', so the second case is impossible too. In this way we can reach the absurdum. □

Note that we can generalize the normal form concept to a general CMTS but in this case the uniqueness is lost, because we have some different possible Normal forms which are as many as implementations of CMTS are. The no-uniqueness derives from missing of no-choice property, in effect if we have some choices we have some different Normal Forms, depending by the made choices.

Theorem 3.25. *The semantics of a No-Choice CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ is a singleton set and it is a LTS $L = (\mathcal{S}, \Sigma, \longrightarrow)$.*

Proof.
Initially, we prove that for each state s_M of M its semantics is a singleton set. We

know that $I \in [\![s_M]\!] \Leftrightarrow \forall c \in \mathfrak{C}(s_M).\ I \in [\![c]\!] \Leftrightarrow \forall c \in \mathfrak{C}(s_M).\ min \leq |I \cap Choice(c)| \leq max$. In the No-Choice context the property $\forall c \in \mathfrak{C}(s_M).\ min \leq |I \cap Choice(c)| \leq max$ is equivalent to $\forall c \in \mathfrak{C}(s_M).\ |I \cap Choice(c)| = |Choice(c)|$. Trivially, we can derive that $\forall c \in \mathfrak{C}(s_M).\ Choice(c) \subseteq I$. Seeing that $\forall t \in Trans(s_M).\ \exists c \in \mathfrak{C}(s_M).\ t \in Choice(s_M)$, then we can conclude that $\forall t \in Trans(s_M).\ t \in I$, where I is the possible solution. Of course, another solution does not exists. In fact suppose for absurdum that I_1 is a solution and it is different from I then $I_1 = Trans(s_M) \setminus \{t\}$ for some transition t. For definition of No-Choice, taken t, surely it exists a constraint c_t such that $t \in Choice(c_t)$ and, therefore, $|I_1 \cap Choice(c_t)| = |I_1 \cap Choice(c_t) \setminus \{t\}|$, since t is not in I_1, deriving that $|I_1 \cap Choice(c_t)| < min_{c_t} = |Choice(c_t)|$. In conclusion $I_1 \notin [\![c_t]\!]$, namely I_1 is not a solution but this is an absurd.

Finally, we know that a LTS L is derived by a CMTS M if, for each state s_L, its outgoing transitions are a possible solution of constraints of the corresponding state s_M. Since s_M has only one possible solution I then $Trans(s_L) = \{(\alpha, s'_L) \mid (\alpha, s'_M) \in I = Trans(s_M)\}$, namely all possible outgoing transitions of s_M are the same of s_L. Hence, a LTS L exists and it has all transitions of M and it is impossible to have another solution.

□

It is clear that the refinement concept in No-Choice CMTS is few significant: the only possible CMTS which refines a No-Choice CMTS is an inconsistent CMTS. In this context it is more interesting to define an equivalence relation between No-Choice CMTSs, using a relation similar to refinement one.

Definition 3.36 (**Equivalence relation between No-Choice CMTSs**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}_M, s_{M_0})$ and $N = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}_N, s_{N_0})$ we say M and N are equivalent, denoted by $M \equiv N$, if and only if $(s_{M_0}, s_{N_0}) \in \mathcal{R}_{\equiv}$ where $(s, t) \in \mathcal{R}_{\equiv}$ if and only if:

1. $\forall(\alpha, s') \in Trans(s).\ \exists(\alpha, t') \in Trans(t).\ (s', t') \in \mathcal{R}_{\equiv}$
2. $\forall(\alpha, t') \in Trans(t).\ \exists(\alpha, s') \in Trans(s).\ (s', t') \in \mathcal{R}_{\equiv}$

■

Definition 3.37:
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}_M, s_{M_0})$ and $N = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}_N, s_{N_0})$ be two CMTSs No-Choice, we say M and N are equivalent, denoted by $M \equiv N$, if and only if $NF(M) \trianglelefteq NF(N)$.

■

This definition allow us to connect the equivalence concept to normal form and refinement relation.

Theorem 3.26. *The Definition 3.36 and Definition 3.37 are equivalent.*

Proof.
First of all, note that for the peculiar structure of the normal form we have only constraints $c =< \{t\}, [1,1] >$ for any $t \in Trans(s)$. So the condition of refinement over constraints can be simplified: $\forall c_t \in \mathfrak{C}_N(t).\ \exists c_s \in \mathfrak{C}_M(s).\ Choice(c_t) = \{(\alpha, t')\} \wedge Choice(c_s) = \{(\alpha, s')\} \wedge (s', t') \in \mathcal{R}$.

Therefore $(s,t) \in \mathcal{R}$ is equivalent to say:

- $(s, \alpha, s') \in \longrightarrow_M \Longrightarrow \exists (t, \alpha, t') \in \longrightarrow_N .\ (s', t') \in \mathcal{R}$
- $\forall c_t \in \mathfrak{C}_N(t).\ \exists c_s \in \mathfrak{C}_M(s).\ Choice(c_t) = \{(\alpha, t')\} \wedge\ Choice(c_s) = \{(\alpha, s')\} \wedge (s', t') \in \mathcal{R}$.

Trivially the first condition of refinement is equivalent to the first condition of relation $R_{\equiv}$. Moreover it is simple to see as the second condition of $R_{\equiv}$ is the same of second condition of simplified refinement relation.

In effect if $(\alpha, t') \in Trans(t)$ then $\exists c_t \in \mathfrak{C}_N(t).\ Choice(c_t) = \{(\alpha, t')\}$, in addition if $(\alpha, t') \in Trans(t)$ implies that $\exists (\alpha, s') \in Trans(s).\ (s', t') \in \mathcal{R}_{\equiv}$ then $\exists c_s \in \mathfrak{C}_N(s).\ Choice(c_s) = \{(\alpha, s')\} \wedge (s', t') \in \mathcal{R}$.

Conversely if exists a $c_t \in \mathfrak{C}_M(t).\ Choice(c_t) = \{(\alpha, t')\}$ then in M it holds $(\alpha, t') \in Trans(t)$, so seeing that $c_t \in \mathfrak{C}_M(t) \Rightarrow \exists c_s \in \mathfrak{C}_N(s).\ Choice(c_t) = \{(\alpha, t')\} \wedge Choice(c_s) = \{(\alpha, s')\} \wedge (s', t') \in \mathcal{R}$ we can say that $c_s \in \mathfrak{C}_N(s).\ c_s = \{(\alpha, s')\}$ exists and therefore in N it holds $(\alpha, s') \in Trans(s)$ and $(s', t') \in \mathcal{R}$, concluding our theorem. □

Theorem 3.27. *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$ and $N = (\mathcal{S}_M, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs No-Choice, such that $M \equiv N$. Then M and N are semantically equivalent*

Proof.
We know that the only LTS L_M derived from M is a LTS with all transition of M, namely $L_M = (\mathcal{S}_M, \Sigma, \longrightarrow_M)$ and the same holds for N, namely $L_N = (\mathcal{S}_N, \Sigma, \longrightarrow_N)$. In addition, we know that $M \equiv N$, hence $L_M \equiv L_N$. Note that, in the LTS context the relation $\equiv$ is the bisimulation, concluding that M and N are semantically equivalent. □

Chapter 4

Extensions of CMTS

In the previous section we described a new formalism, the CMTS, and we saw some interesting properties related to it. In this chapter we will see some simple extensions of CMTSs and how previous definitions will be changed. Of course the CMTS is a very expressive formalism, so it is difficult to understand why we need to define further extensions. The main reason is related to the concept of "*conditional features*", namely features such that their existence (or absence) depends on some conditions. In each model which we saw in Section 2.2 we can only define if a feature must be present or may be present or, taken a set of features, how many of them must be at least taken or at most taken. Therefore it is impossible to define that the presence of a transition is only dependent on the presence of another specific transition. Unlike all other formalisms, the PMTS allows to define conditional requirements and it introduces the negation in the obligation function, in this way we can say if a transition must be present or may be present and, in addition, we can also say if a transition needs not be present, in particular we can define if it must be always absent or only in some special cases. Note that in all formalisms we implicitly assume that if a feature is not represented by some outgoing transitions of the state s, then it is forbidden or absent for the state s. On the other hand, a feature can exist if a transition related to it exists and, in this case, we describe in some way if a transition must or may be present and what sets of transitions are correct. The negation is important because we can reason about the absence of a feature in a direct way, even if a transition related to it exists.

Observe that for all other formalisms this aspect is implicit: a feature is absent if and only if it is not present, however they do not allow to handle directly requirements which need to the absence of a transition in some specific contexts. The possibility of handling the absence in a direct way it is fundamental to define conditional features: we can force two different transitions to be absent or present simultaneously in each possible contexts. This is possible by means of the definition of two different representations of the same transition: one for the absence concept, another one for the presence concept.

Moreover, the introduction of the negation in a PMTS allows to derive a more

interesting property, we can decide in a more selective way what sets of transitions are valid, for example we can require that some transitions are absent only in certain sets, whereas they are present only in other specific sets.

For instance, taken a state s and the set of outgoing transitions $Trans(s) = \{(\alpha, s'), (\beta, s'')\}$, we can consider the requirement R where only $\{(\alpha, s'), (\beta, s'')\}$ and $\emptyset$ are correct sets of transitions. Note that all formalisms in Section 2.2, except PMTS, cannot represent this requirement because surely both (α, s') and (β, s'') are may transitions and, at the same time, they might be not must transitions. From this deduction we can also derive that the set of transitions $\{(\alpha, s')\}$ is possible, but this is wrong for our requirement $\mathcal{R}$.

Unfortunately, the CMTS has this lack too. By means of the CMTS, we can only define the minimal and the maximum number of transitions to be considered, but we are not able to express that a particular transition must be absent in some specific cases or it must be present in other ones, so we are not able to handle in an explicit way the absence and the presence of a transition.

For a better understanding, we consider the requirement R again for a some state s, and for convenience, we suppose that $Trans(s) = \{(\alpha, s'), (\beta, s'')\}$, namely we have only two outgoing transitions. The requirement has only the following correct sets of transitions: $\{\{\emptyset\}, \{(\alpha, s'), (\beta, s'')\}\}$.

In addition we can observe that both (α, s') and (β, s'') are not necessary, therefore we can define the following possible constraints:

- $c_1 =< \{(\alpha, s')\}, [0, 1] >$
- $c_2 =< \{(\beta, s'')\}, [0, 1] >$
- $c_3 =< \{(\alpha, s'), (\beta, s'')\}, [0, 2] >$

Unfortunately, for these constraints, also the set $\{(\alpha, s')\}$ is valid. Of course we cannot modify the minimum of an any constraint, because the set $\{\emptyset\}$ is valid and, in the same way, we cannot modify the maximum of an any constraint seeing that the set $\{(\alpha, s'), (\beta, s'')\}$ is possible. On the other hand, we cannot add further constraints because all possible constraints, which are definable for the state s, can have one of these choice sets: $\{(\alpha, s')\}$, $\{(\beta, s'')\}$, $\{(\alpha, s'), (\beta, s'')\}$.

Intuitively, we realize that the requirement R cannot be expressed by means of a CMTS. Our extensions will be introduced to resolve these lacks.

4.1 CMTS($\mathcal{G}_\mathcal{T}$)

The first step is to understand how we can handle the presence and the absence of a feature in an explicit way. First of all, we observe that the concept of presence and absence of a feature exists in two different levels:

1. **model level**:

- a feature α is *present* in a state s of a CMTS if and only if $\exists(\alpha, s') \in Trans(s)$
- a feature α is *absent* in a state s of a CMTS if and only if $\not\exists(\alpha, s') \in Trans(s)$

2. **solution level**: taken a state s of a CMTS and a solution T for requirements of s

   - a feature α is *present* in T if and only if $\exists(\alpha, s') \in T$
   - a feature α is *absent* in T if and only if $\not\exists(\alpha, s') \in T$

Note that we are only interested to features which are *present* in the model level and, at the same time, *present* or *absent* in the solution level, therefore all *absent* features in the model level are automatically discarded. We want to handle in a direct way the absence of features in the solution level, therefore we must divide in some way the representation of a feature, namely a labelled transition, in two different representations: one for the presence and one for the absence of the feature in the solution. We implicitly assume that all features to be considered are *present* in the model level.

In addition we can remember that, taken a state s, a solution of s describes a piece of an implementation, namely the part of implementation related to the state s. Therefore if a feature α is *present* in a solution T of s then a labelled transition in T exists and it can be **executable**, on the other hand if α is *absent* in T then surely the transition labelled with α is an outgoing transition of s, but it must not be executable for T. From this observation we can deduce that if we choose the representation of the presence of the feature α then α can be executable, whereas if we choose the representation of the absence of α then α must be not executable. At the same time these two representations must be present as outgoing transitions of s, in effect both represents the same feature but they describe two different situations, namely when the feature must be present and when the feature must be absent. It is important to understand that in this context we allow to choose in a direct way if a feature is present or absent, through the choice between two similar labelled transitions: they have the same label but one is always executable, whereas the other one is never executable.

It is clear that we must model in some way the possibility of executing or not the transition. In order to solve this problem we introduce the concept of labelled transitions with guards and, initially, we suppose that our guards can only assume values **tt** or **ff**, namely *true* and *false*, respectively. For convenience we call the set $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\}$ **Guard Set**. Of course, now our transitions have type $\mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$, where $\mathcal{S}$ is the set of states and Σ is the set of actions. To distinguish clearly outgoing transitions with or without guards, in the following we denote by $\mathfrak{Trans}$ the type of outgoing transitions without guards, and $\mathfrak{Trans}(\mathcal{G})$, the type of outgoing transitions with guards belonging to $\mathcal{G}$, namely $\mathfrak{Trans} = \Sigma \times \mathcal{S}$ and $\mathfrak{Trans}(\mathcal{G}) = \mathcal{G} \times \Sigma \times \mathcal{S}$.

Moreover we must update the definition of $Trans$:

Definition 4.1:
Let $s \in \mathcal{S}$ be a state, Σ be a set of actions, $\mathcal{G}$ be a set of guards and $\longrightarrow \subseteq \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ be a transition relation. Then we denote the outgoing transitions of s by $Trans(s)$ where:

$$Trans(s) = \{(g, \alpha, s') \mid (s, g, \alpha, s') \in \longrightarrow\}$$

■

Definition 4.2 (**Enabling of transitions**)**:**
Let $s, s' \in \mathcal{S}$ be two states, $\alpha \in \Sigma$ be a label and $\mathcal{G}$ be a set of guards. We say a transition $t \in \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ is *enabled* if and only if $t = (s, \mathbf{tt}, \alpha, s')$, or graphically $s \xrightarrow{\mathbf{tt} \to \alpha} s'$.

■

Definition 4.3 (**Executable transitions**)**:**
Let $t \in \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ be a transition. We say that t can be *executable* if and only if t is enabled.

■

Definition 4.4 (**Disabling transitions**)**:**
Let $s, s' \in \mathcal{S}$ be two states, $\alpha \in \Sigma$ be a label and $\mathcal{G}$ be a set of guards. We say a transition $t \in \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ is *disabled* if and only if $t = (s, \mathbf{ff}, \alpha, s')$, or graphically $s \xrightarrow{\mathbf{ff} \to \alpha} s'$.

■

In this way a transition t can be always considered but if t is disabled then it is never executed. From our pointview, this is equivalent to say that the feature α does not exist, note that in the MTS this also equals to say that α is forbidden. On the other hand, a transition enabled can be always executed and therefore it is equivalent to the typical transition of a LTS, namely a transition without guards.

This last observation is useful to understand the difference between transition with guards and without guards: both types of transitions hold the property "if an outgoing transition of s is executable then it is in $Trans(s)$". On the other hand, these two types of transitions differ in the contrary property "if a transition t is in $Trans(s)$ then it is an executable outgoing transitions". In effect, in the transition without guards we can guarantee that a transition is an outgoing transition if and only if it is executable, whereas in the transition with guards we cannot guarantee the same thing, because depending on the value of the guard itself, we can determine if the outgoing transition is executable or not.

Of course it is possible to enrich the Guard Set $\mathcal{G}$, but for now we only consider the set $\{\mathbf{tt}, \mathbf{ff}\}$

Definition 4.5 (**Equivalence**):
Let $t = (s_t, g_t, \alpha_t, s'_t)$, $u = (s_u, g_u, \alpha_u, s'_u)$ be two transitions. We can say t is *equivalent* to u if following conditions hold:

- $s_t = s_u$
- $\alpha_t = \alpha_u$
- $s'_t = s'_u$

We can say t is *equivalently enabled* in respect to u if and only if $g_t = g_u$.

■

Trivially, if t is equivalent to u then t and u are the same transition up to the guard, if t is equivalently enabled to u then t is enabled if and only if u is enabled, finally if t and u are equivalent and equivalently enabled then t and u are the same transition.

Now suppose two equivalent transitions $t = (s, \mathbf{tt}, \alpha, s')$ and $u = (s, \mathbf{ff}, \alpha, s')$ exist, where t represents the presence of the feature α and u the absence, trivially we know that concepts of presence and absence are exclusive, namely only one concept is possible in each case. In effect a feature can be present or absent but it cannot be present and absent at the same time. Hence, for our model, we have a further condition of consistency: for any t and u, such that they are equivalent, must exist a constraint $c =< \{t, u\}, [1, 1] >$, namely exactly one of t and u must always exist.

Definition 4.6 (**Constrained Modal Transition System with guarded transitions**):
A *Constrained Modal Transition System with guarded transitions* is a tuple $(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C})$ where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\mathcal{G}$ is a finite set of guards
- $\longrightarrow \subseteq \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathfrak{C} : \mathcal{S} \longrightarrow \mathcal{P}(Constraints(\mathfrak{Trans}(\mathcal{G})))$ is a function which, taken a state s as input, returns a set of possible constraints where constraints are defined over outgoing transitions with guards of s.

Moreover it holds that:

1. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ c$ is a correct constraint.
2. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \emptyset$.

3. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \subseteq Trans(s)$.

4. $\forall s \in \mathcal{S}.\ \forall c, c_1 \in \mathfrak{C}(s).\ Choice(c) \neq Choice(c_1)$.

5. $\forall s \in \mathcal{S}.\ \forall t, u \in Trans(s)$ if t and u are equivalent then $< \{t, u\}, [1, 1] > \in \mathfrak{C}(s)$.

We denote the set of all possible CMTS($\mathcal{G}$) by $\mathbb{CMTS}(\mathcal{G})$.

■

Note that changes introduced by the extension compared to the CMTS definition are minimal: we introduce only guards in transitions and a further consistency requirement. In addition, the syntactic modal refinement and the semantic modal refinement are unchanged, except the definition of refinement between two constraints:

Definition 4.7 (**Refinement between two constraints**)**:**
Let $c =< CS_c, [min_c, max_c] >, c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >$ be two constraints. We say c is a refinement of c_1 regarding a relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, denoted by $c \trianglelefteq_{\mathcal{R}} c_1$ if and only if:

- $\forall (g, \alpha, s') \in CS_c.\ \exists (g, \alpha, t') \in CS_{c_1}.\ (s', t') \in \mathcal{R}$
- c and c_1 are correct constraints
- $min_{c_1} \leq min_c$
- $max_c \leq max_{c_1}$

■

In addition, the introduction of the guard entails a problem with the definition of *Label*, seeing that the function *Label* considers only the label related to a transition, we can have a wrong step refinement such as in Figure 4.1. Note that for our syntactic refinement relation, the step in Figure 4.1 is correct, but it is obvious that CMTSs M and N have two different semantics. In effect, in M the absence of the feature a must be considered, whereas in N the presence of α must be taken into account. It is clear that now a transition is univocally determined by its label and its guard.

Definition 4.8:
Let $M = (\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C}, s_0)$ be a $CMTS(\mathcal{G})$ and s be a state. We denote the set of all labels related to outgoing transitions by $Label(s) = \{(g, \alpha) \in \mathcal{G} \times \Sigma \mid \exists s' \in \mathcal{S}.\ (s, g, \alpha, s') \in \longrightarrow\}$.

■

Again, we can extend this definition to constraints.

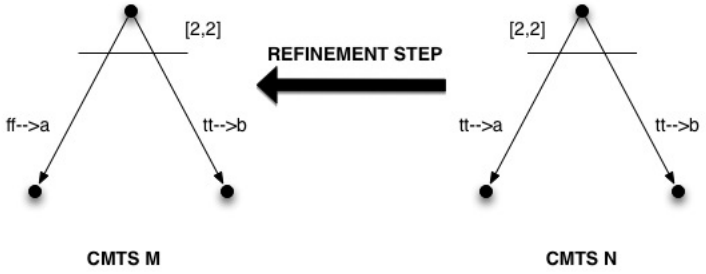


Figure 4.1: Syntactic refinement fails in CMTS($\mathcal{G}$)

Definition 4.9:
Let $M = (\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C}, s_0)$ be a $CMTS(\mathcal{G})$ and $c \in \mathfrak{C}(s)$ be a constraint of a some state s. We denote the set of all labels related to choice set of c by $Label(c) = \{(g, \alpha) \in \mathcal{G} \times \Sigma \mid \exists s' \in \mathcal{S}.\ (g, \alpha, s') \in Choice(c)\}$.

■

In addition, note that an element $t \in Choice(c)$, where c is a constraint, is an element of $\mathcal{G} \times \Sigma \times \mathcal{S}$. Finally, we can observe that the action-determinism concept, defined for CMTS, must be changed for CMTS($\mathcal{G}$):

Definition 4.10 (**Action-Deterministic CMTS**($\mathcal{G}$))**:**
An *Action-Deterministic CMTS*($\mathcal{G}$) is a tuple $(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C}, s_0)$ where:

- $(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C})$ is a CMTS($\mathcal{G}$)
- s_0 is the unique initial state

Moreover it holds another property :

$$\forall s \in \mathcal{S}, g, g' \in \mathcal{G}, \alpha \in \Sigma.\ (s, g, \alpha, s') \in \longrightarrow \wedge\ (s, g', \alpha, s'') \in \longrightarrow \Longrightarrow s' = s''$$

■

In this case the determinism of a CMTS($\mathcal{G}$) depends on the label of a transition and, of course, if we have two transitions with the same guard and label, the requirement of having the same target state is obvious, seeing that this is the requirement of the typical action-determinism of a CMTS, extended to guarded transitions. The special case is when we have two transitions with the same label and different guards, in this case $(\mathbf{tt}, \alpha)$ describes the presence of α, whereas $(\mathbf{ff}, \alpha)$ describes the absence of α, therefore the two guarded transitions must describe the same transition, but one describes only the presence, whereas the other one describes only the absence. From another pointview, the simple transition of a CMTS can stand in two different situations: it can be present or not, by means of the guard we describe in a direct way these situations, distinguishing them by way of two different types of transitions.

Unfortunately, the introduction of guards in the formalism entails some problems. In the previous chapter we said:

1. the semantics of a CMTS is a set of LTSs which can be refined by a CMTS
2. the semantics of a CMTS is defined by means of some refinement relation, which correlates together two CMTSs
3. a CMTS No-Choice is strictly correlates to a LTS

We observe that transitions of a LTS are typical transitions without guards, whereas transitions of CMTS($\mathcal{G}$) can be with or without guards. By means of the refinement, some transitions of a CMTS($\mathcal{G}$) can be lost and other ones can be maintained and therefore a CMTS($\mathcal{G}$) No-Choice derivable by a CMTS($\mathcal{G}$) can have some transitions with guard **ff** and other ones with guard **tt**. It becomes fundamental to find out a way to transform a CMTS($\mathcal{G}$) No-Choice in a LTS, namely a transformation from a system with guarded transitions to a system without guarded transitions, maintaining the semantics. First of all, we define the concept of LTS with guarded transitions.

Definition 4.11 (**Labelled Transition System with guarded transitions**)**:** A *Labelled Transition System with guarded transitions* is a tuple $(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow)$ where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\mathcal{G}$ is a finite set of guards
- $\longrightarrow \subseteq \mathcal{S} \times \mathcal{G} \times \Sigma \times \mathcal{S}$ is a transition relation

We denote this extension by $\mathbb{LTS}(\mathcal{G})$.

■

It is simple to understand that if it exists a relation between a CMTS No-Choice and a LTS, then the same relation must exist between a CMTS($\mathcal{G}$) No-Choice and a LTS($\mathcal{G}$), in effect a CMTS($\mathcal{G}$) No-Choice is equal to a CMTS No-Choice, except for the type of transition relations and the same holds for a LTS and a LTS($\mathcal{G}$).

Therefore we must find out a relation between LTS($\mathcal{G}$) and LTS. In a LTS($\mathcal{G}$) we have two types of transitions:

1. the ones with guard equals to **tt**, which describe executable transitions
2. the ones with guard equals to **ff**, which describe non-executable transitions

Instead, in LTS we know that every transition is executable, hence we can deduce the following property:

Definition 4.12:
A LTS is a LTS($\mathcal{G}$) $M = (\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow)$ where:

$$\forall s \in \mathcal{S}.\ \forall (g, \alpha, s') \in Trans(s) \Rightarrow g = \mathbf{tt}$$

■

In conclusion a LTS is a restriction of a LTS($\mathcal{G}$), where each transition with guard **ff** is deleted.

On the other hand, we do not often handle a LTS but we reason about solutions of a set of constraints related to a state s. We have already explained the relation between these solutions and a LTS, so we try to see the problem derived by the introduction of guards in these sets and how to solve it.

Definition 4.13:
Let $T \subseteq \mathfrak{Trans}(\mathcal{G})$ be a set of outgoing transitions and $H \subseteq \mathcal{G}$ be a set of possible guards. Then we say T is **restricted** by the set H, denoted by $T_{|H}$, to represent the set of all outgoing transitions of T which have a guard in H.

Formally, $T_{|H} = \{t = (g, \alpha, s') \mid (g, \alpha, s') \in T \ \wedge g \in H\}$.

■.

Trivially, $\forall H \subseteq \mathcal{G}.\ T_{|H} \subseteq T$ and $T_{|\emptyset} = T$. Moreover, this definition is easily extendible to a set of sets of transitions.

Definition 4.14:
Let $T, S \subseteq \mathfrak{Trans}(\mathcal{G})$ be two sets of outgoing transitions and $H \subseteq \mathcal{G}$ be a set of possible guards. Then we say T is *equivalent* to S regarding H if and only if $T_{|H} = S_{|H}$.

■

Seeing that, from our pointview, these sets of transitions describe possible outgoing transitions of a state s of a LTS($\mathcal{G}$) and we are interested in only executable transitions, namely transitions with guard equals to **tt**, we must take into account sets of transitions which are equivalent regarding the set $\{\mathbf{tt}\}$.

Definition 4.15:
Let $T, S \subseteq \mathfrak{Trans}(\mathcal{G})$ be two sets of outgoing transitions. Then we say T is *executable-equivalent* to S if and only if $T_{|\{\mathbf{tt}\}} = S_{|\{\mathbf{tt}\}}$

■

Definition 4.16:
Let $T \subseteq \mathfrak{Trans}(\mathcal{G})$ be a set of outgoing transitions and $H \subseteq \mathcal{G}$ a set of guards. Then we define the function $EquivalenceSet : \mathcal{P}(\mathfrak{Trans}(\mathcal{G})) \times \mathcal{G} \longrightarrow \mathcal{P}(\mathcal{P}(\mathfrak{Trans}(\mathcal{G})))$ such that, taken T and H, returns all sets of transitions equivalent to T regarding H.

Formally, $EquivalenceSet(T, H) = \{S \subseteq \mathfrak{Trans}(\mathcal{G}) \mid S_{|H} = T_{|H}\}$.

■

Definition 4.17:
Let $T \subseteq \mathfrak{Trans}(\mathcal{G})$ be a set of outgoing transitions and $H \subseteq \mathcal{G}$ a set of guards. Then we define the function $Ker : \mathcal{P}(\mathfrak{Trans}(\mathcal{G})) \times \mathcal{G} \longrightarrow \mathcal{P}(\mathfrak{Trans}(\mathcal{G}))$ such that, taken T and H, returns a set of transitions equivalent to T regarding H, but minimal with respect to the number of transitions.

Formally, $Ker(T, H) = J$ where:

- $J \in EquivalentSet(T, H)$, namely it is equivalent to T regarding H
- $\forall I \in Ker(T, H).\ |J| \leq |I|$

■

Of course we are interested in *EquivalenceSet* and *Ker* functions related to the set $\{\mathbf{tt}\}$, namely these functions must be based on executable-equivalence.

Definition 4.18:
Let $T \subseteq \mathfrak{Trans}(\mathcal{G})$ be a set of outgoing transitions. Then we define the function $ExecutableKer : \mathcal{P}(\mathfrak{Trans}(\mathcal{G})) \longrightarrow \mathcal{P}(\mathfrak{Trans}(\mathcal{G}))$ such that, taken T, returns a set of transitions executable-equivalent to T, but minimal with respect to the number of transitions.

Formally, $ExecutableKer(T) = Ker(T, \{\mathbf{tt}\})$.

■

Trivially, the set with the minimal number of transitions is the set with all and only transitions with guard equals to $\mathbf{tt}$.

Definition 4.19:
Let $T \subseteq \mathfrak{Trans}(\mathcal{G})$ be a set of outgoing transitions. Then:

$$ExecutableKer(T) = \{(\mathbf{tt}, \alpha, s') \mid (\mathbf{tt}, \alpha, s') \in T\}$$

■

Again, this function can be extended to a set of sets of transitions.

Theorem 4.1. *Let $T, S \subseteq \mathfrak{Trans}(\mathcal{G})$ be two sets of outgoing transitions. Then $\forall H \subseteq \mathcal{G}.\ T \notin EquivalenceSet(S, H) \Leftrightarrow S \notin EquivalenceSet(T, H)$*

Proof.
For definition of *EquivalenceSet*, taken a $H \subseteq \mathcal{G}$:

- $T \notin EquivalentSet(S, H) \Leftrightarrow T_{|H} \neq S_{|H}$
- $S \notin EquivalentSet(T, H) \Leftrightarrow S_{|H} \neq T_{|H}$

The theorem holds trivially. □

In this context, we can define two types of semantics related to states or constraints: one which describes the meaning of constraints, namely all possible solutions for constraints and another one which describes the possible behaviour derivable by the set of constraints, namely all possible sets of executable transitions. In the first case we use the notation $[\![.]\!]_{\mathcal{G}}$ to denote that elements computed by this semantics can have transitions with guards, whereas for the second we use the notation $[\![.]\!]$ to denote that elements computed have only executable transitions.

Definition 4.20:
Let $S \subseteq Constraints(\mathfrak{Trans}(\mathcal{G}))$ be a set of constraints defined over outgoing labelled transition with guards. We denote by $[\![.]\!]_{\mathcal{G}}$ the set of all sets of outgoing transitions which satisfy constraints in S.

Formally, $[\![S]\!]_{\mathcal{G}} = \{I \subseteq \mathfrak{Trans}(\mathcal{G}) \mid \forall c \in S.\ I \models c\}$

■

Definition 4.21:
Let $S \subseteq Constraints(\mathfrak{Trans}(\mathcal{G}))$ be a set of constraints defined over outgoing labelled transition with guard. We denote by $[\![.]\!]$ the set of all sets of outgoing executable transitions which satisfy constraints in S.

Formally, $[\![S]\!] = \{ExecutableKer(I) \mid I \in [\![S]\!]_{\mathcal{G}}\}$

■

Note that this technique, which deletes transitions with guards equal to **ff**, is surely correct for CMTS($\mathcal{G}$) No-Choice, whereas it can be wrong for a generic CMTS($\mathcal{G}$). The reason of the correctness for CMTS($\mathcal{G}$) No-Choice is simple: every state s of a CMTS($\mathcal{G}$) No-Choice has only one possible solution, namely the one which takes into account all outgoing transitions of s, therefore if we delete some outgoing transitions then we do not reduce the number of possible solution for constraints and we do not change the semantics related to a solution, at most we restrict the semantics seeing that some transitions are deleted.

Now suppose to have a generic CMTS($\mathcal{G}$) and, taken a state s, to delete all outgoing transitions with guard equals to **ff**. Trivially, we must also change all constraints of s because the deleted transitions might influence some constraints. Let D be the set of transitions to be deleted and $c =< CS, [min_c, max_c] >$ be a constraint of s. Then we must redefine c in the following way:

- $CS = CS \setminus D$, namely in CS must remain all transitions which are not eliminated
- $min_c = max\{0, min_c - |CS \cap D|\}$, namely the new min_c is derived by considering the worst case: when we have a solution with min_c transitions and the maximum number of transitions in $CS \cap D$ is taken. Of course this maximum number is $|CS \cap D|$. In order to avoid to derive a $min_c < 0$, we introduce the operator max.

- $max_c = min\{max_c, |CS \setminus D|\}$, namely the new max_c is derived by considering the best case: when we have a solution with max_c transitions and we take the maximum possible number of transitions in $CS \setminus D$ Of course this maximum number is $|CS \setminus D|$. Of course if we have more transitions in $CS \setminus D$ in respect to the needed ones to reach the value max_c, then we take max_c. On the other hand if we have not enough transitions to reach max_c then we take more possible transitions, namely all transitions in $|CS \setminus D|$

Now we consider the CMTS($\mathcal{G}$) in Figure 4.2, note that in this figure we can also observe some its LTS($\mathcal{G}$). Trivially, LTSs which can be derived are LTS with a transition labelled with a or with b. Now suppose to delete all transitions with guard

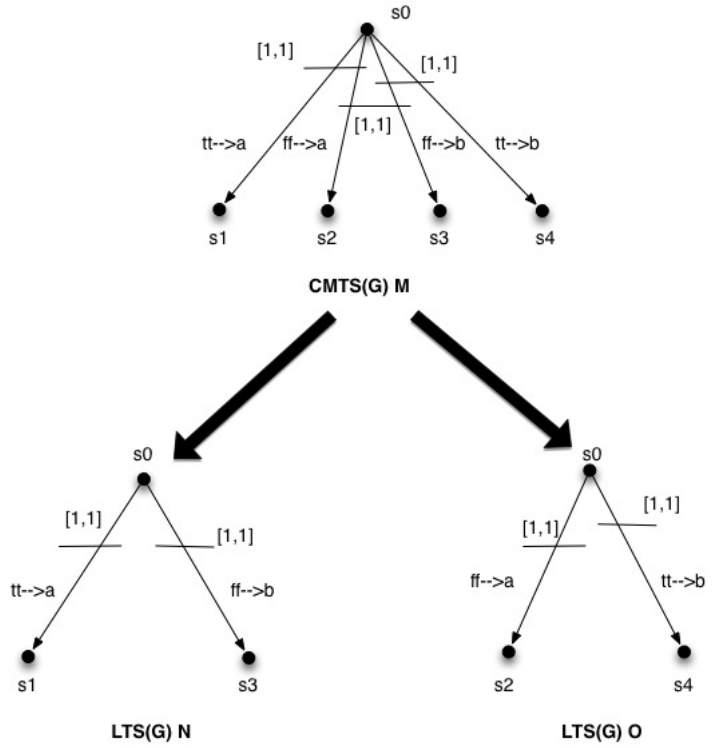


Figure 4.2: An example of CMTS($\mathcal{G}$) and its LTS($\mathcal{G}$)

equals to **ff** in M, then the derived CMTS($\mathcal{G}$) is in Figure 4.3. As we can see, in this case, we have some additional LTS($\mathcal{G}$). Moreover each LTS($\mathcal{G}$) has only transitions with guard equals to **tt**, so they are exactly equivalent to LTSs. Therefore, we can deduce that, deleting transitions with guard **ff**, we can change the semantics and this is obviously wrong. The reason of this mistake is that transitions with guard **ff** and constraints related to them might describe some useful information. Of course if we delete these transitions we also lose this information and in this way the semantics is changed.

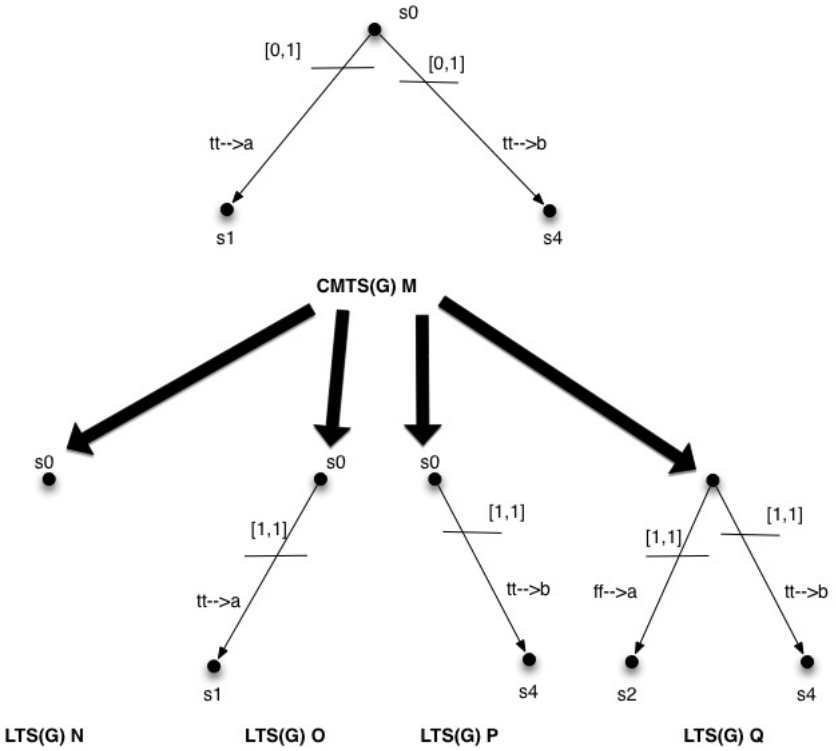


Figure 4.3: The same CMTS($\mathcal{G}$) of Figure 4.2 without disabled transitions and its LTS($\mathcal{G}$)

Now we try to extend the Guard Set with boolean parameters, namely let $\mathcal{Q}$ be a set of parameters we consider our Guard Set $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\} \cup \mathcal{Q}$. To distinguish CMTSs with simple guards and with parametric guards we denote by CMTS($\mathcal{G}$), all CMTSs with Guard Set $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\}$ and CMTS($\mathcal{G}_\mathcal{Q}$), all CMTSs with Guard Set $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\} \cup \mathcal{Q}$, where $\mathcal{Q}$ is a set of parameters. First of all, we try to understand the meaning of a transition with a parametric guard, if the parameter is $\mathbf{tt}$ then the transition is enabled, otherwise is disabled. In this case the presence of a feature depends on some parameters which are the guards of transitions related to the feature.

This change is simple to do from a pointview of the formalism but it hides some troubles.

- in CMTS($\mathcal{G}$) we have a consistency requirement connected to two equivalent transitions, that is two similar transitions with different guards. We know that this requirement is necessary because these two transitions model the presence and the absence of a feature, so it is impossible that these two transitions are simultaneously present. In the parametric context, this situation is slightly more difficult. For example, we suppose to have a state s with the following transitions:

- $(s, \mathbf{tt}, \alpha, s_1)$
- (s, p, α, s_2)
- (s, q, α, s_3)

where $s_1 \neq s_2,\ s_2 \neq s_3,\ s_1 \neq s_3$ and $p, q \in \mathcal{Q}$.

Trivially, if $p = q = \mathbf{tt}$ then all transitions describe an executable transition for the feature α. On the other hand, if $p = \mathbf{ff}$ or $q = \mathbf{ff}$ then we can have two transitions which describe simultaneously the presence and the absence of the feature α, but in this case we do not know a priori which are exactly these transitions.

To solve this problem, we can define a constraint which have the set of all equivalent transitions as choice set and $[1, 1]$ as cardinality

- suppose to have a CMTS($\mathcal{G}_\mathcal{Q}$) M. Then, at each step, we can take a parameter $p \in \mathcal{Q}$ and decide the value of p, namely $p = \mathbf{ff}$ or $p = \mathbf{tt}$. From M, we can derive a CMTS($\mathcal{G}_{\mathcal{Q}_1}$) M', where $\mathcal{Q}_1 = \mathcal{Q} \setminus p$. Now we suppose to iterate this procedure, for every element in $\mathcal{Q}$, until we have a CMTS($\mathcal{G}_{\mathcal{Q}_\mathcal{R}}$) N, where $\mathcal{Q}_\mathcal{R} = \emptyset$. Note that a CMTS($\mathcal{G}_\emptyset$) is trivially a CMTS($\mathcal{G}$). In addition we would like to have only deterministic CMTS($\mathcal{G}_\mathcal{Q}$). Unfortunately, it exists case where the initial CMTS($\mathcal{G}_\mathcal{Q}$) M is deterministic and the CMTS($\mathcal{G}$) N derived by M is not deterministic. For example in Figure 4.4 the CMTS($\mathcal{G}_\mathcal{Q}$) M is deterministic but the derived CMTS($\mathcal{G}_\mathcal{Q}$) O is not deterministic, in effect in O we have two transitions with the same label and guard but target states different.

  To guarantee the determinism, we must transform in some way the set of transitions with the same guard and label in only one transition such that the semantics is unchanged. A possible idea is, taken a set $T = \{(s, g, \alpha, s_i\}$ of transitions with the same guard and label, to define a new transition $t = (s, g, \alpha, s_{fusion})$ where $s_{fusion} = \{s_i\}$ is the state which describes the union of target states of transitions in T. Unfortunately, as we can see in Figure 4.5 this is problematic as this fusion changes the semantics. For example the semantics of O in Figure 4.5 requires the feature a with exactly one features between b and c, whereas the semantics of P requires the feature a or features b and c together.

In conclusion the introduction of parameters on the guard of transitions can create several problems.

For convenience we do not consider anymore CMTS($\mathcal{G}$) with parametric guards, but only with simple guards.

In the following sections we call these models CMTS($\mathcal{G}_\mathcal{T}$) to highlight that the guards are correlated to transitions.

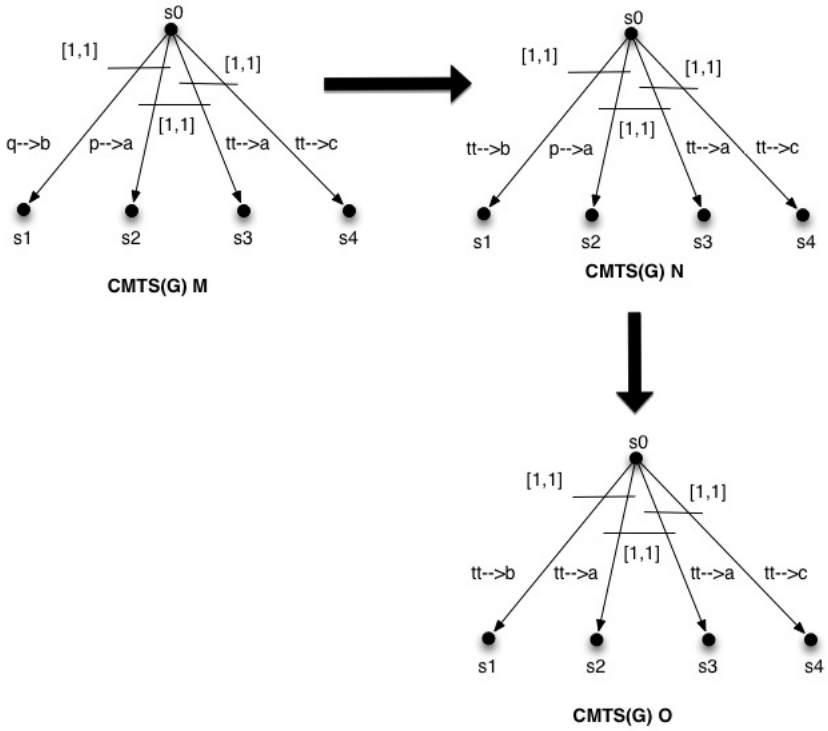


Figure 4.4: An example of transformation of a CMTS($\mathcal{G}_{\mathcal{Q}}$)

Example 4.1. *Now we take again the requirement $\mathfrak{R}$ which has the following semantics: $\{\{\emptyset\}, \{(\alpha, s_1), (\beta, s_3)\}\}$. A possible CMTS($\mathcal{G}_{\mathcal{T}}$) with the same semantics is the one in Figure 4.6. In the figure we also describe all LTS($\mathcal{G}$) which we can derive.*

Of course, from them we can derive some LTSs and, trivially, these LTSs are $\{\{\emptyset\}, \{(\alpha, s_1), (\beta, s_3)\}\}$, namely the semantics of $\mathfrak{R}$.

4.2 CMTS($\mathcal{G}^{\mathcal{Q}}$)

In this context we try to make a different extension: instead of adding guards to transitions, we add guards to constraints. Again, initially we suppose that our set of guards is $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\}$. Obviously, the type of a constraints must be changed.

Definition 4.22 (**Guarded Constraints**)**:**
Let $\mathcal{E}$ be a set of elements and $\mathcal{G}$ be a set of guards. Then a *guarded constraint* c is a tuple $< g, CS, [min, max] >$ where:

- $g \in \mathcal{G}$ is a guard
- $CS \subseteq \mathcal{E}$ is a choice set, namely a set of elements which can be chosen
- $[min, max] \in \mathcal{N} \times \mathcal{N}$ is an interval where min describes the minimum number of required elements of CS and max represents the maximum number of

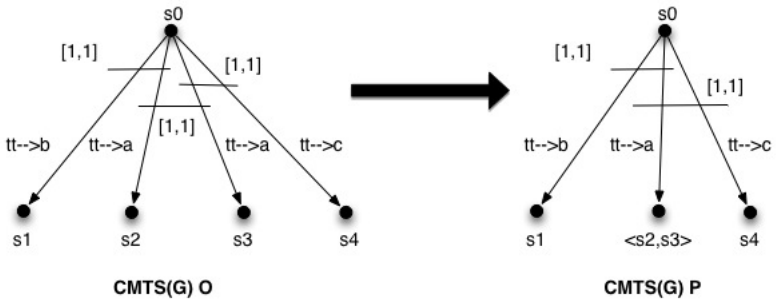


Figure 4.5: An example of transformation from a non-deterministic CMTS($\mathcal{G}_{\mathcal{Q}}$) to a deterministic one

required elements of CS

Graphically, we can denote c as $g \Longrightarrow < CS, [min, max] >$. Moreover, the type of a guarded constraint is $\mathcal{G} \times \mathcal{P}(\mathcal{E}) \times \mathcal{N} \times \mathcal{N}$.

We denote the set of all possible guarded constraints related to the set of guards $\mathcal{G}$ by $Constraints(\mathcal{G}, \mathcal{E})$.

■

In this context the guard of a constraint c describes if c must be considered in the computation of the semantics or not.

Definition 4.23 (**Constraint Satisfaction**)**:**
Let $\mathcal{E}$ be a set of elements, $\mathcal{G}$ be a set of guards, $c =< g, CS, [min, max] >$ be a guarded constraint and $I \subseteq \mathcal{E}$ a possible set of elements of $\mathcal{E}$. Then we define a *satisfaction relation* $\models \subseteq \mathcal{P}(\mathcal{E}) \times Constraints(\mathcal{G}, \mathcal{E})$ as follows:

$$I \models c \Leftrightarrow g \Rightarrow min \leq |I \cap CS| \leq max$$

■

We can note that if $g = \mathbf{ff}$ then any $I \subseteq \mathcal{E}$ satisfies the constraint c, otherwise if $g = \mathbf{tt}$ then $I \subseteq \mathcal{E}$ satisfies the constraint if the number of transition of CS in I is included between min and max.

Let $c =< g, CS, [min, max] >$ be a guarded constraint then we can define some utility functions:

- $Choice$: the function which, taken a constraint, return the choice set of the input constraint, namely $Choice(c) = CS$
- $Card$: the function which, taken a constraint, return the cardinality of the input constraint, namely $Card(c) = [min, max]$
- $Card_{min}$: the function which, taken a constraint, return the minimum of the cardinality of the input constraint, namely $Card_{min}(c) = min$

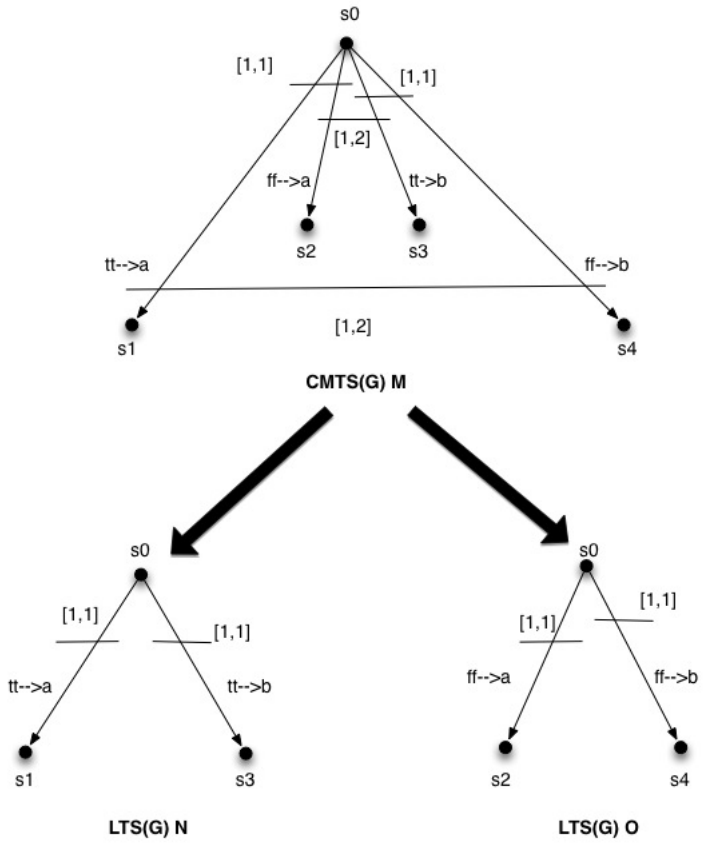


Figure 4.6: A solution for the problematic requirement $\mathcal{R}$

- $Card_{max}$: the function which, taken a constraint, return the maximum of the cardinality of the input constraint, namely $Card_{max}(c) = max$
- $Guard$: the function which, taken a constraint, return the guard of the input constraint, namely $Guard(c) = g$

The concept of semantics is unchanged:

Definition 4.24 (**Constraint Semantics**)**:**
Let $\mathcal{E}$ be a set of elements, $\mathcal{G}$ be a set of guards and $c =< g, CS, [min, max] >$ be a guarded constraint. Then we denote the semantics of c by $[\![c]\!]$ where:

$$[\![c]\!] = \{I \subseteq \mathcal{E} \mid I \models c\}$$

■

Trivially, we can extend these definitions to a set of guarded constraints. Now we can introduce the concept of constraint enabled and disabled.

Definition 4.25 (**Enabling and Disabling of Constraints**)**:**
Let $\mathcal{E}$ be a set of elements and $c \in S$ be a guarded constraint. Then we say that:

- c is *enabled* if and only if $Guard(c) = \mathbf{tt}$
- c is *disabled* if and only if $Guard(c) = \mathbf{ff}$

■

***Corollary* 4.1:** *Let $\mathcal{E}$ be a set of elements and $c =< g, CS, [min, max] >$ be a guarded constraint. Then we say that:*

- *if c is enabled then $[\![c]\!] = \{I \subseteq \mathcal{E} \mid min \leq |I \cap CS| \leq max\}$*
- *if c is disabled then $[\![c]\!] = \mathcal{P}(\mathcal{E})$*

■

Of course, all previous definitions about constraints are equivalent to guarded constraints.

Theorem 4.2. *Let $\mathcal{E}$ be a set of elements, $\mathcal{G}$ be a set of guards, S, U be two sets of guarded constraints such that $U \subseteq S \subseteq Constraints(\mathcal{G}, \mathcal{E})$ and U represents the set of all enabled constraints in S.*

Then it holds $[\![S]\!] = [\![U]\!]$

Proof.
First of all, we define the set $V = S \setminus U$, namely the set of disabled guarded constraints. Moreover we know that:

- $[\![V]\!] = \bigcap_{c \in V} [\![c]\!]$
- any disabled constraint c holds $[\![c]\!] = \mathcal{P}(\mathcal{E})$, namely the powerset of $\mathcal{E}$

Now we can deduce $[\![V]\!] = \bigcap_{c \in V} [\![v]\!] = \bigcap_{c \in V} \mathcal{P}(\mathcal{E}) = \mathcal{P}(\mathcal{E})$. Therefore $[\![S]\!] = [\![V]\!] \cap [\![U]\!] = \mathcal{P}(\mathcal{E}) \cap [\![U]\!] = [\![U]\!]$. □

This theorem is important because it shows us that all disabled constraints can be omitted.

Moreover, in this context, we release the property of the uniqueness of choice set, namely $\forall s \in \mathcal{S}.\ \forall c, c' \in \mathfrak{C}(s).\ Choice(c) \neq Choice(c')$.

Note that in a simple way, taken a set of constraints where all constraints have the same choice set, we can derive a single equivalent constraint, so it is possible to transform a generic CMTS without the uniqueness of choice set in a CMTS with the uniqueness of choice set.

Definition 4.26 (Witness of a set of constraints with the same choice set): Let $\mathcal{E}$ be a set of elements, $CS \subseteq \mathcal{E}$ be a possible choice set, $\mathcal{G}$ be a set of guards and $S \subseteq Constraints(\mathcal{G}, \mathcal{E})$ be a set of guarded constraints such that $\forall c \in S.\ Choice(c) = CS$.

Then we call *witness* of S a constraint $w =< CS_w, [min_w, max_w] >$ such that:

- $CS_w = CS$
- $min_w = \underset{c \in S}{max}\ Card_{min}(c)$
- $max_w = \underset{c \in S}{min}\ Card_{max}(c)$

■

Theorem 4.3. *Let $\mathcal{E}$ be a set of elements, $CS \subseteq \mathcal{E}$ be a possible choice set,$\mathcal{G}$ be a set of guards and $S \subseteq Constraints(\mathcal{G}, \mathcal{E})$ be a set of guarded constraints such that $\forall c \in S.\ Choice(c) = CS$ and w be the witness of S.*

Then $[\![S]\!] = [\![w]\!]$

Proof.
First of all, we note that, for the construction of w, it holds:

$$\forall c =< CS, [min_c, max_c] >\in S.\ min_c \leq min_w \wedge max_w \leq max_c$$

We have three different cases:

1. $[\![S]\!] = \emptyset$.

   In this case we have that $\forall I \subseteq \mathcal{E}.\ \exists c =< CS, [min_c, max_c] \in S.\ I \notin [\![c]\!] \Leftrightarrow |I \cap CS| < min_c \vee |I \cap CS| > max_c$. Then we can deduce $\forall I \subseteq \mathcal{E}.\ |I \cap CS| < min_w \vee |I \cap CS| > max_w$, therefore $[\![w]\!] = \emptyset$.

2. $[\![w]\!] = \emptyset$.

   In this case we have that $\forall I \subseteq \mathcal{E}.\ |I \cap CS| < min_w \vee |I \cap CS| > max_w$. Suppose that $|I \cap CS| < min_w$ then surely $\exists c \in S.\ min_c = min_w$, therefore $|I \cap CS| < min_c$ it is true, namely $\exists c \in S.\ I \notin [\![c]\!]$ holds.

   Suppose that $|I \cap CS| > max_w$ then surely $\exists c \in S.\ max_c = max_w$, therefore $|I \cap CS| > max_c$ it is true, namely $\exists c \in S.\ I \notin [\![c]\!]$ holds.

   Finally we can deduce $\forall I \subseteq \mathcal{E}.\ \exists c =< CS, [min_c, max_c] \in S.\ I \notin [\![c]\!]$ and hence $[\![S]\!] = \emptyset$.

3. $[\![S]\!] \neq \emptyset$ and $[\![w]\!] \neq \emptyset$.

   Now we must demonstrate that, taken a $J \subseteq \mathcal{E}$, then $J \in [\![S]\!] \Leftrightarrow J \in [\![w]\!]$.

   Suppose $J \in [\![S]\!]$ then $\forall c =< CS, [min_c, max_c] > .\ min_c \leq |J \cap CS| \leq max_c$. Seeing that, $\forall c =< CS, [min_c, max_c] > .\ min_c \leq |J \cap CS|$ then $\underset{c \in S}{max}\ min_c = min_w \leq |J \cap CS|$. It is possible to reason in the same way for max_c, therefore we can conclude that $J \in [\![w]\!]$.

   Suppose $J \in [\![w]\!]$ then $min_w \leq |J \cap CS| \leq max_w$. Seeing that, $\forall c =< CS, [min_c, max_c] > .\ min_c \leq min_w$ then $\forall c =< CS, [min_c, max_c] > .\ min_c \leq min_w \leq |J \cap CS|$. It is possible to reason in the same way for max_w, therefore we can conclude that $\forall c =< CS, [min_c, max_c] > .\ J \in [\![c]\!]$, so $J \in [\![S]\!]$.

□

Now we can define the CMTS with guarded constraints.

Definition 4.27 (**Constrained Modal Transition System with guarded constraints**)**:**
A *Constrained Modal Transition System with guarded constraints* is a tuple $(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C})$ where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\mathcal{G}$ is a finite set of guards
- $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathfrak{C} : \mathcal{S} \longrightarrow \mathcal{P}(Constraints(\mathcal{G}, \mathfrak{Trans}))$ is a function which taken a state s as input returns a set of possible guarded constraints where constraints are defined over outgoing transitions of s

Moreover it holds that:

1. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ c$ is a correct constraint.
2. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \emptyset$.
3. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \subseteq Trans(s)$.
4. $\mathcal{G} = \{\mathfrak{tt}, \mathfrak{ff}\}$

We denote the set of all possible CMTS by $\mathbb{CMTS}(\mathcal{G}^{\mathcal{C}})$.

■

Note that in this case the uniqueness property of choice set is not present.

Definition 4.28:
A CMTS is a CMTS$(\mathcal{G}^{\mathcal{C}})$ where the following conditions hold:

- $\forall s \in \mathcal{S}.\ \forall c, c_1 \in \mathfrak{C}(s).\ Choice(c) \neq Choice(c_1)$, namely the uniqueness property
- $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Guard(c) = \mathfrak{tt}$, namely every defined constraints must be enabled.

■

The next step is the introduction of parameters in the set of guards.

Definition 4.29 (Constrained Modal Transition System with parametric guarded constraints):
A *Constrained Modal Transition System with guarded constraints* is a tuple ($\mathcal{S}, \Sigma, \mathcal{Q}$, $\mathcal{G}, \longrightarrow, \mathfrak{C}$) where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\mathcal{Q}$ is a finite set of parameters
- $\mathcal{G}$ is a finite set of guards
- $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathfrak{C} : \mathcal{S} \longrightarrow \mathcal{P}(Constraints(\mathcal{G}, \mathfrak{Trans}))$ is a function which taken a state s as input returns a set of possible guarded constraints where constraints are defined over outgoing transitions of s

Moreover it holds that:

1. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ c$ is a correct constraint.
2. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \emptyset$.
3. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \subseteq Trans(s)$.
4. $\mathcal{G} = \{\mathbf{tt}, \mathbf{ff}\} \cup \mathcal{Q}$

We denote the set of all possible CMTS by $\mathbb{CMTS}(\mathcal{G}^{\mathcal{Q}})$.

■

Trivially, we can make some observation:

- if $\mathcal{Q} = \emptyset$ then a CMTS($\mathcal{G}^{\mathcal{Q}}$) is a CMTS($\mathcal{G}^{\mathcal{C}}$)
- in the CMTS($\mathcal{G}^{\mathcal{C}}$), if two constraints have the same choice set then it is possible to determine directly their witness, seeing that enabled constraints are known "a priori". By introducing of parameters, this direct computation of the witness for a set of constraints with the same choice set is not possible anymore because now the witness depends on the value of parameters.
- a CMTS with parametric guarded transitions can be modelled by a CMTS with parametric guarded constraints. The idea is simple: suppose to have a transition $t = (s, p, \alpha, s')$, therefore if $p = \mathbf{tt}$ then t is enabled, otherwise it is disabled. We can model the same behaviour in the CMTS with parametric guarded constraints: in effect, taken a transition $t_1 = (s, \alpha, s')$, we define a constraint $c =< q, t_1, [0, 0] >$ where $q = \neg p$. In this case if $p = \mathbf{ff}$ then c is enabled and, in each possible solution, t_1 is not present. On the other hand, if $p = \mathbf{tt}$ then c is disabled, then no other one constraint is related to t_1 and we can conclude that t_1 can be present.

Anyway, sometimes we would like that a constraint is disabled when a some parameter assumes value tt, other times we would like that a constraint is disabled when some parameter assumes value ff. Therefore we suppose that a guard can be derived by the following grammar:

$$\varphi :: \mathbf{tt} \mid p \mid \neg\varphi$$

where $p \in \mathcal{Q}$ is a parameter.

We can suppose that this grammar enriches further the power of the model, in effect we might think that the introduction of negation allows us to describe new types of requirements. Instead this supposition proves to be not true: we could simply define a new parameter q, for any negation, such that $q = \neg p$. In this way we can model the same things but using only positive parameters. Therefore the introduction of negation in the guard is only useful to describe in a simpler way the same model.

A concept to be updated due to the introduction of parameters is the refinement relation. Suppose that $\mathcal{Q}$ is the set of parameters and we define a partition of $\mathcal{Q} = \mathcal{Q}^+ \cup \mathcal{Q}^- \cup \mathcal{Q}^\perp$, where $\mathcal{Q}^+$ represents the set of all parameters with value **tt**, $\mathcal{Q}^-$ the set of all parameters with value **ff** and $\mathcal{Q}^\perp$ the set of all parameters without a specific value.

Trivially, taken $\mathcal{Q}$, we can consider a parameter $p \in \mathcal{Q}$, then we can assign a value to p: if $p = \mathbf{tt}$ then $\mathcal{Q}^+ = \mathcal{Q}^+ \cup \{p\}$, otherwise $\mathcal{Q}^- = \mathcal{Q}^- \cup \{p\}$. Finally we consider the set $\mathcal{Q}^\perp = \mathcal{Q} \setminus \{p\}$. Of course we can iterate this procedure until $\mathcal{Q}^\perp = \emptyset$, that is $\mathcal{Q} = \mathcal{Q}^+ \cup \mathcal{Q}^-$.

Definition 4.30 (**Assignment**)**:**
Let $\mathcal{Q}$ be a set of parameters. We call a set of parameters $A \subseteq \mathcal{Q}$ *assignment*.

■

The assignment describes all and only parameters with value **tt**. Note that if we know the set of parameters $\mathcal{Q}$ and the assignment A then we can deduce the set of all parameters with value **ff**. In effect if we see A as the set $\mathcal{Q}^+$, then it is simple to determine the set $\mathcal{Q}^-$.

In addition, we have already said that a CMTS($\mathcal{G}^{\mathcal{Q}}$) with $\mathcal{Q} = \emptyset$ is equivalent to a CMTS($\mathcal{G}^{\mathcal{C}}$) and this last one is strictly related to the classic CMTS.

The idea of refinement exploits this last observation: taken two CMTS($\mathcal{G}^{\mathcal{Q}}$) and an assignment A, then we can instantiate parameters by means of A, obtaining two CMTSs. If these two CMTSs are correlated by some refinement relation, then also two CMTS($\mathcal{G}^{\mathcal{Q}}$) are correlated by the same refinement relation.

Definition 4.31 (**Instantiation of a CMTS**($\mathcal{G}^{\mathcal{Q}}$))**:**
Let $\mathcal{Q}$ be a set of parameters and M be a CMTS($\mathcal{G}^{\mathcal{Q}}$). We define the *instantiation function* $\sigma : \mathbb{CMTS}(\mathcal{G}^{\mathcal{Q}}) \longrightarrow \mathbb{CMTS}$ that, taken a CMTS($\mathcal{G}^{\mathcal{Q}}$) M and an assignment $A \subseteq \mathcal{Q}$, return a CMTS $\sigma(M, A)$ which is derived by enabling and disabling all constraints through parametric guards.

■

Definition 4.32:
Let $\mathcal{Q}$ be a set of parameters, $M = (\mathcal{S}_M, \Sigma, \mathcal{Q}, \longrightarrow_M, \mathfrak{C}_M)$ be a CMTS($\mathcal{G}^{\mathcal{Q}}$) and $\sigma : \mathbb{CMTS}(\mathcal{G}^{\mathcal{Q}}) \longrightarrow \mathbb{CMTS}$ a instantiation function. Then the CMTS $N = \sigma(M, A)$ is defined by the tuple $(\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N)$ such that:

- $\mathcal{S}_M = \mathcal{S}_N$
- $\longrightarrow_M = \longrightarrow_N$
- $\forall s \in \mathcal{S}_N.\ \mathfrak{C}_N(s) = \{(\mathbf{tt}, \alpha, s') \mid (p, \alpha, s') \in \mathfrak{C}_M(s) \wedge p \in (A \cup \{\mathbf{tt}\})\} \cup \{(\mathbf{ff}, \alpha, s') \mid (p, \alpha, s') \in \mathfrak{C}_M(s) \wedge p \in (\mathcal{Q} \setminus A \cup \{\mathbf{ff}\})\}$

■

The first two conditions are obvious, only the last one is slightly more complicated. The third condition requires that taken a constraint c then:

- if in M, $Guard(c) = \{\mathbf{tt}\}$, namely c is enabled, or $Guard(c) \in A$, that is c becomes enabled through A, then c is enabled in N too;
- if in M, $Guard(c) = \{\mathbf{ff}\}$, that is c is disabled or $Guard(c) \notin A$, namely c becomes disabled through A, then in N c is disabled.

Definition 4.33 (Semantic modal refinement relation):
Let $M = (\mathcal{S}_M, \Sigma, \mathcal{Q}_M, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \mathcal{Q}_N, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTS($\mathcal{G}^{\mathcal{Q}}$). We say that $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is a *semantic modal refinement relation* if $\forall A_M \subseteq \mathcal{Q}_M.\ \exists A_N \subseteq \mathcal{Q}_N.\ \mathcal{R}$ is a semantic modal refinement relation for $\sigma(M, A_M)$ and $\sigma(N, A_N)$.

■

Definition 4.34 (Syntactic modal refinement relation):
Let $M = (\mathcal{S}_M, \Sigma, \mathcal{Q}_M, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \mathcal{Q}_N, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTS($\mathcal{G}^{\mathcal{Q}}$). We say that $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ is a *syntactic modal refinement relation* if $\forall A_M \subseteq \mathcal{Q}_M.\ \exists A_N \subseteq \mathcal{Q}_N.\ \mathcal{R}$ is a syntactic modal refinement relation for $\sigma(M, A_M)$ and $\sigma(N, A_N)$.

■

Of course, the guarded constraints can be further generalized: we know that a guarded constraint is a constraint which can be enabled or disabled by its guard. Moreover each constraint has exactly one single guard but clearly this request is very restrictive, in effect in some cases we might need to connect the enabling or disabling of a constraint to some different parameters. Therefore we should allow to have a set of parameters as guard of a constraint.

Definition 4.35 (Constraint with Multi-guard):
Let $\mathcal{E}$ be a set of elements, $\mathcal{G}$ be a set of guards, CS be a choice set and $[min, max]$ be a cardinality. Then a *constraint with multi-guard* c is a tuple $< g, CS, [min, max] >$ where:

- $g \subseteq \mathcal{G}$ is a set of guards, called *multi-guard*
- $CS \subseteq \mathcal{E}$ is a choice set, namely a set of elements which can be chosen
- $[min, max] \in \mathcal{N} \times \mathcal{N}$ is an interval where min describes the minimum number of required elements of CS and max represents the maximum number of required elements of CS

In this case the type of a guarded constraint is $\mathcal{P}(\mathcal{G}) \times \mathcal{P}(\mathcal{E}) \times \mathcal{N} \times \mathcal{N}$.

We denote the set of all possible constraints with multi-guard related to the set of guards $\mathcal{G}$ by $MultiConstraints(\mathcal{G}, \mathcal{E})$.

■

Let $\mathcal{G}$ be the set of guards, c be a constraint with a multi-guard g and $A \subseteq \mathcal{G}$ be an assignment. In this context we must understand which semantics can be related to a set of guards:

- **OR semantics**: a constraint c is enabled if and only if $\mathbf{tt} \in g$ or $\exists p \in g.\ p \in A$, namely c is enabled if at least one guard has value **tt** in an assignment A
- **AND semantics**: a constraint c is enabled if and only if $\forall p \in g.\ p \in A$ and $\mathbf{ff} \notin g$, namely c is enabled if all guards has value **tt** in an assignment A and **ff** is not present in g

Trivially, note that the OR semantics is already present implicitly as we can see in the following theorem.

Theorem 4.4. *Let $c =< g, CS, [min, max] >$ be a constraint with multi-guard with OR-semantics. We define a set S of constraint such that:*

$$S = \{< g_i, CS, [min, max] > \mid g_i \in g\}$$

Then $[\![c]\!] = [\![S]\!]$

Proof.
Let $A \subseteq \mathcal{G}$ be an assignment. We have two different cases:

1. $\mathbf{tt} \notin g$ and $g \cap A = \emptyset$ but then c is disabled and $[\![c]\!] = \mathcal{P}(\mathcal{E})$. The semantics of S is equal to $\bigcap_{c_s \in S} [\![c_s]\!]$. In addition $\forall c_s \in S.\ Guard(c_s) \notin A$ or $Guard(c_s) = \mathbf{ff}$, so any c_s is a disabled constraint. Therefore we can deduce $\forall c_s \in S.\ [\![c_s]\!] = \mathcal{P}(\mathcal{E})$, concluding that $[\![c]\!] = [\![S]\!] = \mathcal{P}(\mathcal{E})$.
2. $\mathbf{tt} \in g$ or $g \cap A \neq \emptyset$. Trivially, $[\![c]\!] = \{I \subseteq \mathcal{E} \mid min \leq |I \cap CS| \leq max\}$. For construction of S, we have a particular $c_s \in S$ such that $Guard(c_s) = \mathbf{tt}$ or $Guard(c_s) \cap A \neq \emptyset$. Now divide S in two subsets S^+, namely the set of enabled constraints and S^- the set of disabled constraints, but then $[\![S]\!] = [\![S^+]\!] \cap [\![S^-]\!] = [\![S^+]\!]$. Seeing that all constraints in S^+ are equal, for construction of S, we can deduce that $[\![S^+]\!] = \{I \subseteq \mathcal{E} \mid min \leq |I \cap CS| \leq max\}$. Again the two semantics are the same.

□

In our context, we know that it is possible to have different guarded constraints with the same choice set and the same cardinality, so the construction described in the theorem is possible. From this observation and the theorem we can derive that our formalism implicitly allows constraints with multi-guard and OR-semantics.

Unfortunately, constraints with multi-guard and AND-semantics cannot be modelled by simple guarded constraints. The reason is simple, taken a constraint c with multi-guard, then we derive a set S of constraints such that $\forall c_s \in S.\, |Guard(c_s)| = 1$, namely each constraint in S has a singleton guard. But from the previous theorem we know that this set S is equivalent to a constraint with multi-guard and OR-semantics, obviously OR-semantics is different from AND-semantics and it is impossible to relate them.

From this observation, we can deduce that the introduction of constraints with multi-guard is possible and this multi-guard is interpreted with an AND-semantics, the OR-semantics is not considered because it is implicitly present in the formalism.

Definition 4.36 (**Constraint with multi-guard Satisfaction**):
Let $\mathcal{E}$ be a set of elements, $\mathcal{G}$ be a set of guards, $c =< g, CS, [min, max] >$ be a guarded constraint such that $g \subseteq \mathcal{G}$ and $I \subseteq \mathcal{E}$ a possible set of elements of $\mathcal{E}$. Then we define a *satisfaction relation* $\models \subseteq \mathcal{P}(\mathcal{E}) \times MultiConstraints(\mathcal{G}, \mathcal{E})$ as follows:

$$I \models c \Leftrightarrow \bigvee_{g_i \in g} g_i \Rightarrow min \leq |I \cap CS| \leq max$$

■

Trivially all other definitions are unchanged.

4.3 CMTS($\mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}$)

In this last section we merge the models described in this chapter. The merge is possible because the model of first section introduces guards in the transitions, whereas the model of second section adds guards to constraints.

Definition 4.37 (**Constrained Modal Transition System with guarded transitions and constraints**):
A *Constrained Modal Transition System with guarded transitions and constraints* is a tuple $(\mathcal{S}, \Sigma, \mathcal{Q}, \mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}, \longrightarrow, \mathfrak{C})$ where:

- $\mathcal{S}$ is a finite set of states
- Σ is a finite set of actions
- $\mathcal{Q}$ is a finite set of parameters

- $\mathcal{G}_\mathcal{T}$, is a finite set of guards of transitions
- $\mathcal{G}^\mathcal{Q}$, is a finite set of guards of constraints
- $\longrightarrow \subseteq \mathcal{S} \times \mathcal{G}_\mathcal{T} \times \Sigma \times \mathcal{S}$ is a transition relation
- $\mathfrak{C} : \mathcal{S} \longrightarrow \mathcal{P}(MultiConstraints(\mathcal{G}^\mathcal{Q}, \mathfrak{Trans}(\mathcal{G}_\mathcal{T}))$ is a function which taken a state s as input returns a set of possible constraints with multi-guard where constraints are defined over outgoing guarded transitions of s

Moreover it holds that:

1. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ c$ is a correct constraint.
2. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \emptyset$.
3. $\forall s \in \mathcal{S}.\ \forall c \in \mathfrak{C}(s).\ Choice(c) \subseteq Trans(s)$.
4. $\mathcal{G}_\mathcal{T} = \{\mathbf{tt}, \mathbf{ff}\}$
5. $\mathcal{G}^\mathcal{Q} = \{\mathbf{tt}, \mathbf{ff}\} \cup \mathcal{Q}$
6. $\forall s \in \mathcal{S}.\ \forall t, u \in Trans(s)$ if t and u are equivalent then $< \{\mathbf{tt}\}, \{t, u\}, [1, 1] > \in \mathfrak{C}(s)$.

We denote the set of all possible CMTS$(\mathcal{G}_\mathcal{T}, \mathcal{G}_\mathcal{Q})$ by $\mathbb{CMTS}(\mathcal{G}_\mathcal{T}, \mathcal{G}_\mathcal{Q})$.

■

Note that this formalism has parameters only for guards of constraints, whereas parameters are lost for guards of transitions for the the same reasons saw in Section 4.1. Moreover constraints has multi-guards with AND-semantics. In addition the refinement of this models is the same of CMTS$(\mathcal{G}^\mathcal{Q})$, seen in Section 4.2, namely we must consider all possible assignments which can be derivable from $\mathcal{Q}$. Seeing that transitions are labelled transitions with guard like in CMTS$(\mathcal{G}_\mathcal{T})$ then all possible products derivable from models are LTS$(\mathcal{G}_\mathcal{T})$

Of course, in this case the definition of semantic modal refinement and syntactic modal refinement must consider that, taken an assignment A and a CMTS$(\mathcal{G}_\mathcal{T}, \mathcal{G}_\mathcal{Q})$ M, then $\sigma(M, A)$ returns a CMTS$(\mathcal{G}_\mathcal{T})$. The definitions changes in the obvious way.

We can add a further observation: if we change the type of transitions by means of adding guards then the type of target product must be changed. Instead if we change the type of constraints, no change is needed for the type of target products. In some way we can say that the adding of guards over constraint is implementation type-free, namely it is independent by the actual type of implementation. This result is very important because, taken a CMTS of any type, then we can extend it by adding guards over constraints and no other changes are needed.

Chapter 5

Hierarchy of Models Expressivity

In the Section 2.2, Chapter 3 and Chapter 4 we have introduced and described several models with some properties like refinement and in some case we sketched an expressivity relation between them

In literature, some works [12], [26] and [27] introduce a more or less detailed hierarchy of expressivity but no work handles all models together. Now in this chapter we try to solve this lack, adding in this hierarchy CMTS and its extensions too. Results reached in this chapter are very important because not only at the end we determine relations of expressivity between all models, namely we will see which models can describe what requirements, but in addition we describe how classical models like LTS, MTS and so on can be represented by means of a CMTS.

First of all, we must understand what we mean about expressivity of a formalism: in the previous chapter we introduced the concept of semantics of a model M, where the semantics is interpreted as the set of LTSs which can be derived by M through a refinement relation. This concept is useful to define the expressivity relation between two formalisms:

Definition 5.1:
Let $\mathcal{F}$ be a formalism, then we denote by $M_{\mathcal{F}}$ a model defined by means of the formalism $\mathcal{F}$.

■

Definition 5.2:
Let $\mathcal{F}$ and $\mathcal{F}_1$ be two different formalisms. Then $\mathcal{F}$ is **less expressive** than $\mathcal{F}_1$, denoted by $\mathcal{F} \rightsquigarrow \mathcal{F}_1$, if and only if it holds:

1. for each model $M_{\mathcal{F}}$ exists a semantically equivalent model $M_{\mathcal{F}_1}$
2. it exists a model $N_{\mathcal{F}_1}$ such that for any model $N_{\mathcal{F}}$, $[\![[\![N_{\mathcal{F}}]\!]]\!] \neq [\![[\![N_{\mathcal{F}_1}]\!]]\!]$

■

Note that each formalism has different characteristics useful to describe some requirements, but all formalisms have a common syntactic structure, namely all formalisms have states and labelled transitions.

Trivially, when we will compare two models $M_{\mathcal{F}}$ and $M_{\mathcal{F}_1}$ of two different formalism, we will assume that these properties hold:

1. for each state s of $M_{\mathcal{F}}$ exists a state t of $M_{\mathcal{F}_1}$ such that $Label(s) = Label(t)$
2. for each state t of $M_{\mathcal{F}_1}$ exists a state s of $M_{\mathcal{F}}$ such that $Label(s) = Label(t)$

Of course, if a model $M_{\mathcal{F}}$ has a state s such that for each state t of $M_{\mathcal{F}_1}$ $Label(s) \neq Label(t)$, then they are not semantically equivalent most likely, because from $M_{\mathcal{F}}$ we can derive a LTS which is not derivable from $M_{\mathcal{F}_1}$.

Definition 5.3:
Let $\mathcal{F}$ and $\mathcal{F}_1$ be two different formalisms. Then $\mathcal{F}$ is **equivalently expressive** to $\mathcal{F}_1$, denoted by $\mathcal{F} \leftrightsquigarrow \mathcal{F}_1$, if and only if it holds:

1. for each model $M_{\mathcal{F}}$ exists a semantically equivalent model $M_{\mathcal{F}_1}$
2. for each model $M_{\mathcal{F}_1}$ exists a semantically equivalent model $M_{\mathcal{F}}$

■

In addition, the understanding of how a LTS can be defined by means of a CMTS becomes a very useful and important result because in this way we can deduce as a product, typically described by a LTS, can be derived from a specification, represented by a CMTS. Finally, we can cover pending topics of Section 3.1.2 related to the thorough refinement and the semantics of a specific CMTS.

Note that all models are divided in two large family: **Modal Family** and **Obligation Family**. This classification has already described in Section 2.2, but we can remember that:

- in Modal Family we have many models: LTS,MTS, DMTS, 1MTS, GEMTS, EMTS
- in Obligation Family we have the most recent models: OTS and PMTS

Note that only for convenience we handle the LTS model in the Section related to the Modal Family. Finally, we assume that the action-determinism property holds in each formalism and in each section, which describes a particular formalism, we also define how this property can be formalized.

5.1 Hierarchy of the Modal Family

A first attempt to compare all models of this family can be found in [26], where all models are compared with the GEMTS formalism. In this section we will try to make a similar comparison but considering CMTS formalism as well. Note that

we will make a description of the relation between models and CMTSs in an increasing way, that is from less-expressive models to more-expressive models and we will also see the relation between the refinement concept of the particular considered model and CMTS. Last but not least observation is that the CMTS is an action-deterministic formalism, hence each formalism which is compared to it must be action-deterministic too. This property, as we will see, will become fundamental in some contexts because it allows us to derive some particular results over the expressivity, which are different regarding the ones known in literature.

Finally, taken two different models $M_{\mathcal{F}}$ and $M_{\mathcal{F}_1}$, described by two different formalisms $\mathcal{F}$ and $\mathcal{F}_1$, we must reason about the semantics of M and N. Of course, the computation of the semantics of a model also depends on the formalism considered. To highlight this aspect, we introduce some further definitions:

Definition 5.4:
We denote by $\mathbb{MODELS}$ the set of formalisms composed by: $\mathbb{LTS}$, $\mathbb{MTS}$, $\mathbb{DMTS}$, $\mathbb{1MTS}$, $\mathbb{GEMTS}$, $\mathbb{OTS}$, $\mathbb{PMTS}$ and $\mathbb{CMTS}$. Formally,

$$\mathbb{MODELS} = \{\mathbb{LTS}, \mathbb{MTS}, \mathbb{DMTS}, \mathbb{1MTS}, \mathbb{GEMTS}, \mathbb{OTS}, \mathbb{PMTS}, \mathbb{CMTS}\}$$

■

Definition 5.5:
Let $\mathcal{F} \in \mathbb{MODELS}$ be a specific formalism and M be a model defined by the formalism $\mathcal{F}$. Then we denote by $[\![M]\!]_{\mathcal{F}}$ the *semantics* of M where the semantics is defined as:

$$[\![M]\!]_{\mathcal{F}} = \{I \mid I \text{ is a LTS} \wedge I \trianglelefteq_{\mathcal{F}} M\}$$

where $\trianglelefteq_{\mathcal{F}}$ is the typical modal refinement relation for the formalism $\mathcal{F}$.

■

For example, $\trianglelefteq_{\mathbb{LTS}}$ is the bisimulation, $\trianglelefteq_{\mathbb{MTS}}$ is the modal refinement for MTS and so on.

5.1.1 LTS

The first model to be considered is surely the LTS and in this context the only expressible requirement is that "*each feature is allowed and necessary*". For example, in [36] a LTS is described as a MTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box})$ such that $\longrightarrow_{\Diamond} = \longrightarrow_{\Box}$.

First of all, we must define the concept of an action-deterministic LTS:

Definition 5.6 (Action-deterministic LTS):
A LTS $L = (\mathcal{S}, \Sigma, \longrightarrow, S_0)$ is an action-deterministic LTS if and only if:

- $\forall s \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow \wedge (s, \alpha, s'') \in \longrightarrow \Longrightarrow s' = s''$

- $|S_0| = 1$

■

This is the typical definition of determinism in the LTS world. Now, we want to understand if it is possible to describe a LTS by means of a CMTS: let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS , then we should find a way to guarantee that, for any state s, each outgoing transition of s is always considered for each possible product.

This is possible if and only if $\forall s \in \mathcal{S}.\ \forall t \in Trans(s).\ \exists c \in \mathfrak{C}(s)$ such that it holds:

$$t \in Choice(c) \wedge |Choice(c)| = Card_{min}(c) = Card_{max}(c)$$

Essentially, we require that each transition t must be handled by a constraint c and this constraint must guarantee that t is always present.

Note that this property is exactly the property being satisfied by a CMTS No-Choice, hence we can deduce that a LTS must be a CMTS No-Choice. Unfortunately, as we seen in Section 3.3, we can have some syntactically different but semantically equivalent CMTS No-Choice. This is a very complicated situation because we would like to have a unique particular description of a LTS by means of a CMTS which identifies univocally the LTS, whereas in this case a LTS is identified by a class of possible CMTS.

In order to solve this ambiguity we might use the "witness" of a class of CMTSs No-Choice (as we saw in Section 3.3), which identifies univocally the class of CMTSs.

Definition 5.7 (LTS):
A LTS $L = (\mathcal{S}, \Sigma, \longrightarrow, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ such that:

- M is a CMTS No-Choice
- M is a CMTS No-Choice Witness

■

An equivalent but more syntactic definition is the following:

Definition 5.8 (LTS):
A LTS $L = (\mathcal{S}, \Sigma, \longrightarrow, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ such that:

$$\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \{< \{(\alpha, s')\}, [1,1] > \ \mid (s, \alpha, s') \in \longrightarrow\}$$

■

In this second definition we requires that for any state s, the set of constraints of s is arranged by all constraints with

- a singleton choice set

- a cardinality equals to $[1, 1]$
- any possible outgoing transition of s is included in the choice set of a specific constraint of s

Trivially, the following theorem holds:

Theorem 5.1. *Let $L = (\mathcal{S}, \Sigma, \longrightarrow_L, s_0)$ be a LTS and $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ be a CMTS. If:*

- *M is a CMTS No-Choice*
- *$\longrightarrow_L = \longrightarrow_M$*

then $[\![L]\!]_{\text{LTS}} = [\![M]\!]_{\text{CMTS}}$

Proof.
We know that a LTS is equivalent to a CMTS No-Choice in Normal Form hence, taken L, we can say N is the CMTS No-Choice in Normal Form related to L. In addition N and M are equivalent, seeing that they have the same transition relation.

Finally, we know that if N and M are two CMTSs No-Choice and they are equivalent then they are semantically equivalent. □

Another interesting topic is to understand if the refinement relation for the CMTS becomes equivalent to the bisimulation when CMTSs describe LTSs or if the refinement relation and the bisimulation are two different relations in any case.

For our purposes, we use the syntactic refinement relation because it is the most difficult from a pointview of formalizations and, at the same time, it is less accurate than the semantic one, hence we consider the worst case.

Theorem 5.2. *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs such that they describe two different LTSs. If it exists a syntactic refinement relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ between M and N then R is a bisimulation.*

Proof.
For convenience, we recall the definition of bisimulation: $\mathcal{R}$ is called bisimulation if, whenever sRt:

1. if $s \xrightarrow{\alpha} s'$ then $\exists t' \in \mathcal{S}$ such that $t \xrightarrow{\alpha} t'$ and $s' \mathcal{R} t'$
2. if $t \xrightarrow{\alpha} t'$ then $\exists s' \in \mathcal{S}$ such that $s \xrightarrow{\alpha} s'$ and $s' \mathcal{R} t'$

Trivially, the first condition of bisimulation is equivalent to the first condition of the syntactic refinement relation. The problem is only in the second condition because in the syntactic refinement we handle constraints.

First of all, note that M and N have a special property:

$$\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \{< \{(\alpha, s')\}, [1, 1] > \ \mid (s, \alpha, s') \in \longrightarrow\}$$

Therefore, if $t \xrightarrow{\alpha} t'$ then, surely, $(t, \alpha, t') \in \longrightarrow$ and $c =< \{(\alpha, t')\}, [1,1] > \in \mathfrak{C}(t)$. Seeing that $\mathcal{R}$ is a syntactic refinement relation then either $Card_{min}(c) = 0 \vee \exists c_s \in \mathfrak{C}(s)$ such that holds some properties. Trivially, $Card_{min}(c) = 0$ is false, hence a constraint c_s exists and it holds the following properties:

1. $\forall(\alpha, s') \in Choice(c_s).\ \exists(\alpha, t') \in Choice(c_t).\ (s', t') \in \mathcal{R}$
2. c_s and c_t are correct constraints
3. $Card_{min}(c_t) \leq Card_{min}(c_s)$
4. $Card_{max}(c_s) \leq Card_{max}(c_t)$
5. $(Label(c_t) \setminus Label(c_s)) \cap Label(s) = \emptyset$

For the structure of a CMTS, when it describes a LTS, we know that conditions (2), (3), and (4) always hold. Since c_s also satisfies conditions (1) and (5) and each constraint has a singleton choice set, we deduce that $c_s =< \{(\alpha, s')\}, [1,1] >$. Therefore, if the state s has the constraint c_s then surely a transition (s, α, s') exists and for the condition (1) we also know that $(s', t') \in \mathcal{R}$

We can conclude that the condition (2) of the definition of the bisimulation holds and $\mathcal{R}$ is also a bisimulation.

□

Now we can solve the pending topic of Section 3.1.2, namely how to define the semantics of a CMTS.

Definition 5.9 (**Semantics of a CMTS**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ be a CMTS. Then the semantics of M is the set of LTSs satisfying the requirements described by M, namely LTSs which can be derived by M through the refinement relation.

We denote by $[\![[\![M]\!]]\!]^{Sem}$ the semantics derived by $\trianglelefteq^{Sem}$ and $[\![[\![M]\!]]\!]^{Syn}$ the semantics derived by $\trianglelefteq^{Syn}$.

Formally,

- $[\![[\![M]\!]]\!]^{Sem} = \{I \text{ is a LTS } \mid I \trianglelefteq^{Sem} M\}$.
- $[\![[\![M]\!]]\!]^{Syn} = \{I = NF(J) \text{ is a LTS } \mid J \text{ is a CMTS No-Choice} \wedge J \trianglelefteq^{Syn} M\}$.

where $I = NF(J)$ describes the transformation of a CMTS No-Choice J in its normal form.

■

Note that, in the semantic refinement relation, we abstract from structure of constraints and we only consider their semantics, in this way we can directly derive the LTS I. On the other hand, in the syntactic refinement relation we must handle constraints and their structure and in the end we reach a CMTS No-Choice, which

is equivalent to a LTS. For this reason we need to transform this derived CMTS No-Choice in the LTS related to it.

Now we can realize why a LTS is a solution of all constraints of a CMTS M. Take a state s, then $\mathfrak{C}(s)$ describes the set of constraints in s. Trivially, a solution I for $\mathfrak{C}(s)$ is derived by choosing, for each constraint $=< CS_c, [min_c, mx_c] >$ in $\mathfrak{C}(s)$, a value K included between min and max and a subset T of transition in CS_c such $|T| = K$. This is equivalent to reduce the constraint c in such a way that all transitions in $CS_c \backslash T$ are deleted and min is increased up to K and max is decreased up to K, the derived constraint is equal to $c' =< T, [K, K] >$. This is true for any $c \in \mathfrak{C}(s)$, seeing that I is a solution. Note that all these derived constraints hold the property of a CMTS No-Choice, moreover we know that a LTS is equivalent to a CMTS No-Choice, hence a LTS can be seen as a composition of solutions for each state of M.

Of course, in this context we can also define the semantics of extensions of CMTSs: first of all, we define a special type of semantics of a CMTS($\mathcal{G}_\mathcal{T}$), namely a semantics which considers all LTS($\mathcal{G}_\mathcal{T}$) derivable from the initial CMTS($\mathcal{G}_\mathcal{T}$) and we call it the **extended semantics**. In effect the CMTS($\mathcal{G}_\mathcal{T}$) formalism changes the type of the transition relation and this semantics keeps unchanged the type of transitions.

Definition 5.10 (**Extended Semantics**)**:**
Let $M(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_\mathcal{T}$). Then the *extended semantics* of M is the set of LTSs($\mathcal{G}_\mathcal{T}$) satisfying the requirements described by M, namely LTS($\mathcal{G}_\mathcal{T}$) which can be derived by M through the refinement relation.

We denote the semantics derived through $\trianglelefteq^{Sem}$ by $[\![[\![M]\!]]\!]^{Sem}_{\mathcal{G}_\mathcal{T}}$ and the semantics derived through $\trianglelefteq^{Syn}$ by $[\![[\![M]\!]]\!]^{Syn}_{\mathcal{G}_\mathcal{T}}$.

Formally,

- $[\![[\![M]\!]]\!]^{Sem}_{\mathcal{G}_\mathcal{T}} = \{I \text{ is a LTS } (\mathcal{G}_\mathcal{T}) \mid I \trianglelefteq^{Sem} M\}$
- $[\![[\![M]\!]]\!]^{Syn}_{\mathcal{G}_\mathcal{T}} = \{I = NF(J) \text{ is a LTS } (\mathcal{G}_\mathcal{T}) \mid J \text{ is a CMTS}(\mathcal{G}_\mathcal{T}) \text{ No-Choice } \wedge J \trianglelefteq^{Syn} M\}$.

where $I = NF(J)$ describes the transformation of a CMTS($\mathcal{G}_T$) No-Choice J in its normal form.

■

Unfortunately, when we must compare two formalisms, we use the semantics defined by means of LTSs, hence we must change the semantics of a CMTS($\mathcal{G}_\mathcal{T}$).

Definition 5.11 (**Semantics of a CMTS**($\mathcal{G}_T$))**:**
Let $M(\mathcal{S}, \Sigma, \mathcal{G}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_T$). Then the semantics of M is the set of LTSs satisfying the requirements described by M, namely LTSs which can be derived by M through the refinement relation.

We denote by $[\![[\![M]\!]]\!]^{Sem}$ the semantics derived by $\trianglelefteq^{Sem}$ and $[\![[\![M]\!]]\!]^{Syn}$ the semantics derived by $\trianglelefteq^{Syn}$.

Formally,

- $[\![[\![M]\!]]\!]^{Sem} = \{ExecutableKer(I) \mid I \in [\![[\![M]\!]]\!]^{Sem}_{\mathcal{G}_\mathcal{T}}\}$
- $[\![[\![M]\!]]\!]^{Syn} = \{ExecutableKer(I) \mid I \in [\![[\![M]\!]]\!]^{Syn}_{\mathcal{G}_\mathcal{T}}\}$.

where $ExecutableKer$ is the function to transform a LTS($\mathcal{G}_T$) in LTS.

■

Finally, when we introduce the parametric guarded constraints, the semantics becomes slightly more complicated, in effect now the semantics depends on the assignment of parameters too. Moreover, taken a CMTS($\mathcal{G}_T, \mathcal{G}^\mathcal{Q}$) M and an assignment $A \subseteq \mathcal{Q}$, we know that $\sigma(M, A)$ is a CMTS($\mathcal{G}_T$). Hence, we can deduce that the semantics of CMTS($\mathcal{G}_T, \mathcal{G}^\mathcal{Q}$) is the union of all possible semantics of CMTS($\mathcal{G}_T$) derived by means of an assignment.

Definition 5.12 (Semantics of a CMTS($\mathcal{G}_T, \mathcal{G}^\mathcal{Q}$)):
Let $M = (\mathcal{S}, \Sigma, \mathcal{Q}, \mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}, \longrightarrow, \mathfrak{C})$ be a CMTS($\mathcal{G}_T, \mathcal{G}^\mathcal{Q}$). Then the semantics of M is the set of LTSs satisfying the requirements described by M, namely LTSs which can be derived by M through the refinement relation.

We denote by $[\![[\![M]\!]]\!]^{Sem}$ the semantics derived by $\trianglelefteq^{Sem}$ and $[\![[\![M]\!]]\!]^{Syn}$ the semantics derived by $\trianglelefteq^{Syn}$.

Formally,

- $[\![[\![M]\!]]\!]^{Sem} = \bigcup_{A \subseteq \mathcal{Q}} [\![[\![\sigma(M, A)]\!]]\!]^{Sem}$
- $[\![[\![M]\!]]\!]^{Syn} = \bigcup_{A \subseteq \mathcal{Q}} [\![[\![\sigma(M, A)]\!]]\!]^{Syn}$

where σ is the assignment function defined for CMTS with parametric guarded constraint.

■

5.1.2 MTS

The next model to be considered is the Modal Transition System (MTS). In this case we have two types of transitions:

1. the necessary ones, namely transitions which must be always present in each possible product
2. the allowed ones, namely transitions which may be present in each possible product

Again, we must define the concept of an action-deterministic MTS:

Definition 5.13 (**Action-deterministic MTS**):
A MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box}, S_0)$ is an action-deterministic MTS if and only if:

- $\forall s \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow_{\Diamond} \wedge (s\alpha, s'') \in \longrightarrow_{\Diamond} \Rightarrow s' = s''$
- $|S_0| = 1$

■

This definition was introduced in [8].

Now, suppose we have a state s and a possible product P then we denote by $P(s) \subseteq Trans(s)$ the set of outgoing transitions of s in the possible product P. In addition suppose we have a transition t then we have two possibilities:

1. t is a necessary transition hence, for any possible product P, $t \in P(s)$ must be always true. Trivially, we must guarantee that $\forall P.\ |P(s) \cap t| = 1$
2. t is an allowed transition hence, for any possible product P, $t \in P(s) \vee t \notin P(s)$ must be always true. Trivially, we must guarantee that $\forall P.\ |P(s) \cap t| = 1 \vee |P(s) \cap t| = 0$

The first condition $\forall P.\ |P(s) \cap t| = 1$ can be seen as a constraint $c =< \{t\}, [1, 1] >$, namely a constraint which always requires the transition t. Instead, the second condition $\forall P.\ |P(s) \cap t| = 1 \vee |P(s) \cap t| = 0$ can be seen as $\forall P.\ 0 \leq |P(s) \cap t| \leq 1$ and this is equivalent to a constraint $c =< \{t\}, [0, 1] >$, namely a constraint which may require the transition t.

The last observation is that in a MTS the must transition relation is a subset of a may transition relation and, seeing that we can have only constraints univocally determined by its choice set then we cannot have two constraints such that $c =< \{t\}, [1, 1] >$ and $c_1 =< \{t\}, [0, 1] >$. In this case c is a more-restrictive constraint compared with c_1, hence, seeing that both c and c_1 must be satisfied, we can only consider the constraint c. Therefore we can deduce that for any must transition t (and therefore also may) we have a constraint $c =< \{t\}, [1, 1] >$, whereas for any may transition t but not must we have $c =< \{t\}, [0, 1] >$.

From these observation, we can derive a MTS described by means of a CMTS.

Definition 5.14 (**MTS**):
A MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_L}, \longrightarrow_{\Box_L}, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ such that:

1. $\longrightarrow_{\Diamond_L} = \longrightarrow_M$
2. $\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \mathfrak{C}_{may}(s) \cup \mathfrak{C}_{must}(s)$ where:
   - $\mathfrak{C}_{may}(s) = \{< \{(\alpha, s')\}, [0, 1] > \mid (s, \alpha, s') \in \longrightarrow_{\Diamond_L} \setminus \longrightarrow_{\Box_L}\}$
   - $\mathfrak{C}_{must}(s) = \{< \{(\alpha, s')\}, [1, 1] > \mid (s, \alpha, s') \in \longrightarrow_{\Box_L}\}$

■

Moreover, we know that, taken a CMTS M, we can have some other CMTSs which are syntactically different but semantically equivalent to M. In Figure 5.1 we describe a MTS and different CMTSs which are semantically equivalent to the MTS. Note that the CMTS described in the Definition 5.14 is the CMTS R in Figure 5.1.

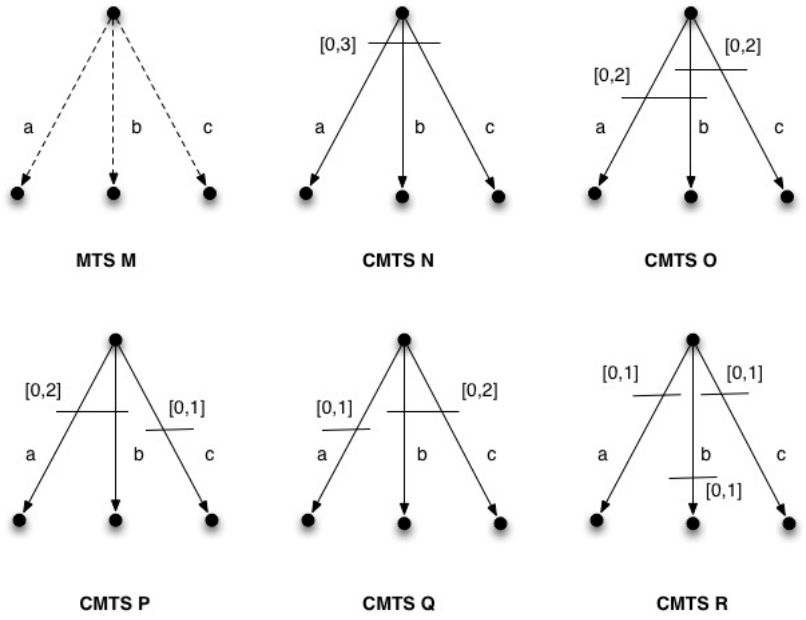


Figure 5.1: An example of different CMTSs semantically equivalent to a MTS

Another similar example is the one described in Figure 5.2. Again, the CMTS described in the Definition 5.14 is the CMTS R. The way to find all these CMTSs, taken a MTS M, is simple: initially we derive the CMTS M_C related to M as we have just see in the Definition 5.14. Then we note that all constraints related to may but not must transitions are special constraints, in effect they are the general non-restrictive constraints because they have the minimum equals to 0 and the maximum equals to the size of the choice set. We know that this type of constraints is very important because adding or deleting a general non-restrictive constraint, the semantics does not change. Therefore, taken M_C, we can derive other semantically equivalent CMTSs simply adding or deleting some general non-restrictive constraints. Again, it is interesting to understand if the refinement relation for the CMTS becomes equivalent to the refinement relation of the MTS when CMTSs describe MTSs or if these two relations are different in any case.

Theorem 5.3. *Let* $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ *be two CMTSs such that they describe two particular MTSs. If it exists a syntactic*

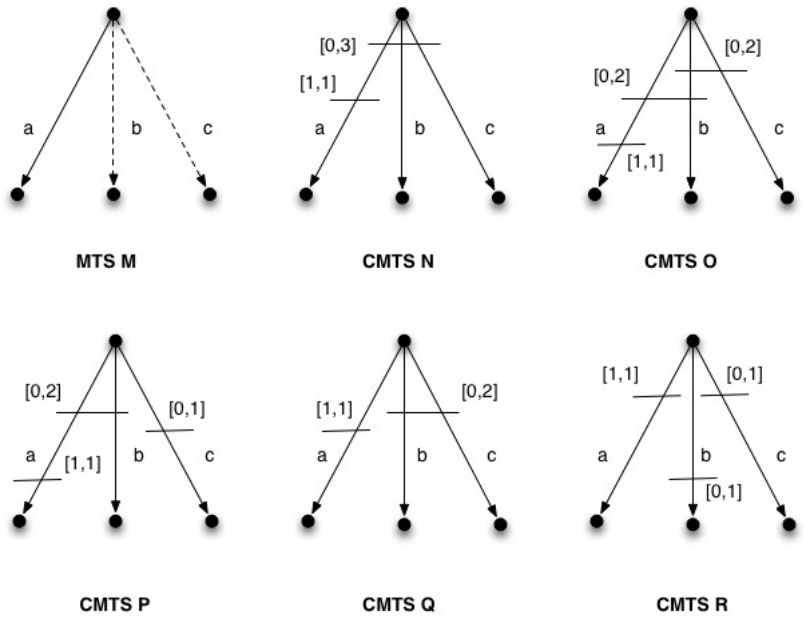


Figure 5.2: Another example of different CMTSs semantically equivalent to a MTS

refinement relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ between M and N then R is a refinement relation of the MTS.

Proof.
For convenience, we recall the definition of refinement relation of MTS: $\mathcal{R}$ is called refinement if, whenever sRt:

1. $s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow t \xrightarrow{\alpha}_{\Diamond} t' \wedge (s', t') \in \mathcal{R}$
2. $t \xrightarrow{\alpha}_{\Box} t' \Rightarrow s \xrightarrow{\alpha}_{\Box} s' \wedge (s', t') \in \mathcal{R}$

First of all, note that each CMTS P equivalent to a MTS R holds the property $\longrightarrow_{\Diamond_R} = \longrightarrow_P$, hence the first condition of refinement of the MTS is equivalent to the first condition of syntactic refinement relation. The problem is in the second condition because in the syntactic refinement we handle constraints.

In this case, we have two types of constraints:

1. constraints with singleton choice set and cardinality $[0, 1]$
2. constraints with singleton choice set and cardinality $[1, 1]$

Since, we cannot have constraints with an empty choice set, we deduce that in each refinement step we cannot delete transitions from choice set of constraints.

Note that, for any constraint c_t with cardinality $[0, 1]$ and choice set $\{(\alpha, t')\}$ we have three possibilities:

1. a constraint c_s which satisfies the condition of refinement exists and in this case $c_s =< \{(\alpha, s')\}, [0,1] >$ and $(s', t') \in \mathcal{R}$

2. a constraint c_s which satisfies the condition of refinement exists and in this case $c_s =< \{(\alpha, s')\}, [1,1] >$ and $(s', t') \in \mathcal{R}$

3. no constraint c_s exists and $Label(c_t) \cap Label(s) = \emptyset$, namely no outgoing transitions with label α can exist

From MTS pointview the previous possibilities describe:

1. the situation where a may outgoing transition, after a refinement step, remains a may transition

2. the situation where a may outgoing transition, after a refinement step, becomes a must transition

3. the situation where a may outgoing transition, after a refinement step, is deleted

Anyway, if $t \xrightarrow{\alpha}_{\square} t'$ then, surely, $(t, \alpha, t') \in \longrightarrow_M$ and $c =< \{(\alpha, t')\}, [1,1] > \in \mathfrak{C}_M(t)$.

Seeing that $\mathcal{R}$ is a syntactic refinement relation then either $Card_{min}(c) = 0 \vee \exists c_s \in \mathfrak{C}(s)$ such that holds some properties. Again, $Card_{min}(c) = 0$ is false, hence a constraint c_s exists and it holds the following properties:

1. $\forall(\alpha, s') \in Choice(c).\ \exists(\alpha, t') \in Choice(c_1).\ (s', t') \in \mathcal{R}$

2. c and c_1 are correct constraints

3. $Card_{min}(c_1) \leq Card_{min}(c)$

4. $Card_{max}(c) \leq Card_{max}(c_1)$

5. $(Label(c_t) \setminus Label(c_s)) \cap Label(s) = \emptyset$

From these properties, we can deduce that $c_s = \{(\alpha, s')\}, [1,1] >$, because the cardinality $[1,1]$ cannot be restricted and we cannot delete transitions from singleton choice set.

Hence, if the state s has the constraint c_s then surely a transition (s, α, s') exists and $(s, \alpha, s') \in \longrightarrow_{\square_S}$. In addition for the condition (1) we also know that $(s', t') \in \mathcal{R}$

We can conclude that the condition (2) of the definition of the refinement holds and $\mathcal{R}$ is also a refinement of MTS.

□

5.1.3 DMTS

The DMTS model is slightly different from the previous ones, in effect it introduces the concept of hypertransition to describe the necessary requirements. By means of a hypertransition we can require that at least one of transitions included in the hypertransition must be present.

In addition, note that in the initial definition of DMTS two types of inconsistency are introduced: the one derived by an hypertransition (s, T) with $T = \emptyset$ and the one derived by the presence of some transition t in hypertransition such that t, at the same time, is not a may transition.

For convenience, we ignore the inconsistent DMTSs because they are less interesting and anyway it is easy to derive a generic inconsistent CMTS.

First of all, we introduce the concept of action-deterministic for DMTS:

Definition 5.15 (**Action-deterministic DMTS**)**:**
A DMTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box}, S_0)$ is an action-deterministic DMTS if and only if:

- $\forall s \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow_{\Diamond} \wedge (s\alpha, s'') \in \longrightarrow_{\Diamond} \Rightarrow s' = s''$
- $|S_0| = 1$

■

This definition was introduced in [10]. Now we can define a DMTS by means of a CMTS:

Definition 5.16 (**DMTS**)**:**
A DMTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_L}, \longrightarrow_{\Box_L}, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ such that:

1. $\longrightarrow_{\Diamond_L} = \longrightarrow_M$
2. $\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \mathfrak{C}_{may}(s) \cup \mathfrak{C}_{must}(s)$ where:
   - $\mathfrak{C}_{may}(s) = \{< \{(\alpha, s')\}, [0, 1] > \mid (s, \alpha, s') \in \longrightarrow_{\Diamond_L} \wedge \nexists (s, V) \in \longrightarrow_{\Box_L} . (\alpha, s') \in V\}$
   - $\mathfrak{C}_{must}(s) = \{< V, [1, |V|] > \mid (s, V) \in \longrightarrow_{\Box_L}\}$

■

In this case L and M must have the same set of transitions too. Compared with the definition of MTS, this time the constraints must change: as for the MTS we must identify outgoing transitions which are may but not must, namely outgoing transitions which are not included in any possible must hypertransition. For the must transitions, instead, we must consider their particular structure and their meaning, namely we must guarantee that all transitions in hypertransition are considered by a constraint and its cardinality is $[1, K]$ where K is the size of the set of

transitions in hypertransition. The reason about this cardinality is simple: at least one of transition must be always considered and at most all transitions must be considered, for the semantics of hypertransition, so the correct cardinality is $[1, K]$. If we decrease the value of the maximum, we might introduce a further restriction about the maximum size of a valid set of transitions and this is clearly wrong in this context.

Again, we can define syntactically different CMTSs such that they are all semantically equivalent to the same DMTS and an example is showed in Figure 5.3. This is

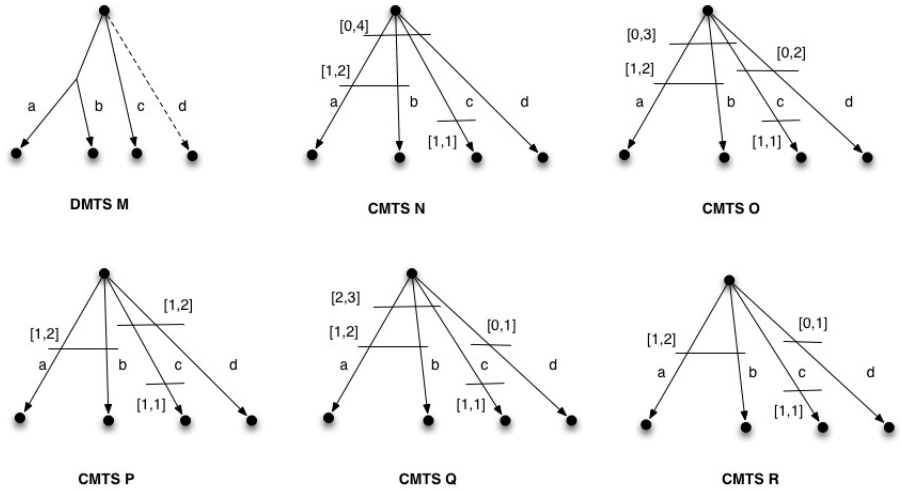


Figure 5.3: An example of different CMTSs semantically equivalent to a DMTS

a very interesting example since the DMTS M in Figure 5.3 has an hypertransition with two transitions, another hypertransition with only one transition and a may transition. The CMTS which can be determined by M through the Definition 5.16 is the CMTS R. Note that all other CMTSs are derived by R adding or deleting the general non-restrictive constraints. Now we see if the refinement relation for the CMTS becomes equivalent to the refinement relation of the DMTS when CMTSs describe DMTSs or if these two relations are different in any case.

Theorem 5.4. *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs such that they describe two particular DMTSs. If it exists a syntactic refinement relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ between M and N then R is a refinement relation of the DMTS.*

Proof.
For convenience, we recall the definition of refinement relation of DMTS: $\mathcal{R}$ is called refinement if, whenever sRt:

1. $s \xrightarrow{\alpha}_{\Diamond} s' \Rightarrow t \xrightarrow{\alpha}_{\Diamond} t' \wedge (s', t') \in \mathcal{R}$
2. $t \rightarrow_{\Box} V \Rightarrow s \rightarrow_{\Box} U$ such that $\forall(\alpha, s') \in U.\ \exists(\alpha, t') \in V \wedge (s', t') \in \mathcal{R}$

First of all, note that each CMTS P equivalent to a DMTS R holds the property $\longrightarrow_{\Diamond_R} = \longrightarrow_P$, hence the first condition of refinement of the MTS is equivalent to the first condition of syntactic refinement relation. The problem is in the second condition because in the syntactic refinement we handle constraints.

In this case, we have two types of constraints:

1. constraints with singleton choice set and cardinality $[0, 1]$
2. constraints with a generic choice set and cardinality $[1, K]$, where K is the size of choice set

Note that, for any constraint c_t with cardinality $[0, 1]$ and choice set $\{(\alpha, t')\}$ we have three possibilities:

1. a constraint c_s which satisfies the condition of refinement exists and in this case $c_s =< \{(\alpha, s')\}, [0, 1] >$ and $(s', t') \in \mathcal{R}$
2. a constraint c_s which satisfies the condition of refinement exists and in this case $c_s =< \{(\alpha, s')\}, [1, 1] >$ and $(s', t') \in \mathcal{R}$
3. no constraint c_s exists and $Label(c_t) \cap Label(s) = \emptyset$, namely no outgoing transitions with label α can exist

From DMTS pointview the previous possibilities describes:

1. the situation where a may outgoing transitions, after a refinement step, remains a may transitions
2. the situation where a may outgoing transitions, after a refinement step, becomes a must transitions
3. the situation where a may outgoing transitions, after a refinement step, is deleted

Anyway, if $t \xrightarrow{\alpha}_{\Box} V$ where $V = \{(\alpha, t'_i)\}$ for some i then, surely, it holds $\forall i.\ (t, \alpha, t'_i) \in \longrightarrow_N$ and $c =< \{V, [1, |V|] >\in \mathfrak{C}_N(t)$.

Seeing that $\mathcal{R}$ is a syntactic refinement relation then either $Card_{min}(c) = 0 \vee \exists c_s \in \mathfrak{C}(s)$ such that holds some properties. Again, $Card_{min}(c) = 0$ is false, hence a constraint c_s exists and it holds the following properties:

1. $\forall(\alpha, s') \in Choice(c).\ \exists(\alpha, t') \in Choice(c_1).\ (s', t') \in \mathcal{R}$
2. c and c_1 are correct constraints
3. $Card_{min}(c_1) \leq Card_{min}(c)$
4. $Card_{max}(c) \leq Card_{max}(c_1)$

5. $(Label(c_t) \setminus Label(c_s)) \cap Label(s) = \emptyset$

From these properties, we can deduce that $c_s =< \{U\}, [1, K] >$, where:

- $\forall(\alpha, s'_i) \in U.\ \exists(\alpha, t'_i) \in V.\ (s'_i, t'_i) \in \mathcal{R}$
- $K \leq |V|$

In addition, seeing that both CMTSs describes DMTSs, then we can deduce that $K = |U|$, hence $|U| \leq |V|$.

Hence, if the state s has the constraint $c_s =< U, [1, |U| >$ then surely $\forall(\alpha, s'_i) \in U.\ (s, \alpha, s') \in \longrightarrow_M$.

Finally, the set of transitions U such that $\forall(\alpha, s'_i) \in U.\ \exists(\alpha, t'_i) \in V.\ (s'_i, t'_i) \in \mathcal{R}$ exists and $(s, U) \in \longrightarrow_{\Box_S}$.

We can conclude that the condition (2) of the definition of the refinement holds and $\mathcal{R}$ is also a refinement of DMTS.

□

5.1.4 1MTS

The next model is slightly different from the DMTS, in effect it introduces the concept of hypertransition to describe not only the necessary requirements but also the allowed requirements. In addition it requires that one and only one transitions in a hypertransition must be considered.

In this context, moreover, the idea of inconsistency is not present because the initial definition of 1MTS handles only hypertransitions (s, V) with $V \neq \emptyset$ and 1MTS has a consistency requirement, that is $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$.

The first problem is to understand what the action-determinism means in hypertransition context. A simple idea is to handle the set of labels and target states as a single label and a single target state, namely taken a hypertransition $t = (s, U)$ we can define the label $L_t = \{\alpha_i \mid (\alpha_i, s_i) \in U\}$ and $S_t = \{s_i \mid (\alpha_i, s_i) \in U\}$, namely a transition with multi-labels and multi-target states. Then the action-determinism is possible if the following property holds:

$$\forall t = (s, U), t_1 = (s, U_1).\ L_t = L_{t_1} \Rightarrow S_t = S_{t_1}$$

The conceptual idea is that we transform each hypertransition in a special transition with the set of labels of transitions in the hypertransition as label and the set of target states of transitions in the hypertransition as target state and then we handle the classic action-determinism property over these new transitions. In this way, we can have some "strange" situations like in Figure 5.4. For example, in the case 2) we have the transition (a, s_3) in two different hypertransitions, then (a, s_3) is considered two times in the computation of corresponding transitions with multi-labels and multi-target states. In the case 3) we have two hypertransitions with the same set of labels but the set of target states is different. Finally, in the case 4) we

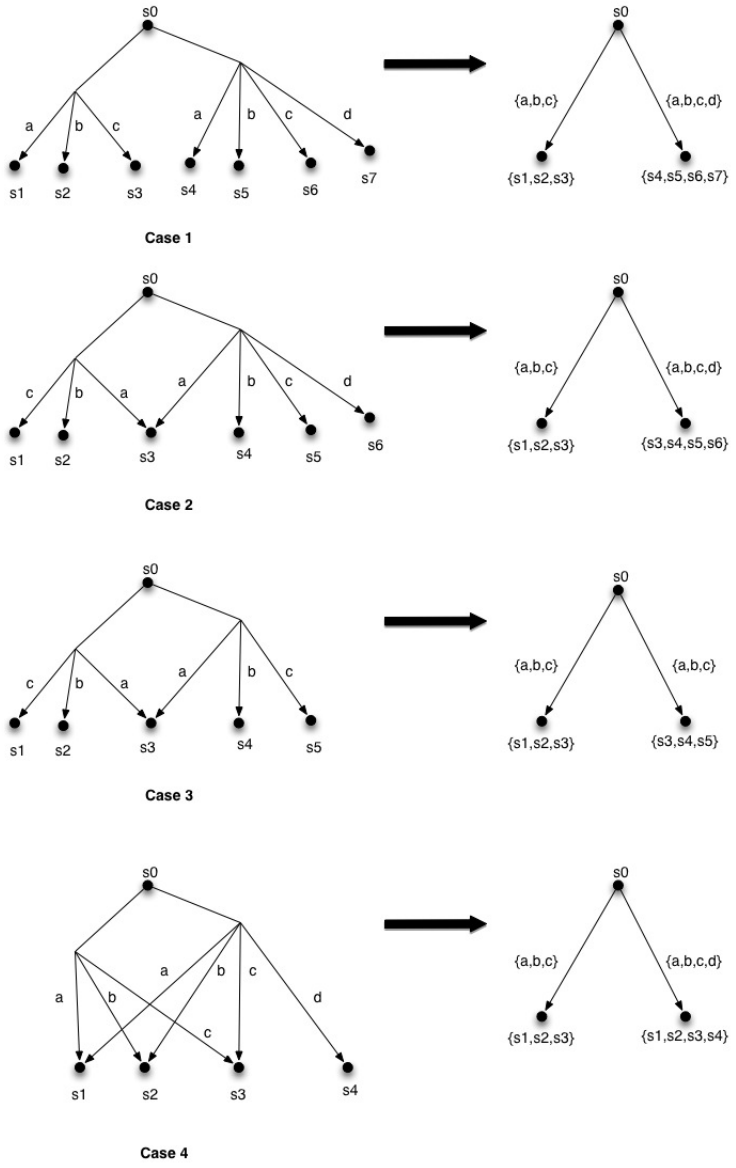


Figure 5.4: Some strange situations in a deterministic hypertransition

have two hypertransitions with the same set of labels and target states except that a hypertransition has a further transition. Anyway, in each case the two derived transitions are always action-deterministic.

Unfortunately, this solution does not work, for example it is possible to have an initial action-deterministic 1MTS that, after a refinement step, loses the property of action-determinism, becoming an action-non deterministic 1MTS as we can see in Figure 5.5. To solve this problem we must handle in a better way the action of a hypertransition, in particular we must guarantee that the target state connected to an action α is always the same in any possible hypertransition.

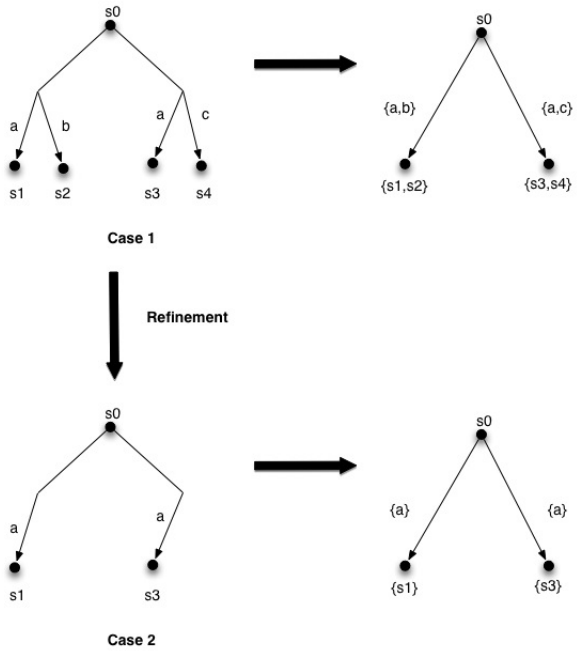


Figure 5.5: The not-maintaining of determinism in the refinement of hypertransition

Definition 5.17 (**Action-deterministic 1MTS**)**:**
A 1MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box}, S_0)$ is an action-deterministic 1MTS if and only if:

- $\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma.\ |\{s' \mid \exists U \in \mathcal{P}(\Sigma \times \mathcal{S}).\ (\alpha, s') \in U\ \wedge (s, U) \in \longrightarrow_{\Diamond}\}| \leq 1$
- $|S_0| = 1$

■

In this way each action identifies univocally the target state, even if the action is present in different hypertransitions. Unfortunately, this solution for 1MTS is not enough because they also introduce the choice function concept and it can handle the same transition in different possible ways. For example, suppose we have a 1MTS M of Figure 5.6 and a choice function γ defined in the following way:

$$\gamma(S) = \begin{cases} (a, s_1) & if\ S = \{(a, s_1), (b, s_2)\} \\ (b, s_2) & if\ S = \{(b, s_2), (c, s_3)\} \end{cases}$$

The derived LTS is the LTS L in Figure 5.6. On the other hand, the possible semantics related to M is that we want to consider exactly one features between a and b and between b and c, of course the possible correct implementations are $\{(a, s_1), (c, s_3)\}$ and $\{(b, s_2)\}$. The problem is raised because for the choice function the same transition in different hypertransitions is different, whereas in this context we would like that the same transition in different hypertransitions is always the same.

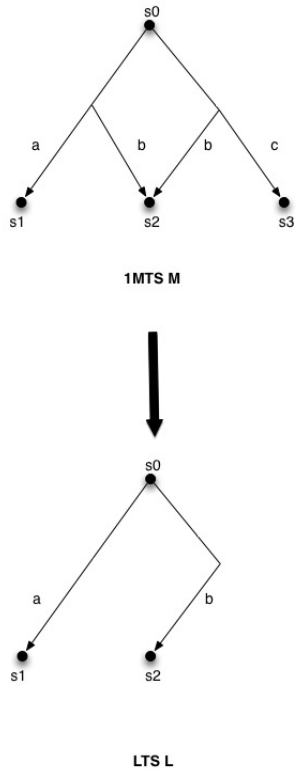


Figure 5.6: An example of why an action-determinism property for 1MTS is not sufficient

Definition 5.18 (**Action-deterministic choice function**)**:**
Let $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond}, \longrightarrow_{\Box}, S_0)$ be an action-deterministic 1MTS , $s \in \mathcal{S}$ be a state of L and $\gamma \in$ **choice**$(s, \longrightarrow_{\Diamond})$ be a choice function. We say that γ is *action-deterministic* if and only if it holds:

$$\forall \alpha \in \Sigma.\ \forall U, V \in \mathcal{P}(\Sigma \times \mathcal{S}).\ (s,U) \in \longrightarrow_{\Diamond} \wedge (s,V) \in \longrightarrow_{\Diamond} \wedge (\alpha, s') \in U \cap V \Rightarrow \gamma(V) \neq (\alpha, s') \Leftrightarrow \gamma(U) \neq (\alpha, s')$$

■

Note that L is action-deterministic hence, taken a label α, we can deduce that it exists a unique target state related to it. This property requires that for each label α, if two hypertransitions U and V exist such that (α, s') is in U and in V, then the choice function make the same choice in respect to α, namely either in both hypertransition α is chosen or in both α is not chosen. In this way we solve the previous problem.

Unfortunately, this solution introduces another problem: consider the 1MTS M in Figure 5.7 then our choice function γ, for the must hypertransition, must choose a transition between (a, s_1) and (b, s_2). At the same time the function γ

for the singleton may hypertransition labelled with b must choose the only possible transition and the same holds for the other may hypertransition. Then we can conclude that each possible choice function is not action-deterministic. Note that

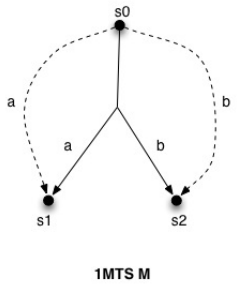


Figure 5.7: An example of the problem of action-deterministic choice functions

this problem is present when we handle singleton hypertransitions where the choice is obligated. Trivially, we have two possibilities:

- a singleton hypertransition is must: in this case the choice is obligated, we must take the hypertransition
- a singleton hypertransition is may: in this case our choice function allows to consider the transition if the transition is taken in some other hypertransition, or it guarantees that the single hypertransition may is discarded and this is described by a new element $\perp$.

Definition 5.19:
Let A be a set of element, $\mathcal{P}_A \subseteq \mathcal{P}(A)$ then we define an *extended choice function* a choice function $\gamma : \mathcal{P}_A \longrightarrow A \cup \{\perp\}$, where $\perp \notin A$.

We denote by $\mathbf{extendedchoice}(\mathcal{P}_A)$ the set of all possible extended choice functions on $\mathcal{P}_A$

■

Now we implicitly assume that, for each state s of a 1MTS:

$$\begin{aligned}\mathbf{choice}(s, \longrightarrow_{\Diamond}) = &\mathbf{choice}(s, \longrightarrow_{\Box}) \cup \\ &\{\mathbf{choice}(U) \mid (s,U) \in \longrightarrow_{\Diamond} \setminus \longrightarrow_{\Box} \wedge |U| > 1\} \cup \\ &\{\mathbf{extendedchoice}(U) \mid (s,U) \in \longrightarrow_{\Diamond} \setminus \longrightarrow_{\Box} \wedge |U| = 1\}\end{aligned}$$

Definition 5.20 (1MTS):
A 1MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_L}, \longrightarrow_{\Box_L}, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ such that:

1. $\longrightarrow_M = \{(s, \alpha, s') \mid \exists U \in \Sigma \times \mathcal{S}.\ (\alpha, s') \in U \wedge (s,U) \in \longrightarrow_{\Diamond_L}\}$
2. $\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \mathfrak{C}_{may}(s) \cup \mathfrak{C}_{must}(s)$ where:

- $\mathfrak{C}_{may}(s) = \{< U, [0,1] > \mid (s,U) \in \longrightarrow_{\Diamond_L} \setminus \longrightarrow_{\Box_L}\}$
- $\mathfrak{C}_{must}(s) = \{< U, [1,1] > \mid (s,V) \in \longrightarrow_{\Box_L}\}$

■

In this case the transition relation of M is equal to the union of all possible transition including in some may hypertransitions. Compared with the definition of DMTS, this time the constraints must change: from each hypertransition we can derive at most one transitions, hence the maximum for may and must hypertransitions is 1. On the other hand, the must hypertransition require that exactly one transition must be always taken, whereas the may hypertransition requires that exactly one transition may be taken. Hence the minimum value for these two types of hypertransitions is obvious: 0 for may hypertransitions, 1 for must ones.

Also in this case, we can define syntactically different CMTSs such that they are all semantically equivalent to the same 1MTS and an example is showed in Figure 5.8. The CMTS which can be determined by M through the Definition 5.20

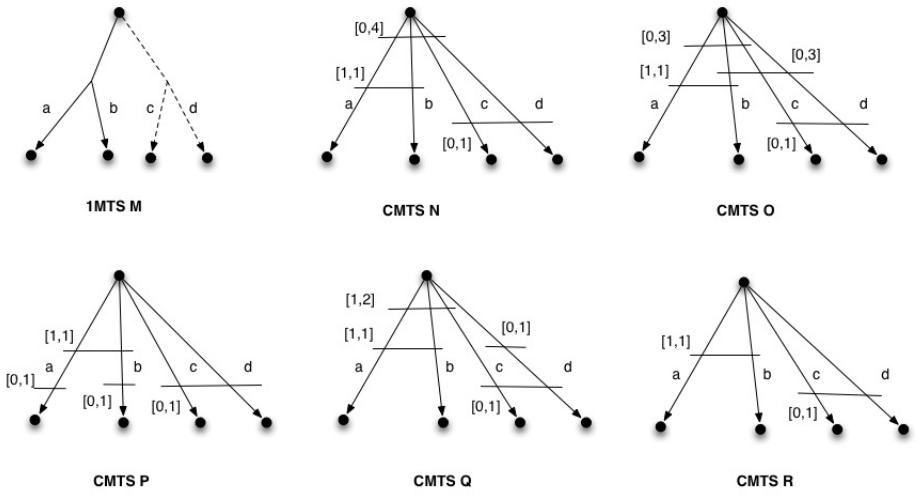


Figure 5.8: An example of different CMTSs semantically equivalent to a 1MTS

is the CMTS R. Note that all other CMTSs are derived by R adding or deleting the general non-restrictive constraints.

Finally, we see if the refinement relation for the CMTS becomes equivalent to the refinement relation of the 1MTS when CMTSs describe 1MTSs or if these two relations are different in any case.

Theorem 5.5. *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{M_0})$, $N = (\mathcal{S}_N, \Sigma, \longrightarrow_N, \mathfrak{C}_N, s_{N_0})$ be two CMTSs such that they describe two particular 1MTSs. If it exists a syntactic refinement relation $\mathcal{R} \subseteq \mathcal{S}_M \times \mathcal{S}_N$ between M and N then R is a refinement relation of the 1MTS.*

Proof.
For convenience, we recall the definition of refinement relation of 1MTS: $\mathcal{R}$ is called refinement if, whenever sRt: $\forall\gamma \in \mathbf{choice}(s \longrightarrow_\Diamond)$. $\exists\widehat{\gamma} \in \mathbf{choice}(t \longrightarrow_\Diamond)$ such that the following holds:

1. $\forall\Theta_s \in (s \longrightarrow_\Diamond)$. $\exists\Theta_t \in (t \longrightarrow_\Diamond)$. $(\gamma(\Theta_s), \widehat{\gamma}(\Theta_t)) \in \mathcal{R}$
2. $\forall\Theta_t \in (t \longrightarrow_\Box)$. $\exists\Theta_s \in (s \longrightarrow_\Box)$. $(\gamma(\Theta_s), \widehat{\gamma}(\Theta_t)) \in \mathcal{R}$

In this context we do not reason about choice functions because they represent a possible choice among all transitions in a hypertransition, effectively we can abstract from this choice because it is implicitly made in the refinement, namely once we choose a transition t in a hypertransition then we can refine our CMTS by means of reducing of the cardinality and deleting transitions from choice sets and states unless the transition t.

Now suppose to have a set $\Theta_s \in (s \longrightarrow_\Diamond)$ then, in the corresponding CMTS, we have a constraint c_s with a choice set equivalent to Θ_s. Of course, this hypertransition (s, Θ_s) can be:

- a must hypertransition but then $Card(c_s) = [1, 1]$
- a may but not must hypertransition but then $Card(c_s) = [0, 1]$

Since c_s exists and a refinement relation $\mathcal{R}$ exists then we can deduce that also a constraint c_t related to c_s exists such that choice set of c_t can have the same transitions of c_s and possibly some more transitions and the cardinality is an extension of $[1, 1]$ and $[0, 1]$. Suppose that $[min_t, max_t]$ is the cardinality of c_t then it holds:

- $min_t \leq max_t$
- $min_t \leq 0$ or $min_t \leq 1$
- $1 \leq max_t$

Hence we can deduce that surely $min_t = 0$ or $min_t = 1$ and $max_t = 1$, seeing that all possible constraints of a CMTS representing a 1MTS have the maximum equals to 1. In each case we can conclude that in t exists a constraint c_t related to c_s, and a may hypertransition with $\Theta_t = Choice(c_t)$ exists. In addition $\forall(\alpha, s') \in Choice(c_s)$. $\exists(\alpha, t') \in Choice(c_t)$. $(s', t') \in \mathcal{R}$ for the refinement definition, so for any possible choice function of s and t $\gamma(\Theta_s), \widehat{\gamma}(\Theta_t)) \in \mathcal{R}$ holds.

Note that the first condition of refinement for 1MTS implicitly assume that no new transition is added in s: if a new transition u_s is added, then u_s is included in some hypertransition (s, V) and some choice function such that $\gamma(V) = u_s$ exists. Of course, in t this transition does not exist hence for any possible choice function and for any possible hypertransition we do not succeed to compare $\gamma(V)$, deducing that the refinement of 1MTS fails.

In our context this property is directly verified by the first condition of CMTS refinement.

Now suppose $\Theta_t \in (t \longrightarrow_{\Box})$ then a constraint c_t such that $c_t =< \Theta_t, [1,1] >$ exists. Trivially, for the definition of refinement, a constraint $c_s =< \Theta_s, [1,1] >$ must exist in s such that $\forall(\alpha, s') \in \Theta_s.\ \exists(\alpha, t') \in \Theta_t.\ (s', t') \in \mathcal{R}$. Also in this case, for any possible choice function of s we can derive a choice function in t such that $\gamma(\Theta_s), \widehat{\gamma}(\Theta_t) \in \mathcal{R}$. □

5.1.5 GEMTS

The last model which we see is the Generalized Extended Modal Transition System (GEMTS). In this case we have two types of transition relations:

1. $\Diamond$ which describes the requirement "at most k of n"
2. $\Box$ which describes the requirement "at least k of n"

First of all, we define the concept of an action-deterministic GEMTS:

Definition 5.21 (**Action-deterministic MTS**)**:**
A GEMTS $L = (\mathcal{S}, \Sigma, \Diamond, \Box, S_0)$ is an action-deterministic MTS if and only if:

- $\forall s \in \mathcal{S}.\ (\alpha, s') \in PossibleTrans(s) \wedge (\alpha, s'') \in\in PossibleTrans(s) \Rightarrow s' = s''$
- $|S_0| = 1$

where $PossibleTrans(s) = \bigcup_{\exists k \in \mathcal{N}.\ (s,U,k) \in \Diamond} U \ \cup \bigcup_{\exists k \in \mathcal{N}.\ (s,C,k) \in \Box} U.$

■

As we said in the hyperref[ChapterCMTS]Chapter 3, CMTS is a formalism which is introduced to describe in a different way the GEMTS concepts in order to study properties and possible extensions in a more simple way.

Hence it is trivial to understand the equivalence between GEMTS and CMTS. From these observation, we can derive a MTS described by means of a CMTS.

Definition 5.22 (**MTS**)**:**
A GEMTS $L = (\mathcal{S}, \Sigma, \Diamond_L, \Box_L, s_0)$ is a particular CMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_M, \mathfrak{C}, s_0)$ such that:

1. $\longrightarrow_M = \{(s, \alpha, s') \mid \exists k \in \mathcal{N}, U \in \Sigma \times \mathcal{S}.\ (\alpha, s') \in U \wedge (s, U, k) \in (\Diamond \cup \Box)\}$
2. $\forall s \in \mathcal{S}.\ \mathfrak{C}(s) = \mathfrak{C}_{\Diamond}(s) \cup \mathfrak{C}_{\Box}(s) \cup \mathfrak{C}_{\Diamond \wedge \Box}(s)$ where:
   - $\mathfrak{C}_{\Diamond}(s) = \{< U, [0, max] > \mid (s, U, max) \in \Diamond_L \wedge \forall min > 0.\ (s, U, min) \notin \Box_L\}$
   - $\mathfrak{C}_{\Box}(s) = \{< U, [min, |U|] > \mid \forall max \in \mathcal{N}.\ (s, U, max) \notin \Diamond_L \wedge (s, U, min) \in \Box_L\}$

- $\mathfrak{C}_{\Diamond\wedge\Box}(s) = \{< U, [min, max] > \mid (s, U, max) \in \Diamond_L \wedge (s, U, min) \in \Box_L\}$

■

Moreover, in this case, we do not handle the refinement relation because in the GEMTS the defined relation connect the GEMTS to product directly, whereas our refinement definition describe a step-by-step refinement. Anyway in [26] a product is described by a LTS and we know that a CMTS No-Choice is equivalent to a LTS, hence it is clear that the refinement relation of a GEMTS is equivalent to the union of all refinement steps necessary to transform a CMTS in a CMTS No-Choice.

5.1.6 Hierarchy

In the previous sections we have seen that we can represent all models of the Modal Family by means of a CMTS. Now in this section we determine the hierarchy of expressivity existing among all models of the Modal Family, note that for hypothesis all considered models are action-deterministic. Since some proofs are very long, in some case we will describe only the theorem and the proof, instead, will be presented in Appendix B.

Theorem 5.6. *The formalism* $\mathbb{LTS}$ *is less expressive of a* $\mathbb{MTS}$*, namely* $\mathbb{LTS} \rightsquigarrow \mathbb{MTS}$

Proof.
First of all, we know that, taken a LTS $L = (\mathcal{S}, \Sigma, \longrightarrow_L)$, L is equivalent to a MTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_M}, \longrightarrow_{\Box_M})$ such that:

$$\longrightarrow_L = \longrightarrow_{\Diamond_M} = \longrightarrow_{\Box_M}$$

In addition no possible LTS can describe the MTS in Figure 5.9. The reason is simple: taken a LTS L then the semantics of L is the set of LTSs which are bisimilar to L. Instead the semantics of a MTS is the set of LTSs which can be derived by means of the refinement relation and these LTSs could also be not bisimilar between them as we can see in Figure 5.9. Hence the semantics of a MTS can represent more LTSs compared with the semantics of a LTS itself. □

Theorem 5.7. *The formalism* $\mathbb{MTS}$ *is less expressive of a* $\mathbb{DMTS}$*, namely* $\mathbb{MTS} \rightsquigarrow \mathbb{DMTS}$

Proof.
First of all, we know that, taken a MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_L}, \longrightarrow_{\Box_L})$, L is equivalent to a DMTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_M}, \longrightarrow_{\Box_M})$ such that:

- $\longrightarrow_L = \longrightarrow_{\Diamond_M}$
- $\forall s \in \mathcal{S}.\ (s, U) \in \longrightarrow_{\Box_M} \ \wedge |U| = 1$

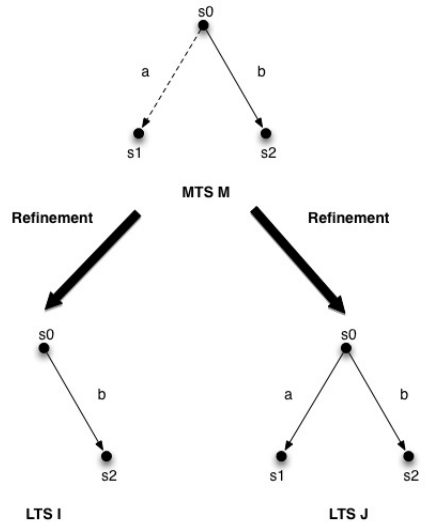


Figure 5.9: An example of a MTS and its derived LTS

Moreover no possible MTS can describe the DMTS in Figure 5.10. Suppose that a MTS L which describes the DMTS M in Figure 5.10 exists then we can deduce that transitions $(a, s_1), (b, s_2)$ are may transitions because they must not be always present. If it is true then the LTS with no transitions is correct for L but is wrong for M, because as we can see in Figure 5.10 all possible LTSs must have at least one transition. On the other hand, if we suppose that (a, s_1) is the must transition in L then the LTS J in Figure 5.10 is wrong for L and the same reasoning is true if we suppose (b, s_2) as must transition or if we suppose that all transitions are must.

Finally, we can deduce that this hypothetical MTS does not exist. □

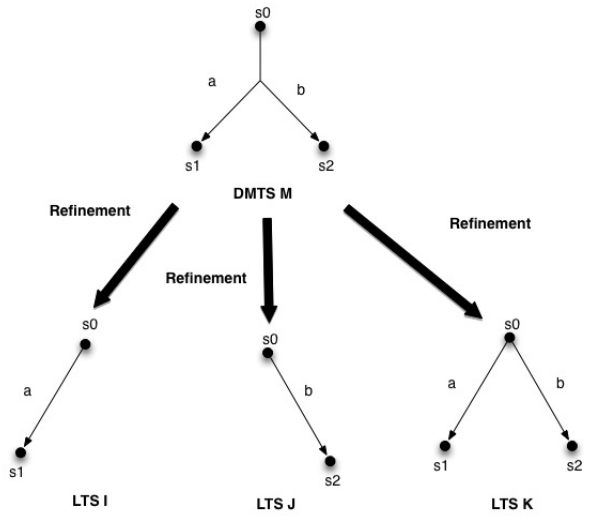


Figure 5.10: An example of a DMTS and its derived LTS

Note that in [27] it is demonstrated that taken a generic DMTS is possible to derive a semantically equivalent 1MTS and the vice versa holds too, namely DMTS and 1MTS are equivalently expressive. Unfortunately, if we consider only DMTS and 1MTS action-deterministic this equivalence is not maintained.

Theorem 5.8. *The formalism* $\mathbb{MTS}$ *is less expressive of a* $1\mathbb{MTS}$*, namely* $\mathbb{MTS} \rightsquigarrow 1\mathbb{MTS}$

Theorem 5.9. *The formalism* $\mathbb{DMTS}$ *and* $1\mathbb{MTS}$ *are not comparable, namely* $\mathbb{DMTS} \not\rightsquigarrow 1\mathbb{MTS}$ *and* $1\mathbb{MTS} \not\rightsquigarrow \mathbb{DMTS}$

Theorem 5.10. *The formalism* $\mathbb{DMTS}$ *is less expressive of the* $\mathbb{CMTS}$*, namely* $\mathbb{DMTS} \rightsquigarrow \mathbb{CMTS}$

Theorem 5.11. *The formalism* $1\mathbb{MTS}$ *is less expressive of the* $\mathbb{CMTS}$*, namely* $1\mathbb{MTS} \rightsquigarrow \mathbb{CMTS}$

Finally, we have seen that GEMTS and CMTS are the same model, represented in two different ways.

We can conclude that our family has the following hierarchy:

$$\mathbb{LTS} \rightsquigarrow \mathbb{MTS} \rightsquigarrow \mathbb{DMTS} \rightsquigarrow 1\mathbb{MTS} \rightsquigarrow \mathbb{GEMTS} \leftrightsquigarrow \mathbb{CMTS}$$

5.2 Hierarchy of the Obligation Family

In this context we focus on OTS and PMTS formalisms and, in addition, we also introduce a new simple formalism: OTS*. The idea is simple: the OTS formalism has the property that, for any state s, the obligation function related to s is a positive boolean formula, namely $\mathbf{\Omega}(s) \in \mathcal{B}(\Sigma \times \mathcal{S})$. This is a very important restriction as we will see, then we would like to understand how the expressivity changes for an OTS with a generic boolean formula. Formally,

Definition 5.23 (Obligation formula syntax):
A *boolean formula* over set X of atomic propositions is given by the following syntax:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \tag{5.1}$$

where $x \in X$. The set of all boolean formulae over X is denoted as $\mathcal{B}(X)$.

■

Definition 5.24 (OTS*):
An *OTS** is a tuple $(\mathcal{S}, \Sigma, \dashrightarrow, \mathbf{\Omega})$ where:

- $\mathcal{S}$ is a set of states
- Σ is a set of actions

- $\dashrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the may transition relation
- $\mathbf{\Omega} : \mathcal{S} \longrightarrow \mathcal{B}(\Sigma \times \mathcal{S})$ is the set of obligations

We denote the set of all possible OTSs* by $\mathbb{OTS}^*$.

■

In addition, seeing that OTS, OTS* and PMTS have the same structure and changes are only the type of obligation function we can define the action-determinism property which is valid for all models:

Definition 5.25 (**Action-deterministic LTS**)**:**
A OTS $L = (\mathcal{S}, \Sigma, \longrightarrow, \mathbf{\Omega}, S_0)$ is an action-deterministic OTS if and only if:

- $\forall s \in \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow \wedge (s, \alpha, s'') \in \longrightarrow \Longrightarrow s' = s''$
- $|S_0| = 1$

■

The definition is the same for OTS* and PMTS.

5.2.1 OTS

In [12] Beneš and Křetínký proved that the OTS is semantically equivalent to a DMTS. The idea is simple: a DMTS is an OTS with obligation functions in a (positive) conjunctive normal form (CNF). Moreover, we know that each boolean formula can be transformed in an equivalent CNF formula, hence taken a generic OTS M we can derive a OTS N where each obligation function is in CNF and M and N are equivalent, but trivially N also describes a DMTS, concluding that $\mathbb{OTS} \leftrightsquigarrow \mathbb{DMTS}$.

Adding the action-determinism property, of course, the result does not change and the proof is similar to the one of a generic OTS.

Note that, for example, no 1MTS can be effectively modelled by a OTS because for each hypertransition (s, T) the semantics of the 1MTS requires that exactly one transition of T must be considered in a valid implementation. By means of positive obligation formula this requirement is impossible to describe:

- $(\alpha, s') \wedge (\beta, s'')$: requires that both transitions are always present
- $(\alpha, s') \vee (\beta, s'')$: requires that at least one transition is always present

We need to say "*take a transition t and not take all other transitions of the set*", but unfortunately we do not have a way to describe "*not take a particular transition*".

5.2.2 OTS*

To solve the problem of describing the concept "*not take a particular transition*", we introduce the negation in the obligation formula. First of all, we must understand which is the level of the expressivity of an OTS*, surely it is true $\mathbb{OTS} \rightsquigarrow \mathbb{OTS}^*$, hence we must compare it to $\mathbb{CMTS}$.

Theorem 5.12. *The formalism* $\mathbb{CMTS}$ *is less expressive of* $\mathbb{OTS}^*$*, namely* $\mathbb{CMTS} \rightsquigarrow \mathbb{OTS}^*$

A typical example where $\mathbb{CMTS}$ is less expressive than $\mathbb{OTS}^*$ is in the case of conditional requirement. As we saw in Chapter 4, the conditional requirement cannot be modelled by means of a CMTS, whereas a OTS* can easily model it, using the typical implication operator of the logic formulae. In Chapter 4, in addition, we saw that to solve this problem, we must introduce the CMTS($\mathcal{G}_\mathcal{T}$), hence we want to understand if a relation between CMTS($\mathcal{G}_\mathcal{T}$) and OTS* exists.

In this context

Theorem 5.13. *The formalism* $\mathbb{OTS}^*$ *is as much expressive as* $\mathbb{CMTS}(\mathcal{G}_\mathcal{T})$*, namely* $\mathbb{OTS}^* \leftrightsquigarrow \mathbb{CMTS}(\mathcal{G}_\mathcal{T})$

5.2.3 PMTS

The last formalism that we see is the PMTS. Trivially, PMTS is an extension of OTS* where parameters are introduced in the obligation formula and of course $\mathbb{OTS}^* \rightsquigarrow \mathbb{PMTS}$, because they cannot handle the parameter in OTS*.

For example, we consider the PMTS in Figure 5.11 and now we try to find out a OTS* equivalent to PMTS. Of course we must focus on the obligation function of the state s_1. Suppose that OTS* which is semantically equivalent to PMTS exists,

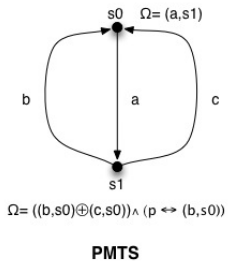


Figure 5.11: An example of a PMTS

then surely the state s_1 of the OTS* must have an obligation formula equals to $((b, s_0) \oplus (c, s_0)) \wedge \varphi$, for some logic formula φ. Of course, we must guarantee that, for each cycle step, exactly one and always the same transition t must be taken. Then, in order to guarantee always the same choice, we can deduce that $\varphi = (b, s_0)$ or $\varphi = (c, s_0)$.

Suppose $\varphi = (b, s_0)$, hence if $p = true$ then PMTS and OTS* are equivalent, otherwise PMTS and OTS* are completely different and we have no common solution effectively. Essentially we can see a PMTS as a set of possible different OTS*, namely we have as many OTS* as possible different assignments are.

Finally, we want to understand the relation between the $\mathbb{CMTS}(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$ and the $\mathbb{PMTS}$, where the first one is an extension of $\mathbb{CMTS}(\mathcal{G})$

Theorem 5.14. *The formalism* $\mathbb{PMTS}$ *is as much expressive as* $\mathbb{CMTS}(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$*, namely* $\mathbb{PMTS} \leftrightsquigarrow \mathbb{CMTS}(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$

In Figure 5.12 we describe all expressivity results where each arrow from a formalism M to a formalism N, describes the concept of $M \rightsquigarrow N$, whereas the double arrow describes that the formalism M and N are expressively equivalent.

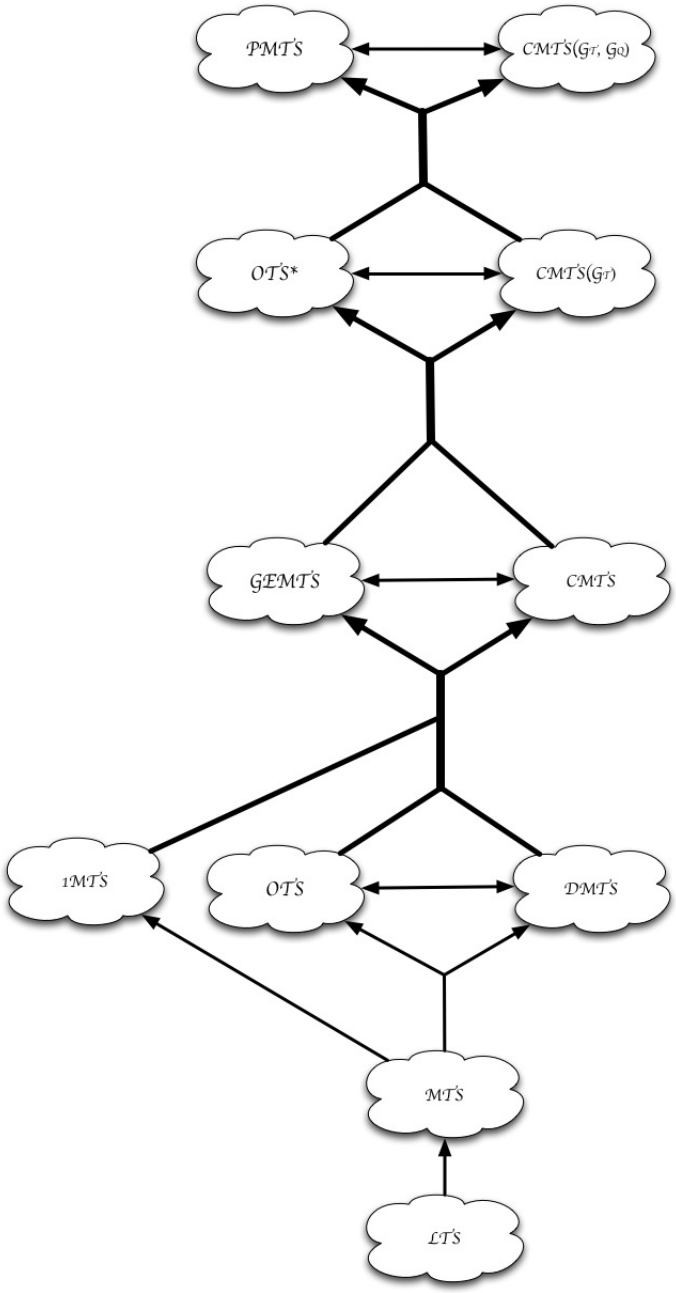


Figure 5.12: The hierarchy of expressivity of models

Chapter 6

Logics for Model Checking

In this chapter we introduce a new logic, based on the CTL* and the Deontic Logic, in particular we will consider the typical operators $\mathbf{O}, \mathbf{P}$ and $\mathbf{F}$ which mean "*it is obligatory that*", "*it is permissible that*" and "*it is forbidden that*", respectively.

Contrary to CTL* where the used models are state-based, in this context we will use CMTS and its extensions which are typically action-based. This is not a restriction and we do not lose expressive power in respect to CTL* and the reason is simple, instead of using CTL* we can use ACTL* and in [21] De Nicola and Vaandrager prove that CTL* and ACTL* are equivalent.

In addition, we will see in an incremental way how our logic can be derived by ACTL* and the Deontic Logic. Then we will introduce some optimizations which reduces the computational cost of algorithms useful to verify properties over CMTSs. Finally we will extend these logics to the CMTS extensions and we will see some further possible optimizations.

Finally, in this context, we reuse the concept of path and other definitions related to it described in Section 2.3. For convenience, we recall them:

Definition 6.1 (**Path(s)**)**:**
Let s be a generic state of a some kind of transition system. We denote by $Path(s)$ the set of all possible paths with s as initial state. We denote the set of all possible paths by $Path$.

■

Definition 6.2 (**Suffix**)**:**
Let $\sigma = s_0, s_1 \ldots$ be a generic path and let $i \in \mathcal{N}$ be an index. We denote by $\sigma[i] = s_i$ the i-th state of σ, whereas we denote by $suffix(\sigma, i) = \varsigma$ the suffix of the path σ from the i-th state, that is if $\varsigma = t_0, t_1 \ldots$ then $\forall j \in \mathcal{N}.\ \varsigma[j] = t_j = s_{i+j} = \sigma[i+j]$.

■

Note that a path is a simple sequence of states.

6.1 Logic for CMTS

The logic which we consider as starting point is a special logic because it is derived by a merge of the syntax of the ACTL* logic and the HML logic. From another pointview, we can see this logic as the HML logic extended with the until operator and path quantifiers, which are typical operators of CTL*. Moreover, as we could see in Section 2.3, the ACTL* logic introduce a new operator $\mathbf{X}_\alpha$, which describes a operator "*next*" with a further requirement, namely we can move in the next state only by means of the action α, and in addition the typical operator next of CTL* can be easily derived by this new operator.

Anyway, we call this logic **Hennessy Milner with Until and path quantifier (HMUL)** [5].

Definition 6.3 (**Syntax of HMUL**)**:**
A correct HMUL formula can be defined according to the following grammar:

$$\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \exists\pi \mid \forall\pi$$
$$\pi ::= \varphi \ U \varphi_1$$

where α is a label describing an action.

The formulae derived by φ is called *state formulae*, whereas the formulae derived by π is called *path formulae*.

■

Trivially, we can derive from this initial set some other derived formulae like $\mathbf{ff}, \varphi \vee \varphi_1$ and so on, using the known rules of propositional logic. In addition we introduce an axiom, typical in the HML context: $\langle\alpha\rangle\varphi = \neg[\alpha]\neg\varphi$. The "*Until*" operator is defined by U. In addition, note that this logic cannot have nested path operator because it has a structure typical of CTL and not of the extension CTL*.

The next step is the definition of the semantics of HMUL logic formulae over CMTS and it is described by means of a satisfaction relation denoted by $\models$, in a more detailed way we have two satisfaction relations: one for state formulae and one for path formulae. Of course, the intended meaning of $s \models \varphi$ is: it is true if and only if s satisfies the formula φ and the same holds for path formulae.

Definition 6.4 (**Semantics of HMUL**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS, $s \in \mathcal{S}$ be a state and ϕ be a state formula. The satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ for state formulae is defined by:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s'.\ s \xrightarrow{\alpha} s' \wedge s' \models \varphi$

- $s \models [\alpha]\varphi \Leftrightarrow \forall s'.\ s \xrightarrow{\alpha} s' \Rightarrow s' \models \varphi$
- $s \models \exists\pi \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \pi$
- $s \models \forall\pi \Leftrightarrow \forall\sigma \in Path(s).\ \sigma \models \pi$

Let σ be a path, then the satisfaction relation $\models \subseteq Path \times \pi$ is defined by:

- $\sigma \models \varphi\ U\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models_S \varphi_1\ \wedge\ \forall 0 \leq i < j.\ \sigma[i] \models \varphi$

■

Note that $\langle\alpha\rangle\varphi$ is semantically equivalent to the ACTL* operator $\mathbf{X}_\alpha\varphi$. Moreover, as we can do in CTL and in CTL*, we can add some further operator like $\mathcal{F}\varphi = \mathbf{tt}U\varphi$ called **eventually** or **finally** and $\mathcal{G}\varphi = \neg\mathcal{F}\neg\varphi$ called **always** or **globally**.

Finally, we want to highlight a strange case: in general the operator $\langle.\rangle$ is the weaker than [.] because $\langle.\rangle$ requires that the property is true for at least one element, whereas [.] requires the same for all elements. In the context of CMTS, which we implicitly assume to be action-deterministic, the roles are inverted. In effect, seeing that we can have at most only one possible target state related to an action α, then $\langle\alpha\rangle\varphi$ requires that a transition labelled with α exists and the target state satisfies the property φ, whereas $[\alpha]\varphi$ is true if either no transition labelled with α exists or a transition exists and the target state satisfies φ. It is clear that, in this case, [.] is weaker than $\langle.\rangle$.

We keep operators $\langle.\rangle$ and [.] for two different reasons:

1. they are "*standard*" operator in the literature
2. we keep them for possible future extensions to a non-deterministic case

Anyway, this logic might be also interpreted in L^2TS because it uses in no way the characteristics of CMTS. Effectively, taken a CMTS M, we can derive a set of LTSs which satisfies constraints described in M. It is clear that in general we are interested in which possible transitions can be executed and which requirements are described by means of constraints.

For example, taken a LTS L, we might say that L, in order to be a correct product for a CMTS M, must have certain transitions whereas all other remaining transitions may be present. In this way we can reason about the deontic logic.

Unfortunately, in the CMTS context is not so clear when an action is obligatory or permissible, on the contrary in the MTS these two concepts are directly represented by two different types of transitions.

From the previous chapter we know that an action α is allowed but not obligatory if a transition t labelled with α has a constraint c related to it such that the choice set of c is singleton and the cardinality is equal to $[0, 1]$, α is obligatory if t is related to a constraint c such that the choice set of c is singleton and the cardinality is

equal to $[1, 1]$. Moreover in the MTS context an action α is forbidden if no outgoing transition labelled with α exists . In the CMTS context, another condition for the prohibition is possible, in effect to say that a transition t is not an outgoing transition is equivalent to have t as outgoing transition and a constraint related to t such that the choice set is singleton and the cardinality is $[0, 0]$. Note that in this case, each possible valid solution I must guarantee that $0 \leq |I \cap t| \leq 0$, namely $t \notin I$. These conditions are not sufficient to guarantee the permission, the obligation or the prohibition.

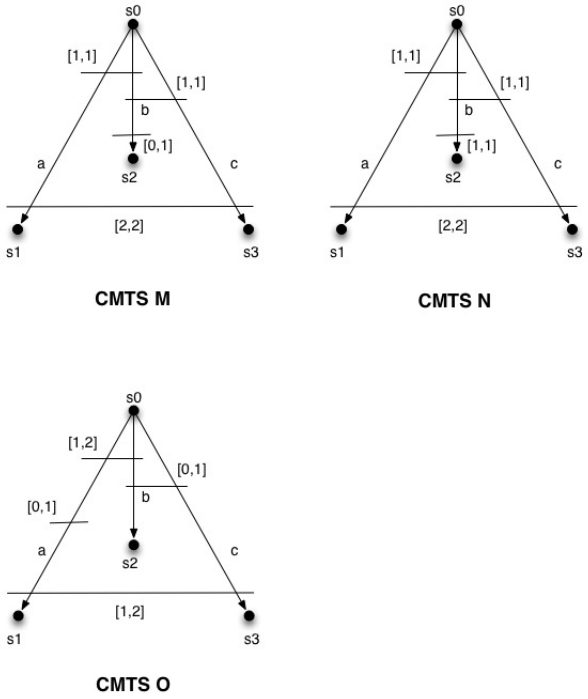


Figure 6.1: Some examples of CMTS with obligatory or forbidden transition not directly visible

Suppose to have the CMTS M of Figure 6.1: in this case the action b appears to be a permitted action but if we see all possible solutions for M, we can observe that (b, s_2) is never present, namely an outgoing transition labelled with b exists and b is forbidden. From this observation we deduce that conditions:

1. if a transition t labelled with α is related to a constraint $c =< \{t\}, [0, 0] >$
2. if no transition with label α exists

are not sufficient conditions for a forbidden action.

Definition 6.5 (**Property 1**)**:**
Let $\alpha \in \Sigma$ be an action and s be a state. If

$$(\nexists s'.\ (\alpha, s') \in Trans(s)) \vee (\exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s'')\} \wedge Card(c) = [0, 0])$$

then α is forbidden. The vice versa is not true.

■

Suppose to have the CMTS N of Figure 6.1: in this case the action b appears to be an obligatory action but if we see all possible solutions for M, we can observe that (b, s_2) is never present, namely an outgoing transition t labelled with b exists and a constraint exists with a singleton choice set related to t and the cardinality equals to $[1, 1]$ but b is forbidden. This is possible when we have an inconsistent CMTS, but we implictly assume that each considered CMTS is consistent.

Finally, suppose to have the CMTS O of Figure 6.1: in this case the action a appears to be a permitted transition but not obligatory, on the contrary if we see all possible solutions for M, we can observe that (a, s_1) is always present, namely an outgoing transition t labelled with a exists, a constraint exists with a singleton choice set related to t and the cardinality equals to $[0, 1]$ but a is obligatory. From this observation we deduce two properties:

Definition 6.6 (**Property 2**)**:**
Let $\alpha \in \Sigma$ be an action and s be a state. If

$$\exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge Card(c) = [1, 1]$$

then α is obligatory. The vice versa is not true.

■

Definition 6.7 (**Property 3**)**:**
Let $\alpha \in \Sigma$ be an action and s be a state. If α is permitted but not obligatory then

$$\exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge Card(c) = [0, 1]$$

The vice versa is not true.

■

Hence, we can deduce that to reason about only transitions is not enough to determine the obligatory, permission or prohibition in a correct way.

A CMTS is a model describing a set of valid LTSs, hence we can define the concept of obligatory, permission and prohibition from the pointview of this set of valid LTSs.

Definition 6.8:
Let M be a CMTS. Then we say:

- the action α is **obligatory** for the state s of M if and only if for each valid LTS, derived by M, the action α is an outgoing transition of the state s_I, where s_I is the corresponding state of s in LTS
- the action α is **forbidden** for the state s of M if and only if for each valid LTS, derived by M, the action α is not an outgoing transition of the state s_I, where s_I is the corresponding state of s in LTS

- the action α is **permitted** for the state s of M if and only if exists a valid LTS, derived by M, such that the action α is an outgoing transition of the state s_I, where s_I is the corresponding state of s in LTS

Formally,

- the action α is **obligatory** for the state s of M if and only if $\forall I \in [\![s]\!]. (\alpha, s') \in I$
- the action α is **forbidden** for the state s of M if and only if $\forall I \in [\![s]\!]. \neg\exists s'. (\alpha, s') \in I$, namely $\forall I \in [\![s]\!]. \forall s'. (\alpha, s') \in I$
- the action α is **permitted** for the state s of M if and only if $\exists I \in [\![s]\!]. \exists s'. (\alpha, s') \in I$

■

Now we want to define operators $\mathbb{O}, \mathbb{P}$ and $\mathbb{F}$. Note that operators $\mathcal{F}$ and $\mathbb{F}$ are two different operators: the first is the eventually temporal operator, whereas the second describes the deontic operator of prohibition. In addition we want to define these operators in such a way that deontic axioms holds:

- $\mathbb{F}\alpha = \mathbb{O}\neg\alpha$
- $\mathbb{P}\alpha = \neg\mathbb{O}\neg\alpha$
- $\mathbb{O}\alpha \Rightarrow \mathbb{P}\alpha$

First of all, we must define what means the formula α in our context: trivially, α should represent the presence of a transition labelled with α in some set of transitions I and, hence, $\alpha = \exists s'. (\alpha, s') \in I$. Note that the semantics of α operator depends on I, therefore the semantics of α in some way is parametric.

Trivially, we can suppose that the semantics of the operator $\mathbb{O} = \forall I \in [\![s]\!]$ for some state s, namely the property must be always true, for each possibility. Then the formula $\mathbb{O}\alpha$ can be seen as the composition of $\mathbb{O}$ and α, deriving $\forall I \in [\![s]\!]. \exists s'. (\alpha, s') \in I$.

In this way, we can derive operators $\mathbb{F}$ and $\mathbb{P}$:

- $\mathbb{F} = \mathbb{O}\neg\alpha$, namely $\forall I \in [\![s]\!]. \neg(\exists s'. (\alpha, s') \in I) = \forall I \in [\![s]\!]. \forall s'. (\alpha, s') \notin I$. This is exactly the conceptual semantics of $\mathbb{F}$
- $\mathbb{P} = \neg\mathbb{O}\neg\alpha$, namely $\neg(\forall I \in [\![s]\!]. \neg(\exists s'. (\alpha, s') \in I)) = \exists I \in [\![s]\!]. \exists s'. (\alpha, s') \in I$. Again, this is exactly the conceptual semantics of $\mathbb{P}$

Anyway, often we do not only want to understand if a transition with label α is executed but we want to know if a transition with label α is executed and the reached target state satisfies some property φ. Hence, our atomic deontic operator is not α but it is $\alpha(\varphi)$, where its semantics is $\alpha(\varphi) = \exists s'. (\alpha, s') \in I \wedge s' \models \varphi$.

Trivially the semantics of operators $\mathbb{O}, \mathbb{F}$ and $\mathbb{P}$ changes in the correct way:

- $\mathbb{O}\alpha(\varphi)$, namely $\forall I \in [\![s]\!].\ \exists s'.\ (\alpha, s') \in I \wedge s' \models \varphi$, namely "it is obligatory that a transition labelled with α is executed and the reached state satisfies φ"
- $\mathbb{F}\alpha(\varphi) = \mathbb{O}\neg\alpha(\varphi)$, namely $\forall I \in [\![s]\!].\ \neg(\exists s'.\ (\alpha, s') \in I \wedge s' \models \varphi) = \forall I \in [\![s]\!].\ \forall s'.\ (\alpha, s') \in I \Rightarrow s' \not\models \varphi$. This is exactly the conceptual semantics of $\mathbb{F}$, namely "it is forbidden that a transition labelled with α is executed and the reached state satisfies φ"
- $\mathbb{P}\alpha(\varphi) = \neg\ \mathbb{O}\neg\alpha(\varphi)$, namely $\neg(\forall I \in [\![s]\!].\ \neg(\exists s'.\ (\alpha, s') \in I \wedge s' \models \varphi)) = \exists I \in [\![s]\!].\ \exists s'.\ (\alpha, s') \in I \wedge s' \models \varphi$. Again, this is exactly the conceptual semantics of $\mathbb{P}$, namely "it is permitted that a transition labelled with α is executed and the reached state satisfies φ"

Of course, we can extend the semantics of the deontic operator to a set of actions A or we can change the semantics of the deontic operator $\alpha(\varphi) = \forall s'.\ (\alpha, s') \in I \Rightarrow s' \models \varphi$ but, if we also want to satisfy the deontic axioms, the formal semantics which we can derive is slightly different with respect to the conceptual semantics. For example, this difference can be derived by the ambiguity of the redefined $\alpha(\varphi)$ and this is the typical ambiguity between the logic implication and the implication of natural language.

Now we can define our deontic extension of HMUL: the **DHMUL**.

Definition 6.9 (**Syntax of DHMUL**)**:**
A correct DHMUL formula can be defined according to the following grammar:

$$\begin{aligned}
&\delta ::= \mathbf{tt} \mid \neg\delta \mid \alpha(\varphi)\\
&\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi\ \wedge\ \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \mathbb{O}\delta \mid \mathbb{F}\delta \mid \mathbb{P}\delta \mid \exists\pi \mid \forall\pi\\
&\pi ::= \varphi\ U\varphi_1
\end{aligned}$$

where α is a label describing an action.

The formulae derived from φ are called *state formulae*, the formulae derived by π is called *path formulae* and the formulare derived from δ are called *atomic deontic formulae*.

■

Note that deontic operators are related to state formulae because they describe a property of the state and not of the path. In addition the semantics of δ cannot only be computed by means of a state s or a path π, we also need of having a set of transitions.

Of course, we can derive operator like $\mathcal{F}\varphi = \mathbf{tt}U\varphi$, $\mathcal{G}\varphi = \neg\mathcal{F}\neg\varphi$ and all other one typical of CTL logic.

Definition 6.10:
Let $M = (S, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS, s be a state and ϕ be a state formula. The satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ for state formulae is defined by:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s'.\ s \xrightarrow{\alpha} s' \wedge s' \models \varphi$
- $s \models [\alpha]\varphi \Leftrightarrow \forall s'.\ s \xrightarrow{\alpha} s' \Rightarrow s' \models \varphi$
- $s \models \mathbb{O}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{P}\delta \Leftrightarrow \exists I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{F}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \not\models \delta$
- $s \models \exists\pi \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \pi$
- $s \models \forall\pi \Leftrightarrow \forall\sigma \in Path(s).\ \sigma \models \pi$

Let s be a state, I be a set of transitions and φ be a state formula then the satisfaction relation $\models \subseteq \mathcal{S} \times \mathcal{P}(\Sigma \times \mathcal{S}) \times \delta$ for deontic formulae is defined by:

- $s, I \models \mathbf{tt}$
- $s, I \models \neg\delta \Leftrightarrow s, I \not\models \delta$
- $s, I \models \alpha(\varphi) \Leftrightarrow \exists(\alpha, s') \in Trans(s).\ (\alpha, s') \in I \wedge s' \models \varphi$

Let σ be a path and φ, φ_1 be two state formulae then the satisfaction relation $\models \subseteq Path \times \pi$ for path formulae is defined by:

- $\sigma \models \varphi\ U \varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \varphi_1\ \wedge \forall 0 \leq i < j.\ \sigma[i] \models \varphi$

■

The next step is to understand if deontic operators hold after a refinement step, that is if we have two CMTSs M and N, such that N is a refinement of M and we have a deontic operator D and a deontic formula δ then $s_M \models D\delta$ implies that $s_N \models D\delta$, where $(s_N, s_M) \in \mathcal{R}$ for some refinement relation $\mathcal{R}$.

Theorem 6.1. *Let s_M be a state of a CMTS M and δ be a deontic formula. Then for any s_N of a CMTS N, such that N is a refinement of M and $(s_N, s_M) \in \mathcal{R}$ for some refinement relation $\mathcal{R}$, it holds:*

$$s_M \models \mathbb{O}\delta \Rightarrow s_N \models \mathbb{O}\delta$$

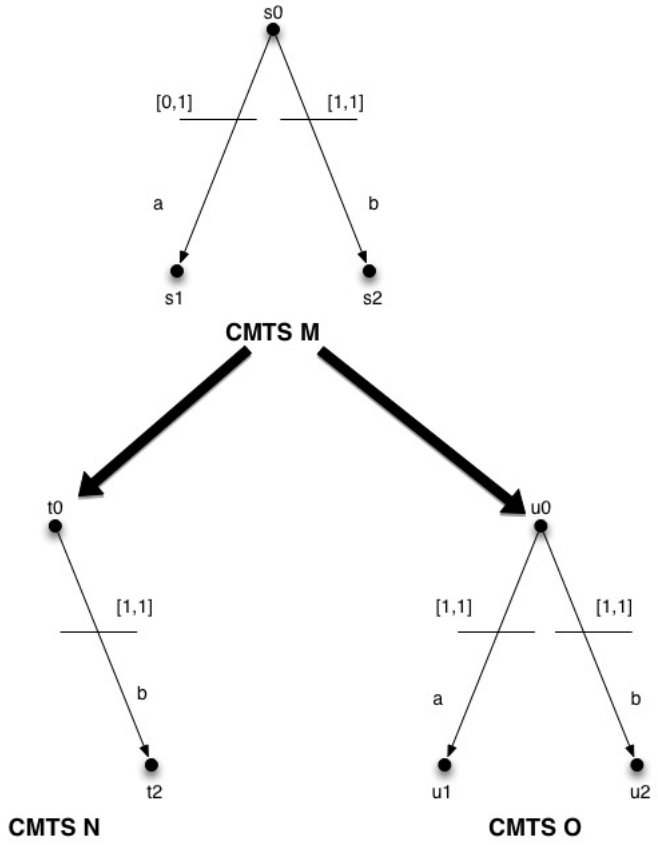


Figure 6.2: An example of a CMTS and its implementations

Proof.
We know that $s_M \models \mathbb{O}\delta \Leftrightarrow \forall I \in [\![s_M]\!].\ s, I \models \delta$. In addition, we know that if $(s_N, s_M) \in \mathcal{R}$ for some refinement relation then $\forall I.\ I \in [\![s_N]\!] \Rightarrow I \in [\![s_M]\!]$. For convenience, we suppose that M and N have the same set of states, in this way we do not verify if each couple of states is in the refinement relation $\mathcal{R}$.

Trivially, we can deduce that $\forall I \in [\![s_N]\!].\ s, I \models \delta$. □

The vice versa is not true: for example, taken the CMTS M and the formula $\phi = \mathbb{O}a(\mathbf{tt})$, trivially $s_0 \not\models \phi$. On the other hand, in the CMTS O, which is a refinement of M, $u_0 \models \phi$. The reason is simple: by means of a refinement some permitted actions becomes obligatory and, hence the vice versa does not hold.

Theorem 6.2. *Let s_M be a state of a CMTS M and δ be a deontic formula. Then for any s_N of a CMTS N, such that N is a refinement of M and $(s_N, s_M) \in \mathcal{R}$ for some refinement relation $\mathcal{R}$, it holds:*

$$s_M \models \mathbb{F}\delta \Rightarrow s_N \models \mathbb{F}\delta$$

Proof.
Since $\mathbb{F}\delta = \mathbb{O}\neg\delta$ then the theorem is true for Theorem 6.1. □

Again, the vice versa is not true: for example, taken the CMTS M and the formula $\phi = \mathbb{F}a(\mathbf{tt})$, trivially $s_0 \not\models \phi$. On the other hand, in the CMTS N, which is a refinement of M, $t_0 \models \phi$. The reason is simple: by means of a refinement some permitted actions becomes forbidden and, hence the vice versa does not hold. Unfortunately, this property is not maintained for the operator $\mathbb{P}$: for example, taken the CMTS M and the formula $\phi = \mathbb{P}a(\mathbf{tt})$, trivially $s_0 \models \phi$. On the other hand, in the CMTS O, which is a refinement of M, $u_0 \not\models \phi$. The reason is simple: by means of a refinement some permitted actions is deleted and, hence, some actions becomes forbidden.

6.1.1 Optimizations

It is clear that DHMUL is a very expensive logic from computational pointview because, for each state, we must compute its semantics, on the other hand, for the model checking technique the computational cost is fundamental. In this section we try to modify the initial CMTS in such a way that the verification of the property is less expensive.

First of all, we note that a label α is obligatory, namely it is always present, if a constraint $c =< \{t\}, [1,1] >$ for the transition t related to α exists, in effect in this way we say that t is a must transition. On the other hand, if the constraint $c =< \{t\}, [0,1] >$ for a transition t related to α exists then t is a may transition and t can be present in some solutions.

Unfortunately, properties **Property 1**, **Property 2** and **Property 3**, defined in the previous section, prove us that the obligatoriness, the permissibility and the prohibition, which we can derive from the semantics, are not always computed in a correct way if we consider only outgoing transitions and constraint related to them. Anyway, if we succeed to solve these properties, then we can verify the obligatory, the permission and the prohibition directly by means of outgoing transitions and singleton constraints.

Definition 6.11 (**Forbidden action-free**)**:**
Let M be a CMTS, s be a state of M and α be a label of some outgoing transition of s. Then we say that α is *forbidden* if and only if $\forall I \in [\![s]\!]. \forall s'. (\alpha, s') \notin I$.

We say that a state s is *forbidden-action free* if and only if it holds that, for each action α: if $\exists s'. (\alpha, s') \in Trans(s)$ then α is not an action forbidden.

We say that a CMTS M is *forbidden-action free* if and only if for each state s, s is forbidden action-free.

■

Trivially, the following corollary holds:

***Corollary* 6.1:** *Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS forbidden action-free then:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma. \alpha \ is\ forbidden \Leftrightarrow \forall s' \in \mathcal{S}.\ (\alpha, s') \notin Trans(s)$$

Of course, it is possible to define an algorithm which transforms a generic CMTS M in a CMTS forbidden action-free N such that M and N are semantically equivalent. The hint of how to develop this algorithm is: for each state s, we can compute the semantics of s, then we determine a set $TransPerm$ of outgoing transitions of s which are present in some solution of constraints of s. Hence, we delete, in a correct way, each transition in $Trans(s) \setminus TransPerm$. In this way, we solve all problems related to the property **Property 1**.

Definition 6.12 (**Hidden must action-free**)**:**
Let M be a CMTS, s be a state of M and α be a label of some outgoing transition of s. Then we say that α is *obligatory* if and only if $\forall I \in [\![s]\!]. \exists s'. (\alpha, s') \in I$.

We say that a state s is *hidden must-action free* if and only if for each action α it holds: $\exists c \in \mathfrak{C}(s).\overline{\overline{\in}} < \{(\alpha, s')\}, [1,1] >$ if and only if α is an obligatory action.

We say that a CMTS M is *hidden must-action free* if and only if for each state s, s is hidden must action-free.

■

In this way, we describe in an explicit way the obligatory of a transition t labelled with α.

Trivially, the following corollary holds:

***Corollary* 6.2:** *Let $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS hidden must action-free then:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma. \alpha \textit{ is obligatory} \Leftrightarrow \exists c \in \mathfrak{s}.\ c = \{(\alpha, s')\}, [1,1]1 >$$

Of course, it is possible to define an algorithm which transforms a generic CMTS M in a CMTS hidden must action-free N such that M and N are semantically equivalent. The hint of how to develop this algorithm is: for each state s, we can compute the semantics of s, then we determine a set $TransObb$ of outgoing transitions of s which are always present. At this point, we add a new constraint $< \{t\}, [1,1] >$ for each transition in $TransObb$. In this way, we solve all problems related to the property **Property 2**.

In addition, we suppose that, for each transition t, a constraint c such that $Choice(s) = \{t\}$ always exists, then also the **Property 3** is solved:

Theorem 6.3. *Let M be a CMTS hidden must-action free and forbidden action free. Taken a state s of M, then an action α is allowed but not obligatory for s if and only if $\exists s'. (\alpha, s') \in Trans(s) \wedge \nexists c \in \mathfrak{C}(s).\ c =< \{(\alpha, s'), [1,1] >$.*

Proof.
Since M is forbidden action-free then surely if $\exists s'. (\alpha, s') \in Trans(s)$ then α is not forbidden, moreover seeing that M is hidden must-action free then α is obligatory if and only $\exists c =< \{(\alpha, s')\}, [1,1] >$. Note that for a CMTS we can have only one constraint for any outgoing transition, hence the possible constraints for t are:

- $c =< \{(\alpha, s')\}, [1,1] >$: in this case α is obligatory

- $c =< \{(\alpha, s')\}, [0,1] >$: in this case (α, s') is present in some solution, namely α is an allowed but not obligatory action
- $c =< \{(\alpha, s')\}, [0,0] >$: in this case (α, s') is never present in some solution, then α is forbidden but for property of forbidden action-free, this case is impossible
- $c =< \{(\alpha, s')\}, [1,0] >$: in this case c is a incorrect constraint and this is impossible

□

Note that $c =< \{(\alpha, s')\}, [0,1] >$ is a general non-restrictive constraint, hence it can be present in $\mathfrak{C}(s)$.

Theorem 6.4. *Let M be a CMTS hidden must-action free and forbidden action free. Taken a state s of M, then an action α is allowed if and only if $\exists s'.\ (\alpha, s') \in Trans(s)$*

Proof.
An action α is permitted if and only if it is obligatory or allowed but not obligatory. In the first case we have that in the state s we must have a constraint $c =< \{(\alpha, s')\}, [1,1] >$ and $(\alpha, s') \in Trans(s)$.

On the other hand, if α is allowed but not obligatory we must have and $(\alpha, s') \in Trans(s)$ and a constraint $c =< \{(\alpha, s')\}, [1,1] >$ does not exists.

Then we can deduce that an action is permitted if and only if $\exists(\alpha, s') \in Trans(s) \wedge (\exists c =< \{(\alpha, s')\}, [1,1] >\in \mathfrak{C}(s) \vee \neg(\exists c =< \{(\alpha, s')\}, [1,1] >\in \mathfrak{C}(s)))$, deducing the theorem. □

Definition 6.13 (**Fully described action**)**:**
Let M be a CMTS. Then we call M is a *fully described action* CMTS if and only if M is hidden must-action free and forbidden action free.

■

Moreover, note that the transformation algorithm from CMTS to CMTS forbidden action free holds the hidden must-action free property, in effect this transformation takes into account only forbidden action, and the transformation algorithm from CMTS to CMTS hidden must action free holds the forbidden action free property, this time the transformation takes into account only obligatory action.

Now we want to highlight that the computational cost related to the computation of the semantics of each state s is not deleted obviously, but now it is reduced to only one initial computation, namely the computation which transforms a CMTS in a fully described action CMTS.

In addition, we can further delete this computation, in effect in the first phase of handling of a CMTS, we would like to check some properties like the consistency property. To verify the consistency property we must derive all possible solutions

for constraints of s, in this phase we can add the needed operations for the CMTS fully described action. In this way we compute only one time the semantics of states and in a previous phase with respect to the verification of property described by means of the logic.

Now we extend the action to the action with state property and we try to understand what means $\alpha(\varphi)$ is permitted, obligatory or forbidden:

Definition 6.14:
Let $\alpha(\varphi)$ be an action with state property and s be a state of a CMTS. Then we say:

- $\alpha(\varphi)$ is permitted for s if and only if the action α is permitted and the target state satisfies φ, namely $\alpha(\varphi)$ is permitted if and only if $\exists I \in [\![s]\!]. \exists(\alpha, s') \in Trans(s). (\alpha, s') \in I \wedge s' \models \varphi$

- $\alpha(\varphi)$ is obligatory for s if and only if the action α is obligatory and the target state satisfies φ, namely $\alpha(\varphi)$ is obligatory if and only if $\forall I \in [\![s]\!]. \exists(\alpha, s') \in Trans(s). (\alpha, s') \in I \wedge s' \models \varphi$

- $\alpha(\varphi)$ is forbidden for s if and only if the action α is forbidden or the target state does not satisfies φ, namely $\alpha(\varphi)$ is forbidden if and only if $\forall I \in [\![s]\!]. \forall(\alpha, s') \in Trans(s). (\alpha, s') \notin I \vee s' \not\models \varphi$

■

Now we see these definitions in fully described action CMTS:

Definition 6.15:
Let $\alpha(\varphi)$ be an action with state property and s be a state of a fully described action CMTS. Then we say:

- $\alpha(\varphi)$ is permitted for s if and only if the action α is permitted and the target state satisfies φ, namely $\alpha(\varphi)$ is permitted if and only if $\exists s'. (\alpha, s') \in Trans(s) \wedge s' \models \varphi$

- $\alpha(\varphi)$ is obligatory for s if and only if the action α is obligatory and the target state satisfies φ, namely $\alpha(\varphi)$ is obligatory if and only if $\exists s'. (\alpha, s') \in Trans(s) \wedge (\exists c \in \mathfrak{C}(s). c =< \{(\alpha, s')\}, [1, 1] >) \wedge s' \models \varphi$

- $\alpha(\varphi)$ is forbidden for s if and only if the action α is forbidden or the target state does not satisfies φ, namely $\alpha(\varphi)$ is forbidden if and only if $\forall s'. (\alpha, s') \notin Trans(s) \vee s' \not\models \varphi$

■

Note that the definition of permitted and obligatory actions with state property are very similar, the only difference is the condition

$$(\exists c \in \mathfrak{C}(s).\ c =< \{(\alpha, s')\}, [1,1] >)$$

which describes the obligatory.

Unfortunately, this definition does not hold the deontic axiom $\mathbb{F}\alpha(\varphi) = \mathbb{O}\neg\alpha(\varphi)$, because we do not know what means $\neg\alpha$ in the condition of obligatory $(\exists c \in \mathfrak{C}(s).\ c =< \{(\neg\alpha, s')\}, [1,1] >)$.

The first step is to observe that the condition $(\exists c \in \mathfrak{C}(s).\ c =< \{(\alpha, s')\}, [1,1] >)$ can be rewritten in another way: $\exists s'.\ \exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge Card(c) = [1,1])$. Now we focus on the obligatory condition which is described by $Card(c) = [1,1]$.

The idea of the obligatory is that "*for each possible solution a transition t must be always present*" and this concept is modelled with two different conditions: suppose that $Card(c) = [min, max]$ then

- $min = max$ describes the concept of "*for each possible solution a transition t always satisfies the same property*"
- $max = 1$ describes the concept of "*a transition t can be present in some solution*", in effect, seeing that $0 \leq min \leq max$ for each constraint and the constraint has a singleton choice set, then in this way we require that for each solution the transition t can be present.

Hence the conjunction $max = min \wedge max = 1$ describes the property "*for each possible solution a transition t always satisfies the property of being present*". Note, for example, that $max = min \wedge max = 0$ describes the property "*for each possible solution a transition t always satisfies the property of not being present*".

Moreover, we can observe that $max = min \wedge max = 1$ is composed by two different conditions:

- $max = min$ is a condition strictly related to the operator $\mathbb{O}$
- $max = 1$ is the condition related to the deontic formula $\alpha(\varphi)$, because $\alpha(\varphi)$ requires the existence of a transition labelled with α.

Now we try to understand the meaning of $\mathbb{O}\alpha(\varphi)$. In this case we want to describe that it is obligatory the presence of a transition labelled with α and the reached state satisfies φ. This requires, hence, that a constraint c exists such that it describes a transition labelled with α and the target state satisfies φ, in addition we want that this constraint describes the obligatoriness.

We know that our deontic operators reason about only the actions, that is $\mathbb{O}\alpha(\varphi)$ can be seen as $\mathbb{O}(\alpha \wedge (\varphi_\alpha)) = \mathbb{O}\alpha \wedge \varphi_\alpha$, where φ_α describes the idea that a target state, reached by means of α satisfies φ.

We can say that a constraint c satisfies $\mathbb{O}\alpha(\varphi)$ if c satisfies $\mathbb{O}\alpha \wedge \varphi_\alpha$. Trivially, c satisfies (φ_α) if and only if it describes a transition labelled with α and the target state satisfies φ. On the other hand, c satisfies $\mathbb{O}\alpha$ if and only if it describes a transition labelled with α and its cardinality $[min, max]$ is defined in such a way that $min = max \wedge max = 1$.

Now we try to consider $\mathbb{O}\neg\alpha(\varphi)$. Conceptually, this equivalent to say that a transition with label $\neg\alpha$ is obligatory and the target state $s_{\neg\alpha}$ satisfies φ. Seeing that $\neg\alpha$ describe the not-presence of a transition α, this means that for each possible executions which we can do from s we can never execute an action α and reach a state $s_{\neg\alpha}$ satisfying φ.

In the "real world", we have only transitions with "positive" label then we must guarantee that for each possible transition or the transition is not labelled by α or the reached state does not satisfy φ. Note that these hypotheses are very restrictive because we only want that the not-executability of α is always true, therefore it is not required that outgoing transitions (α, s') of s must be necessarily absent but simply it requires that such outgoing transitions must be never executable.

The second step is to understand what means $\mathbb{O}\neg\alpha$, that is it is obligatory the not-presence of α, so it is required that no transition labelled with α can be executed. This is possible if no outgoing transition with label α exists or, if some transition t exists but it is never executed in some LTS and this is possible if and only if $t = (\alpha, s')$ is correlated to a constraint with cardinality $[0, 0]$, that is it is obligatory to take never the transition. Seeing that the the obligatoriness is defined by $max = min$ and the absence of the transition is described by $max = 0$, then we can deduce that the condition becomes $max = min \wedge max = 0$.

We want to highlight as conditions for $\mathbb{O}\neg\alpha$ and conditions for $\mathbb{O}\alpha$ are strange because some are negated, whereas other ones are not. In effect the condition about action is correctly changed, whereas the condition about the operator is correctly unchanged.

The last step is to understand the semantics of $\mathbb{O}\neg\alpha(\varphi)$. Previously, we have said that the meaning of $\mathbb{O}\alpha(\varphi)$ is that a constraint c satisfies $\mathbb{O}\alpha(\varphi)$. Conceptually, in the context of $\mathbb{O}\neg\alpha(\varphi)$ we should have a constraint c which satisfies $\varphi_{\neg\alpha}$ and $\mathbb{O}\neg\alpha$, that is a transition with label $\neg\alpha$ must exist, the reached state must satisfy φ and it is obligatory. Again, in the real world we have only positive label, hence a constraint $c =< \{\neg\alpha, s'\}, [1, 1]$ such that $s' \models \varphi$ is equivalent to say that any possible executions which we can do for any possible product must always verify that the not-executability of α satisfies φ, that is a possible executability of α must not satisfy φ. Hence, in this context, we must check all possible outgoing transitions and verify that a transition labelled with α cannot exist, or the target state does not satisfy φ, or iif it exists and satisfies φ, it never can be executed.

Definition 6.16:
Let $c =< CS, [min, max] >$ be a constraint. Then we call *Obligatoriness condition* over c the expression $min = max \wedge max = k$ for some $k \in \{0, 1\}$. We call $min = max$

Operator condition, denoted by OP_c and $max = k$ *Action Condition*, denoted by $ACTION_c$

The negation for an operation condition OP_c of a constraint c is denoted by $\overline{OP_c}$ and it is equal to OP_c, whereas the negation for an action condition ACT_c of a constraint c is denoted by $\overline{ACT_c}$ and it is equal to $\neg ACT_c$.

The negation for an obligatoriness condition $Cond_c$ of a constraint c is denoted by $\overline{Cond_c}$ and it is defined in the following way:

$$\overline{Cond_c} = \overline{OP_c \wedge ACTION_c} = \overline{OP_c} \wedge \overline{ACTION_c} = OP_c \wedge \overline{ACTION_c}$$

■

Moreover, note that $\overline{\overline{Cond_c}} = Cond_c$

Now we define a predicate which describes the semantics of $\mathbb{O}\alpha$ and $\mathbb{O}\neg\alpha$ and call it $OBB(c)$. Taken a constraint c, $OBB(c) = Cond_c$

The negation concept for $OBB(c)$ exploits the negation of obligatoriness condition, in particular $\neg OBB(c) = \overline{Cond_c}$.

The next step is defined a predicate to describe $\mathbb{O}\alpha(\varphi)$ and we call it *DeonticObb*. Taken a state s, a satisfaction relation $\models$ and the deontic formula $\alpha(\varphi)$, then it is inductively defined by:

- $DeonticObb(s, \models, \alpha(\varphi)) = \exists c \in \mathfrak{C}.\ OBB(c) \wedge s, c \models \alpha(\varphi)$
- $DeonticObb(s, \models, \neg\delta) = \neg(DeonticObb(s, \models, \delta)$

Definition 6.17:
Let $M = (S, \Sigma, \longrightarrow, \mathfrak{C}, s_0)$ be a fully described action CMTS, s be a state and ϕ be a state formula. The satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ for state formulae is defined by:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s'.\ s \xrightarrow{\alpha} s' \wedge s' \models \varphi$
- $s \models [\alpha]\varphi \Leftrightarrow \forall s'.\ s \xrightarrow{\alpha} s' \Rightarrow s' \models \varphi$
- $s \models \mathbb{O}\delta \Leftrightarrow DeonticObb(s, \models, \delta)$
- $s \models \mathbb{P}\delta \Leftrightarrow \neg DeonticObb(s, \models, \neg\delta)$
- $s \models \mathbb{F}\delta \Leftrightarrow DeonticObb(s, \models, \neg\delta)$
- $s \models \exists\pi \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \pi$

- $s \models \forall\pi \Leftrightarrow \forall\sigma \in Path(s).\ \sigma \models \pi$

Let s be a state, c be a constraint and φ be a state formula then the satisfaction relation $\models \subseteq \mathcal{S} \times Constraints(\Sigma \times \mathcal{S}) \times \delta$ for deontic formulae is defined by:

- $s, c \models \mathbf{tt}$
- $s, c \models \alpha(\varphi) \Leftrightarrow Choice(c) = \{(\alpha, s')\} \wedge s' \models \varphi$
- $s, c \models \neg\delta \Leftrightarrow s, c \not\models \alpha(\varphi)$

Let σ be a path and φ, φ_1 be two state formulae then the satisfaction relation $\models \subseteq Path \times \pi$ for path formulae is defined by:

- $\sigma \models \varphi\ U\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \varphi_1\ \wedge \forall 0 \leq i < j.\ \sigma[i] \models \varphi$

■

As we can see, from formalization pointview this change introduces a few complication, but from computation pointview we improve the computational cost, because now we must to check only outgoing transitions and the deontic axioms hold.

Finally, note that if we expand previous definition, we derive:

- $\mathbb{O}a(\varphi) = \exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge s' \models \varphi \wedge (max = min \wedge max = 1)$
- $\mathbb{F}a(\varphi) = \forall c \in \mathfrak{C}(s).\ Choice(c) \neq \{(\alpha, s')\} \vee s' \not\models \varphi \vee (max = min \wedge max \neq 1)$
- $\mathbb{P}a(\varphi) = \exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge s' \models \varphi \wedge (max \neq min \vee max = 1)$

In the CMTS fully described action, we can derive that:

- $\mathbb{O}a(\varphi)$ describes exactly the concept of obligatoriness
- $\mathbb{F}a(\varphi)$ is equal to $\forall c \in \mathfrak{C}(s).\ Choice(c) \neq \{(\alpha, s')\} \vee s' \not\models \varphi$, seeing that no possible constraint has a cardinality such that $min = max$ and $max \neq 1$. It is describes exactly the prohibition.
- $\mathbb{P}a(\varphi)$ is equal to $\exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\alpha, s')\} \wedge s' \models \varphi$, seeing that any possible constraint has a cardinality $min \leq max$ or $max = 1$. It is describes exactly the permission.

6.2 Logic for CMTS($\mathcal{G}_\mathcal{T}$)

The introduction of guards in the transitions does not add new special properties, of course now the interesting transitions are the ones with guard equals to $\mathbf{tt}$, hence in the general semantics of DHMUL the only changes are the following:

- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s'.\ s \xrightarrow{\mathbf{tt}\to\alpha} s' \wedge s' \models \varphi$

- $s \models [\alpha]\varphi \Leftrightarrow \forall s'.\ s \xrightarrow{\mathbf{tt}\to\alpha} s' \Rightarrow s' \models \varphi$
- $s \models \mathbb{O}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{P}\delta \Leftrightarrow \exists I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{F}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \not\models \delta$
- $s, I \models \alpha(\varphi) \Leftrightarrow \exists(\mathbf{tt}, \alpha, s') \in Trans(s).\ (\mathbf{tt}, \alpha, s') \in I \wedge s' \models \varphi$

Note that for deontic operators, we consider the semantics of constraints $[\![.]\!]$, namely the semantics where for each element, the transition with guards equal to **ff** is deleted.

The next step is to understand how the fully described action CMTSs change. In this context we have:

Definition 6.18:
Let M be a CMTS($\mathcal{G}_\mathcal{T}$), s be a state and α be an action. Then we say that α is a forbidden action if and only if $\forall s'.\ \forall I \in [\![s]\!].\ (\mathbf{tt}, \alpha, s') \notin I$.

■

Definition 6.19:
Let M be a CMTS($\mathcal{G}_\mathcal{T}$), s be a state. Then we say that s is forbidden action free if $\forall \alpha.\ \exists s'.\ (\mathbf{tt}, \alpha, s') \in Trans(s)$ then α is not forbidden.

We say that M is forbidden action if and only if each state s is forbidden action free.

■

In this case the algorithm to derive a CMTS($\mathcal{G}_\mathcal{T}$) forbidden action free from a generic CMTS($\mathcal{G}_\mathcal{T}$) is slightly complicated, in effect we cannot simply delete all transitions with guard equals to **ff**, because for definition of CMTS($\mathcal{G}_\mathcal{T}$) for each transition with guard **ff** we also have a similar transition but with guard **tt** and, hence, the action related to these transition might be present. On the other hand, a transition t with guard **ff** and a constraint with choice set singleton equals to t and cardinality equals to $[1, 1]$ describe a forbidden action, because we must choose, for each possible solution, the transition t and then it is lost when the LTS($\mathcal{G}_\mathcal{T}$) is transformed in LTS. This algorithm can be derived anyway but it is more complicated than the one for CMTS.

***Corollary* 6.3:** *Let $M = (\mathcal{S}, \Sigma, \mathcal{G}_\mathcal{T}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_\mathcal{T}$) forbidden action free then the following property holds:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma.\ \alpha \textit{ is a forbidden action if and only if } \forall s'.\ (\mathbf{tt}, \alpha, s') \notin Trans(s)$$

Definition 6.20:
Let M be a CMTS($\mathcal{G}_\mathcal{T}$), s be a state and α be an action. Then we say that α is an obligatory action if and only if $\forall I \in [\![s]\!].\ \exists s'.\ (\mathbf{tt}, \alpha, s') \in I$.

■

Definition 6.21:
Let M be a CMTS($\mathcal{G}_\mathcal{T}$), s be a state. Then we say that s is a hidden must action free if α is an obligatory action then $\exists s'.\ \exists c \in \mathfrak{C}(s).\ Choice(c) = \{(\mathbf{tt}, \alpha, s')\} \wedge Card(c) = [1,1]$.

We say that M is a hidden must action free if and only if each state s is hidden must action free.

■

In this case the algorithm to derive a hidden must action free CMTS($\mathcal{G}_\mathcal{T}$) from a generic CMTS is very similar to the one for CMTS.

***Corollary* 6.4:** *Let $M = (\mathcal{S}, \Sigma, \mathcal{G}_\mathcal{T}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_\mathcal{T}$) hidden must action free then the following property holds:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma.\ \alpha \text{ is an obligatory action if and only if } \exists c \in \mathfrak{C}(s).\ \exists s' \in \mathcal{S}.\ Choice(c) = \{(\mathbf{tt}, \alpha, s')\} \wedge Card(c) = [1,1]$$

As for the CMTS the following two theorems hold:

Theorem 6.5. *Let $M = (\mathcal{S}, \Sigma, \mathcal{G}_\mathcal{T}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_\mathcal{T}$) hidden must action free and forbidden action free. Then the following property holds:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma.\ \alpha \text{ is a permitted but not obligatory action if and only if } \exists c \in \mathfrak{C}(s).\ \exists s' \in \mathcal{S}.\ Choice(c) = \{(\mathbf{tt}, \alpha, s')\} \wedge Card(c) \neq [1,1]$$

Theorem 6.6. *Let $M = (\mathcal{S}, \Sigma, \mathcal{G}_\mathcal{T}, \longrightarrow, \mathfrak{C}, s_0)$ be a CMTS($\mathcal{G}_\mathcal{T}$) hidden must action free and forbidden action free. Then the following property holds:*

$$\forall s \in \mathcal{S}.\ \forall \alpha \in \Sigma.\ \alpha \text{ is a permitted if and only if } \exists c \in \mathfrak{C}(s).\ \exists s' \in \mathcal{S}.\ Choice(c) = \{(\mathbf{tt}, \alpha, s')\} \wedge Card(c) \neq [1,1]$$

Trivially, the general semantics of deontic operators changes because now an action α is obligatory if a transition t with label α exists, a constraint with a singleton choice set equals to t exists and its cardinality is equal to $[1,1]$, in addition this transition must have $\mathbf{tt}$ as guard.

For our optimized logic, the interpretation is modified but in a simple way: clearly, no change is directly needed for deontic operators, the only modification concerns the way of defining the satisfaction relation about the deontic formula, indeed now a constraint c satisfies a formula $\alpha(\varphi)$ if and only if $Choice(c) = \{(g, \alpha, s')\} \wedge s' \models \varphi \wedge g = \mathbf{tt}$.

Hence, the second optimized version of logic for fully described action CMTS ($\mathcal{G}_\mathcal{T}$), compared to the one for fully described action CMTS, requires only one change:

$$s, c \models \alpha(\varphi) \Leftrightarrow Choice(c) = \{(g, \alpha, s')\} \wedge s' \models \varphi \wedge g = \mathbf{tt}$$

6.3 Logic for CMTS($\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}}$)

The last logic is the extension of DHMUL to CMTS($\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}}$).

First of all, we can observe that in this case we have parametric guarded constraints and, hence, it is interesting to develop a logic which takes into account of parameters and to allow to verify properties based on parameters and assignments.

Of course, the set of considered parameters is $\mathcal{Q}$, namely the set related to the CMTS($\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}}$). An interesting type of properties to be checked is the one where we want to check if properties must be true for all or some possible assignment of parameters. The simple idea is, taken a generic assignment A, we verify typical DHMUL formulae over CMTS($\mathcal{G}_{\mathcal{T}}$), derived by the initial CMTS($\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}}$) and the assignment A.

Hence, the new logic is an extension of the DHMUL, which simply introduces parametric formulae and we call this logic **PDHMUL** (Parametric DHMUL).

Definition 6.22 (**Syntax of PDHMUL**)**:**
A correct PDHMUL formula can be defined according to the following grammar:

$$\begin{aligned}
&\phi ::= p.\phi \mid \overline{p}.\phi \mid \rho \\
&\rho ::= \forall^P \ \varphi \mid \exists^P \varphi \\
&\varphi ::= \mathbf{tt} \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \mathbb{O}\delta \mid \mathbb{F}\delta \mid \mathbb{P}\delta \mid \exists\pi \mid \forall\pi \\
&\delta ::= \mathbf{tt} \mid \neg\delta \mid a(\varphi) \\
&\pi ::= \varphi \ U \varphi_1
\end{aligned}$$

where α is a label describing an action, p is a parameter of $\mathcal{Q}$

The formulae derived from:

- ϕ is called *parameter formulae*
- ρ is called *assignment formulae*
- φ is called *state formulae*
- δ is called *deontic formulae*
- π is called *path formulae*

■

The intended meaning of parameter formulae and assignment formulae is the obvious one.

Taken a parameter formula ϕ then:

- $p.\phi$ means " *defined* $p = \mathbf{tt}$ *then* ϕ *holds*"
- $\overline{p}.\phi$ means " *defined* $p = \mathbf{ff}$ *then* ϕ *holds*"

Consider an assignment formula then:

- $\forall^P \varphi$ means " *for any possible assignment,* φ *holds*"
- $\exists^P \varphi$ means " *exists a possible assignment such that* φ *holds*"

Note that the formula $p.\ q.\ \overline{r}.\ \forall^P \varphi$ is interpreted as "fixed $p = q = \mathbf{tt}$ and $r = \mathbf{ff}$ then for all possible assignment A, φ holds". Note that each assignment must be defined in such a way that $\{p, q\} \subseteq A$ and $r \notin A$ hold.

First of all, we define some useful concept to handle the parameters:

Definition 6.23 (**Condition Set**)**:**
Let $\mathcal{Q}$ be a set of parameters, we call $C = \{p \mid p \in \mathcal{Q}\} \cup \{\overline{p} \mid p \in \mathcal{Q}\}$ a condition set.

We denote by $\mathcal{COND}$ the set of all possible condition set. ■

Trivially, a condition set is a possible subset of parameters, that describes which parameters assume value **tt** and which ones assume value **ff**.

Definition 6.24 (**Satisfiability of a condition set**)**:**
Let $\mathcal{Q}$ be a set of parameters, C be a condition set and $A \subseteq \mathcal{Q}$ be an assignment. Then we say that

$$A \models C \Leftrightarrow \forall p \in C.\ p \in A \wedge \forall \overline{p} \in C.\ p \notin A$$

■

Definition 6.25:
Let $\mathcal{Q}$ be a set of parameters, M be a $CMTS(\mathcal{G}_T, \mathcal{G}^{\mathcal{Q}})$, s be a state of M and A be an assignment.

Then we denote by $\sigma(s, A)$ the state s in the $CMTS(\mathcal{G}_T)$ $N = \sigma(M, A)$.

■

Definition 6.26:
Let $M = (S, \Sigma, \mathcal{P}, \mathcal{G}_T, \mathcal{G}^{\mathcal{Q}}, \longrightarrow, \mathfrak{C}, s_0)$ be a $CMTS(\mathcal{G}_T, \mathcal{G}^{\mathcal{Q}})$, s be a state, C be a condition set.

Then the satisfaction relation $\models \subseteq \mathcal{S} \times \mathcal{COND} \times \varphi$ for parameter formula is defined by:

- $s, C \models p.\ \phi \Leftrightarrow s, (C \cup p) \models \phi$
- $s, C \models \overline{p}.\ \phi \Leftrightarrow s, (C \cup \overline{p}) \models \phi$
- $s, C \models \rho \Leftrightarrow s, C \models \rho$

The satisfaction relation $\models \subseteq \mathcal{S} \times \mathcal{COND} \times \rho$ for an assignment formula is defined by:

- $s, C \models \forall^P\ \varphi \Leftrightarrow \forall A \subseteq \mathcal{P}. A \models C \Rightarrow\ \sigma(s, A) \models \varphi$

- $s, C \models \exists^P \varphi \Leftrightarrow \exists A \subseteq \mathcal{P}.A \models C \wedge \sigma(s, A) \models \varphi$

The satisfaction relation $\models \subseteq \mathcal{S} \times \varphi$ for a state formula is defined by:

- $s \models \mathbf{tt}$
- $s \models \neg\varphi \Leftrightarrow s \not\models \varphi$
- $s \models \varphi \wedge \varphi_1 \Leftrightarrow s \models \varphi$ and $s \models \varphi_1$
- $s \models \langle\alpha\rangle\varphi \Leftrightarrow \exists s'.\ s \xrightarrow{\alpha} s' \wedge s' \models \varphi$
- $s \models [\alpha]\varphi \Leftrightarrow \forall s'.\ s \xrightarrow{\alpha} s' \Rightarrow s' \models \varphi$
- $s \models \mathbb{O}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{P}\delta \Leftrightarrow \exists I \in [\![s]\!].\ s, I \models \delta$
- $s \models \mathbb{F}\delta \Leftrightarrow \forall I \in [\![s]\!].\ s, I \not\models \delta$
- $s \models \exists\pi \Leftrightarrow \exists\sigma \in Path(s).\ \sigma \models \pi$
- $s \models \forall\pi \Leftrightarrow \forall\sigma \in Path(s).\ \sigma \models \pi$

Let I be a set of outgoing transitions of s, then the satisfaction relation $\models \subseteq \mathcal{P}(\Sigma \times \mathcal{S}) \times \delta$ for a deontic formula is defined by:

- $s, I \models tt$
- $s, I \models \neg\delta \Leftrightarrow s, I \not\models \delta$
- $s, I \models \alpha(\varphi) \Leftrightarrow \exists(\alpha, s') \in Trans(s).\ (s, \alpha, s') \in I \wedge s' \models \varphi$

Let σ be a path of s, then the satisfaction relation $\models \subseteq Path \times \pi$ for a path formula is defined by:

- $\sigma \models \varphi\ U\varphi_1 \Leftrightarrow \exists j \geq 0.\ \sigma[j] \models \varphi_1\ \wedge \forall 0 \leq i < j.\ \sigma[i] \models \varphi$

■

Despite of all possible optimization described in the previous section, in this case, this logic has an high computational cost because we must generate several possible assignments.

We want to highlight that, for each assignment $A \subseteq \mathcal{Q}$, we derive a $CMTS(\mathcal{G}_T)$ N from M and then we must verify properties over N and, unfortunately, the set of all possible $CMTS(\mathcal{G}_T)$ can be very large.

In effect, if we suppose that the computational cost of a single formula φ over a $CMTS(\mathcal{G}_T)$ equals to K_φ, the set of our parameters is $\mathcal{Q}$ and its size is $N_\mathcal{Q}$

and for each formula ρ we define an initial set of parameters of size N_P then the computational cost of $\rho = \forall^P \varphi$ or $\rho = \exists^P \varphi$ is equivalent to $2^{N_{\mathcal{Q}} - N_P} * K_\varphi$

Trivially, this is a very high computational cost. Of course, once we determine a CMTS($\mathcal{G}_T$) from an assignment A, then we can verify properties using the optimization previously seen, nevertheless the transformation from the CMTS($\mathcal{G}_T$) to the CMTS($\mathcal{G}_T$) fully described action must be computed for each new derived CMTS($\mathcal{G}_T$). Again, we have the cost of transformation repeated for each possible assignment.

Chapter 7

Conclusion

In this thesis, we have presented a new specification formalism deriving by the Generalized Extended Modal Transition System (GEMTS). The Constrained Modal Transition System (CMTS) revisits the GEMTS by defining the same concept but in a different way, namely introducing the concept of constraint over outgoing transitions. By means of a constraint we can decide how many transitions must be present in a correct implementation, moreover this new approach allow us to find out some useful properties of the model in a more direct way.

In addition, we have defined a concept of refinement step-by-step, which is absent in the GEMTS formalism and we have presented two different types of refinement: one slightly more semantic, which uses the concept of semantics of constraints, the other one more syntactic, which uses the syntactic concept of constraints and we have also described how we can refine a constraint in another one in a correct way.

For convenience, it has been assumed that the CMTS is action-deterministic, hence in the Chapter 3 we have shortly described as a CMTS can change in a non-deterministic context and finally we have studied the problem of minimalization, namely if taken a CMTS M, it is possible to find a CMTS N semantically equivalent but with a reduced number of constraints. Furthermore, an initial definition of parallel composition is described in the Appendix.

Thereafter we have introduced some further extensions of the CMTS formalism, introducing guards both in transition and constraint concepts.

Hence, by using CMTS and its extensions we have compared some different specification formalisms, which can be found in literature: **LTS, MTS, DMTS, 1MTS, GEMTS, OTS** and **PMTS**, deriving a hierarchy of expressivity of all these models, considered in a context of action-determinism.

Finally, we have introduced a new modal logic, which has both deontic and temporal operator, and it allows to verify both typical properties of CTL and properties like "it is obligatory that", or "it is permitted that", or "it is forbidden that", namely typical deontic property. In addition we have described how the CMTS can be improved in order to decrease the computational cost. Then this logic has been extended to CMTS extensions, deriving a deontic-temporal logic with parameters.

7.1 Future work

As for future work, there are many possibilities. One possibility is to study and to understand if it is possible to determine the exact minimal number of constraint for each possible CMTS M, because our theorem is only an existential theorem. Moreover, we might define some useful algorithms to compute the consistency of a CMTS and how to derive the minimal CMTS from a generic one, for example by means of the use in a smart way of constraints. On the other hand, it is interesting to study how the CMTS, its extensions and the derived hierarchy change by deleting of action-determinism requirement. In particular if this non-determinism is useful and how to handle constraints with some multiple transition with the same label. Note that the most part of definitions allow to handle the non-determinism context and we have rarely used to the action-determinism requirement in view of a possible general extension.

Another possibility is to determine if other extensions of CMTS are possible or if it is interesting and useful to introduce some further conditions over constraints and transitions.

Moreover, it is possible to study the complexity problem about the refinement introduced for CMTSs and possibly compare these results to the ones related to the PMTS and OTS formalisms. The study of complexity can be a useful tool to find some other special properties or some useful extension/restriction of CMTS which reduces the possible computational costs.

Another interesting topic is the one related to the logic introduced in Chapter 6, in effect we might study in a deeper way some useful properties of the logic itself or of the logic related to the refinement. On the other hand, it is interesting to understand if the optimization which we have described can be further extended or improved, in particular in order to derive some less-expensive algorithms to check logic formulae and to handle, in a smart way, the logic with parameters. Again, it is possible to try to realize how the interpretation of logic formulae, in particular the deontic ones, changes with the introduction of non-determinism and if this change is significant or not.

Last but not least, it is possible to develop these concepts from a practical point view and, hence, to define some algorithms and/or programs which allow to verify properties and/or to describe a CMTS and derive its implementations in a simple way in order to understand in which real contexts these models are useful or are too much expressive and what lacks they have for an effective utilization in a company/product-family context.

Appendix A

Refinement Properties

All refinement definitions which we have seen are preorders, namely they have a reflexive and transition property, so it is interesting to know if semantic and syntactic modal refinements are preorders too.

Theorem A.1. *The semantic modal refinement is a preorder.*

Proof.
Reflexive: taken a state s, we must demonstrate that $(s, s) \in \mathcal{R}$, for some semantic refinement relation $\mathcal{R}$. Now we suppose $\mathcal{R} = \{(s, s) \mid s \in \mathcal{S}\}$ then we try to see if R is a semantic refinement relation.

Trivially, the first condition of $\mathcal{R}$ is satisfied. In addition for the second $[\![\mathfrak{C}(s)]\!] \sqsubseteq_{\mathcal{R}} [\![\mathfrak{C}(s)]\!]$ holds. In effect, taken a set of transitions $I \in [\![\mathfrak{C}(s)]\!]$ then $\exists J \in [\![\mathfrak{C}(s)]\!]$ such that $I \sqsubseteq_{\mathcal{R}} J$, simply it is sufficient to consider J such that $J = I$.
Transitive: let s_0, s_1, s_2 be three states such that $(s_0, s_1) \in \mathcal{R}_1$ and $(s_1, s_2) \in \mathcal{R}_2$, we must demonstrate that $(s_0, s_2) \in \mathcal{R}$, for some semantic refinement relation $\mathcal{R}$. We suppose $\mathcal{R} = \{(s, s_2) \mid \exists s_1.\ (s, s_1) \in \mathcal{R}_1 \ \wedge (s_1, s_2) \in \mathcal{R}_2\}$.

The first condition is simple, we know:

- $s_0 \xrightarrow{\alpha} s_0' \Rightarrow s_1 \xrightarrow{\alpha} s_1' \wedge (s_0', s_1') \in \mathcal{R}_1$
- $s_1 \xrightarrow{\alpha} s_1' \Rightarrow s_2 \xrightarrow{\alpha} s_2' \wedge (s_1', s_2') \in \mathcal{R}_2$

It is simple to understand that $s_0 \xrightarrow{\alpha} s_0' \Rightarrow s_2 \xrightarrow{\alpha} s_2' \wedge (s_0', s_2') \in \mathcal{R}$.

Now we consider the second condition: if $(s_0, s_1) \in \mathcal{R}_1$ then $\forall I \in [\![s_0]\!].\ \exists J \in [\![s_1]\!].\ I \sqsubseteq_{\mathcal{R}_1} J$. Moreover if $(s_1, s_2) \in \mathcal{R}_2$ then $\forall J \in [\![s_1]\!].\ \exists K \in [\![s_2]\!].\ J \sqsubseteq_{\mathcal{R}_2} K$.

Therefore, taken $I \in [\![s_0]\!]$, we can derive $J \in [\![s_1]\!]$ and so $K \in [\![s_2]\!]$. We need to understand if $I \sqsubseteq_{\mathcal{R}} K$, namely $\forall(\alpha, s') \in I.\ \exists(\alpha, t') \in K.\ (s', t') \in \mathcal{R}$.

We consider a generic $(\alpha, s') \in I$ then we know that $(\alpha, s_J') \in J$ and $(s', s_J') \in \mathcal{R}_1$, for hypothesis. In addition, for hypothesis, $(\alpha, s_J') \in J$ then $(\alpha, t') \in K$ such that $(s_J', t') \in \mathcal{R}_2$, so $(s', t') \in \mathcal{R}$, deducing $\forall(\alpha, s') \in I.\ \exists(\alpha, t') \in K.\ (s', t') \in \mathcal{R}$. □

Theorem A.2. *The syntactic modal refinement is a preorder.*

Proof.
Reflexive: taken a state s, we must demonstrate that $(s, s) \in \mathcal{R}$, for some syntactic refinement relation $\mathcal{R}$. Suppose $\mathcal{R} = \{(s, s) \mid s \in \mathcal{S}\}$ then we try to see if $\mathcal{R}$ is a syntactic refinement relation.

Trivially, the first condition of $\mathcal{R}$ is satisfied. In addition for the second, $\forall c \in \mathfrak{C}(s)$ we can find a $c' \in \mathfrak{C}(s)$ such that $c \trianglelefteq_{\mathcal{R}} c'$ and $(Label(c) \setminus Label(c')) \cap Label(s) = \emptyset$. In effect it is sufficient to consider $c' = c$.
Transitive: let s_0, s_1, s_2 be three states such that $(s_0, s_1) \in \mathcal{R}_1$ and $(s_1, s_2) \in \mathcal{R}_2$, we must demonstrate that $(s_0, s_2) \in \mathcal{R}$, for some syntactic refinement relation $\mathcal{R}$. We suppose $\mathcal{R} = \{(s, s_2) \mid \exists s_1.\ (s, s_1) \in \mathcal{R}_1 \ \wedge (s_1, s_2) \in \mathcal{R}_2\}$.

The first condition is simple, in effect we know:

- $s_0 \xrightarrow{\alpha} s_0' \Rightarrow s_1 \xrightarrow{\alpha} s_1' \wedge (s_0', s_1') \in \mathcal{R}_1$
- $s_1 \xrightarrow{\alpha} s_1' \Rightarrow s_2 \xrightarrow{\alpha} s_2' \wedge (s_1', s_2') \in \mathcal{R}_2$

Again it is simple to understand that $s_0 \xrightarrow{\alpha} s_0' \Rightarrow s_2 \xrightarrow{\alpha} s_2' \wedge (s_0', s_2') \in \mathcal{R}$.

As we saw in other previous theorems, we suppose that if it exists a constraint $c =< CS, [min, max] >$ such that $min = 0$ and it does not exist a refined constraint of c, then this is equivalent to have a constraint $c' =< CS', [0, 0] >$ where $c' \trianglelefteq c$. Therefore we can modify the second refinement condition in this way: $\forall c_t \in \mathfrak{C}_N(t).\ \exists c_s \in \mathfrak{C}_M(s)$ such that: $c_s \trianglelefteq_{\mathcal{R}} c_t$ and $(Label(c_t) \setminus Label(c_s)) \cap Label(s) = \emptyset$.

Since $(s_0, s_1) \in \mathcal{R}_1$ then $\forall c_0 \in \mathfrak{C}(s_0).\ \exists c_1 \in \mathfrak{C}(s_1)$ such that: $c_0 \trianglelefteq_{\mathcal{R}_1} c_1$ and $(Label(c_1) \setminus Label(c_0)) \cap Label(s_0) = \emptyset$. Moreover if $(s_1, s_2) \in \mathcal{R}_2$ then $\forall c_1 \in \mathfrak{C}(s_1).\ \exists c_2 \in \mathfrak{C}(s_2)$ such that: $c_1 \trianglelefteq_{\mathcal{R}_2} c_2$ and $(Label(c_2) \setminus Label(c_1)) \cap Label(s_1) = \emptyset$.

Now consider $c_0 =< CS_0, [min_0, max_0] >, c_1 =< CS_1, [min_1, max_1] >$ and $c_2 =< CS_2, [min_2, max_2] >$. Trivially all constraints are correct.

Since $c_0 \trianglelefteq_{\mathcal{R}_1} c_1$ then $min_1 \leq min_0 \leq max_0 \leq max_1$, in addition seeing that $c_1 \trianglelefteq_{\mathcal{R}_2} c_2$ then $min_2 \leq min_1 \leq max_1 \leq max_2$. It is simple to conclude that $min_2 \leq min_0 \leq max_0 \leq max_2$.

Furthermore for $c_0 \trianglelefteq_{\mathcal{R}_1} c_1$ we have that $\forall(\alpha, s_0') \in CS_0.\ \exists(\alpha, s_1') \in CS_1 \wedge (s_0', s_1') \in \mathcal{R}_1$ and the same holds for $c_1 \trianglelefteq_{\mathcal{R}_2} c_2$. So we can deduce that $\forall(\alpha, s_0') \in CS_0.\ \exists(\alpha, s_1') \in CS_1 \wedge \exists(\alpha, s_2') \in CS_2.\ (s_0', s_1') \in \mathcal{R}_1 \wedge (s_1', s_2') \in \mathcal{R}_2$, concluding $\forall(\alpha, s_0') \in CS_0.\ \exists(\alpha, s_2') \in CS_2.\ (s_0', s_2') \in \mathcal{R}$.

The last thing to be demonstrated is $(Label(c_2) \setminus Label(c_0)) \cap Label(s_0) = \emptyset$. First of all note that if $(s_0, s_1) \in \mathcal{R}_1$, for definition of syntactic modal refinement, $Label(s_0) \subseteq Label(s_1)$ and the same holds for (s_1, s_2), hence we can deduce $Label(s_0) \subseteq Label(s_2)$. From set theory we know that $S \setminus S_1$ is equivalent to say that for each element x, $x \in S \wedge x \notin S_1$ holds. In addition we know:

1. $(Label(c_1) \setminus Label(c_0)) \cap Label(s_0) = \emptyset$ is true, namely $\forall\alpha.\ \alpha \in Label(c_1) \wedge \alpha \notin Label(c_0) \Rightarrow \alpha \notin Label(s_0)$
2. $(Label(c_2) \setminus Label(c_1)) \cap Label(s_1) = \emptyset$ and $Label(s_0) \subseteq Label(s_1)$, so we can say $(Label(c_2) \setminus Label(c_1)) \cap Label(s_0) = \emptyset$ is true, hence $\forall\alpha.\ \alpha \in Label(c_2) \wedge \alpha \notin Label(c_1) \Rightarrow \alpha \notin Label(s_0)$

It is simple to deduce that taken $\alpha \in \Sigma$ then $\alpha \in Label(c_1) \vee \alpha \notin Label(c_1)$. Therefore if $\alpha \in Label(c_2) \wedge \alpha \notin Label(c_0)$ we have two possibilities: if $\alpha \in Label(c_1)$ then surely $\alpha \notin Label(s_0)$, otherwise $\alpha \notin Label(c_1)$ again surely $\alpha \notin Label(s_0)$.

In conclusion we can say that $(Label(c_2) \setminus Label(c_0)) \cap Label(s_0)$ is always true. $\square$

Appendix B

Theorems and proofs of Chapter 5

In these theorems and proofs we implicitly assume that each model is action-deterministic.

Theorem B.1. *The formalism* $\mathbb{MTS}$ *is less expressive of a* $1\mathbb{MTS}$*, namely* $\mathbb{MTS} \rightsquigarrow 1\mathbb{MTS}$

Proof.
First of all, we know that, taken a MTS $L = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_L}, \longrightarrow_{\Box_L})$, L is equivalent to a 1MTS $M = (\mathcal{S}, \Sigma, \longrightarrow_{\Diamond_M}, \longrightarrow_{\Box_M})$ such that:

- $\forall s \in \mathcal{S}.\ (s, U) \in \longrightarrow_{\Diamond_M} \ \wedge |U| = 1$
- $\forall s \in \mathcal{S}.\ \forall (\alpha, s') \in \Sigma \times \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow_{\Diamond_L} \Leftrightarrow \exists (s, U) \in \longrightarrow_{\Diamond_M} .U = \{(\alpha, s')\}$
- $\forall s \in \mathcal{S}.\ \forall (\alpha, s') \in \Sigma \times \mathcal{S}.\ (s, \alpha, s') \in \longrightarrow_{\Box_L} \Leftrightarrow \exists (s, U) \in \longrightarrow_{\Box_M} .U = \{(\alpha, s')\}$

Moreover no possible MTS can describe the 1MTS in Figure B.1. Suppose that a MTS L which describes the 1MTS M in Figure B.1 exists then we can deduce that transitions $(a, s_1), (b, s_2)$ are may transitions because they must not be always present. If it is true then the LTS with no transitions is correct for L but is wrong for M, because as we can see in Figure B.1 all possible LTSs must have at least one transition. On the other hand, if we suppose that (a, s_1) is the must transition in L then the LTS J in Figure B.1 is wrong for L and the same reasoning is true if we suppose (b, s_2) as must transition or if we suppose that all transitions are must.

Finally, we can deduce that this hypothetical MTS does not exist. □

Theorem B.2. *The formalism* $\mathbb{DMTS}$ *and* $1\mathbb{MTS}$ *are not comparable, namely* $1\mathbb{MTS} \not\rightsquigarrow \mathbb{DMTS}$ *and* $\mathbb{DMTS} \not\rightsquigarrow 1\mathbb{MTS}$.

Proof.
No possible DMTS can describe the 1MTS in Figure B.1. Suppose that a DMTS L which describes the 1MTS M in Figure B.1 exists then we can deduce that transitions $(a, s_1), (b, s_2)$ are may transitions because they must not be always present.

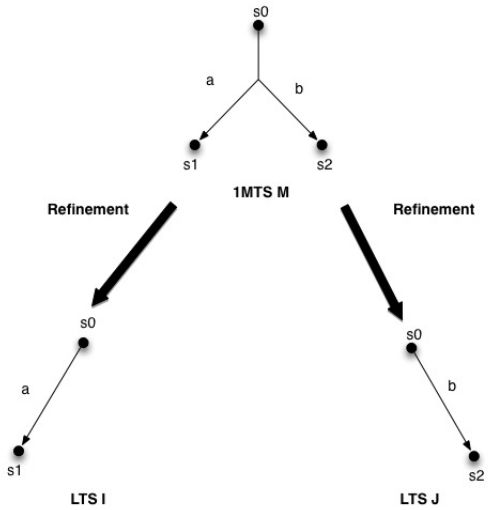


Figure B.1: An example of 1MTS and its derived LTSs

If we have not a must hypertransition then the LTS with no transitions is correct for L but is wrong for M. If we introduce the must hypertransition we have three possibilities:

1. we add only one must hypertransition $(s, \{(a, s_1)\})$
2. we add only one must hypertransition $(s, \{(b, s_2)\})$
3. we add only one must hypertransition $(s, \{(a, s_1), (b, s_2)\})$

Of course, we can combine these cases, adding more hypertransitions but these three cases are the base ones. In the first case the LTS K is never derivable, in the second one the LTS J is never derivable, whereas in the third case J and K are derivable but, unfortunately, we can derive another LTS with both transitions and this is obviously wrong.

Finally, we can deduce that this hypothetical DMTS does not exist.

On the other hand, no possible 1MTS can describe the DMTS in Figure 5.10. Suppose that a 1MTS L which describes the DMTS M in Figure 5.10 exists. We have some possibilities:

- we have two may singleton hypertransitions, one for (a, s_1) and one for (b, s_2)
- we have a may hypertransition which considers both (a, s_1) and (b, s_2)

In each case we can derive the LTS wit no-transitions and it is impossible for M. Hence, we need of some must transition but again we have two possibilities:

- we have two must singleton hypertransitions, one for (a, s_1) and one for (b, s_2)
- we have a must hypertransition which considers both (a, s_1) and (b, s_2)

In the first case LTSs I and J are impossible for L, in the second case the LTS K is impossible.

The last chance is if we have:

- two may singleton hypertransitions, one for (a, s_1) and one for (b, s_2)
- a may hypertransitions, which considers both (a, s_1) and (b, s_2)
- a must hypertransition which considers both (a, s_1) and (b, s_2)

Now LTS K is possible, because if the must hypertransition choose, for example, (a, s_1) then we have in any case the may hypertransition (b, s_2). In this way K is derivable. But as we said, we must have action-deterministic choice functions and they must be extended for the singleton may hypertransition. Hence, if $\gamma(\{(a, s_1), (b, s_2)\}) = (a, s_1)$ then $\gamma(\{(a, s_1)\}) = (a, s_1)$ and $\gamma(\{b, s_2)\}) = \bot$, deducing that in this case the LTS K is not derivable. □

Theorem B.3. *The formalism* 1MTS *is less expressive of the* CMTS*, namely* 1MTS $\rightsquigarrow$ CMTS

Proof.
Trivially, no possible 1MTS can describe the CMTS in Figure B.2. Suppose that a 1MTS L which describes the CMTS M in Figure B.1 exists then we can deduce that transitions $(a, s_1), (b, s_2), (c, s_3)$ are may transitions because they must not be always present. If we have not a must hypertransition then the LTS with no transitions is correct for L but it is wrong for M. If we introduce the must hypertransition we have three possibilities:

1. we add a single must hypertransition which handles only one transition
2. we add a single must hypertransition which handles only two transitions
3. we add a single must hypertransition which handles all transitions

Of course, we can add more hypertransitions simultaneously but this three cases are the base cases. In the first case some LTSs are not derivable, for example if the must hypertransition has only (a, s_1), then the LTS K is never derivable.

In the second case some possible LTSs are not derived, for example if we consider a hypertransition with $(a, s_1), (b, s_2)$ then the LTS I is impossible.

In the third case if we interpret the must hypertransition with XOR semantics then no derived LTS is possible.

Finally, we can deduce that this hypothetical 1MTS does not exist. □

Theorem B.4. *The formalism* DMTS *is less expressive of the* CMTS*, namely* DMTS $\rightsquigarrow$ CMTS

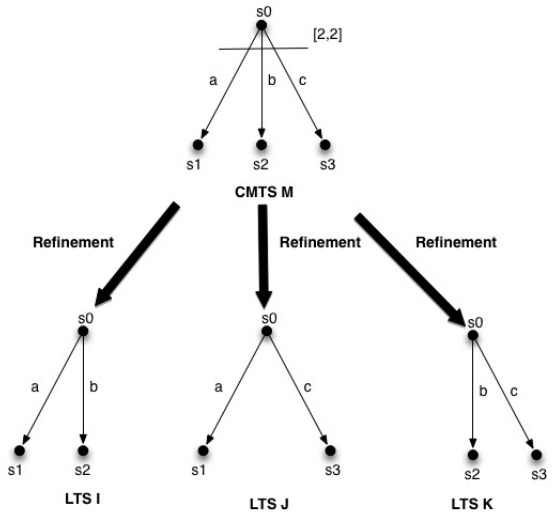


Figure B.2: An example of a CMTS and its derived LTSs

Proof.
Trivially, no possible DMTS can describe the CMTS in Figure B.2. Suppose that a DMTS L which describes the CMTS M in Figure B.1 exists then we can deduce that transitions $(a, s_1), (b, s_2), (c, s_3)$ are may transitions because they must not be always present. If we have not a must hypertransition then the LTS with no transitions is correct for L but it is wrong for M. If we introduce the must hypertransition we have three possibilities:

1. we add a single must hypertransition which handles only one transition
2. we add a single must hypertransition which handles only two transitions
3. we add a single must hypertransition which handles all transitions

Of course, we can add more hypertransitions simultaneously but this three cases are the base cases. In the first case some LTSs are not derivable, for example if the must hypertransition has only (a, s_1), then the LTS K is never derivable.

In the second case we have a must hypertransition with two transitions and, for each transition in the hypertransition, we have a further may transition, then the result depends on the type of remaining transition. If it is may then we can derive a LTS with only one transition and this is wrong. If it is must we can derive a LTS with all transitions and again, this is wrong. For example, if the hypertransition has $(a, s_1), (b, s_2)$ then:

- if (c, s_3) is may then LTS with only (a, s_1) is possible
- if (c, s_3) is must then LTS with $(a, s_1), (b, s_2), (c, s_3)$ is possible

In the third case if we have a must hypertransition with all transitions then a LTS with all transitions is possible and this is obviously wrong.

Finally, we can deduce that this hypothetical DMTS does not exist. □

Now we consider a generic OTS O and a LTS I derived from O. We know that a LTS is a OTS where, for each state, the obligation function is a conjunction of all outgoing transitions.

Suppose to consider a state s_O of O and the corresponding s_I of I then, seeing that I is derived by O, we have that each set of transitions which satisfies the obligation formula of s_I also satisfies the obligation formula of s_O. Trivially, we have only one set of transitions which satisfies the obligation formula of s_I, namely the set of all outgoing transitions and, hence, we deduce that the set of all outgoing transitions of s_I is included in the set of all possible sets of transitions of s_O. We can conclude that each possible LTS I derived from O has the property that, for each state s_I, $Trans(s_I) \in [\![\mathbf{\Omega}(s_O)]\!]$, namely for each state s_I, its set of outgoing transitions is a correct set for the obligation formula of corresponding state s_O.

Conceptually, this is the same idea used to describe that a LTS I, derived from a CMTS M, is composed by states s_I such that its outgoing transitions are solutions of constraints of the corresponding state s_M of M.

In addition, taken two sets of sets of transitions S and S_1, we say that $S \sqsubseteq_{\mathcal{R}} S_1$ if and only if the following property holds:

$$\forall I \in S.\ \exists J \in S_1.\ \forall(\alpha, s') \in I.\ \exists(\alpha, s'_1) \in J \wedge (s', s'_1) \in \mathcal{R} \wedge \\ \forall(\alpha, s'_1) \in J.\ \exists(\alpha, s') \in I \wedge (s', s'_1) \in \mathcal{R}$$

Note that this operator has already been described in Section 2.2.7. We say that $S =_{\mathcal{R}} S_1$ if and only if $S \sqsubseteq_{\mathcal{R}} S_1$ and $S_1 \sqsubseteq_{\mathcal{R}} S$

First of all, we introduce a special construction that we will use in the subsequent three theorems. This construction allow us to determine that the CMTS model can be represented by an OTS* and we call it **construction by semantics**.

Theorem B.5 (**Construction by semantics**). *Let $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{0_M})$ be a CMTS. Then we can deduce an OTS* $O = (\mathcal{S}_O, \Sigma, \longrightarrow_O, \mathbf{\Omega}_M, s_{0_O})$ such that it holds:*

$$[\![[\![M]\!]]\!]_{\text{CMTS}} = [\![[\![O]\!]]\!]_{\text{OTS}^*}$$

Proof.
Suppose to compute the semantics $[\![[\![M]\!]]\!]_{\text{CMTS}}$. Seeing that the CMTS is an action deterministic CMTS, then each computed LTS in the semantics is action-deterministic. Now we try to derive the OTS* O in the following way:

- $\mathcal{S}_O = \mathcal{S}_M$
- $\longrightarrow_O = \longrightarrow_M$

- $\forall s_O \in \mathcal{S}_O$. $\mathbf{\Omega}(s_O)$ is derived in this way: we denote by s_M the corresponding state in M of s_O. If $[\![\mathfrak{C}(s_M)]\!] = \emptyset$ then $\mathbf{\Omega}(s_O) = \mathbf{ff}$, otherwise taken a $I \in [\![\mathfrak{C}(s_M)]\!]$, we denote $NotPres_I = Trans(s_M) \setminus I$.

  Now we define the formula $\varphi_I = \bigwedge_{(\alpha,s'_M)\in I} (\alpha, s'_O) \wedge \bigwedge_{(\alpha,s'_M)\in NotPres_I} \neg(\alpha, s'_O)$.

  The obligation formula $\mathbf{\Omega}(s_O) = \bigvee_{I\in[\![\mathfrak{C}(s_M)]\!]} \varphi_I$.

  Note that this obligation formula is in DNF.

  In addition if $[\![\mathfrak{C}(s_M)]\!] = \mathcal{P}(Trans(s_M))$, namely each possible combination of outgoing transitions of s_M is possible, then our obligation formula is an OR of all possible combinations. Trivially, it is possible to prove that this obligation formula is also equivalent to the obligation formula $\mathbf{tt}$.

Now we must prove the semantic equivalence. Instead of checking if a LTS derived from O is also derived by M and the vice versa too, we can note that in both formalisms a LTS can be derived by the semantics of constraints and obligation formula, hence if we take a generic state s and we prove that $[\![\mathfrak{C}(s_M)]\!] =_{\mathcal{R}} [\![\not\Leftarrow(s_O)$ where $\mathcal{R} = \{(s_M, s_O)\}$ and s_M, s_O describe the same state s in M and O, respectively. In this way we implicitly deduce the semantic equivalence.

Initially, we prove the inconsistency:

- $[\![\mathfrak{C}(s_M)]\!] = \emptyset$, in this case, for construction, we impose that $\mathbf{\Omega}(s_O) = \mathbf{ff}$ and hence $[\![\mathbf{\Omega}(s_O)]\!] = \emptyset$

- $[\![\mathbf{\Omega}(s_O)]\!] = \emptyset$ hence $\mathbf{\Omega}(s_O) = \mathbf{ff}$ and for construction, we can derive that $[\![\mathfrak{C}(s_M)]\!] = \emptyset$

Now we consider the consistent case:

Case 1) $\forall I.\ I \in [\![\mathfrak{C}(s_M)]\!] \Rightarrow I \in [\![\mathbf{\Omega}(s_O)]\!]$.

Suppose $I \in [\![\mathfrak{C}(s_M)]\!]$, then for construction a disjunct φ of $\mathbf{\Omega}(s_O)$ exists such that $I \in [\![\varphi]\!]$. Trivially, we can deduce that $I \in [\![\mathbf{\Omega}(s_O)]\!]$.

Case 2) $\forall I.\ I \in [\![\mathbf{\Omega}(s_O)]\!] \Rightarrow I \in [\![\mathfrak{C}(s_M)]\!]$.

Suppose $I \in [\![\mathbf{\Omega}(s_O)]\!]$, then a disjunct φ of $\mathbf{\Omega}(s_O)$ exists such that $I \in [\![\varphi]\!]$. Trivially, for construction, φ is a conjunction of all possible outgoing transitions of s_O and, hence, of s_M. We can deduce that each set of transitions J satisfying φ holds this property: each positive atom in φ is in J, whereas each negative atom is not in J. Therefore we can deduce that $[\![\varphi]\!]$ has only one possible valid set of transitions, namely I. For construction, a set of transition formed by only positive atoms of φ exists in $[\![\mathfrak{C}(s_M)]\!]$ too, hence $I \in [\![\mathfrak{C}(s_M)]\!]$ must be true. □

Theorem B.6. *The formalism* $\mathbb{CMTS}$ *is less expressive of* $\mathbb{OTS}^*$*, namely* $\mathbb{CMTS} \rightsquigarrow \mathbb{OTS}^*$

Proof.
In Theorem B.5 we prove that for each CMTS is possible to find a semantically equivalent OTS^*. The vice versa is not true. For example, we consider the OTS* in Figure B.3, in this case we can have three different possible constraints:

1. the constraint with choice set $\{(a, s_1)\}$
2. the constraint with choice set $\{(b, s_2)\}$
3. the constraint with choice set $\{(a, s_1), (b, s_2)\}$

In the first and second case we must have the cardinality equals to $[0, 1]$ because (a, s_1) and (b, s_2) can be taken or not, and the same reasoning holds for the third case, deducing that its cardinality is $[0, 2]$. Trivially, the LTS with only (a, s_1) is possible for our CMTS and this is wrong. To solve this problem we can change some cardinality, but each change reduces the semantics of the CMTS in a wrong way. □

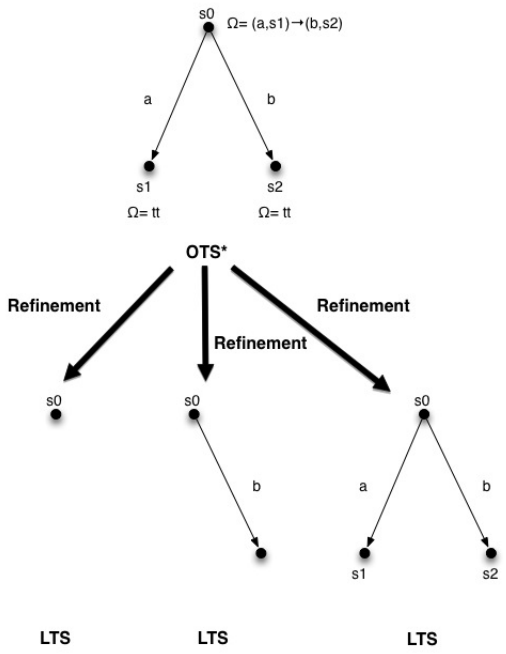


Figure B.3: An example of an OTS* and its derived LTSs

We want to highlight that in some case in the next proof we compare LTS and LTS($\mathcal{G}$) where each transition has guard equals to **tt**. Conceptually, they represent the same model, hence we handle indistinctly a LTS as LTS($\mathcal{G}$) and vice versa.

Lemma B.1. *Let M be a CMTS($\mathcal{G}_\mathcal{T}$) then exists an OTS* O such that $[\![[\![O]\!]]\!]_{\mathbb{OTS}^*} = [\![[\![M]\!]]\!]_{\mathbb{CMTS}_{\mathcal{G}_\mathcal{T}}}$.*

Proof.
In this case we have guards over transitions and, for each state s_M, the semantics of $\mathfrak{C}(s_M)$ is a set of guarded transitions. We can derive that for each state s of a LTS($\mathcal{G}_\mathcal{T}$) derived by M through a refinement relation, the outgoing transitions of s are a solution of $\mathfrak{C}(s_M)$. On the other hand, the semantics of M takes into account only LTSs, which are derived by LTS($\mathcal{G}_\mathcal{T}$) deleting each transition with guard equals to $\mathbf{ff}$. Then we easily derive a new set $[\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}} = \{ExecutableKer(I) \mid I \in [\![\mathfrak{C}(s_M)]\!]\}$.

We use the same construction used for Theorem B.5:

- $\mathcal{S}_O = \mathcal{S}_M$
- $\longrightarrow_O = \{(s_O, \alpha, s'_O) \mid (s_M, g, \alpha, s'_M) \in \longrightarrow_M\}$
- $\forall s_O \in \mathcal{S}_O$. $\mathbf{\Omega}(s_O)$ is derived in this way: we denote by s_M the corresponding state in M of s_O.

  If $[\![\mathfrak{C}(s_M)]\!] = \emptyset$ then $\mathbf{\Omega}(s_O) = \mathbf{ff}$, otherwise taken a $I \in [\![\mathfrak{C}(s_M)]\!]$. We denote $Pres_I = \{(\alpha, s'_O) \mid (\mathbf{tt}, \alpha, s'_M) \in I\}$ and $NotPres_I = Trans(s_O) \setminus Pres_I$. In this case $Pres_I$ represents all possible transitions which is surely present in the LTS derived by LTS($\mathcal{G}_\mathcal{T}$) which are refinable from M.

  Now we define the formula $\varphi_I = \bigwedge_{(\alpha,s'_M)\in Pres_I} (\alpha, s'_O) \wedge \bigwedge_{(\alpha,s'_M)\in NotPres_I} \neg(\alpha, s'_O)$.

  The obligation formula $\mathbf{\Omega}(s_O) = \bigvee_{I\in[\![\mathfrak{C}(s_M)]\!]} \varphi_I$.

  Note that this obligation formula is in DNF.

  In addition if $[\![\mathfrak{C}(s_M)]\!] = \mathcal{P}(Trans(s_M))$, namely each possible combination of outgoing transitions of s_M is possible, then our obligation formula is an OR of all possible combinations. Trivially, it is possible to prove that this obligation formula is also equivalent to the obligation formula $\mathbf{tt}$.

Now we prove that $[\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}} =_\mathcal{R} [\![\mathbf{\Omega}(s_O)]\!]$, where $\mathcal{R} = \{(s_M, s_O)\}$ and s_M, s_O describe the same state s in M and O, respectively.

Trivially if $[\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}} = \emptyset$ then we can derive that $[\![\mathfrak{C}(s_M)]\!] = \emptyset$, hence for construction $\mathbf{\Omega}(s_O) = \mathbf{ff}$ and its semantics is empty. Otherwise, if $[\![\mathbf{\Omega}(s_O)]\!] = \emptyset$ then, the only possibility is $\mathbf{\Omega}(s_O) = \mathbf{ff}$ and, for construction we can conclude that $[\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}} = \emptyset$.

Case 1) $\forall I$. $I \in [\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}} \Rightarrow I \in [\![\mathbf{\Omega}(s_O)]\!]$.

Suppose $I \in [\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}}$, then a $J \in [\![\mathfrak{C}(s_M)]\!]$ such that $I = ExecutableKer(J)$ exists. For construction, we compute $Pres_J$ and $NotPres_J$ and we derive a formula φ from these two sets. Note that this property holds $(\alpha, s'_O) \in Pres_J \Leftrightarrow (\mathbf{tt}, \alpha, s'_M) \in J \Leftrightarrow (\mathbf{tt}, \alpha, s'_M) \in I$.

In addition φ is a disjunct of $\mathbf{\Omega}(s_O)$ and, seeing that φ is a conjunction of all outgoing transitions of s_O, where all positive atoms are all and only the guarded transitions in I, then $I \in [\![\varphi]\!]$. Trivially, we can deduce that $I \in [\![\mathbf{\Omega}(s_O)]\!]$.

Case 2) $\forall I.\ I \in [\![\mathbf{\Omega}(s_O)]\!] \Rightarrow I \in [\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}}$.

Suppose $I \in [\![\mathbf{\Omega}(s_O)]\!]$, then a disjunct φ of $\mathbf{\Omega}(s_O)$ exists such that $I \in [\![\varphi]\!]$. Trivially, for construction φ is a conjunction of all possible outgoing transitions of s_O and we can deduce that $[\![\varphi]\!]$ has only one possible valid set of transitions, namely I. For construction, then $\exists J \in [\![\mathfrak{C}(s_M)]\!]$ such that each positive atom in φ is related to a transition with guard $\mathbf{tt}$ in J. Seeing that each positive atom must be present in I, then we can deduce that $I = ExecutableKer(J)$ is true and, therefore, $I \in [\![\mathfrak{C}(s_M)]\!]_{|\{\mathbf{tt}\}}$ holds □

Lemma B.2. *Let O be an OTS* then exists a CMTS($\mathcal{G}_\mathcal{T}$) M such that $[\![[\![O]\!]]\!]_{\mathbb{OTS}^*} = [\![[\![M]\!]]\!]_{\mathbb{CMTS}_{\mathcal{G}_\mathcal{T}}}$.*

Proof.
First of all, taken $O = (\mathcal{S}_O, \Sigma, \longrightarrow_O, \mathbf{\Omega}_M, s_{0_O})$ we change conveniently its obligation function, for each state s, in the following way:

- if $[\![\mathbf{\Omega}(s)]\!] = [\![\mathbf{tt}]\!]$ then we define $\mathbf{\Omega}(s) = \mathbf{tt}$
- if $[\![\mathbf{\Omega}(s)]\!] = [\![\mathbf{ff}]\!]$ then we define $\mathbf{\Omega}(s) = \mathbf{ff}$
- if $[\![\mathbf{\Omega}(s)]\!] \neq [\![\mathbf{ff}]\!]$ and $[\![\mathbf{\Omega}(s)]\!] \neq [\![\mathbf{tt}]\!]$ then we simplify $\mathbf{\Omega}(s)$ such that in $\mathbf{\Omega}(s)$ we have only atomic literals and it is in CNF

These changes do not modify the semantics and not introduce some type of restriction, they are only useful to simplify the proof.

Since, for any state s, the obligation formula φ is in CNF then we can say $\varphi = \bigwedge_{1 \leq i \leq K} \varphi_i$ for some K. Note that if $\varphi = \mathbf{tt}$ then $K = 1$ and $\varphi_1 = \mathbf{tt}$, the same holds for $\mathbf{ff}$.

Now taken a generic φ_j, we define three categories:

1. $ATOM_j^+$ is the set of all positive atoms in φ_j, namely elements like (α, s')
2. $ATOM_j^-$ is the set of all negative atoms in φ_j, namely elements like $\neg(\alpha, s')$
3. $NOATOM_j$ is the set of all possible values which are not atomic literals, namely $\mathbf{tt}$ or $\mathbf{ff}$

Note that an atom is an element of $\Sigma \times \mathcal{S}$ and $ATOM_j^+ \cup ATOM_j^- \subseteq Trans(s_j)$ because we can have may transitions which are not considered in the obligation function.

In addition, we define $ATOM^+ = \bigcup_{1 \leq j \leq K} ATOM_j^+$ and $ATOM^- = \bigcup_{1 \leq j \leq K} ATOM_j^-$.

Now we define a CMTS($\mathcal{G}_\mathcal{T}$) $M = (\mathcal{S}_M, \Sigma, \longrightarrow_M, \mathfrak{C}_M, s_{0_M})$ such that:

- $\mathcal{S}_M = \mathcal{S}_O$

- $\longrightarrow_M = \{(s_M, g, \alpha, s'_M) \mid (g, \alpha, s'_M) \in DerivTrans(s_M)\}$ where:

$$DerivTrans(s_M) = DerivTrans^{+}(s_M) \cup DerivTrans^{-}(s_M) \cup DerivTrans^{\perp}(s_M)$$

These sets are computed in the following way: initially, we take the state s_M and determine the related state $s_O \in \mathcal{S}_O$ and the set $ATOM^+$ and $ATOM^-$, then

  - $DerivTrans^{+}(s_M) = \bigcup_{(\alpha, s'_O) \in ATOM^+} \{(\mathbf{tt}, \alpha, s'_M)\}$, where s'_M is the state related to s'_O
  - $DerivTrans^{-}(s_M) = \bigcup_{(\alpha, s'_O) \in ATOM^-} \{(\mathbf{tt}, \alpha, s'_M), (\mathbf{ff}, \alpha, s'_M), \}$, where s'_M is the state related to s'_O
  - $DerivTrans^{\perp}(s_M) = \{(\mathbf{tt}, \alpha, s'_M) \mid (\alpha, s'_O) \in Trans(s_O) \setminus (ATOM^+ \cup ATOM^-)\}$, where s'_M is the state related to s'_O. This set describes all may transitions which are not considered in the obligation function

- $\forall s_M$ we compute the related state s_O and $\mathbf{\Omega}(s_O)$, suppose that $\mathbf{\Omega}(s_O) = \varphi$ then:

  - if $\varphi = \mathbf{tt}$ then we define only one constraint $c =< CS, [min, max] >$ such that $CS = \{(s_M, \mathbf{tt}, \alpha, s'_M) \mid (s_M, \mathbf{tt}, \alpha, s'_M) \in Trans(s_M)\}$, $min = 0$ and $max = |CS|$
  - if $\varphi = \mathbf{ff}$ then we define only one constraint $c =< CS, [min, max] >$ such that $CS = Trans(s_M)$, $min = |CS| + 1$ and $max = |CS| + 1$
  - if φ is in CNF and it has only atomic literals then $\varphi = \bigwedge_{1 \leq j \leq K} \varphi_j$. Now we define the following sets of constraints:
    * $\forall (\alpha, s'_O) \in ATOM^-$ we define:

$$c =< \{(\mathbf{tt}, \alpha, S'_M), (\mathbf{ff}, \alpha, S'_M)\}, [1, 1] >$$

      . We call the set of all these constraints C_{cons}
    * $\forall 1 \leq j \leq K$ we define $c_j =< CS_j, [min_j, max_j] >$ where:
      1. $min_j = 1$
      2. $max_j = |CS|$
      3. $CS_j = \{(\mathbf{tt}, \alpha, s'_M) \mid (\alpha, s'_O) \in ATOM^+_j\} \cup \{(\mathbf{ff}, \alpha, s'_M) \mid (\alpha, s'_O) \in ATOM^-_j\}$

      We call the set of all these constraints $C_{disjuncts}$

    In addition $\forall (\alpha, s'_O) \in Trans(s_O) \setminus (ATOM^+ \cup ATOM^-)$ we define $c =< \{(\mathbf{tt}, \alpha, s'_M)\}, [0, 1] >$ and we call the set of all these constraints $C_{no-atom}$. The set $\mathfrak{C}(s_M) = C_{cons} \cup C_{disjuncts} \cup C_{no-atom}$.

Now we prove the semantic equivalence and again we consider the semantics of constraints and obligation formula. If for each state s_M and s_O the semantics of constraints of s_M is equivalent to the semantics of the obligation formula of s_O then M and O have the same semantics, because they can derive the same set of LTSs.

Take states s_M and s_O, we can note that the semantics of s_O is a set of LTS, whereas the semantics of constraints of s_M is a set of LTS($\mathcal{G}_\mathcal{T}$). In Section4.1 we define two types of semantics of constraints: $[\![.]\!]_\mathcal{G}$ to denote the set of guarded transitions with both types of guard, whereas $[\![.]\!]$ to denote the set of guarded transition with the only guard $\mathbf{tt}$. Then we prove that $[\![\mathfrak{C}(s_M)]\!] =_\mathcal{R} [\![\mathbf{\Omega}(s_O)]\!]$, where $\mathcal{R} = \{(s_M, s_O)\}$.

Trivially, suppose that $[\![\mathbf{\Omega}(s_O)]\!] = \emptyset$ then this is possible if and only if $\mathbf{\Omega}(s_O) = \mathbf{ff}$ but, for construction, the set of constraints in s_M is formed by only one constraint, which is inconsistent and then $[\![\mathfrak{C}(s_M)]\!]_\mathcal{G} = [\![\mathfrak{C}(s_M)]\!] = \emptyset$.

Now suppose that $[\![\mathfrak{C}(s_M)]\!] = \emptyset$ and this is possible if and only if we have at least one inconsistent constraint. Note that all constraints in $C_{cons}, C_{disjuncts}$ and $C_{no-atom}$ are consistent, the only possible inconsistent constraint is the defined ones when $\mathbf{\Omega}(s_O) = \mathbf{ff}$, concluding that $[\![\mathbf{\Omega}(s_O)]\!] = \emptyset$.

At this point, we want to prove that $[\![\mathbf{\Omega}(s_O)]\!] =_\mathcal{R} [\![\mathfrak{C}(s_M)]\!]$, when $\mathbf{\Omega}(s_O)$ is a CNF formula $\varphi = \bigwedge_{1 \leq i \leq K} \varphi_i$ for some K.

Suppose that $I \in [\![\mathbf{\Omega}(s_O)]\!]$ then $I \in [\![\varphi_j]\!]$ for any $1 \leq j \leq K$. Now try to verify if $I \in [\![\mathfrak{C}(s_M)]\!]$, namely if $\exists J \in [\![\mathfrak{C}(s_M)]\!]_\mathcal{G}$ such that $I = ExecutableKer(J)$. Trivially, J satisfies a constraint in C_{cons} if two equivalent transitions with different guards are not both in J, this is an important restriction to guarantee the consistency. Note that in I a transition cannot be simultaneously present and not, so J trivially can satisfy these constraints. Constraints in $C_{no-atom}$ are general non-restrictive constraints, hence any possible J satisfies these constraints. Finally, we take a constraint $c \in C_{disjuncts}$ and $J \in [\![c]\!] \Leftrightarrow 1 \leq |J \cap CS_c| \leq |CS_c|$. For definition, taken the constraint c then exists some disjunct φ_h related to c. Seeing that $I \in [\![\varphi_h]\!]$ then we can deduce that $\exists(\alpha, s'_O) \in ATOM_h^+.\ I \in [\![(\alpha, s'_O)]\!] \vee \exists(\alpha, s'_O) \in ATOM_h^-.\ I \in [\![\neg(\alpha, s'_O)]\!]$.

Suppose that $\exists(\alpha, s'_O) \in ATOM_h^+.\ I \in [\![(\alpha, s'_O)]\!]$ is true then $(\alpha, s'_O) \in I$. Since $I = ExecutableKer(J)$ then $(\mathbf{tt}, \alpha, s'_M) \in J$, deducing that surely $1 \leq |J \cap CS_c|$.

On the other hand, if $\exists(\alpha, s'_O) \in ATOM_h^-.\ I \in [\![\neg(\alpha, s'_O)]\!]$ is true then $(\alpha, s'_O) \notin I$. Since $(\alpha, s'_O) \in ATOM_h^-$ then $(s_M, \mathbf{ff}, \alpha, s'_M)$ exists.

In addition $I = ExecutableKer(J)$ then $(\mathbf{tt}, \alpha, s'_M) \notin J$ and for some consistency constraint we can derive that $(\mathbf{ff}, \alpha, s'_M) \in J$, deducing that surely $1 \leq |J \cap CS_c|$.

Now suppose that $I \in [\![\mathfrak{C}(s_M)]\!]$ then $\exists J \in [\![\mathfrak{C}(s_M)]\!]_\mathcal{G}$ and $I = ExecutableKer(J)$.

Trivially, J satisfies constraints in $C_{no-atom}$ and C_{cons} but in the first case we consider may transitions which are not handled by an obligation function, whereas in the second case we require that a transition is not simultaneously present and not present and this is always true.

Now we consider a constraint $c \in C_{disjuncts}$, we know that a φ_h exists and φ_h is

a disjunct of φ. If $J \in [\![c]\!]_{\mathcal{G}}$ then at least one transition of $Choice(c)$ exists in J. Therefore if this transition is $(\mathbf{ff}, \alpha, s'_M)$, then in $I = ExecutableKer(J)$ it does not exist, in addition a negative atom (α, s'_O) exists in φ and, trivially, $I \in [\![\neg(\alpha, s'_O)]\!]$. On the other hand, if this transition is $(\mathbf{tt}, \alpha, s'_M)$, then in $I = ExecutableKer(J)$ it exists, in addition a positive atom (α, s'_O) exists in φ and, trivially, $I \in [\![(\alpha, s'_O)]\!]$.

Hence, if J satisfies each constraint in $C_{disjuncts}$ then I satisfies each constraint in φ, then $I \in [\![\mathbf{\Omega}(s'_O)]\!]$.

Finally, the special case is when $\mathbf{\Omega}(s_O) = \mathbf{tt}$. Trivially the semantics of this logic formula is equal to $\mathcal{P}(Trans(s_O))$, namely all possible sets of transitions. In addition, we note that by means functions $DerivTrans$ if (s_O, α, s'_O) exists then $(s_M, \mathbf{tt}, \alpha, s'_M)$ exists too. If $\mathbf{\Omega}(s_O) = \mathbf{tt}$ then the derived constraint is a general non-restrictive constraint and its choice set has all possible transitions with guard equal to $\mathbf{tt}$. It is clear that, in this way, we have all possible sets of transitions with guard equal to $\mathbf{tt}$, deducing that $[\![\mathfrak{C}(s_M)]\!] = \mathcal{P}(\{(s_M, \mathbf{tt}, \alpha, s'_M)\})$.

On the other hand, if $[\![\mathfrak{C}(s_M)]\!] = \mathcal{P}(\{(s_M, \mathbf{tt}, \alpha, s'_M)\})$ then we have two possibilities:

- exists a general non-restrictive constraint which describes the powerset and in this case we can derive that $\mathbf{\Omega}(s_O) = \mathbf{tt}$
- the powerset is derived by a set of possible constraints. The only possibility of having a set of constraints is if $[\![\mathbf{\Omega}(s_O)]\!] \neq [\![\mathbf{ff}]\!]$ and $[\![\mathbf{\Omega}(s_O)]\!] \neq [\![\mathbf{tt}]\!]$. But we have just seen that, in this case, $[\![\mathfrak{C}(s_M)]\!] = [\![\mathbf{\Omega}(s_O)]\!]$, hence $[\![\mathbf{\Omega}(s_O)]\!] = \mathcal{P}(\{(s_O, \alpha, s'_O)\})$. In this case we can deduce that $[\![\mathbf{\Omega}(s_O)]\!] = [\![\mathbf{tt}]\!]$ and, for hypotheses, we should have changed $\mathbf{\Omega}(s_O) = \mathbf{tt}$ and hence, we cannot have a set of possible constraints, deriving an absurd

□

Theorem B.7. *The formalism* $\mathbb{OTS}^*$ *is as many expressive as* $\mathbb{CMTS}(\mathcal{G}_{\mathcal{T}})$*, namely* $\mathbb{OTS}^* \leftrightsquigarrow \mathbb{CMTS}(\mathcal{G}_{\mathcal{T}})$

Proof.
The theorem holds for Lemma B.1 and Lemma B.2.

□

Theorem B.8. *The formalism* $\mathbb{PMTS}$ *is as many expressive as* $\mathbb{CMTS}(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$*, namely* $\mathbb{PMTS} \leftrightsquigarrow \mathbb{CMTS}(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$

Proof.
We know that, taken an assignment A, then we can derive an OTS* from PMTS and the assignment A. The same reasoning holds for a CMTS$(\mathcal{G}_{\mathcal{T}}, \mathcal{G}^{\mathcal{Q}})$.

Hence, the hint is touse these two observations and useful constructions determined for Theorem B.7.

We must prove two cases:

1. from a CMTS($\mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}$) M we can derive a PMTS P
2. from a PMTS P we can derive a CMTS($\mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}$) M

Case 1) we use a similar construction to the one seen in Lemma B.1. Take an assignment A, we derive a logic formula related to A, namely $\varphi_A = \bigwedge_{p \in A} p \wedge \bigwedge_{p \notin A} \neg p$.

Then we define the formula logic $\mathbf{\Omega}(s_P, A) = \mathbf{ff} \wedge \varphi_A$, if $[\![\sigma(s_M, A)]\!] = \emptyset$, otherwise $\mathbf{\Omega}(s_P, A) = \bigvee_{I \in [\![\sigma(s_M, A)]\!]} \varphi_I \wedge \varphi_A$ where $\sigma(s_M, A)$ is the state s_M such that each guarded constraints is assigned in respect to the value A. Then $\mathbf{\Omega}(s_P) = \bigvee_{A \subseteq \mathcal{Q}} \mathbf{\Omega}(s_P, A)$.

Trivially, for an assignment A the CMTS($\mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}$) M becomes a CMTS($\mathcal{G}_\mathcal{T}$), namely $\sigma(M, A)$. At the same time, for each assignment $B \neq A \subseteq \mathcal{Q}$, we have that $\varphi_B = \mathbf{ff}$ and, hence, $\mathbf{\Omega}(s_P)$ becomes equal to $\mathbf{\Omega}(s_P, A)$, in particular it is equivalent $\bigvee_{I \in [\![\sigma(s_M, A)]\!]} \varphi_I \wedge \mathbf{tt}$. Then the PMTS becomes the same OTS* used in Lemma B.1. Trivially this OTS* and CMTS($\mathcal{G}_\mathcal{T}$) have the same semantics and this is true for a generic assignment A, hence it is always true, for any possible assignment.

Case 2) we use a similar construction to the one seen in Lemma B.2. Take an assignment A, our PMTS becomes an OTS*, seeing that we can simplify logic formulae of P by means of A. Hence, derived this OTS*, we use the same construction of Lemma B.2 to determine a CMTS($\mathcal{G}_\mathcal{T}$) but this time each constraint derived by means of the construction of Lemma B.2 becomes a guarded constraint where the guard is a multi-guard equals to A. Note that, in this case, a certain constraint could have different multi-guard simultaneously.

Now, take an assignment A, the PMTS becomes an OTS* and the CMTS($\mathcal{G}_\mathcal{T}, \mathcal{G}^\mathcal{Q}$) becomes a CMTS($\mathcal{G}_\mathcal{T}$). In particular, note that in $\sigma(M, A)$ enabled constraints are all and only constraints which we can derive by means of the construction of Lemma B.2 when we have an OTS*. Trivially, the derived OTS* and CMTS($\mathcal{G}_\mathcal{T}$) are equivalent for Lemma B.2 and this holds for any possible assignment A. □

Appendix C

Parallel Composition

In this chapter we introduce the idea of how to handle the parallel composition in CMTS context, in particular how, taken two CMTSs M and M_1, we can represent the CMTS $M||M_1$, where $||$ is the parallel composition. This topic is very important because we might have a system L composed by several subsystems L_i, so we might describe several subsystems separately by means of suitable specifications S_i then we might connect all together these subsystems in order to obtain the specification S of the general system L.

The idea of parallel composition is simple, indeed if two states s and t must be composed in parallel way then:

- if both states have a transition labelled with the same action α and the same transition type (for example both transitions are may transitions or both are must transitions), then the two states can be synchronized over the execution of this particular transition.
- otherwise we execute a possible transition of only one state, whereas the other state is unchanged, namely a subsystem makes progress, whereas the other one staying in its current state.

First of all, we introduce the composition in the LTS world [39] and in the MTS world [11] [28] [36] [44]:

Definition C.1 (**LTS composition**)**:**
Let $M = (\mathcal{S}, \Sigma, \longrightarrow), M_1 = (\mathcal{S}_1, \Sigma_1, \longrightarrow_1)$ be two LTSs. Then we denote the parallel composition of M and M_1 by $M||M1$.

$M||M_1$ is a LTS described by the tuple $(\mathcal{S}_P, \Sigma_P, \longrightarrow_P)$ where:

- $\mathcal{S}_P = \mathcal{S} \times \mathcal{S}_1$
- $\Sigma_P = \Sigma \cup \Sigma_1$
- $\longrightarrow$ is inductively defined by:
  (1) $\dfrac{s \xrightarrow{\alpha} s' \quad \nexists t'.\ t \xrightarrow{\alpha} t'}{s||t \xrightarrow{\alpha}_P s'||t}$ (2) $\dfrac{t \xrightarrow{\alpha} t' \quad \nexists s'.\ s \xrightarrow{\alpha} s'}{s||t \xrightarrow{\alpha}_P s||t'}$ (3) $\dfrac{s \xrightarrow{\alpha} s' \quad t \xrightarrow{\alpha} t'}{s||t \xrightarrow{\alpha}_P s'||t'}$

■

Definition C.2 (MTS composition):
Let $M = (\mathcal{S}, \Sigma, \longrightarrow_\Diamond, \longrightarrow_\Box), M_1 = (\mathcal{S}_1, \Sigma_1, \longrightarrow_{\Diamond_1}, \longrightarrow_{\Box_1})$ be two MTSs. Then we denote the parallel composition of M and M_1 by $M||M1$.

$M||M_1$ is a MTS described by the tuple $(\mathcal{S}_P, \Sigma_P, \longrightarrow_{\Diamond_P}, \longrightarrow_{\Box_P})$ where:

- $\mathcal{S}_P = \mathcal{S} \times \mathcal{S}_1$
- $\Sigma_P = \Sigma \cup \Sigma_1$
- $\longrightarrow_{\Diamond_P}$ is inductively defined by:
  1. $\dfrac{s \xrightarrow{\alpha}_\Diamond s' \quad \nexists t'.\ t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Diamond_P} s'||t}$ and vice versa
  2. $\dfrac{s \xrightarrow{\alpha}_\Diamond s' \quad t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Diamond_P} s'||t'}$
  3. $\dfrac{s \xrightarrow{\alpha}_\Box s' \quad t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Diamond_P} s'||t'}$ and viceversa
- $\longrightarrow_{\Box_P}$ is inductively defined by:
  1. $\dfrac{s \xrightarrow{\alpha}_\Box s' \quad \nexists t'.\ t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Box_P} s'||t}$ and viceversa
  2. $\dfrac{s \xrightarrow{\alpha}_\Box s' \quad t \xrightarrow{\alpha}_{\Box_1} t'}{s||t \xrightarrow{\alpha}_{\Box_P} s'||t'}$

  All rules are simple except the following:

  – $\dfrac{s \xrightarrow{\alpha}_\Box s' \quad t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Diamond_P} s'||t'}$: in this case we have a must outgoing transition labelled with α from s and a may outgoing transition labelled with α from t. Since $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$ then we can deduces $s \xrightarrow{\alpha}_\Diamond s'$ exists, so we can only compose transitions of the same types, that is may transitions of s and t, reaching $s||t \xrightarrow{\alpha}_{\Diamond_P} s'||t'$

  – $\dfrac{s \xrightarrow{\alpha}_\Box s' \quad \nexists t'.\ t \xrightarrow{\alpha}_{\Diamond_1} t'}{s||t \xrightarrow{\alpha}_{\Box_P} s'||t}$: the condition $\not\exists t'.\ t \xrightarrow{\alpha}_{\Diamond_1} t'$ means that from t is impossible to execute the action α, since $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$, in this way we guarantee that it exists neither a may transition nor must transition labelled with α.

■

As we can see, the parallel composition both for LTSs and for MTSs is a very simple operation. Unfortunately, in CMTS context this operation becomes slightly more complicated and the reason is related to the constraint concept. In effect the bigger problem which we must resolve is, taken two states s and t of two different CMTSs, to understand how derive the set $Choice(s||t)$ by sets $Choice(s)$ and $Choice(t)$.

We want to point out again that the constraint concept is "strange": it does not express which transitions must be considered from a particular predefined set but it expresses the minimum and maximum number of transitions which must be taken from the set, allowing a "free choice" about the exact transitions to be considered. This free choice entails obvious problems when we try to reason about parallel composition, but we will try to solve them in a simple way.

First of all, suppose we have two CMTSs $M = (\mathcal{S}, \Sigma, \longrightarrow, \mathfrak{C})$ and $M_1 = (\mathcal{S}_1, \Sigma_1, \longrightarrow_1, \mathfrak{C}_1)$ which have a particular property: $\forall s \in \mathcal{S}, s_1 \in \mathcal{S}_1$ the following holds:

- $\forall c \in \mathfrak{C}(s).\ Label(c) \subseteq Label(s_1) \Rightarrow \exists c_1 \in \mathfrak{C}_1(s_1).\ Label(c) = Label(c_1)$
- $\forall c_1 \in \mathfrak{C}_1(s_1).\ Label(c_1) \subseteq Label(s) \Rightarrow \exists c \in \mathfrak{C}(s).\ Label(c) = Label(c_1)$

This property is simple: it requires that for any constraint c of a state s of M, if the choice set of c, which is composed by a set of outgoing transitions of s, also describes a set of possible outgoing transitions of a some state s_1 of M_1, then in s_1 must exist a constraint which handle the same choice set and this property must be true for states of M_1 too. The property is not a restriction because if a constraint related to a set of transitions S is absent then we know that it is equivalent to $< S, [0, |S|] >$, moreover it is very useful because it allow us to reason about the same constraint, even if it is simultaneously related to two different states when these states have a common subset of outgoing transitions. It is possible to make another simple observation: suppose we have a constraint $c =< CS, [min, max] >\in \mathfrak{C}(s)$ of a some state s and a set $T \subseteq CS$ such that $|T| = max$. In this case c implicitly requires that the set $CS \setminus T$ has only transitions "*forbidden*", namely transitions which must not be executed by the state s. In effect if we have $I \subseteq Trans(s)$ such that $T \subseteq I$, then $I \models c \Leftrightarrow I \cap (CS \setminus T) = \emptyset$, that is if I describes a possible solution which includes the set T and if I satisfies c then we can simply deduce that transitions in $CS \setminus T$ are not present surely. In conclusion we can suppose to have two types of transitions: *executable* and *forbidden*.

From all these observations we have the needed knowledge to solve the problem of parallel composition in CMTSs. Our aim is to determine the set $\mathfrak{C}(s||t)$, taken two states s and t of two different CMTSs and their constraints $\mathfrak{C}(s)$ and $\mathfrak{C}(t)$.

We consider these two states s and t, then three situations are possible:

1. exists a constraint c of s such that $Label(c) \not\subseteq Label(t)$
2. exists a constraint c of t such that $Label(c) \not\subseteq Label(s)$

3. exists a constraint c of s such that $Label(c) \subseteq Label(t)$ and so, for hypothesis, exists a constraint c_1 of t such that $Label(c) = Label(c_1)$

In first case the chosen constraint c introduces some restrictions over transitions but, unfortunately, c must handle some outgoing transitions of s which do not exist in t. Therefore c cannot be deleted, seeing that it describes some type of restriction in s not present in t, and hence we can deduce that c must be included in $\mathfrak{C}(s||t)$. The second case is obviously symmetric to the first one. The third is the special case: we have a constraint c of s and a constraint c_1 of t such that $Label(c) = Label(c_1)$, furthermore we suppose $c =< CS, [min_c, max_c] >$ and $c_1 =< CS_1, [min_{c_1}, max_{c_1}] >$. Now we imagine to have a set of transitions $T \subseteq Trans(s)$ and a set of transitions $T_1 \subseteq Trans(t)$ such that $Label(T) = Label(T_1)$. Moreover we know that $Label(c) = Label(c_1)$, hence for convenience we can reason about only labels, ignoring target states. Note that this is not a restrictive assumption for two reasons:

1. CMTSs are action-deterministic, so labels identify univocally transitions
2. in non-deterministic CMTS we can change the label of a single transitions with a couple $< label, targetstate >$, returning to a CMTS action-deterministic

We call $L = Label(T), L_1 = Label(T_1), L_c = Label(c) = Label(c_1)$, in addition note that $L = L_1$. Suppose that $min_c \leq |L \cap L_c| \leq max_c$, then we have three possibilities:

1. $min_{c_1} \leq |L_1 \cap L_c| \leq max_{c_1}$
2. $max_{c_1} < |L_1 \cap L_c|$
3. $|L_1 \cap L_c| < min_{c_1}$

In the first case we have a set of transitions in T and T_1 and all together can be executed both from s and from t, so s and t can be synchronized by means of the set of actions L. In the second case we have too many transitions if we want to satisfy c_1, so the correct number of transitions which we can consider is $min\{max_c, max_{c_1}\}$. For a better understanding of this result, we try to reason in a different way: our set T satisfies the constraint c but the corresponding T_1 does not satisfy c_1, therefore we divide the set L in two components L_V and L_R, where $L_V \subseteq L$ and $|L_V \cap L_c| = max_{c_1}$ and $L_R = L \backslash L_V$. From the pointview of c_1, actions in L_V can be executable, whereas actions in L_R are forbidden. The reached situation is the following:

- in s all actions in L are executable, so their type is executable
- in t actions in L_V are executable, whereas actions in L_R are not executable

So if we compose s and t, we reach $s||t$ such that it can execute all actions in L_V, whereas all transitions in L_R cannot be executed. Note that the transitions in L_R have type as "executable" in s, as 'forbidden" in t and as "forbidden" in $s||t$, so it is possible to deduce that the type "forbidden" is prioritary compared to type

"executable", and the reason is simple: in parallel composition we can synchronize only transitions with the same type and the same action, in this case some transitions have different types.

Now we try to reason about the third case: this time all actions in L can be carried out both in s and in t, unfortunately they are not enough for t. Therefore we can try to add new actions in L to reach the value min_{c_1}. In this context we must remember that constraints c and c_1 have the same actions, so if we add a new action to satisfy c_1, implicitly we add a new action to c too. As before, we suppose that our set T satisfies the constraint c, namely $min_c \leq |L \cap L_c| \leq max_c$ but the corresponding T_1 does not satisfy c_1, because $|L_1 \cap L_{c_1}| < min_{c_1}$. Now we focus on T: once we choose a set of transition from the set $Trans(s)$, we implicitly assume that all other transitions must be missing. In particular, taken T and L, we can deduce that all labels in $L \cap L_c$ must be present, whereas all labels in $L_c \setminus L$ must be absent. Again we can divide the set L_c in two components $L_T = L \cap L_c$, namely all labels in L_c and in L, and $L_R = L_c \setminus L$. In addition we suppose that $min_c \leq |L_T| < max_c$ then we can deduce that all labels in L_R would be still valid for the constraint c because $|L_T| < max_c$, therefore we might still add k labels at most, where $k = max_c - |L_T|$. Unfortunately, seeing that we have chosen exactly the set L_T of labels then we have assumed implicitly that all transitions labelled by a label in L_R are forbidden. Therefore, s and t can be only synchronized over labels L_T, even though they are not enough for t. Of course, in the computation of the minimum value of parallel composition we must take into account this situation too. The worst case is when $|L \cap L_c| = max_c$ and $|L \cap L_c| < min_{c_1}$, in effect in this situation if we try to add new k actions of L_c to T, where $k = min_{c_1} - |L_1 \cap L_c|$ that is the needed number of transitions to obtain $|L_1 \cap L_c| = min_{c_1}$, then obviously $|L \cap L_c| > max_c$. But, as before, we know that if $|L \cap L_c| = max_c$ then, for c, the actions in $L_c \setminus L$ are forbidden. On the other hand, we may take k actions from set $L_c \setminus L_1$ and they are executable for t, concluding that for $s||t$ this k transitions are not allowed. The minimum value for $s||t$ is, therefore, equals to $min\{min_c, min_{c_1}\}$.

A simple way to see why these choices are correct is: we generate all possible implementations for s and t, then we compose in parallel all together and eventually we compute the value of $min_{s||t}$ and $max_{s||t}$ by means of all possible parallel compositions which can be created. For parallel composition of implementations we can use the rules which we have seen for LTS, but we must note that each outgoing transition, existing in the CMTS but not in the derived LTS, is a transition which we establish being *forbidden*. Now we see an example of these concepts: suppose we have two CMTSs M Figure C.1 and N Figure C.2. In these figures we also show all possible derived LTSs. In Figure C.3, Figure C.4 and Figure C.5 we describe all possible compositions between derived LTS of M and N. As we can see, the state $s_0||t_0$ has at most one of actions of set $\{(a, b)\}$, so the constraints related to $s_0||t_0$ should be $c_{||} =< \{(a, s_1||t_1), (b, s_2||t_2)\}, [0, 1] >$. We can derive the same result if we compute $min_{c_{||}} = min\{min_{c_s}, min_{c_t}\}$ and $max_{c_{||}} = min\{max_{c_s}, max_{c_t}\}$.

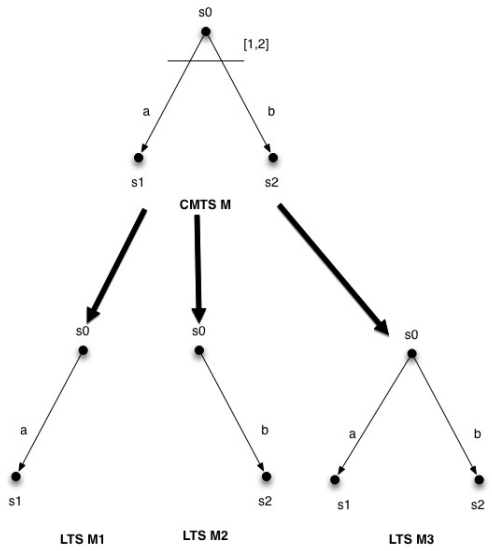


Figure C.1: A possible CMTS with its derived LTSs

Definition C.3:
Let $c =< CS_c, [min_c, max_c] >, c_1 =< CS_{c_1}, [min_{c_1}, max_{c_1}] >$ be two constraints of two different states such that $Label(c) = Label(c_1)$ then we denote their parallel composition by $c||c_1 =< CS_{||}, [min_{||}, max_{||}] >$ where:

- $CS_{||} = \{(\alpha, s||s_1 \mid (\alpha, s) \in CS_c \wedge (\alpha, s_1) \in CS_{c_1}\}$
- $min_{||} = min\{min_c, min_{c_1}\}$
- $max_{||} = max\{min_c, min_{c_1}\}$

■

Definition C.4:
Let $M = (\mathcal{S}_M, \Sigma_M, \longrightarrow_M, \mathfrak{C}_M), N = (\mathcal{S}_N, \Sigma_N, \longrightarrow_N, \mathfrak{C}_N)$ be two CMTSs. We call their parallel composition $M||N = (\mathcal{S}_{||}, \Sigma_{||}, \longrightarrow_{||}, \mathfrak{C}_{||})$ where:

- $\mathcal{S}_{||} = \mathcal{S}_M \times \mathcal{S}_N$
- $\Sigma_{||} = \Sigma_M \cup \Sigma_N$
- $\forall s_M \in \mathcal{S}_M, s_N \in \mathcal{S}_N$ we have:
  1. $c \in \mathfrak{C}_M(s_M) \wedge Label(c) \not\subseteq Label(s_N) \Rightarrow c \in \mathfrak{C}_{||}(s_M||s_N)$
  2. $c \in \mathfrak{C}_N(s_N) \wedge Label(c) \not\subseteq Label(s_M) \Rightarrow c \in \mathfrak{C}_{||}(s_M||s_N)$
  3. $\exists c_M \in \mathfrak{C}_M(s_M), c_N \in \mathfrak{C}_N(s_N).\ Label(c_M) = Label(c_N) \Rightarrow (c||c_1) \in \mathfrak{C}_{||}(s_M||s_N)$

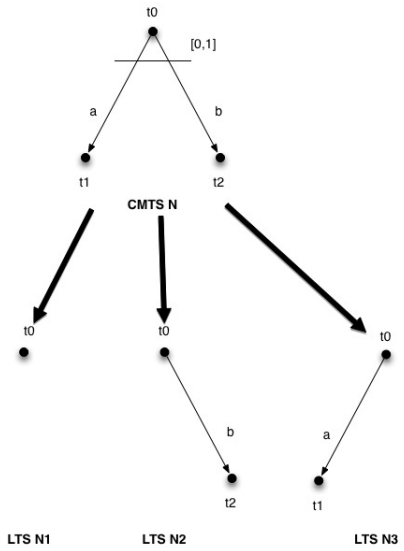


Figure C.2: Another possible CMTS with its derived LTSs

- $\longrightarrow_{||}$ is inductively defined by:
  (1) $\dfrac{s \xrightarrow{\alpha} s' \quad \nexists t'.\ t \xrightarrow{\alpha} t'}{s||t \xrightarrow{\alpha}_P s'||t}$ (2) $\dfrac{t \xrightarrow{\alpha} t' \quad \nexists s'.\ s \xrightarrow{\alpha} s'}{s||t \xrightarrow{\alpha}_P s||t'}$ (3) $\dfrac{s \xrightarrow{\alpha} s' \quad t \xrightarrow{\alpha} t'}{s||t \xrightarrow{\alpha}_P s'||t'}$

■

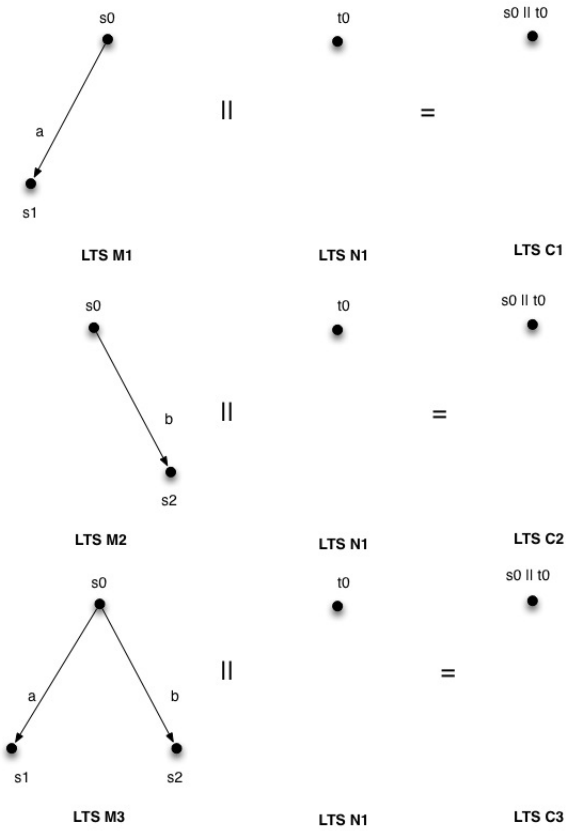


Figure C.3: Composition of derived LTSs

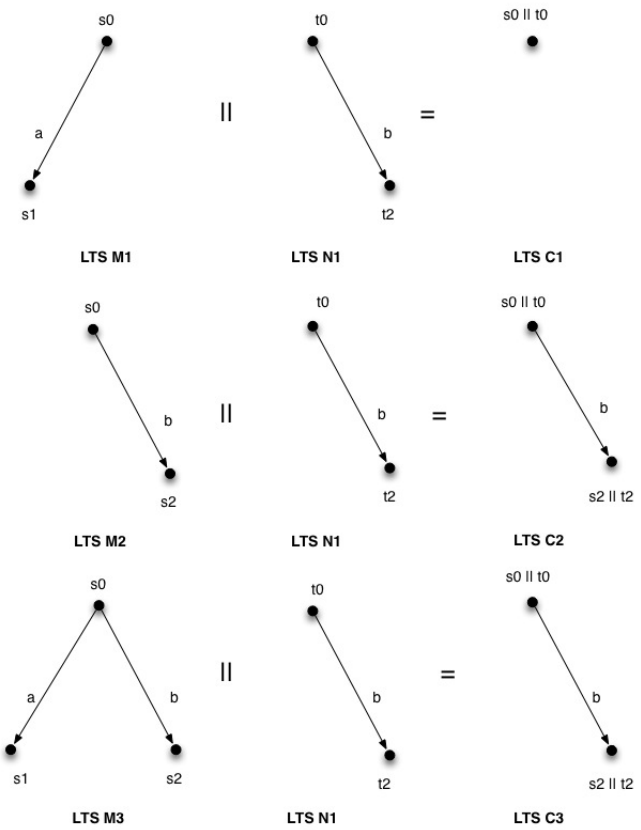


Figure C.4: Composition of derived LTSs

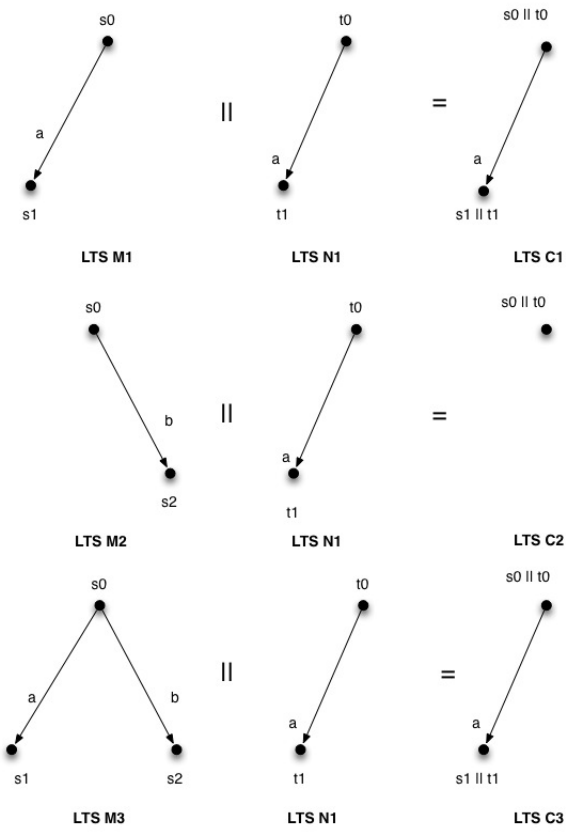


Figure C.5: Composition of derived LTSs

Bibliography

[1] Antonik, A., Huth, M., Larsen, K. G., Nyman, U., and Wasowski, A. 20 years of modal and mixed specifications. *Bulletin of the EATCS 95* (2008), 94–129.

[2] Aqvist, L. Deontic logic. In *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*, D. Gabbay and F. Guenthner, Eds. Reidel, Dordrecht, 1984, pp. 605–714.

[3] Asirelli, P., ter Beek, M. H., Fantechi, A., and Gnesi, S. A logical framework to deal with variability. In *IFM* (2010), pp. 43–58.

[4] Asirelli, P., ter Beek, M. H., Fantechi, A., and Gnesi, S. Formal description of variability in product families. In *SPLC* (2011), pp. 130–139.

[5] Asirelli, P., ter Beek, M. H., Fantechi, A., and Gnesi, S. A model-checking tool for families of services. In *FMOODS/FORTE* (2011), pp. 44–58.

[6] Baier, C., and Katoen, J. *Principles of model checking*. MIT Press, 2008.

[7] Batory, D. Feature models, grammars, and propositional formulas. In *SPLC* (2005), pp. 7–20.

[8] Benes, N., Kretínský, J., Larsen, K. G., and Srba, J. On determinism in modal transition systems. *Theor. Comput. Sci. 410*, 41 (2009), 4026–4043.

[9] Beneš, N. *Disjunctive Modal Transition Systems*. Disertační práce, Masarykova univerzita, Fakulta informatiky, 2012.

[10] Beneš, N., Cerna, I., and Kretínský, J. Disjunctive modal transition systems and generalized ltl model checking. Technical report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, Brno, 2010.

[11] Beneš, N., Cerna, I., and Kretínský, J. Modal transition systems: Composition and ltl model checking. In *ATVA* (2011), pp. 228–242.

[12] Beneš, N., and Kretínský, J. Process algebra for modal transition systems. In *MEMICS* (2010), pp. 9–18.

[13] Beneš, N., Kretínský, J., Larsen, K. G., Møller, M. H., and Srba, J. Parametric modal transition systems. In *ATVA* (2011), pp. 275–289.

[14] Blackburn, P., de Rijke, M., and Venema, Y. *Modal Logic*, vol. 53 of *Cambridge Tracts in Theoretical Computer Scie*. Cambridge University Press, Cambridge, 2001.

[15] Boudol, G., and Larsen, K. Graphical versus logical specifications. In *CAAP* (1990), pp. 57–71.

[16] Bruni, R., Montanari, U., and Sassone, V. Open ended systems, dynamic bisimulation and tile logic. In *IFIP TCS* (2000), pp. 440–456.

[17] Clarke, E., and Emerson, E. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop* (London, UK, UK, 1982), Springer-Verlag, pp. 52–71.

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

[19] Clements, P. C., and Northrop, L. *Software product lines: practices and patterns*. Addison-Wesley, August 2001.

[20] Dams, D., Gerth, R., and Grumberg, O. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst. 19*, 2 (1997), 253–291.

[21] De Nicola, R., and Vaandrager, F. Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes* (New York, NY, USA, 1990), Springer-Verlag New York, Inc., pp. 407–419.

[22] De Nicola, R., and Vaandrager, F. Three logics for branching bisimulation. *J. ACM 42*, 2 (Mar. 1995), 458–487.

[23] Emerson, E., and Halpern, J. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing* (New York, NY, USA, 1982), STOC '82, ACM, pp. 169–180.

[24] Emerson, E., and Halpern, J. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM 33* (1986), 151–178.

[25] Fantechi, A., and Gnesi, S. A behavioural model for product families. In *ESEC/SIGSOFT FSE* (2007), pp. 521–524.

[26] Fantechi, A., and Gnesi, S. Formal modeling for product families engineering. In *SPLC* (2008), pp. 193–202.

[27] Fecher, H., and Schmidt, H. Comparing disjunctive modal transition systems with an one-selecting variant.

[28] Fischbein, D., D'Ippolito, N., Brunet, G., M.Chechik, and Uchitel, S. Weak alphabet merging of partial behavior models. *ACM Trans. Softw. Eng. Methodol. 21*, 2 (2012), 9.

[29] Hennessy, M., and Milner, R. On observing nondeterminism and concurrency. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming* (London, UK, UK, 1980), Springer-Verlag, pp. 299–309.

[30] Hilpinen, R. Deontic logic. In *The Blackwell Guide to Philosophical Logic*, L. Goble, Ed., Blackwell Philosophy Guides. Wiley Blackwell, 2001, pp. 159–183.

[31] Jazayeri, M., Ran, A., and Van Der Linden, F. *Software Architecture for Product Families: Principles and Practice*. Addison-Wesley, 2000.

[32] Keller, R. M. Formal verification of parallel programs. *Commun. ACM 19*, 7 (1976), 371–384.

[33] K.G.Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *Theor. Comput. Sci. 72*, 2&3 (1990), 265–288.

[34] Larsen, K. Modal specifications. In *Automatic Verification Methods for Finite State Systems* (1989), pp. 232–246.

[35] Larsen, K. G., Nyman, U., and Wasowski, A. On modal refinement and consistency. In *CONCUR* (2007), pp. 105–119.

[36] Larsen, K. G., and Thomsen, B. A modal process logic. In *LICS* (1988), pp. 203–210.

[37] Larsen, K. G., and Xinxin, L. Equation solving using modal transition systems. In *LICS* (1990), pp. 108–117.

[38] Meyer, J. J. C. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic 29* (1988), 109–136.

[39] Milner, R. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[40] P, F. C., and Maibaum, T. S. E. A complete and compact propositional deontic logic. In *ICTAC* (2007), pp. 109–123.

[41] Park, D. M. R. Concurrency and automata on infinite sequences. In *Theoretical Computer Science* (1981), pp. 167–183.

[42] Pnueli, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1977), SFCS '77, IEEE Computer Society, pp. 46–57.

[43] Turner, D. *Modal Logic: Possible Worlds Semantics.* Knox College, Galesburg, Ill., 1984.

[44] Uchitel, S., and Chechik, M. Merging partial behavioural models. In *SIGSOFT FSE* (2004), pp. 43–52.