

Ph.D. Thesis

# Modal and Component-Based System Specifications

Mikael Harkjær Møller

*Aalborg University*  
*Department of Computer Science*



# Abstract

Formal methods are not always applicable in software system design processes. We aim at improving this by extending existing abstract modelling formalisms. The main problem is that existing system specification formalisms is not always capable of modelling the actual system behaviour and concepts at the required abstraction level. Furthermore, the existing frameworks often do not scale to larger and more complex software systems.

We study extensions of *Modal Transition Systems*(MTS), a system modelling framework that support system refinements. This is beneficial in the design process where the system is designed gradually from a loose specification into a complete system specification with a fixed implementation behaviour.

Initially, we extend the MTS framework with a set of parameters that enable system designers to express persistency in their system models. Second, we extend the model with time and cost measures, that can then be used to analyse the implementation cost and average running cost. Early in a development process such analysis can assist the designers to keep their system within a given budget. Finally, we equip this extended MTS framework with a logic for expressing system requirements. This logic can be used by designers to verify whether their system model meet given requirements. In case, the system model is parametrised it can also clarify for which configurations, the requirements will be met.

We also study communication properties in a component-based design approach. Component-based design is favourable when designing large and complex systems as it splits the system into communicating components. This gives a better overview, but causes a new problem, namely ensuring that the composition of the components is correct. We study a series of communication properties of asynchronously composed Petri nets and prove that these channel properties are preserved by asynchronous composition. This means that the modelling framework support incremental design. Furthermore, we prove that all the properties are decidable for our asynchronous I/O Petri nets.

Finally, we extend the asynchronous I/O Petri nets with modalities. This gives us a formal modelling framework with the advantages from MTS, where we can still reason about channel properties even during the refinement process. With all these results, we present a methodology for the specification of distributed systems that supports incremental design, encapsulation of components and stepwise refinement.



# Dansk sammenfatning

Formelle metoder er ikke altid lige anvendelige i software designprocesser. Vi ønsker at forbedre dette ved at udvide eksisterende modelleringsformalismer. Hovedproblemet er, at de eksisterende formalismer til systemspecifikation ikke altid er i stand til at modellere de reelle systemkoncepter i det rigtige abstraktionsniveau. Derudover skalerer de eksisterende frameworks ikke godt nok til at de kan anvendes på store og mere komplekse systemer.

Vi studerer udvidelser af *Modale Transitionssystemer* (MTS), et framework til systemmodellering, der understøtter system-raffinering. Det er fordelagtigt i designprocesser, hvor et system er designet gradvist fra en overordnet specifikation ned til en komplet systemspecifikation med en fastlagt implementering.

Til at starte med udvider vi MTS-frameworket med en række konfigurationsparametre. Disse parametre gør det muligt at modellere persistente valg i systemmodellen. Dernæst udvider vi modelleringssproget med tid og priser, som kan bruges til at analysere omkostningerne ved at bygge et system og køre systemet. Det kan hjælpe udviklere, allerede tidligt i designfasen, med at sikre, at systemet overholder budgetterne. Til sidst designer vi en logik til MTS-frameworket. I denne logik kan man udtrykke formelle systemkrav og verificere, hvorvidt systemmodellen opfylder disse. Hvis systemmodellen har konfigurationsparametre, så viser vi også, hvordan man kan bruge logikken til at afgøre for hvilke konfigurationer af systemet kravene er opfyldt.

Vi studerer også kommunikationsegenskaber i komponent-baserede designmetoder. Komponent-baserede designs er favorable, når man udvikler større og komplekse systemer, da systemet bliver delt op i mindre moduler, der kommunikerer. Det kan være med til at give et bedre overblik, men det medfører også et nyt problem, nemlig det at modulerne skal kunne kommunikere korrekt. Derfor studerer vi en række kommunikationsegenskaber på asynkront sammensatte systemer. Vi betragter systemer modelleret som asynkrone I/O Petri nets, og vi beviser, at alle vores kommunikationsegenskaber er afgørbare i dette setup. Derudover beviser vi, at de egenskaber et modul har, ikke ændres, hvis det bliver sat sammen med et andet modul. Det betyder, at vores framework understøtter trinvis design.

Til sidst udvider vi vores asynkrone I/O Petri nets med modaliteter. Det giver os et formelt modelleringsframework med fordelene fra MTS, hvor vi kan undersøge kommunikationsegenskaber selv under raffinering af modellen. I alt giver det os en metode til at designe distribuerede systemer, der understøtter, trinvis design, indkapsling af moduler og trinvis raffinering.



# Acknowledgements

First I would like to express my sincere gratitude to my supervisor Jiří Srba. Thank you for giving me the best possible introduction to the field of formal verification during my master studies and for believing in me as a Ph.D. student. You are one of the very most thorough guys I know, and this has been a great help for me during my Ph.D. You have always been helpful, supporting and encouraging when I needed it.

An equal big thank you goes to my co-supervisor Kim Guldstrand Larsen. I am thankful for our great discussions and the fact that you have always been able to provide extra insight and additional ideas whenever I asked for your opinions.

Next I would like to send a big thank you to Rolf Hennicker for making my six month visit at Ludwig-Maximilians-Universität in Munich a very inspiring and productive stay. In this context, I would also like to thank Annabelle Klarl, Andreas Schroeder, Phillip Mayer, Christian Kroiss and Sebastian Bauer for showing me Munich, the Alps, feuerzangenbowle and in general making my visit very nice and pleasant.

During my stay in Munich I had the pleasure of working with Serge Haddad. I would like to thank you for our fruitful discussions and your deep insight in Petri Nets, which has been a great help.

I would also like to thank Nikola Beneš and Jan Křetínský for our collaboration. You guys helped me get back on track when I changed the focus of my Ph.d. study. I have really enjoyed working with you and travelling with Jan in Curaçao and Venezuela.

In December 2011, I visited ISCAS, Beijing. I would like to thank Liu Xinxin for arranging everything and for our interesting discussions.

I would also like to thank the F-klub and Hal9k. It has been vital for me to be able to flee to these places when my Ph.d. study got too stressful and troublesome. Thanks to the members of “Madklubben”, for our fun and great lunches. A special thank you to my friends and colleges Mads Chr. Olesen and Alex Birklykke, who have been in the same situation as me. I am thankful for our discussions about the life of being a Ph.D. student, this have helped me survive the hard times of my study.

Thank you, mom and dad for supporting me and letting me choose my own future. You have always inspired me to curiosity, wonder and inventiveness. Thanks to my siblings, Jesper and Louise for all our crazy ideas.

Finally, a huge thank you Sisse, my dear girlfriend. Without you, I doubt that I would have been able to succeed in completing this Ph.D. You are always supportive when I am stressed and worn out. But most of all thank you for making my everyday life fun and cheerful. You are the best.



September 2013





# Mandatory Page

**Title:** Modal and Component-Based System Specifications

**Author:** Mikael H. Møller

**Supervisor:** Associate Professor Jiří Srba

**Co-Supervisor:** Professor Kim G. Larsen

## Published Papers:

### Parametric Modal Transition Systems

Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, Jiří Srba

Published in proceedings of ATVA'11, volume 6996 of LNCS, pages 275–289, Springer-Verlag 2011.

### A Characterizing Logic for Boolean Modal Transition Systems

Mikael H. Møller, Kim G. Larsen, Xinxin Liu

Published in the local proceedings of MEMICS'12.

### Dual-Priced Modal Transition Systems with Time Durations

Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, Jiří Srba

Published in proceedings of LPAR'12, volume 7180 of LNCS, pages 122–137, Springer-Verlag 2012.

### Channel Properties of Asynchronously Composed Petri Nets

Serge Haddad, Rolf Hennicker, Mikael H. Møller

Published in the proceedings of Petri Nets'13, volume 7927 of LNCS, pages 369–388, Springer-Verlag 2013.

### Specification of Asynchronous Component Systems with Modal I/O-Petri Nets

Serge Haddad, Rolf Hennicker, Mikael H. Møller

Published in the local pre-proceedings of TGC'13. Planned to be published in the post-proceedings of TGC'13, that will appear as a volume in Springer's LNCS series.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.



# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Motivation</b>	<b>3</b>
1.1	Safety-critical Systems . . . . .	3
1.2	Model Checking . . . . .	4
<b>2</b>	<b>Modal System Specification and Refinement</b>	<b>7</b>
2.1	Labelled Transition Systems . . . . .	7
2.2	Modal Transition Systems and Refinement . . . . .	8
2.3	Extending Modal Transition Systems . . . . .	10
2.4	Logics for Extended Modal Transition Systems . . . . .	14
2.5	Time and Cost in Modal Transition Systems . . . . .	16
2.6	Main Contributions . . . . .	18
<b>3</b>	<b>Component-Based Development</b>	<b>19</b>
3.1	Synchronous and Asynchronous Composition . . . . .	19
3.2	Channel Properties . . . . .	22
3.3	Asynchronous I/O Petri Nets . . . . .	23
3.4	Channel Properties of Asynchronous I/O Petri Nets . . . . .	27
3.5	Modal Asynchronous I/O Petri Nets . . . . .	32
3.6	Main Contributions . . . . .	36
<b>4</b>	<b>Publication Overview</b>	<b>39</b>
<b>II</b>	<b>Papers</b>	<b>47</b>
<b>A</b>	<b>Parametric Modal Transition Systems</b>	<b>49</b>
1	Introduction . . . . .	50
2	Parametric Modal Transition Systems . . . . .	51
3	Complexity of Modal Refinement Checking . . . . .	57
4	Conclusion and Future Work . . . . .	68
<b>B</b>	<b>A Characterizing Logic for Boolean Modal Transition Systems</b>	<b>69</b>
1	Introduction . . . . .	70
2	Boolean Modal Transition Systems . . . . .	71

3	Characterizing Logic . . . . .	74
4	Parametric Modal Transition System . . . . .	77
5	Conclusion and Future Work . . . . .	80
<b>C</b>	<b>Dual-Priced Modal Transition Systems with Time Durations</b>	<b>81</b>
1	Introduction and Motivating Example . . . . .	82
2	Modal Transition Systems with Durations . . . . .	85
3	Dual-Price Scheme . . . . .	88
4	Complexity Results . . . . .	89
5	Conclusion and Future Work . . . . .	99
<b>D</b>	<b>Channel Properties of Asynchronously Composed Petri Nets</b>	<b>101</b>
1	Introduction . . . . .	102
2	Asynchronous I/O-Petri Nets . . . . .	104
3	Compositional Semantics . . . . .	109
4	Channel Properties and Their Compositionality . . . . .	112
5	Decidability of Channel Properties . . . . .	123
6	Conclusion and Future Work . . . . .	129
<b>E</b>	<b>Specification of Asynchronous Component Systems with Modal I/O-Petri Nets</b>	<b>131</b>
1	Introduction . . . . .	132
2	Illustrating Example . . . . .	133
3	Modal Asynchronous I/O-Petri Nets . . . . .	137
4	Modal Refinement . . . . .	141
5	Message Consuming Systems . . . . .	145
6	Conclusion and Future Work . . . . .	154

# Part I

## Introduction



# 1

## Motivation

During the last decade, software has become a regular part in all types of products, both consumer- and industrial scale products. Twenty years ago software were generally regarded as programs running on personal computers. Today however, many products rely on software running inside it. For instance, all new cars contain a software system that triggers emergency functionality on the basis of inputs from sensors in the car. Furthermore, coffee machines, dishwashers, elevators, heating/cooling systems and medical monitors are all examples of products that rely on software-based control systems. Such applications are grouped under the common term *embedded software*.

### 1.1 Safety-critical Systems

As with any other software application, embedded software can contain errors. Most people have experienced a computer program crashing, become unresponsive or doing something unexpected. Embedded software can have the same problems. As embedded software is not always directly visible, it can be hard to determine whether the system is running correctly. This can be extremely dangerous if the embedded software is responsible for triggering safety mechanisms in an emergency situation. We say that a system is *safety critical* if an error in the system can result in a costly outcome, cause injury or even lead to death. Software errors can occur due to various reasons. One of these reasons is that the size of software systems can make them so complex that it is hard to avoid errors.

Guaranteeing that a piece of software does not contain any errors and only behaves as intended is a non-trivial task. It is crucial to describe exactly how the system should function, but also specify the potential errors of the system. By formally defining the behaviour of the system, one can examine the system specification using mathematically-based techniques. Techniques building upon these concepts are known in software engineering as *formal methods*.

## 1. Motivation

Formal methods rely on a precise system specification expressed in a well-defined mathematical formalism. Mathematical reasoning can then prove the absence of the specified errors. Formal method techniques can be applied in several stages of a design process in software development. A software design process is a tool that helps developers make better software. The design processes define a number of steps that the development must go through, for instance requirement specification, system design, implementation etc. Model-based development is a design process centred around the model of a system. This central model can then be altered and refined throughout the first steps of the design process and used for comparison when testing the implemented software.

By expressing this central model in a mathematical formalism, we can apply formal methods in the development process and avoid errors early in the design phase. In order to do this, there must be a formalism capable of expressing the behaviour and concepts of the system. If a system must have deadlines and timing constraints in order to meet its requirements, then the modelling framework must also be able to express such timing features. Further, if a system manages a resource of some kind, the models should also include such mechanisms. However, having complex modelling formalisms will also make the mathematical reasoning harder and often impossible.

Complexity is a significant problem in the field of formal methods, but there are “divide and conquer” techniques that attempt to alleviate the problem. The idea is to split the system model into simpler components and then analyse them individually. When splitting a model into components, a new problem arises: how do one ensure that the composition of these components will meet the overall requirements? We need to make sure that they cooperate in a correct manner.

Some systems are constructed by individual components communicating together. Such systems are known as distributed systems. For instance, an intelligent heating control for a house is a distributed system. It consists of heating regulators that open and close the radiators, thermometers that report the room temperature, window sensors telling if the window is open and a control unit. The components communicate on a wireless connection, so it is essential that the communication is reliable.

The main subject of this thesis is to enhance the ability to use formal methods in system design of embedded systems. This is done by extending and improving existing techniques for system specification, in order to make them more applicable in a development process of a real system.

## 1.2 Model Checking

The contribution of this thesis lies within *model checking*, which is a formal method. Given a formal specification of a system and precise requirements, model checking is a mathematically-based technique that verifies whether the system fulfils its requirements. The general aspects of this approach were introduced independently by Emmerson and Clarke [EC80; CE82] and Queille and Sifakis [QS82]. Clark, Emmerson and Sifakis have all made a substantial contribution to model checking and were honoured with the Turing Award in 2007. The model checking technique can be seen as a three phase method.



First, a model of the system must be defined in a mathematical formalism. Second, the system requirements and properties must be expressed in a formal manner. Given such formal model and requirement, a model checking algorithm can check whether the property holds for the system model. The algorithm gives its answer on the basis of a complete search of the model in which the algorithm checks whether the requirements are met throughout the model. If there is a part of the system where a requirement does not hold, model checking will be able to provide information about this.

Model checking does not examine an actual system, but a model of a system. This simplifies the complex task of checking whether a property is satisfied by a system. Therefore, the model must be as close to the system as possible. If the system model lack a vital detail, the results of the model checking might be useless. In order to specify a model that captures all the core concepts of a system, the modelling formalism must be expressive enough. For instance, consider an emergency system that brakes trains that are too close to each other on the same track. Using model checking, it is possible to verify that both trains must be stopped within 1 minute if they get too close. Additionally, model checking can make sure that two trains will never get closer than 2 km. To do this, however, the modelling formalism must be able to model timing constraints and quantities such as distance. Furthermore, some system models can be infinite which can make the model checking problematic or even undecidable, for an overview of this see [BE01; Mol96].

Since model checking mathematically proves that a system model has certain properties, developers can trust this method. For example, it is common to aim for a system that has no deadlocks. A deadlock in a system is a situation where the system cannot progress in any way. Using model checking, the developer can confirm that there are no deadlocks in the system analysed. Hence, developers can use model checking to prove the absence of error in the system model. However, this does not guarantee that the actual implementation of the system will be error free, as it only reason about a model of the system. On the other hand, it can still be better than well-known techniques such as testing and simulation, where developers monitor a large number of executions of a systems. These executions can reveal problems, but there is no way to make sure that these executions will reveal all errors. On the other hand, to prove absence of errors using model checking, the developers must know the possible errors and be able to express them.



# 2

## Modal System Specification and Refinement

In order to use formal methods in a system design process it is crucial to use a formal modelling framework that fits the needs and is expressive enough to model all the abstracts of the system being designed. Initially in a design process the system specification can be loose and abstract, meaning that some features or system behaviour is not completely settled. During the design process these specification are refined into more and more precise specifications. There exist formal models that support this style of system developing, however they are not always expressive enough.

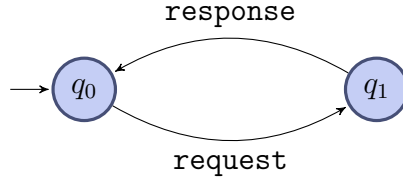
### 2.1 Labelled Transition Systems

One of the most used modelling formalisms within formal methods is the *Labelled Transition System* (LTS), introduced by Keller [Kel76] in 1976 and later applied by Plotkin [Plo81] in 1981. The basic idea of an LTS is to give a description of a system behaviour by a set of states and a set of labelled transitions between these states. The labelled transitions describe the possible actions the system can perform. By performing an action the system might change its state. LTSs are represented graphically by a directed-graph structure, where each state is a node and each transition is an arc with an attached label.

For instance, Figure 2.1 shows a transition system specification of a very simple web-service, where you can send a request for data to a server, the server can then respond to the request by sending back the data. The small arrow going into state  $q_0$  illustrates the initial state of the system, in which state the system is started.

As this example is very simple, there is actually only one execution of this system. An execution of such system is a sequence of transitions. In the web-service example the only one possible execution is the infinite and alternating sequence of **request** and **respond** transitions.

## 2. Modal System Specification and Refinement



**Figure 2.1:** Simple transition system specification of a file web-service.

Formally an LTS is defined as follows.

**Definition 1** (Labelled Transition System). A labelled transition system (LTS) is a 4-tuple  $(Q, \Sigma, \longrightarrow, q_0)$  such that

- $Q$  is a set of states,
- $\Sigma$  is a set of labels,
- $\longrightarrow \subseteq Q \times \Sigma \times Q$  is a set of labelled transitions, and
- $q_0$  is the initial state.

## 2.2 Modal Transition Systems and Refinement

Even though LTS is a very useful formalism it is not always that well suited for software development processes. An LTS can be a very precise low-level system specification. However, in a development process, like model-driven development, it is natural to start with a loose and simple specification of the system and then refine the specification during the process. For example, by removing optional features. In such case LTSs are too strict as they cannot express that some parts are allowed but not necessarily required.

In order to fix this problem, Larsen and Thomsen introduced the *Modal Transition System* (MTS) [LT88], to integrate model checking more easily into model-driven development. An MTS is like an LTS, the only difference being that it has two types of transitions, *may-transitions* and *must-transitions*, whereas an LTS only has one type. May-transitions represent allowed, but not required, system behaviour and must-transitions represent mandatory behaviour. MTSs are represented graphically like LTSs with may-transitions drawn as dashed arcs and must-transitions drawn as solid arcs.

In Figure 2.2a, a modal transition system specification of a web-service is shown. This specification is an extension of the LTS in Figure 2.1. The MTS specification aims at capturing two types of web-services a secure one and a non-secure one. The system *may* receive a **request** for data in two different ways, where it in one case *must validate* the client, before it *must* reply with the requested data. The main idea of the specification is to model that the web-service can either be secure or not. This is achieved by having

the two **request** transitions as may-transitions in the specification. Formally an MTS is defined as follows.

**Definition 2** (Modal Transition System). A modal transition system (MTS) is a 5-tuple  $(Q, \Sigma, \dashrightarrow, \longrightarrow, q_0)$ , such that

- $Q$  is a set of states,
- $\Sigma$  is a set of labels,
- $\dashrightarrow \subseteq Q \times \Sigma \times Q$  is a set of labelled may-transitions,
- $\longrightarrow \subseteq \dashrightarrow$  is a set of labelled must-transitions, and
- $q_0$  is the initial state.

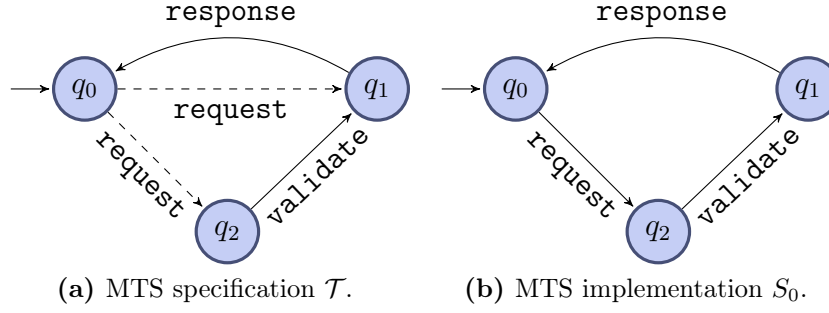
Note that each must-transition is also a may-transition. This is done for consistency, as it unclear what it means if a transition is required but not allowed. On the other hand, when starting to compose modal transition systems it can be the case that two MTS specifications do not fit, i.e. thier composition becomes incositent.

The main concept of MTSs is that a system specification can be loose in the sense that the complete system behaviour is not yet completely fixed, like with the **request** transitions in Figure 2.2a. An MTS containing may-transitions is called an *abstract* system specification. Abstract specifications can be refined into *implementations* by removing some may-transitions, i.e. leaving out optional behaviour, and converting other may-transitions into must-transitions, i.e. making optional system behaviour mandatory. An implementation is an MTS where every transition is a must-transition, i.e. the system behaviour is fixed. In our example we can for instance convert the may-transition **request** from  $q_0$  to  $q_2$  into a must-transition and remove the may-transition **request** from  $q_0$  to  $q_1$ . This will give us the MTS implementation in Figure 2.2b. Note that any MTS implementation is actually an LTS.

As there are several ways to refine the abstract MTS specification, each MTS yields a set of different implementations. For instance the LTS in Figure 2.1 is also an implementation of  $\mathcal{T}$  in Figure 2.2a. Observe that the must-transition **validate** has been removed in that last implementation. The only way we can remove this transition is by removing the may-transition **request** going from  $q_0$  to  $q_2$ . Then we can no longer reach  $q_2$  from the initial state, which mean that  $q_2$  and its outgoing transition is not contributing to the specification any more, thus they can be removed.

We have briefly mentioned how to refine an abstract specification into an implementation. However, the process of specification refinement can be done in a sequence of steps where each abstract specification is refined into a less abstract specification. In each refinement step there are two requirements that must hold to ensure that the refinement is valid: (1) everything that is mandatory in the abstract specification must also be mandatory in the refined specification, and (2) the refined specification cannot allow behaviour that is not allowed by the abstract specification. The process of refin-

## 2. Modal System Specification and Refinement



**Figure 2.2:** Examples of modal transition systems.

ing an MTS is formally defined as *modal refinement* and is formally defined as follows.

**Definition 3** (Modal refinement). Let  $\mathcal{S} = (Q_{\mathcal{S}}, \Sigma, \dashrightarrow_{\mathcal{S}}, \rightarrow_{\mathcal{S}}, q_{\mathcal{S}}^0)$  and  $\mathcal{T} = (Q_{\mathcal{T}}, \Sigma, \dashrightarrow_{\mathcal{T}}, \rightarrow_{\mathcal{T}}, q_{\mathcal{T}}^0)$  be two MTSs with the same set of labels  $\Sigma$ . A relation  $R \subseteq Q_{\mathcal{S}} \times Q_{\mathcal{T}}$  is a modal refinement relation between  $\mathcal{S}$  and  $\mathcal{T}$  if for all  $(q_{\mathcal{S}}, q_{\mathcal{T}}) \in R$  and for all  $a \in \Sigma$ :

1.  $q_{\mathcal{T}} \xrightarrow{a}_{\mathcal{T}} q'_{\mathcal{T}} \implies \exists q'_{\mathcal{S}} \in Q_{\mathcal{S}} . q_{\mathcal{S}} \xrightarrow{a}_{\mathcal{S}} q'_{\mathcal{S}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R,$
2.  $q_{\mathcal{S}} \dashrightarrow_{\mathcal{S}} q'_{\mathcal{S}} \implies \exists q'_{\mathcal{T}} \in Q_{\mathcal{T}} . q_{\mathcal{T}} \dashrightarrow_{\mathcal{T}} q'_{\mathcal{T}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R,$

We say that  $\mathcal{S}$  is a modal refinement of  $\mathcal{T}$ , written  $\mathcal{S} \leq_m \mathcal{T}$ , if there exists a modal refinement relation  $R$  between  $\mathcal{S}$  and  $\mathcal{T}$  such that  $(q_{\mathcal{S}}^0, q_{\mathcal{T}}^0) \in R$ .

Note that modal refinement does not require a one to one mapping between states in the two specifications, it only considers the allowed and required behaviour in a given state. Thus, the MTSs in Figure 2.3 are all refinements and implementations of the MTS in Figure 2.2a. It might seem strange that the MTS  $\mathcal{S}_2$  in Figure 2.3b is a valid implementation, but this is actually the case since all outgoing transitions from  $q_0$  are may-transitions and can be omitted.

### 2.3 Extending Modal Transition Systems

MTS is a simple formalism and it is not always expressive enough to model complex systems. It models the behaviour of a system by modelling the different states of the system and transitions between these different states. Each transition can then be either required or optional. As MTS is that simple, there is no way to model that two features are exclusive or give timing constraints on the transitions. This means that the MTS formalism can be limited in its applications.

Even though, the specification  $\mathcal{T}$  in Figure 2.2a looks natural, it is not as precise as

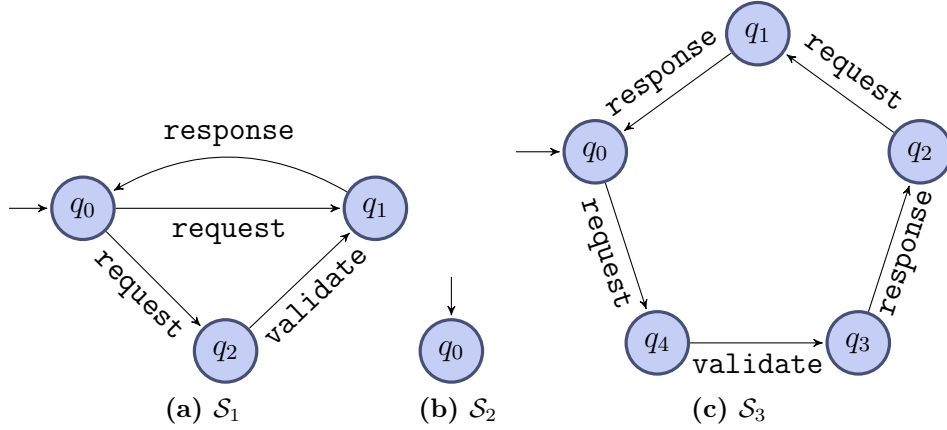


Figure 2.3: Examples of MTS implementations of  $\mathcal{T}$ .

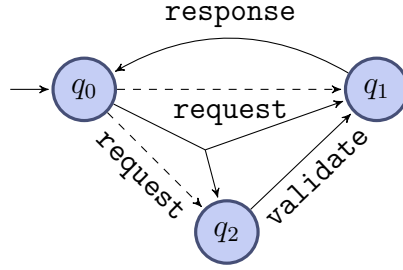
we can hope for. The three implementations  $\mathcal{S}_1$ ,  $\mathcal{S}_2$  and  $\mathcal{S}_3$  in Figure 2.3 are actually all problematic. Recall that the main concept of the specification  $\mathcal{T}$  is to specify a web-service that can be either secure or non-secure. The only intended implementations are those shown in Figure 2.1 and Figure 2.2b.

There is a unique problem with each of the implementations  $\mathcal{S}_1$ ,  $\mathcal{S}_2$  and  $\mathcal{S}_3$ . In  $\mathcal{S}_1$  both **request** transitions has been implemented. This is a problem as we do not know what happens when the server receives a request. It might not validate the client even though that might have been what we wanted. The problem is that there is no way to express an exclusive choice between transitions. In  $\mathcal{S}_2$ , it is again a problem that both **request** transitions are may-transitions, as we do not need to implement them. However, having them both as must-transitions will not fix the problem, as seen with  $\mathcal{S}_1$ . The problem is that we can not express that we must implement at least one of the transitions. Finally in  $\mathcal{S}_3$  the problem is that the server is alternating between being a secure and non-secure web-server. Every second request requires validation of the client. This is not an expected behaviour.

Since the introduction of MTS, several extensions of the formalism have been proposed [LX90; FS08; BK11], and they solve some of the issues mentioned above. In [LX90] Larsen and Xinxin present an extension called *disjunctive modal transition system* (DMTS). The extension is disjunctive in the way that must-transitions can be a disjunction of must-transitions with the same source state. An example of a DMTS is shown in Figure 2.4, it is an extension of our previous MTS example. The new full lined arc that is splitting from  $q_0$  to  $q_1$  and  $q_2$  illustrates the disjunction between the **request** transitions. This means that at least one of the **request** transitions must be part of any implementation. Note that the underlying may-transitions are kept in the figure. By utilizing this disjunction technique on the **request** transitions we can avoid the problematic implementation  $\mathcal{S}_2$ . However,  $\mathcal{S}_1$  and  $\mathcal{S}_3$  are still valid implementations that we may wish to eliminate.

Recently, Beneš and Křetínský suggested a generalisation of the MTS framework, Transition Systems with Obligations (OTS) [BK11]. Their idea is to add constraints or

## 2. Modal System Specification and Refinement



**Figure 2.4:** Disjunctive MTS specification  $\mathcal{T}$ .

*obligations* on transitions instead of having may-transitions and must-transitions, as this is more expressive than MTS. The transition modalities in OTSs are expressed by an obligation, saying which transitions are allowed and how to combine them in valid implementations. Therefore, there is only one type of transition, as the obligation function can express both may and must modalities. An obligation is a positive Boolean formula over transitions bound to a state. The obligation specifies the allowed combinations of outgoing transitions from a state, in the way that each truth assignment of the obligation is a valid implementation.

An OTS specification of the previously introduced web-service is shown in Figure 2.5. Even though OTSs only have one type of transitions we still use full and dashed arcs for transitions. Transitions drawn as full lines are those that must be in any implementation, i.e. the obligation on the source state requires it. For instance, the obligation for  $q_1$  is  $(\mathbf{response}, q_0)$  meaning that the **response** must be there, thus drawn as a full line. On the other hand, the obligation for  $q_0$  is  $(\mathbf{request}, q_2) \vee (\mathbf{request}, q_1)$ , which specifies that at least one of the **request** transitions must be present in any refinement. If we needed a classical may-transition equivalent to those in an MTS we can do that by leaving the transition out of the obligation, then the obligation can be satisfied independently of that transitions.

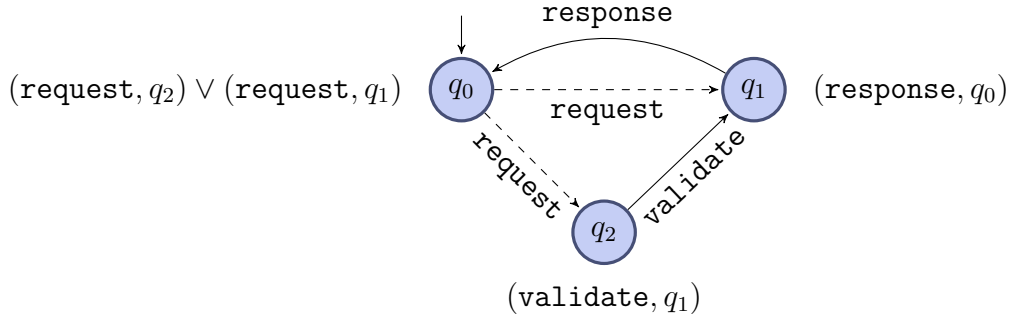
Note that it is important that each formula is satisfiable by the present transitions, as it will be inconsistent otherwise. The OTS formalism is in fact a generalisation of the MTS, but it is just as expressive as DMTS. This is the case, as a DMTS is just a OTS with the obligations in *conjunctive normal form* and any positive Boolean formula can be converted to a formula in CNF. However, such conversion might lead to an exponential blow-up in size. Thus OTS will not help us to get rid of the two problematic implementations  $\mathcal{S}_1$  and  $\mathcal{S}_3$  shown in Figure 2.3.

We started by extending the OTS framework to support arbitrary Boolean formula, and call it Boolean MTS (BMTS). With BMTS we can express an exclusive choice between transitions, which is what we need to avoid  $\mathcal{S}_1$  as a valid implementation. Such BMTS will be identical to the one in Figure 2.5 except that the obligation for  $q_0$  will be

$$((\mathbf{request}, q_2) \wedge \neg(\mathbf{request}, q_1)) \vee (\neg(\mathbf{request}, q_2) \wedge (\mathbf{request}, q_1)) ,$$

meaning that each time we are in a  $q_0$  we can only implement at most one of the **request** transitions, and we have to implement at least one of them. The implementation  $\mathcal{S}_1$  is





**Figure 2.5:** Transition system with obligations specification  $\mathcal{T}_2$ .

not a valid implementation of such a specification. However,  $\mathcal{S}_3$  is still valid, as we have not violated the obligation on  $q_0$ .

The problem with implementation  $\mathcal{S}_3$  stems from the fact that we try to make a system specification that captures 2 different web-services, a secure and a non-secure. System development is not always about designing a single piece of software. Companies often release a series of similar products. These products can be of different built-quality, have a different physical design and vary in functionality. Sometimes, two different products are built upon the same hardware, but with different software embedded in the product. The software can provide the core functionality of both products, as well as enable unique features that are only available in the expensive version. A concrete example of this is the GPS navigator. Some manufacturers of GPS navigators offer a series of products on the basis of one single hardware design. Each of the products can assist with navigation using its GPS module, but only some products can provide additional information such as heavy traffic or roadwork. The strategy of developing software for such a family of products is known as a Software Product Line (SPL) [CN01].

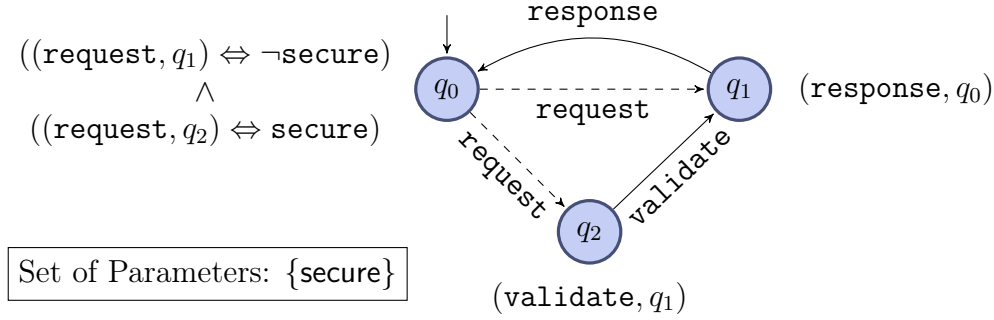
Modal transition systems are particularly interesting in the context of software product lines. There is a natural correspondence between optional features in SPL and may-transitions in MTS. In SPLs, System developers can specify one abstract specification that fits all the products in the SPL, like in our small web-service example. Furthermore, as it is possible to prove that certain system requirements are satisfied by all products in the SPL by applying model checking to the abstract system model, it is clearly a significant benefit which can save developers some time [BCK11; Ant+08].

However, the MTS formalism in itself is not directly suited for SPL since it can be refined into implementations that do not correspond with any product in the SPL, as shown in our example. BMTS solves two out of our three problematic implementations. Nevertheless, we still need the ability to express a persistent choice, where we can set which features should be present and how this affects the modalities on the transitions.

Therefore, we suggest a generalisation of the MTS family, Parametric MTS (PMTS). A PMTS is basically a BMTS with a set of Boolean parameters that can be used in the obligations. For instance, in the example shown in Figure 2.6 we can introduce the parameter *secure*. This parameter is now used in the obligation for state  $q_0$ . This obligation restricts the possible implementations, based on how we set the parameter

## 2. Modal System Specification and Refinement

secure. If `secure` is set to true, we can only satisfy the obligation for  $q_0$  by implementing  $(\text{request}, q_2)$ . This forces requests to the web-server to be validated before a response is sent back. Furthermore, if `secure` is set to false, we cannot implement  $(\text{request}, q_2)$  as this will violate the obligation on  $q_0$ . Note that parameters can be used in more than one obligation, even though we only use `secure` in the obligation for  $q_0$ .



**Figure 2.6:** Parametric MTS specification  $\mathcal{T}_3$ .

Unfortunately, as we increase the expressivity of the modal specification formalism, the complexity of checking modal refinement also increases. However, we have proven that even for our most general formalism PMTS the problem of modal refinement checking remains decidable.

Similar to this approach the authors of [Cla+10] introduce Feature Transition Systems (FTS). In FTSs, each transition is labelled with its corresponding feature and each combination of features then yields an implementation. This means that an FTS cannot be refined, you can only select a subset of features that combined yield an implementation. This means that it is not possible to give an abstract specification of a feature. In relation to our example an FTS of the web-service would have the same set of states and transitions. The two `request` transitions would each be attached with their own feature, non-secure and secure respectively. Even though there is a close relationship between the features of FTSs and the parameters in PMTSs, the parameters are a more general notion, that can be used to represent not only features, but for instance also the presence of a particular piece of hardware.

It is important to notice that MTS, disjunctive MTS, OTS and Boolean MTS are all sub-classes of parametric MTS. We study the computational complexity of checking modal refinement between these different sub-classes of parametric MTS.

**Result 1.** *We provide a comprehensive overview of the complexity of refinement checking on different sub-classes of parametric MTS. The complexity ranges from P-complete in the simplest case to  $\Pi_4^P$  in the most general setting.*

## 2.4 Logics for Extended Modal Transition Systems

Given a PMTS and a system requirement, it is convenient to deduce which instantiations of the parameters yield implementations that fulfil the requirement. In order to achieve

this, we define two new logics one for BMTS and one for PMTS. Both are developed on the basis of the well known Hennessy Milner Logic with recursion [Lar88]. With the logic for BMTS we can express certain requirements for the specification and verify whether they are met. For example, consider the requirement that our simple web-service must be able to do a **request** from the initial state. Such requirement can be expressed in our logic as the following formula:

$$\langle \text{request} \rangle \text{tt} .$$

The “diamond” shaped brackets states that we must be able to do a **request** in any implementation of the specification, and the **tt** in the end just means that we only care about the first step in this requirement. With such requirement we can formally state the question whether a state e.g.  $q_0$  of  $\mathcal{T}_2$  in Fig. 2.5 satisfies the requirement:

$$q_0 \models \langle \text{request} \rangle \text{tt} .$$

In fact  $q_0$  satisfies the requirement even though there is no fixed outgoing must-transition labelled **request**. This is the case as the obligation for  $q_0$  requires that we at least implement one the two **request** transitions going out the state. Note that by satisfying this requirement, we also ensure that  $\mathcal{S}_2$  in Fig. 2.3b is not a valid implementation of our specification.

As another example, consider the following formula

$$[\text{request}] \langle \text{validate} \rangle \text{tt} .$$

In order to satisfy this requirement, a system specification may be able to do a **request** and when ever it can do a **request** it must be able to do a **validate** afterwards. This is not the case for the  $\mathcal{T}_2$  in Fig. 2.5, as we can implement the **request** going from  $q_0$  to  $q_1$ , i.e. the system in Fig. 2.1 is a valid implementation, thus  $\mathcal{T}_2$  does not satisfy the requirement in all its implementations.

**Result 2.** *We present a characteristic formula for a BMTS specification and prove that our logic characterises modal refinement on BMTS. Meaning that given two specifications  $\mathcal{S}$  and  $\mathcal{T}$  we can express  $\mathcal{T}$  as a logic formula  $D_{\mathcal{T}}$  such that*

$$\mathcal{S} \models \mathcal{T} \text{ iff } \mathcal{S} \leq_m \mathcal{T} .$$

Now consider PMTS instead of BMTS specification. We define the logic for PMTSs such that a formula is satisfied if there exists a configuration of the parameters such that the requirement is met. But more interestingly we can also compute all possible configurations that fulfil the requirement. If we want to know how to set the parameters of  $\mathcal{T}_3$  in Fig. 2.6, such that valid implementations must always perform a **validate** after a **request**, we can check the following:

$$\mathcal{T}_3 \models_P [\text{request}] \langle \text{validate} \rangle \text{tt} .$$

An answer to such a problem is a set of instantiations of the parameters that yield a BMTS for which each implementation satisfies the requirement. In the concrete example

## 2. Modal System Specification and Refinement

it is clear that if `secure` is true, then the requirement is met. Note that a PMTS with a fixed configuration of the parameters is actually a BMTS. Similar to our approach, the authors of the FTS formalism [Cla+10] have also presented a technique where they decide which implementations satisfy a given requirement, i.e. they can deduce which combinations of features fulfil the requirement.

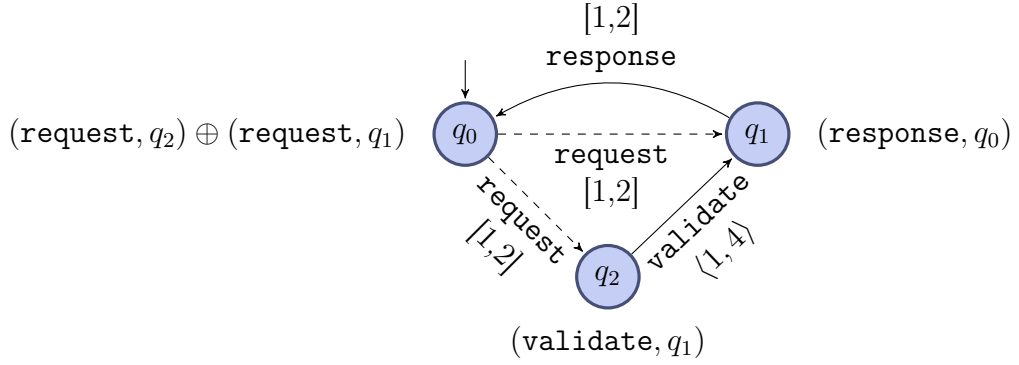
**Result 3.** *Given a PMTS specification and a requirement expressed in our logic we can decide for which parameter-configurations the requirement will be fulfilled.*

## 2.5 Time and Cost in Modal Transition Systems

Having a precise and formal system specification as a system model can enhance a system development process in many ways. Formal system models are great for formal verification and usually give a better overview of the system being designed. Furthermore, if we extend a system model to carry information about cost and time resources, we can use the specification to reason about expenses related to implementing and running the system, and how including different features will affect these costs. We have made such dual-priced extension to the MTS family. We consider this extension on the BMTS. However, it does not rely on any special feature of BMTS so we could have used it on PMTS without problems.

The extension consists of two parts; first, extend the model with time duration for each transition and secondly, add a price scheme on top of the model. We add the timings to the BMTS formalism by appending a time interval in each transitions. The time interval can either be *controllable* or *uncontrollable*, depending on whether we can refine the interval. In Fig. 2.7 we have extended our running example of a web service with time intervals. Note that the interval  $\langle 1, 4 \rangle$  on the `validate` transition is different from the rest. This is a controllable interval, that we can refine the interval, either by making it uncontrollable or narrowing the interval. Developers can use controllable intervals to model the lower- and upper-bound of the duration a transition which is not yet completely specified. We allow uncontrollable intervals and not just a fixed time duration as some transition might depend on parts we cannot control. For instance, we model the `request` and `response` transitions with a uncontrollable interval of  $[1, 2]$  time units as for instance network latency can affect the timing. All uncontrollable intervals stay the same in all refinements of a specification.

Besides this time extension we also add a *price-scheme*. A price-scheme consists of four parts, a cost per time unit for each transition, a set of possibly required hardware, a price tag for each hardware piece, and a hardware requirement list for each transition. An example of a price-scheme for our running example is shown in Fig. 2.8. The web-service can require one or two different servers. All three transitions require the Server but the `validation` transition also requires a `ValidationServer`. Thus we must purchase both servers if we plan to implement a secure web-service. Furthermore, we can see from the price-scheme that a `Validation` transition is cheaper per time unit than the two other transitions.

**Figure 2.7:** BMTS specification  $\mathcal{T}_4$ , with time durations.

Hardware	Price
ValidationServer	100
Server	50

Transition	Running Cost	Required Hardware
Validation	5	Server and ValidationServer
Request	10	Server
Response	10	Server

**Figure 2.8:** Price Scheme.

Given a modal specification annotated with time intervals and a price-scheme we can start reasoning about the cost of implementing a system and compute the expenses of running it. The implementation cost is computed on the basis of which hardware is required for a given implementation and its costs. The running expenses are computed as the worst-case long run average, on the basis of the transition durations and their running cost. With these computations we prove that for a fixed hardware budget we can decide which system implementation will yield a system with the cheapest running cost on a long run average. Furthermore, we can decide if there exists an implementation within a fixed hardware budget and running cost budget.

This method can be used to make sure a product stays within the desired price range. For instance, consider an embedded system that relies on battery-power. In such case, it is interesting to determine how implementing a feature affects the size of the battery needed. This can be done with our approach, by using the battery consumption as running-cost on transitions. Moreover, if the model contains information about the cost implementing a feature, one can verify which solution is the cheapest one and meets the basic requirements. However, we prove that the problem of finding such implementation is hard.

**Result 4.** *The problem of deciding whether there exists an implementation within a fixed hardware budget and running cost budget is NP-complete.*

## 2.6 Main Contributions

Before continuing with next part, we will just briefly summarize the main contribution of this thesis within the area of modal transition systems. First, we have introduced and defined the Parametric MTS formalism. Further, we have defined modal refinements of PMTS specification and done a complexity analysis and overview of modal refinement checking of PMTSs and subclasses hereof. Second, we have presented two logics to reason about Boolean MTSs and Parametric MTS. The first logic was proved to characterise BMTSs and the second can compute which configurations of a PMTS will yield a correct system specification with respect to a given requirement. Finally, we have presented a way to extend the MTS family of specifications with time and cost. This was done by adding a time duration to actions and including price scheme in the modelling formalisms. We showed how to use this additional information to check if there exists a valid system implementation within the budget.

# 3

## Component-Based Development

Component-based development is a popular method when designing software systems, as it is a method that has many advantages. The system design is split into individual modules such that the system is a composition of loosely connected components. This means that each component is isolated and responsible for a specific functionality of the system. The component-based development method is interesting also from a formal verification point of view. There is a significant benefit when it is possible to draw a conclusion for a composed system on the basis of an analysis of the individual components. However, it is very important that the composition of these components is done properly, especially the communication as this is the way they synchronise.

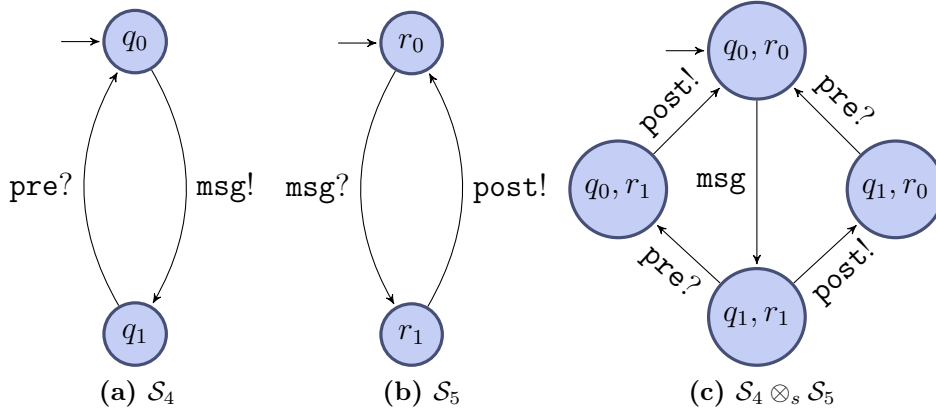
### 3.1 Synchronous and Asynchronous Composition

There are mainly two types of composition: synchronous and asynchronous composition. Synchronous composition is a composition where all participating components are aware of the actual communication at the same time, this can simplify verification of the communication. On the other hand, synchronous communication is not very suitable for larger and more complex systems, as components can block until the all participants of a synchronisation are ready. This blocking problem can be avoided by using asynchronous composition instead. In asynchronous composition a component can leave a message on a channel which can later be consumed by another component. However, when using asynchronous composition, the sending component can generally not know if the receiving component will ever take the message from the channel.

Formal verification can also be applied in a component-based development setting. In order to do this it is important that there is a suitable modelling formalism that can express the composition needed, and distinguish between internal behaviour of a component and the behaviour related to communication between the components. One way is to use the previously introduced LTS, but with each transition labelled by a label

### 3. Component-Based Development

from a special I/O-alphabet [AH05]. An I/O alphabet is disjoint union of *input* labels, *output* labels and *internal* labels. The input and output labels define the interface of the component, whereas internal labels are used on transitions that are not supposed to be visible outside of the component, but are needed to model the actual behaviour of the component. In our examples we will identify input labels by an appended “?” and output by an appended “!”. In Figure 3.1a we show a simple component with two transitions. It can send a `msg` message and then receive a `pre` message.



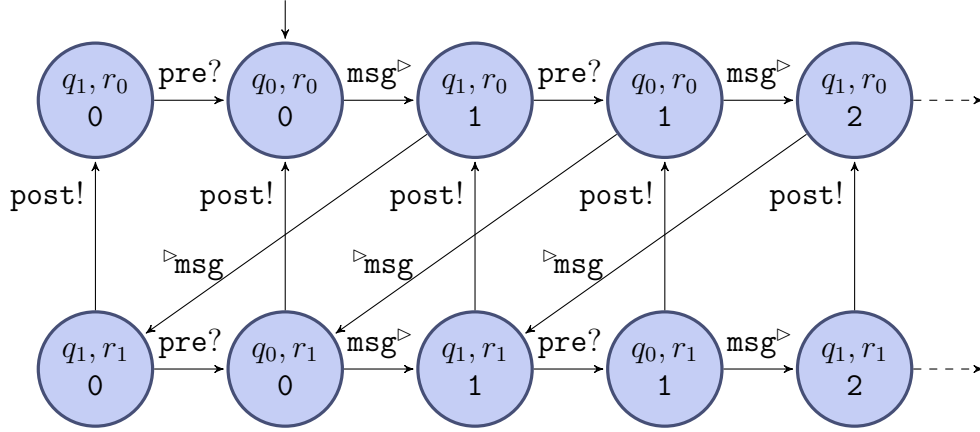
**Figure 3.1:** Example of synchronously composed LTSs.

The composition of two LTS components is also an LTS, but with each state being a pair of states, one from each component. The outgoing transitions of such state, is a union of the possible outgoing transitions in each component. However, in case of a synchronous composition, if a state has an input and an output transitions with the same label, then such two transitions are merged into one transition. As an example consider the LTS in Figure 3.1c, this is a synchronous composition of  $\mathcal{S}_4$  and  $\mathcal{S}_5$  in Figure 3.1a and 3.1b, denoted  $\mathcal{S}_4 \otimes_s \mathcal{S}_5$ . The initial state is the pair  $(q_0, r_0)$  as these are the initial states of  $\mathcal{S}_4$  and  $\mathcal{S}_5$  respectively. Further, as the two states have matching input and output transitions labelled `msg`, they are merged into an internal `msg` transition which goes to the state pair  $q_1, r_1$ , as these states are reached in the original component. The other transitions are interleaving, as they do not match any transitions in the opposite component. The example also shows one of the main problems with synchronous composition, namely that the communication is blocking. This is the case when  $\mathcal{S}_5$  is in  $r_1$  and not ready to receive a `msg` input, then  $\mathcal{S}_4$  will be stuck in  $q_0$  until  $\mathcal{S}_5$  is ready to synchronise. Note that the composition  $\mathcal{S}_4 \otimes_s \mathcal{S}_5$  is also an LTS component with I/O-alphabet  $\{\text{pre}, \text{post}, \text{msg}\}$ , where `pre` is an input label, `post` is an output label and `msg` is an internal label.

In an asynchronous composition of two LTS components no transitions are merged. Instead, a new *channel* is created for each matching input and output label pair in the two components. Such channel can be seen as a counter that counts the number of messages on a communication channel in the composition. Each output transition that has a new channel is converted to an internal transition and will increase the counter of



the corresponding channel and likewise the input transitions are converted to internal transitions and decrease the counter. However, an input transition is only available if the corresponding channel is not empty. In order to distinguish the internal transitions that were previously input transitions and output transitions, the labels are renamed by pre-pending an  $\triangleright$  on inputs and appending an  $\triangleright$  on outputs.



**Figure 3.2:** Example of an asynchronously composed LTS,  $\mathcal{S}_4 \otimes_{as} \mathcal{S}_5$ .

As an example consider the LTS in Figure 3.2, this is an asynchronous composition of  $\mathcal{S}_4$  and  $\mathcal{S}_5$ , denoted  $\mathcal{S}_4 \otimes_{as} \mathcal{S}_5$ . As  $\text{msg}$  is an output transition of  $\mathcal{S}_4$  and an input of  $\mathcal{S}_5$ . The two components will communicate on a new channel  $\text{msg}$ , and the output transition label is renamed to  $\text{msg}^\triangleright$  and converted to an internal label. Likewise, the input transition label is renamed to  $\triangleright\text{msg}$  and converted to an internal label. As in the synchronous composition each state contains a state pair, one from each of the two components. Additionally each state carries a counter representing the number of messages on the the channel  $\text{msg}$ . Thus, the initial state is  $q_0, r_0, 0$  as channels are initially empty. From the initial state only  $\mathcal{S}_4$  can move as  $\mathcal{S}_5$  is waiting for a message on the channel  $\text{msg}$ . Note that the composition is drawn so the horizontal transitions change the state of  $\mathcal{S}_4$  and the vertical and diagonal transitions change the state of  $\mathcal{S}_5$ . This asynchronous composition is also an LTS component in it self, with I/O-alphabet  $\{\text{msg}^\triangleright, \triangleright\text{msg}, \text{pre}, \text{post}\}$ . Unlike synchronous compositions, an asynchronous composition is not necessarily finite even if both components are finite. This happens because the channel is unbounded, i.e. it can hold infinitely many messages.

An unbounded and unordered channel is not the only way to model a communication channel. Alternatively, a channel can be modelled as a FIFO-buffer, but such channel requires more resources and might decrease the performance in larger systems. Furthermore, a finite LTS and a single infinite FIFO channel has full Turing power, as the FIFO-buffer can simulate the infinite tape of a Turing machine. This means that we cannot hope for any decidable results when using FIFO channels. Another alternative are *lossy channel systems*, with a FIFO channel that can loose messages. Surprisingly, some properties of the lossy channel become decidable [AJ96], however reachability and termination have a non-primitive recursive complexity [Sch02].

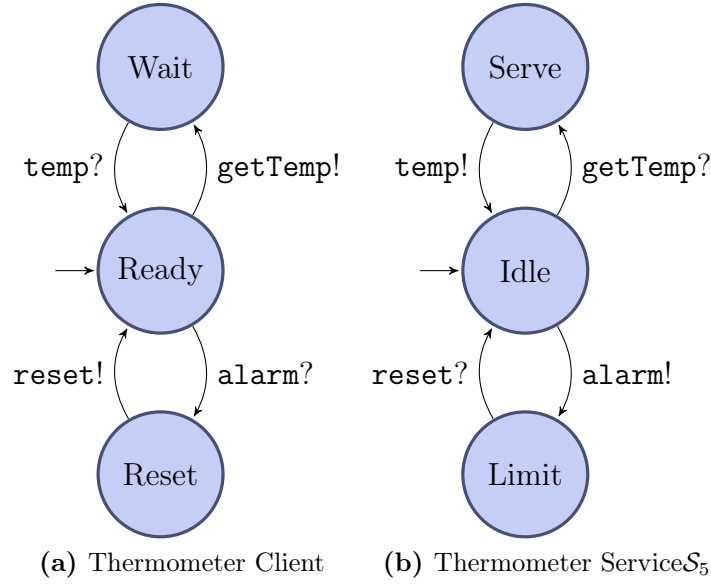
## 3.2 Channel Properties

When using component-based approach to system design it is important that the components are interacting and communicating correctly. Thus, it is interesting to verify certain properties of the communication. For synchronous communication these are properties like the blocking issue mentioned above and for asynchronous composition this means considering properties related to the communication channels. Overall these communication properties can be considered as general system properties which are not related to a particular system design or application.

In [BZ83] the authors consider a property called *specified reception*. The concept of the property is that a component must know how to handle an incoming message, i.e. the action taken on reception of a message must be specified. Consider the two components in Figure 3.3, they model a temperature service and a user that can request the current temperature. Furthermore, the temperature service can send an alarm to the user in case it reaches a certain limit. After an alarm the service must receive a reset message in order to go back to its idle state. In [BZ83] they consider finite components and compose them asynchronously with two FIFO channels, one for each direction, opposed to the unordered channels introduced earlier. Just by looking at the two components in Figure 3.3, such composition might seem fine. However, as the communication is asynchronous it can happen that the user sends a `getTemp` at the same time as the temperature service send out an `alarm`. Assuming that the channels were both empty prior to this, we have a situation with an `alarm` on the channel for the user and an `getTemp` on the channel for the temperature service. This is problematic as it is not specified how the user should deal with an alarm when he is waiting for a temperature from the service. This problem is characterized by the *specified reception* property, meaning that this particular composition does not have this property. Such property is interesting, but unfortunately the authors only consider finite components and they stick to FIFO channels. Furthermore, they do not allow the components to perform any internal behaviour before reception. This means that the property can be a too strong requirement for more complex systems.

Similarly, [AH05] studies the *compatibility* of components, stating whether two components fit together in a given environment. For synchronous systems this means that when one component is ready so send a message `a` then the other component must be in a state where it is ready to receive an `a`. Note that this is very related to the blocking mentioned earlier, nevertheless two components can be non-blocking but not compatible. This is the case if the sending component has a choice between two actions, where one of them can synchronise properly. For asynchronously composed systems such requirement must be adapted as messages do not arrive immediately at the receiving component. Instead, the authors of [HJK10] suggest that any message put on a channel must eventually be taken by the receiver, they call this the *buffered compatibility* property. Note that the two components in Figure 3.3 are neither compatible nor buffered compatible for the same reason as their composition does not have the specified reception property. This notion of compatibility is also interesting, but is only considered for finite components.

In [CF05] the authors focus on a different channel property, the *half-duplex* property.



**Figure 3.3:** Example of two LTSs modelling a client and a thermometer service.

This is a property of two finite systems composed asynchronously with two unbounded channels (one for each direction of the communication). A composition has the half-duplex property if there is no state where there is a message on both channels at the same time. This property is mainly interesting as problems like reachability and boundedness, that in general are undecidable for FIFO-channels, are decidable for a half-duplex composition.

### 3.3 Asynchronous I/O Petri Nets

Labelled Transition Systems are simple and straightforward as formal system models, however they tend to be very large when the system complexity rises. Furthermore, we cannot use LTSs directly to reason formally about infinite systems, as we need a finite representation of the system. In order to do so we use the *labelled Petri net* (PN) formalism in our study. Petri nets were introduced by C. A. Petri in his dissertation [Pet62]. Petri nets are a popular formal model for specification and analysis, and the advantage is that PN systems specifications are more compact than LTSs, in fact the size of a LTS corresponding to a finite PN specification can be exponentially larger. Additionally, many verification problems are decidable for PNs, for a survey see [EN94].

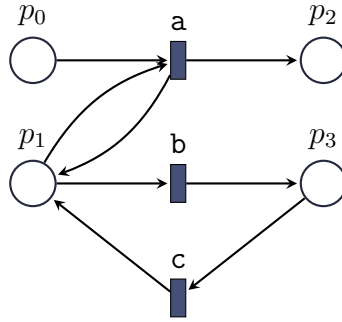
A labelled Petri net is a finite graph with two types of nodes; *places* and *transitions*, and with directed arcs between the nodes. The arcs are only allowed to go from places to transitions (*input-arcs*) or from transitions to places (*output-arcs*). These input- and output-arcs defines the *input-places* and *output-places* of a transition. The transitions of a PN model the actions that a system can perform, as it was the case for LTSs, i.e. firing a transition can change the state of the system. Each place can contain a number

### 3. Component-Based Development

of *tokens*, and a state of a PN, referred to as a *marking*, is a distribution of tokens in the places of the PN. In general the places are unbounded, which means that a PN can have infinitely many markings. The input-arcs model the precondition of a transition, so a transition is *enabled* if there is a token in each of the input-places of that transition. An enabled transition can be *fired*, and when a it is fired, a token is consumed from each input place, and a new token is produced into each output place. Formally a Petri net is defined as follows.

**Definition 4** (Labelled Petri Net). A labelled Petri net is a tuple  $\mathcal{N} = (P, T, \Sigma, W^-, W^+, \lambda, m^0)$ , such that

- $P$  is a finite set of places,
- $T$  is a finite set of transitions with  $P \cap T = \emptyset$ ,
- $\Sigma$  is a finite alphabet,
- $W^-$  (resp.  $W^+$ ) is a matrix indexed by  $P \times T$  with values in  $\mathbb{N}$ ; it is called the backward (resp. forward) incidence matrix,
- $\lambda : T \rightarrow \Sigma_\tau$  is a transition labelling function, and
- $m^0$  is a vector of natural numbers, indexed by  $P$  and called the initial marking.



**Figure 3.4:** Example of a Petri Net  $\mathcal{N}_1$ .

Figure 3.4 shows an example of a Petri net  $\mathcal{N}_1$ . The circular nodes  $p_0, p_1, p_2, p_3$  are places and the squared nodes are transitions, each labelled with a label from an action alphabet. Note that no arcs are going directly between two places or two transitions as this is not allowed in a PN. In Figure 3.5a  $\mathcal{N}_1$  is shown with a marking  $m$ . The marking  $m$  specifies that there are two tokens in  $p_0$  and one in  $p_1$ . In the following markings of  $\mathcal{N}_1$  are represented by a 4-tuples with a natural number for each place, for instance  $m = (2, 1, 0, 0)$ . As there is at least one token in  $p_0$  and  $p_1$  in marking  $m$ , which are the input-places of the transition **a**, then **a** is enabled and can be fired from  $m$ .

Likewise **b** is enabled in marking  $m$ , but **c** is not as there are no tokens in  $p_3$ . Enabled transitions are striped in all examples. Since **b** is enabled in  $m$ , we can fire it. By firing **b** we consume one token from  $p_1$ , and produce a new token in  $p_3$ . This results in a new marking  $m^1 = (2, 0, 0, 1)$ , shown in Figure 3.5b. We write  $m \xrightarrow{b} m^1$ , when **b** is enabled in  $m$  and  $m^1$  is reached by firing it. Note that **a** is disabled in  $m^1$ , since the firing of **b** removed the required token from  $p_1$ . When analysing the behaviour of a PN we consider *traces*. A trace is an execution of a PN denoted as a sequence of transitions and the visited markings. For instance,  $(2, 1, 0, 0) \xrightarrow{b} (2, 0, 0, 1) \xrightarrow{c} (2, 1, 0, 0) \xrightarrow{a} (1, 1, 1, 0) \xrightarrow{a} (0, 1, 2, 0)$  is a trace of  $\mathcal{N}_1$ .

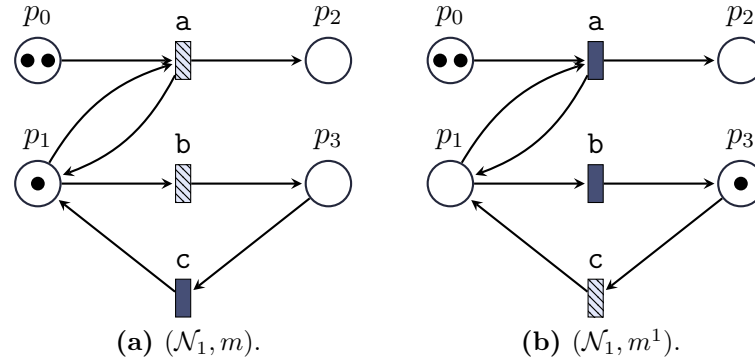


Figure 3.5: Examples of marked Petri nets.

We study general properties of asynchronous communication over unbounded and unordered channels and consider both primitive components with an open interface for communication with other components as well as larger composed systems. Therefore, we define *Asynchronous I/O-Petri nets* (AIOPNs) as a simple extension of the PNs briefly described above. In addition to a basic PN, AIOPNs have a finite set of channels, which is a subset of the places, and the transitions are labelled with labels from an special I/O-alphabet. In a primitive component there are no channels, as channels are introduced during composition of components. The I/O-alphabet is very similar to the I/O-alphabet we introduced for LTSs, however now we have a distinct set of communication labels. For a given AIOPN with channels  $C$  the set of communication labels is  $\text{com} = \{\triangleright a, a^\triangleright \mid a \in C\}$ . For a primitive component the set of communication labels is empty, as there are no channels. Formally an AIOPN is defined as follows.

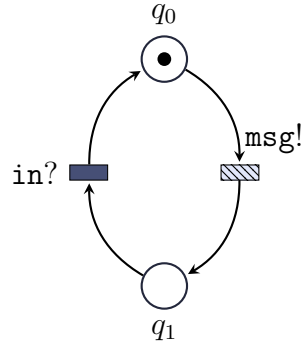
**Definition 5** (Asynchronous I/O-Petri net). *An asynchronous I/O-Petri net (AIOPN) is a tuple  $\mathcal{N} = (C, P, T, \Sigma, W^-, W^+, \lambda, m^0)$ , such that*

- $(P, T, \Sigma, W^-, W^+, \lambda, m^0)$  is a labelled Petri net,
- $C$  is a finite set of channels,
- $C \subseteq P$ , i.e. each channel is a place,

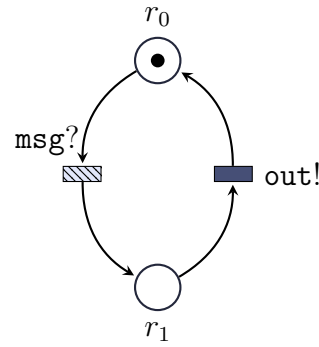
### 3. Component-Based Development

- $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is an I/O-alphabet over  $C$ ,
- for all  $a \in C$  and  $t \in T$ ,  

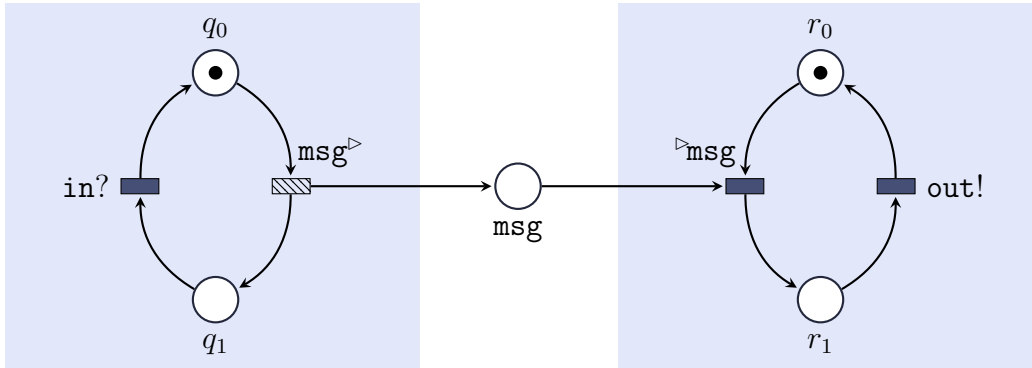
$$W^-(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = \triangleright a, \\ 0 & \text{otherwise} \end{cases} \quad W^+(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = a \triangleright, \\ 0 & \text{otherwise} \end{cases}$$
- for all  $a \in C$ ,  $m^0(a) = 0$ .



(a) Example of a primitive AIOPN component  $\mathcal{N}_2$ .



(b) Example of a primitive AIOPN component  $\mathcal{N}_3$ .



(c) Example of a composed AIOPN component  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$ .

**Figure 3.6:** Example of asynchronous I/O Perti nets.

As an example consider the two primitive AIOPN components  $\mathcal{N}_2$  and  $\mathcal{N}_3$  in Figure 3.6a and Figure 3.6b. It should be clear that these two components correspond to the two previously described LTS components  $\mathcal{S}_4$  and  $\mathcal{S}_5$  in Figure 3.1a and Figure 3.1b. Even though the components are very similar to the LTS components, the asynchronous composition of AIOPNs is somewhat different. When composing two AIOPNs asynchronously, the two AIOPNs are basically unioned into one AIOPN and for every matching input- and output-label a new channel-place is created with an arc from the output-labelled transition and an arc to the input-labelled transition. A token in such channel-place models a message on that channel. Note that this means there

is a communication channel for each type of message exchanged between the composed components. Figure 3.6c shows the asynchronous composition of  $\mathcal{N}_2$  and  $\mathcal{N}_3$ , we show the original primitive components by drawing them in frames. It is important to observe that this composition is a regular PN opposed to the corresponding LTS composition in Figure 3.1c, which is infinite even though it is a composition of two finite components. Nevertheless, the behaviour of this composed AIOPN is infinite, since the `msg` channel can hold unboundedly many messages. It should be easy to see the correspondence between a state of the composed LTS (Figure 3.1c) and a marking of the composed AIOPN (Figure 3.6c). The counter in the LTS state corresponds to the number of tokens in the `msg` place of the AIOPN composition and the state-pair in the LTS state corresponds to the marking of the places  $q_0, q_1, r_0$  and  $r_1$  of the AIOPN. For each new channel, the input- and output-transitions are renamed as it was done for the composed LTS.

Our AIOPN formalism is very similar to open Petri nets introduced by Baldan, Corradini, Ehrig, and Heckel in [Bal+01]. They also use places as interfaces between components for asynchronous composition. However, there are two main difference. We explicitly distinguish the channel-places, this allows us to reason about communication properties of the channels. Furthermore, in open Petri nets interface places are already present in the components before composition, matching interface places are then collapsed during composition. Therefore, open Petri nets are bound to use asynchronous composition, whereas we could define a synchronous composition of AIOPNs or even a mix of both asynchronous and synchronous composition. This means that the designer of a component does not have to consider whether the component will be used in an asynchronous or synchronous environment, which we believe is an advantage as it gives a better separation of concerns.

## 3.4 Channel Properties of Asynchronous I/O Petri Nets

With AIOPN as a tool for modelling asynchronously composed systems we consider two classes of properties concerning the asynchronous communication via unbounded and unordered channels: *consuming properties* and *communication stopping properties*. The first class of properties considers situations where a message is already sent from one component on a channel and we set requirements for how and when this message should be consumed by the receiving component. The second class of properties identify systems, where the components can reach consensus about terminating communication.

Our goal is to define a set of properties that are compositional, in the sense that if a composed system has some property, then this property should not be broken if this composed system is composed with a new component. Note that a composed system might have open input- and output-transitions, which can be matched in a future composition with another component. Hence, the communication properties must take into account that a composed system must have a given property in any possible environment. This means that a property must hold even though a “future” component in the

### 3. Component-Based Development

final system might not provide a required input to this particular part of the system. We say that a component must be able to do something *autonomously* if we cannot rely on input from an environment. Open output-transitions are not a problem as we are dealing with asynchronous communication, so an output can never be blocking. We identify markings of an AIOPN where only open input-labelled transitions are enabled and call them *pure input states*. A pure input state is a possible deadlock in composed systems, as there are no guaranties that there will ever be any messages for these open inputs when needed.

We study four main properties within the class of consuming properties; *Consuming*, *Decreasing*, *Emptying*, and *Wholly Emptying*. Each property has a successively stronger requirement than its predecessor. Furthermore, we study each of these four basic properties in two additional variations; a *strong* and a *necessarily* variant. We start by going through the consuming property in its basic form and the strong and necessarily variations.

The *consuming* property is one of the simplest asynchronous communication requirements. It requires that whenever there is a message on a channel then the receiving component must be able to take it. More precisely for our AIOPNs.

**Definition 6** (Consuming). *An AIOPN is consuming on a channel  $c$  if for any reachable marking  $m$  with some messages on  $c$ , the AIOPN can from  $m$  autonomously reach a marking  $m'$  such that  $m' \xrightarrow{c} m''$ .*

We say that an AIOPN is *consuming* if it is consuming on all its channels.

As an example, consider the marked PN  $(\mathcal{N}_2 \otimes_{as} \mathcal{N}_3, m)$  in Figure 3.7. In order for  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$  to be *consuming*, we must prove that we can always take a token from the **msg**, whenever there are tokens present. This is clearly the case as  $\mathcal{N}_3$  can always fire  $\triangleright\mathbf{msg}$  followed by **out!** when there is a message in **msg**.

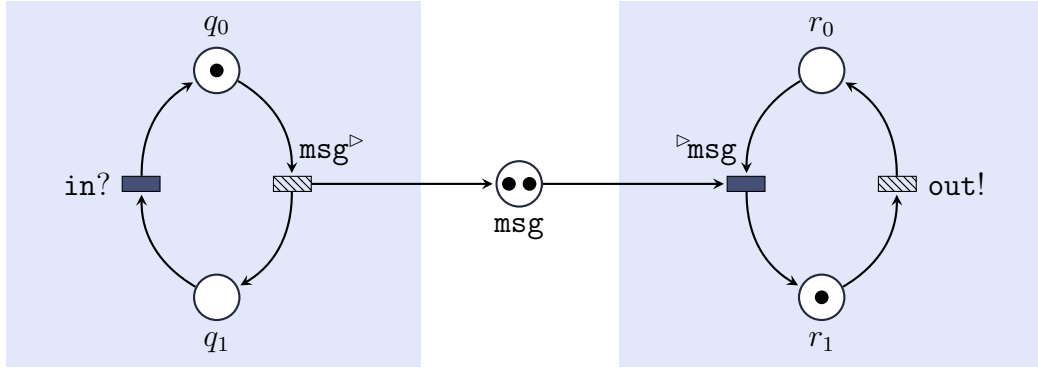
As mentioned above we also consider a *strong consuming* property. This property is stronger in the sense that the system must immediately be able to consume a message from a channel, whenever a message is present on a channel.

**Definition 7** (Strong Consuming). *An AIOPN is strong consuming on a channel  $c$  if for any reachable marking  $m$  with some messages on  $c$ , it is the case that  $m \xrightarrow{c} m'$ .*

Clearly, the  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$  shown in Figure 3.7 is not *strongly consuming* as this would have required  $\triangleright\mathbf{msg}$  to be enabled in any marking where the channel **msg** is not empty.

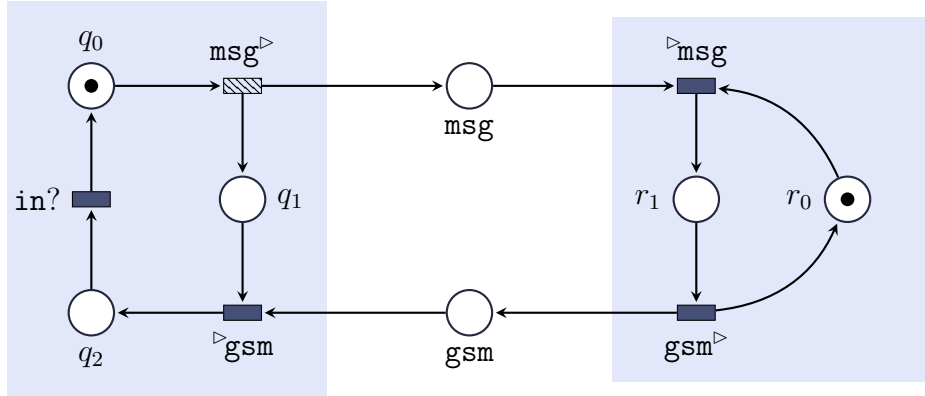
Both the consuming and strong consuming property require that a system must be able to consume a message whenever a message is present on a channel, but there is no guarantee that this will happen. In some situation such guarantee is important. In order to make such guarantee an AIOPN must be consuming in all possible *runs* of the




 Figure 3.7:  $(\mathcal{N}_2 \otimes_{as} \mathcal{N}_3, m)$ .

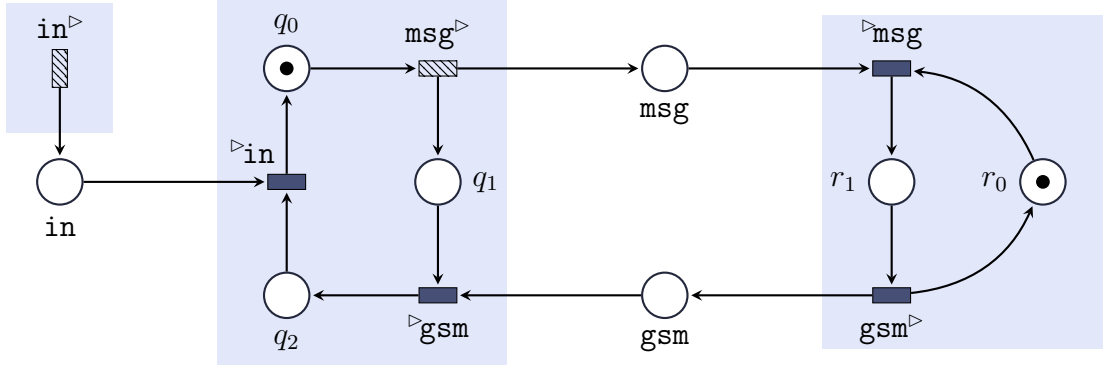
system. A run of an AIOPNs is either an infinite trace or a finite trace ending in a pure input state, i.e. a possible deadlock. Furthermore, we must assume that the systems we consider are running in an environment with a fair task-scheduler. If we do not assume fairness the consuming guarantees might not be compositional. The problem is illustrated by the AIOPN component  $\mathcal{N}_4$  shown in Figure 3.8. It is a composed component with two channels  $\text{msg}$  and  $\text{gsm}$  and a single open input  $\text{in}$ . Note that there can be at most one message in transit at a time, as it must consume a message on a channel in order to progress. Thus, any run of  $\mathcal{N}_4$  will be a firing of the following sequence of transitions repeatedly;

$$\text{msg}^\triangleright, \triangleright\text{msg}, \text{gsm}^\triangleright, \triangleright\text{gsm}, \text{in?}.$$


 Figure 3.8: The composed AIOPN  $\mathcal{N}_4$ .

Furthermore, a run of  $\mathcal{N}_4$  will either be infinite or finite and ending a pure input state, i.e. the marking with a single token in  $q_2$  and all other places empty. Hence, it will eventually take a message from a channel if there is a message on that channel. Thus,  $\mathcal{N}_4$  is consuming in all possible runs. However, if we compose  $\mathcal{N}_4$  with a simple component,  $\mathcal{N}_5$  that can always send an  $\text{in}$  message, shown in Figure 3.9, we no longer have a system that is consuming in all possible runs. This is the case as we by firing

### 3. Component-Based Development



**Figure 3.9:** The composed AIOPN  $\mathcal{N}_4 \otimes_{as} \mathcal{N}_5$ .

$\text{msg}^\triangleright$  reach a marking with a token on the channel  $\text{msg}$ , from this marking, there is a valid run that is just firing  $\text{in}^\triangleright$  to infinity, i.e. the token in the channel  $\text{msg}$  will never be consumed in this run. Thus we have a composition that broke a property of one of the components. This is undesirable and as mentioned above we only want to consider properties that are compositional.

In order to fix this issue we assume weak fairness. By this assumption we can omit runs that are not weakly fair runs. A run is a *weakly fair run* if any transition that is always enabled in a run is also infinitely often taken. With this we can define our necessarily consuming property.

**Definition 8** (Necessarily Consuming). *An AIOPN is necessarily consuming on a channel  $c$  if for any reachable marking  $m$  with some messages on  $c$ , the AIOPN will do a  $\triangleright c$  action on any weakly fair run from  $m$ .*

Clearly, the problematic run, that was just firing the  $\text{in}$  transition to infinity is not a weakly fair run, as  $\triangleright \text{msg}$  is enabled in all markings of the run, but never taken.

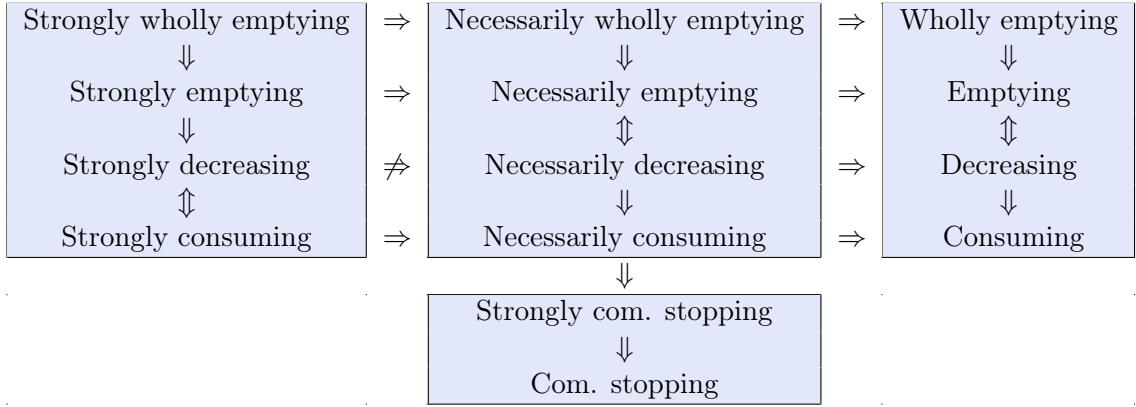
As mentioned in earlier, we define three more consuming properties that increase the requirement of how an AIOPN should be able to consume messages on a channel.

**Definition 9** (Decreasing). *An AIOPN is decreasing on a channel  $c$  if for any reachable marking  $m$  with some messages on  $c$ , the AIOPN can autonomously from  $m$  reach a marking  $m'$  such that the number of messages on  $c$  is less than it was in  $m$ .*

**Definition 10** (Emptying). *An AIOPN is emptying on a channel  $c$  if for any reachable marking  $m$  with some messages on  $c$ , the AIOPN can autonomously from  $m$  reach a marking  $m'$  such that the channel  $c$  is empty.*

**Definition 11** (Wholly Emptying). *An AIOPN is wholly emptying if for any reachable marking  $m$ , the AIOPN can autonomously from  $m$  reach a marking  $m'$  where all channels are empty at once.*

Each of the three properties decreasing, emptying, and wholly emptying are also defined in the strong and necessarily variant as it was done for the consuming property. This means that we have 12 different properties regarding the consumption of messages on a communication channel. Beside these 12 consumption properties, we define a *communication stopping* property, which requires that if a component stops consuming properties on a channel, then the sending component should be aware and stop sending messages on that channel, after a finite number of steps. If it immediately stops we say that the system is *strongly communication stopping*. For example,  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$  in Figure 3.6c is not communication stopping, as  $\mathcal{N}_2$  can always send new messages on the channel.



**Table 3.1:** Relationships between channel properties.

We have now presented 14 different properties that we believe are interesting for compositional system design regardless of the system application. We have also considered the relationships between the different properties, an overview is shown in Table 3.1. For more discussion and details about these relations see Section D. We prove that all of these properties are compositional. Meaning that if a composed component  $\mathcal{N}$  has a property  $P$ , then a composition of  $\mathcal{N}$  and any other component will also have the property  $P$  on the channels from  $\mathcal{N}$ . This is a crucial result and it means that AIOPNs support a modular verification approach, where we can verify each component by itself, and only consider new channels when we need to prove that a composition has a property. This modular verification approach holds for all properties except the wholly emptying, as we cannot prove that the new channels can be emptied at the same time as the channels of each component. On the other hand, wholly emptying is more a system requirement than a component requirement.

**Result 5** (Incremental design). *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two AIOPNs and let  $P$  be an arbitrary*

### 3. Component-Based Development

*channel property, except wholly emptying. If both  $\mathcal{N}$  and  $\mathcal{M}$  have property  $P$  and if  $\mathcal{N} \otimes_{as} \mathcal{M}$  has property  $P$  with respect to the new channels of the composition, then  $\mathcal{N} \otimes_{as} \mathcal{M}$  has property  $P$ .*

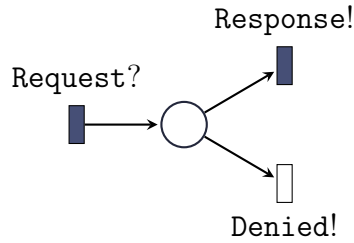
This result means that system engineers and designers can verify the communication of their component-based system design in an incremental way. First, they start by proving that compositions of primitive components have the desired property. Then, they prove that composition of verified composed components has the desired property, but the result above means that they only need to prove it for the new channels. This process continues until the whole system is proved to have the desired property.

Finally, we prove that all 14 properties are decidable for AIOPNs. We obtain these results by reusing existing decidability results and approaches for regular Petri nets.

**Result 6** (Decidability). *Given an AIOPN  $\mathcal{N}$  and an arbitrary channel property  $P$ , it is decidable if  $\mathcal{N}$  has property  $P$ .*

## 3.5 Modal Asynchronous I/O Petri Nets

Up till now we have been working with two different formal methods that can be used in system design processes. The modal transition systems formalise the refinement steps of a design process and our study of properties directly related to asynchronous composition ensures that the communication in a component-based design is done properly. We will now join these two fields by extending our asynchronous I/O Petri nets with modalities, define modal refinement of such models and adapt the channel properties to cope with the modalities. This gives us a convenient framework where we can refine a system component-wise while preserving channel properties.



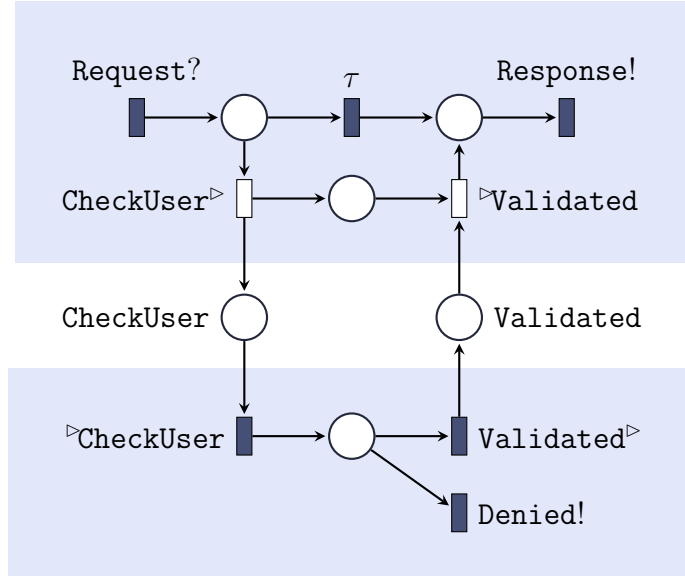
**Figure 3.10:** Example of a primitive MAIOPN component  $\mathcal{N}_{\text{interface}}$ .

We extend AIOPNs with modalities simply by having two types of transitions instead of one, so we can distinguish may-transitions and must-transitions. Everything else stays the same and we call it *modal asynchronous I/O Petri nets* (MAIOPNs). Formally an MAIOPN is defined as follows.

**Definition 12** (Modal Asynchronous I/O-Petri Net). A modal asynchronous I/O-Petri net (MAIOPN)  $\mathcal{N} = (C, P, T, T^\square, \Sigma_\tau, W^-, W^+, \lambda, m^0)$  is a modal Petri net such that,

- $(C, P, T, \Sigma_\tau, W^-, W^+, \lambda, m^0)$  is a AIOPN,
- $T$  is a finite set of may-transitions with  $P \cap T = \emptyset$ , and
- $T^\square$  is a finite set of must-transitions with  $T^\square \subseteq T$ .

A simple example of a primitive MAIOPN component is shown in Figure 3.10. The black transitions are must-transitions and the white transition is a may-transition. The MAIOPN  $\mathcal{N}_{\text{interface}}$  models the interface of a simple web-service that *must* be able to receive **Request** messages and once for each request it must be able to send a **Response**. Furthermore, it *may* send a **Denied** message instead of a response, but this is not a required behaviour.

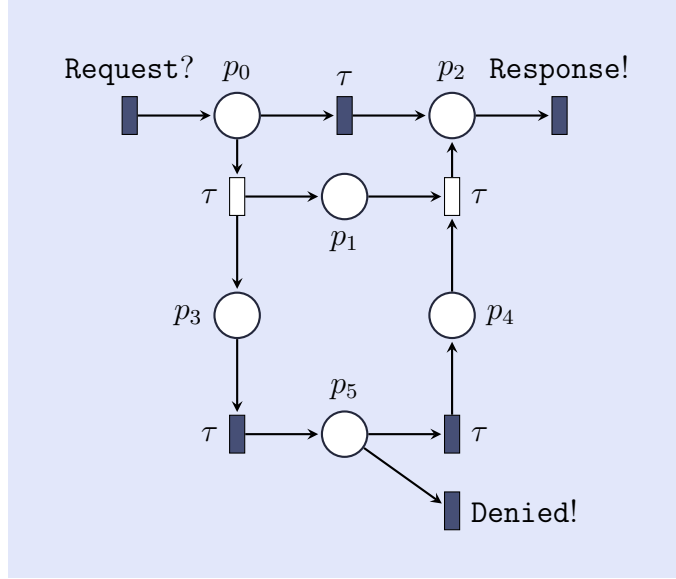


**Figure 3.11:**  $\mathcal{N}_{\text{WS}} \otimes_{as} \mathcal{N}_{\text{validator}}$ .

The next step is to refine this interface into a more detailed specification of our web-service. We do this by introducing two components: the main web-service component  $\mathcal{N}_{\text{WS}}$  and a validation service  $\mathcal{N}_{\text{validator}}$  that will approve or deny users interacting with the web-service. The composition of these two components is shown in Figure 3.11. This composed system has the same open inputs and outputs as the web-service interface  $\mathcal{N}_{\text{interface}}$ , but a more complex internal behaviour. Note that we use a special  $\tau$  transition to model hidden internal behaviour of a component. The internal behaviour of this composed component is not relevant when we want to verify that it is a realisation of the interface. Therefore, we define a hiding operation that can relabel communication on certain channels to hidden  $\tau$  channel. In this web-service example we hide commu-

### 3. Component-Based Development

nication on the **CheckUser** and **Validated** channels shown in Figure 3.12 and denote it by  $(\mathcal{N}_{WS} \otimes_{as} \mathcal{N}_{\text{interface}}) \setminus \{\text{CheckUser}, \text{Validated}\}$ .



**Figure 3.12:**  $(\mathcal{N}_{WS} \otimes_{as} \mathcal{N}_{\text{validator}}) \setminus \{\text{CheckUser}, \text{Validated}\}$ .

We cannot directly reuse the modal refinement introduced earlier for MTS, as it does deal with the hidden internal transitions. Instead, we define *weak modal refinement* of MAIOPNs that allows for observational abstractions. The main difference is that we now consider observable transitions, known as *weak transitions*. A weak transition is a finite trace, where at most one transition is visible, i.e. not a  $\tau$ -transition. For instance, in  $(\mathcal{N}_{WS} \otimes_{as} \mathcal{N}_{\text{interface}}) \setminus \{\text{CheckUser}, \text{Validated}\}$ , we can execute the following weak sequence

$$(0, 0, 0, 0, 0, 0) \xrightarrow{\text{Request?}} (1, 0, 0, 0, 0, 0) \xRightarrow{\text{Denied!}} .$$

The actual sequence of transitions fired by these weak transitions is

$$(0, 0, 0, 0, 0, 0) \xrightarrow{\text{Request?}} (1, 0, 0, 0, 0, 0) \xrightarrow{\tau} (0, 0, 0, 1, 0, 0) \xrightarrow{\tau} (0, 0, 0, 0, 0, 1) \xrightarrow{\text{Denied!}} .$$

Weak modal refinement, denoted  $\leq_m^*$  is similar to modal refinement. It requires that every mandatory behaviour of the abstract specification should also be mandatory in the concrete specification, but it allows must-transitions to be machted by weak must-transitions. Likewise, any allowed behaviour in the concrete specification should be allowed by the abstract specification, but again weak may-transitions are allowed. For a precise definition and discussion see Section E. With weak modal refinement we can now prove that

$$(\mathcal{N}_{WS} \otimes_{as} \mathcal{N}_{\text{validator}}) \setminus \{\text{CheckUser}, \text{Validated}\} \leq_m^* \mathcal{N}_{\text{interface}} .$$

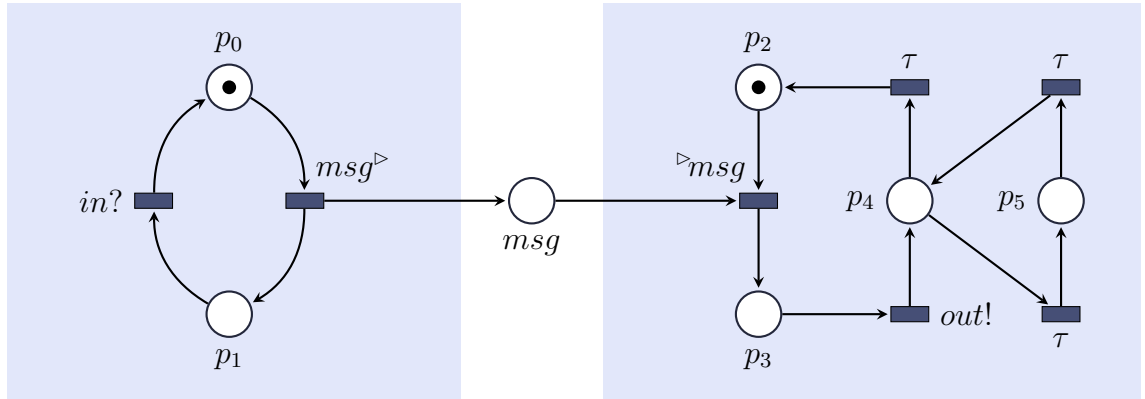
The goal is now to show that we can refine the  $\mathcal{N}_{\text{validator}}$  component in any way, and preserve this refinement by composition and hiding, so we do not need to prove refinement

of the interface again. Furthermore, we want to have a consuming channel property that also preserves this refinement. This will give us a modular approach where we can

1. define a system interface,
2. design the system behaviour with a component-based architecture,
3. prove that this system refines the interface by hiding internal behaviour, and
4. prove that the components are communicating properly with the consuming property.

Now, with all the above done, we can start refining the components individually without violating the interface while maintaining the channel properties.

Before we can reach this goal we have to adapt the consuming property to deal with modalities and hidden internal behaviour. We want the consuming property to be preserved by refinement, thus we need to make sure that a system has the consuming property in any possible implementation. To do this, we require that in any marking  $m$  where there is some message on a channel  $c$  then the system *must* autonomously be able to take this message. Hence, there should be a sequence  $\rho$  of non-input must-transitions from  $m$  leading to a marking  $m'$  such that  $m' \xrightarrow{\triangleright_c} m''$ . Note that we also allow must  $\tau$  transitions in  $\rho$ .



**Figure 3.13:** MAIOPN  $\mathcal{N}_2 \otimes_{as} \mathcal{N}'_3$ .

For the necessarily consuming property, the adjustment is not as simple. There is a problem with the weak fairness assumption, which is not sufficient any more. Consider the composed AIOPN  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$  in Figure 3.6c on page 26 as an MAIOPN, where every transition is a must-transition. Using the definition of necessarily consuming for AIOPN  $\mathcal{N}_2 \otimes_{as} \mathcal{N}_3$  is necessarily consuming. However, if we refine the component  $\mathcal{N}_3$  by adding some hidden internal behaviour, we can break the necessarily consuming property. For instance in Figure 3.13, we have exchanged  $\mathcal{N}_3$  with a weak refinement  $\mathcal{N}'_3$ . In  $\mathcal{N}_2 \otimes_{as} \mathcal{N}'_3$  we can reach a marking  $m$  with a token in  $p_0$ ,  $p_4$  and  $\text{msg}$ . From this marking we can repeatedly fire the two rightmost  $\tau$ -transitions followed by  $\text{msg}^\triangleright$  and  $\text{in?}$  to infinity.

### 3. Component-Based Development

This is in fact a weakly fair run, but there is never taken a message from `msg`. Hence,  $\mathcal{N}_2 \otimes_{as} \mathcal{N}'_3$  is not necessarily consuming as defined for AIOPNs.

The problem is that  $\mathcal{N}'_3$  is a valid weak refinement of  $\mathcal{N}_3$ , but the introduced  $\tau$ -loop means that  $\mathcal{N}'_3$  can loop forever without consuming messages from `msg`, even in a weak fair run. The core of this problem is that we allow weak transitions in the definition of weak modal refinement, and this must be reflected in the necessarily property. In order to do this we introduce a notion of *observationally weak fair runs*, where we require that any weak transition that is always enabled in a run is also infinitely often taken. In this concrete example this means that the weak transition  $\xRightarrow{\text{msg}}$  is always enabled but never taken. Thus, the infinite run above does not qualify as an observationally weak fair run. With this notion of fairness we can define the necessarily consuming for MAIOPNs by requiring that from any marking  $m$  with some messages on a channel  $c$ , the system must on all observationally weak fair runs eventually consume a message from  $c$ .

We have now presented the four techniques for component-based system design with MAIOPNs; composition, hiding, refinement, and consuming properties. With these we prove that following main results. The first result shows that hiding internal communication preserves refinement.

**Result 7.** *For two MAIOPNs  $\mathcal{N}$  and  $\mathcal{M}$  such that  $\mathcal{N} \leq_m^* \mathcal{M}$  and  $H$  a subset of their channels it holds that  $(\mathcal{N} \setminus H) \leq_m^* (\mathcal{M} \setminus H)$ .*

The next result shows that refinement is preserved by composition.

**Result 8.** *Given four MAIOPNs,  $\mathcal{N}, \mathcal{N}', \mathcal{M}$  and  $\mathcal{M}'$  such that  $\mathcal{N}' \leq_m^* \mathcal{N}$  and  $\mathcal{M}' \leq_m^* \mathcal{M}$ , then it holds that  $\mathcal{N}' \otimes_{as} \mathcal{M}' \leq_m^* \mathcal{N} \otimes_{as} \mathcal{M}$ .*

Like for AIOPNs, MAIOPNs do also support incremental design.

**Result 9 (Incremental design).** *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two MAIOPNs such that they are both (necessarily) consuming. If  $\mathcal{N} \otimes_{as} \mathcal{M}$  is (necessarily) consuming with respect to the new channels of the composition, then  $\mathcal{N} \otimes_{as} \mathcal{M}$  is (necessarily) consuming.*

Finally, we prove that (necessarily) consuming is preserved by refinement.

**Result 10.** *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two MAIOPNs such that  $\mathcal{N}$  is (necessarily) consuming. If  $\mathcal{M} \leq_m^* \mathcal{N}$  then  $\mathcal{M}$  is (necessarily) consuming.*

With all these results we believe that modal asynchronous I/O Petri nets is a suitable framework for component-based system design of complex systems. We have not studied the remaining channel properties in this modal framework, but we expect that they can be adapted in a similar manner and that the results above will continue to hold.

## 3.6 Main Contributions

We will now briefly summarize the main contributions of this thesis within the area of component-based system specification. First, we have introduced and defined the



Asynchronous I/O Petri Nets and composition of such. Furthermore, we have introduced 14 different communication properties that assist in ensuring that component are communicating properly. We prove that these properties are all decidable for AIOPNs and that they are preserved by composition. Second, we extend the AIOPN formalism with modalities in order support refinement steps. We prove that this is an interesting framework for designing component-based systems, since our communication properties are preserved by both refinement and composition. In addition to this, we introduce a way to encapsulate composed components as black boxes with an open interface. This step does also preserve the communication properties and enables us to prove that a composition of specific components is in fact a refinement of a general specification.



# 4

## Publication Overview

This thesis is based on five published papers. Minor changes have been done to the papers; layout adjustments to make them fit the format of this thesis, proofs has been moved to the main text from appendix or technical report and typos have been fixed. Each paper will now be summarised by its abstract, contribution and details about publication.

In addition to these five papers, I have published the following papers during my Ph.D. study.

### **Verification of Timed-Arc Petri Nets**

Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jiří Srba

Published in proceedings of the 37th international conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11) volume 6543 of LNCS, pages 46–72. Springer-Verlag, 2011

### **Compositional Verification of Real-time Systems Using Ecdar.**

Alexandre David, Kim G. Larsen, Axel Legay, Mikael H. Møller, Ulrik Nyman, Anders P. Ravn, Arne Skou and Andrzej Wąsowski

Published in international journal on Software Tools for Technology Transfer (STTT), vol.14, no. 6, p. 703–720. Springer-Verlag, 2012

### **TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets.**

Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Y. Jørgensen Mikael H. Møller, and Jiří Srba

Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, Jiří Srba

Published in proceedings of the 18th international Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12) volume 7214 of LNCS, pages 492–497. Springer-Verlag, 2012

**TCTL-preserving translations from timed-arc Petri nets to networks of timed automata.**

Joakim Byg, Lasse Jacobsen, Morten Jacobsen, Kenneth Y. Jørgensen Mikael H. Møller, and Jiří Srba

Published in journal Theoretical Computer Science. Currently, only available on-line, printed version in press. doi: 10.1016/j.tcs.2013.07.011

# Paper A: Parametric Modal Transition Systems

Nikola BENEŠ      Jan KŘETÍNSKÝ      Kim G. LARSEN  
Mikael H. MØLLER      Jiří SRBA

Modal transition systems (MTS) is a well-studied specification formalism of reactive systems supporting a step-wise refinement methodology. Despite its many advantages, the formalism as well as its currently known extensions are incapable of expressing some practically needed aspects in the refinement process like exclusive, conditional and persistent choices. We introduce a new model called parametric modal transition systems (PMTS) together with a general modal refinement notion that overcome many of the limitations and we investigate the computational complexity of modal refinement checking.

## Contributions

- Definition of parametric modal transition systems, an extension of MTS that is general enough to express several features missing in previous extensions of MTS.
- A comprehensive overview of complexity results for modal refinement checking of PMTSs and its subclasses.

## Publication History

The paper was accepted and presented at the 9th International Symposium on Automated Technology for Verification and Analysis (ATVA'11) and published in the proceedings of ATVA'11, volume 6996 of LNCS, pages 275–289, Springer-Verlag 2011.

## Paper B: A Characterizing Logic for Boolean Modal Transition Systems

Mikael H. MØLLER      Kim G. LARSEN      Xinxin LIU

Modal transition systems (MTS) is a specification formalism with support for stepwise refinement. It is a well-studied formalism, and several different extensions and generalizations have been developed. The original MTS has been shown to be in close relation with Hennessy-Milner Logic (HML). We will in this paper define a logic for Boolean MTS (BMTS), and present the characteristic formula for such specifications, using a version of HML with recursion. We prove that our logic characterizes the modal refinement of BMTS. Further we apply our logic on the more general Parametric MTS (PMTS), where we show how to select the parameters of a PMTS under certain requirements expressed in the logic.

### Contributions

- Defined a logic for Boolean modal transition systems and proved that it characterises modal refinement on BMTSs.
- Presented a way to apply the logic on parametric modal transition systems in order to determine which parameter configurations will result in a BMTS that satisfy the requirement.

### Publication History

The paper was accepted and presented at the 8th Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science 2012 (MEMICS'12) and published in the local proceedings of MEMICS'12.

# Paper C: Dual-Priced Modal Transition Systems with Time Durations

Nikola BENEŠ      Jan KŘETÍNSKÝ      Kim G. LARSEN  
Mikael H. MØLLER      Jiří SRBA

Modal transition systems are a well-established specification formalism for a high-level modelling of component-based software systems. We present a novel extension of the formalism called modal transition systems with durations where time durations are modelled as controllable or uncontrollable intervals. We further equip the model with two kinds of quantitative aspects: each action has its own running cost per time unit, and actions may require several hardware components of different costs. We ask the question, given a fixed budget for the hardware components, what is the implementation with the cheapest long-run average reward. We give an algorithm for computing such optimal implementations via a reduction to a new extension of mean payoff games with time durations and analyse the complexity of the algorithm.

## Contributions

- Defined an extension of modal transition systems that include variable time durations of actions and a price-scheme with a one-shot investment price for the hardware and a running cost per time unit for each action.
- Presented an algorithm to determine the cheapest implementation w.r.t. running expenses, on a fixed budget for hardware investment.

## Publication History

The paper was accepted and presented at the 18th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'12) and published in the proceedings of LPAR'12, volume 7180 of LNCS, pages 122-137, Springer-Verlag, 2012.

## Paper D: Channel Properties of Asynchronously Composed Petri Nets

Serge HADDAD      Rolf HENNICKER      Mikael H. MØLLER

We consider asynchronously composed I/O-Petri nets (AIOPNs) with built-in communication channels. They are equipped with a compositional semantics in terms of asynchronous I/O-transition systems (AIOTSs) admitting infinite state spaces. We study various channel properties that deal with the production and consumption of messages exchanged via the communication channels and establish useful relationships between them. In order to support incremental design we show that the channel properties considered in this work are preserved by asynchronous composition, i.e. they are compositional. As a crucial result we prove that the channel properties are decidable for AIOPNs.

### Contributions

- Defined asynchronously composed I/O-Petri nets and studied a series of 14 communication properties of such systems models.
- Proven that these communication properties are compositional, meaning that properties are preserved by new compositions.
- Proven that all 14 communication properties are decidable for asynchronously composed I/O-Petri nets.

### Publication History

The paper was accepted and presented at the 34th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets'13) and published in the proceedings of Petri Nets'13, volume 7927 of LNCS, pages 369-388, Springer-Verlag, 2013.



# Paper E: Specification of Asynchronous Component Systems with Modal I/O-Petri Nets

Serge HADDAD      Rolf HENNICKER      Mikael H. MØLLER

Modal transition systems are an elegant way to formalise the design process of a system through refinement and composition. Here we propose to adapt this methodology to asynchronous composition via Petri nets. The Petri nets that we consider have distinguished labels for inputs, outputs, internal communications and silent actions and “must” and “may” modalities for transitions. The input/output labels show the interaction capabilities of a net to the outside used to build larger nets by asynchronous composition via communication channels. The modalities express constraints for Petri net refinement taking into account observational abstraction from silent transitions. Modal I/O-Petri nets are equipped with a modal transition system semantics. We show that refinement is preserved by asynchronous composition and by hiding of communication channels. We study compatibility properties which express communication requirements for composed systems and we show that these properties are decidable, they are preserved in larger contexts and also by modal refinement. On this basis we propose a methodology for the specification of distributed systems in terms of modal I/O-Petri nets which supports incremental design, encapsulation of components, stepwise refinement and independent implementability.

## Contributions

- Extended the asynchronously composed I/O-Petri nets with modalities and studied communication properties and refinements of such modal specifications.
- Developed a complete methodology for designing asynchronously composed systems using our new formalism.
- Proven that channel properties are preserved by refinement and composition. Furthermore that refinement is preserved by composition and hiding.

## Publication History

The paper was accepted and presented at the 8th International Symposium on Trustworthy Global Computing (TGC’13) and published in the local pre-proceedings of TGC’13. Will also be published in the post-proceedings of TGC’13, that will appear as a volume in Springer’s LNCS series.



Part II

Papers





# Parametric Modal Transition Systems

Nikola BENEŠ

*Masaryk University, Department of Computer Science, Czech Republic*

Jan KŘETÍNSKÝ

*Institut für Informatik, Technische Universität München, Germany*

Kim G. LARSEN      Mikael H. MØLLER      Jiří SRBA

*Aalborg University, Department of Computer Science, Denmark*

**Abstract** Modal transition systems (MTS) is a well-studied specification formalism of reactive systems supporting a step-wise refinement methodology. Despite its many advantages, the formalism as well as its currently known extensions are incapable of expressing some practically needed aspects in the refinement process like exclusive, conditional and persistent choices. We introduce a new model called parametric modal transition systems (PMTS) together with a general modal refinement notion that overcome many of the limitations and we investigate the computational complexity of modal refinement checking.

# 1 Introduction

The specification formalisms of Modal Transition Systems (MTS) [LT88; Ant+08] grew out of a series of attempts to achieve a flexible and easy-to-use compositional development methodology for reactive systems. In fact the formalism of MTS may be seen as a fragment of a temporal logic [BL92], while having a behavioural semantics allowing for an easy composition with respect to process constructs.

In short, MTS are labelled transition systems equipped with two types of transitions: *must* transitions which are mandatory for any implementation, and *may* transitions which are optional for an implementation. Refinement of an MTS now essentially consists of iteratively resolving the unsettled status of may transitions: either by removing them or by turning them into must transitions.

It is well admitted (see e.g. [Rac+09]) that MTS and their extensions like disjunctive MTS (DMTS) [LX90], 1-selecting MTS (1MTS) [FS08] and transition systems with obligations (OTS) [BK11] provide strong support for a specification formalism allowing for step-wise refinement process. Moreover, the MTS formalisms have applications in other contexts, which include verification of product lines [GLS08; LNW07], interface theories [UC04; Rac+09] and modal abstractions in program analysis [GHJ01; HJS01; NNN08].

Unfortunately, all of these formalisms lack the capability to express some intuitive specification requirements like exclusive, conditional and persistent choices. In this paper we extend considerably the expressiveness of MTS and its variants so that it can model arbitrary Boolean conditions on transitions and also allows to instantiate persistent transitions. Our model, called *parametric modal transition systems* (PMTS), is equipped with a finite set of parameters that are fixed prior to the instantiation of the transitions in the specification. The generalized notion of modal refinement is designed to handle the parametric extension and it specializes to the well-studied modal refinements on all the subclasses of our model like MTS, disjunctive MTS and MTS with obligations.

To the best of our knowledge, this is the first sound attempt to introduce persistence into a specification formalism based on modal transition systems. The most related work is by Fecher and Schmidt on 1-selecting MTS [FS08] where the authors allow to model exclusive-or and briefly mention the desire to extend the formalism with persistence. However, as in detail explained in the appendix of [Ben+11b], their definition does not capture the notion of persistence. Our formalism is in several aspects semantically more general and handles persistence in a complete and uniform manner.

The main technical contribution, apart from the formalism itself, is a comprehensive complexity characterization of modal refinement checking on all of the practically relevant subclasses of PMTS. We show that the complexity ranges from P-completeness to  $\Pi_4^P$ -completeness, depending on the requested generality of the PMTS specifications on the left-hand and right-hand sides.

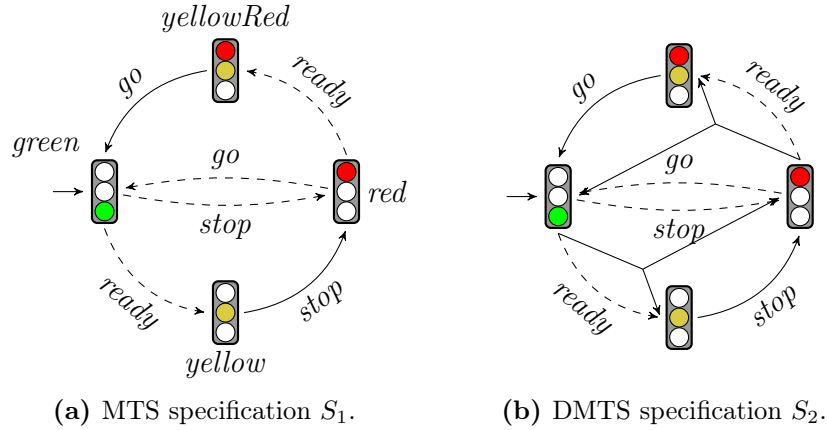
## 2 Parametric Modal Transition Systems

In this section we present the formalism of parametric modal transition systems (PMTS), starting with a motivating example and continuing with the formal definitions, followed by the general notion of modal refinement.

### 2.1 Motivation

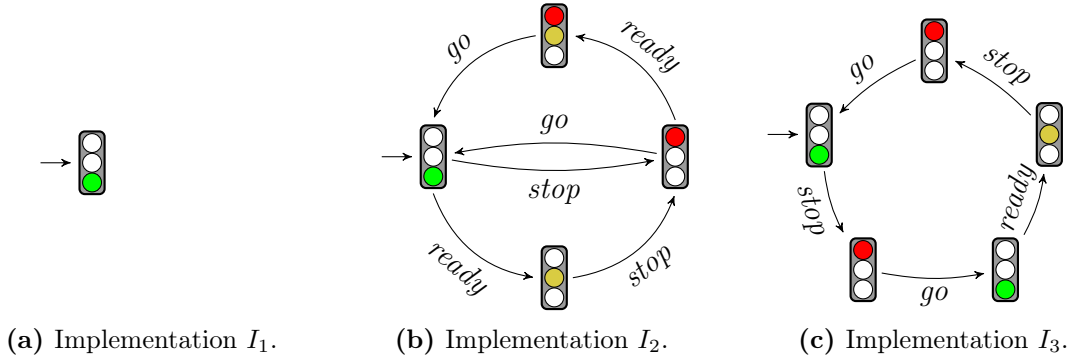
Modal transition systems and their extensions described in the literature are lacking the capability to express several specification requirements like exclusive, conditional and persistent choices. We shall now discuss these limitations on an example as a motivation for the introduction of parametric MTS formalism with general Boolean conditions in specification requirements.

Consider a simple specification of a traffic light controller that can be at any moment in one of the four predefined states: *red*, *green*, *yellow* or *yellowRed*. The requirements of the specification are: when *green* is on the traffic light may either change to *red* or *yellow* and if it turned *yellow* it must go to *red* afterward; when *red* is on it may either turn to *green* or *yellowRed*, and if it turns *yellowRed* (as it is the case in some countries) it must go to *green* afterwards.



**Figure 1:** Specifications of a traffic light controller.

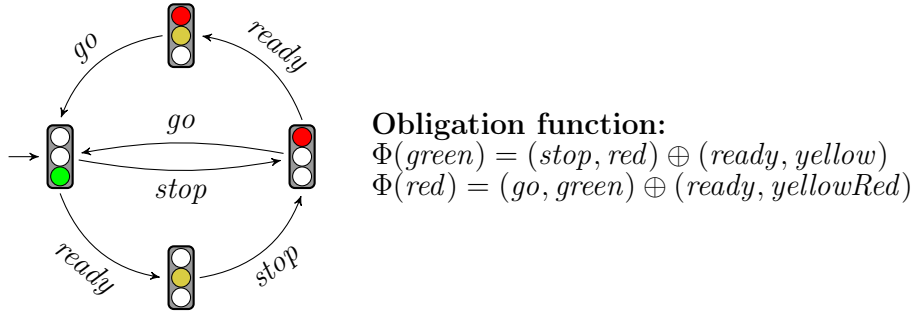
Figure 1a shows an obvious MTS specification (defined formally later on) of the proposed specification. The transitions in the standard MTS formalism are either of type may (optional transitions depicted as dashed lines) or must (required transitions depicted as solid lines). In Figure 2a, Figure 2b and Figure 2c we present three different implementations of the MTS specification where there are no more optional transitions. The implementation  $I_1$  does not implement any may transition as it is a valid possibility to satisfy the specification  $S_1$ . Of course, in our concrete example, this means that the light is constantly *green* and it is clearly an undesirable behaviour that cannot be, however, easily avoided. The second implementation  $I_2$  on the other hand implements all may transitions, again a legal implementation in the MTS methodology but not a



**Figure 2:** MTS implementations of a traffic light controller.

desirable implementation of a traffic light as the next action is not always deterministically given. Finally, the implementation  $I_3$  of  $S_1$  illustrates the third problem with the MTS specifications, namely that the choices made in each turn are not persistent and the implementation alternates between entering *yellow* or not. None of these problems can be avoided when using the MTS formalism.

A more expressive formalism of disjunctive modal transition systems (DMTS) can overcome some of the above mentioned problems. A possible DMTS specification  $S_2$  is depicted in Figure 1b. Here the *ready* and *stop* transitions, as well as *ready* and *go* ones, are disjunctive, meaning that it is still optional which one is implemented but at least one of them must be present. Now the system  $I_1$  in Figure 2a is not a valid implementation of  $S_2$  any more. Nevertheless, the undesirable implementations  $I_2$  and  $I_3$  are still possible and the modelling power of DMTS is insufficient to eliminate them.



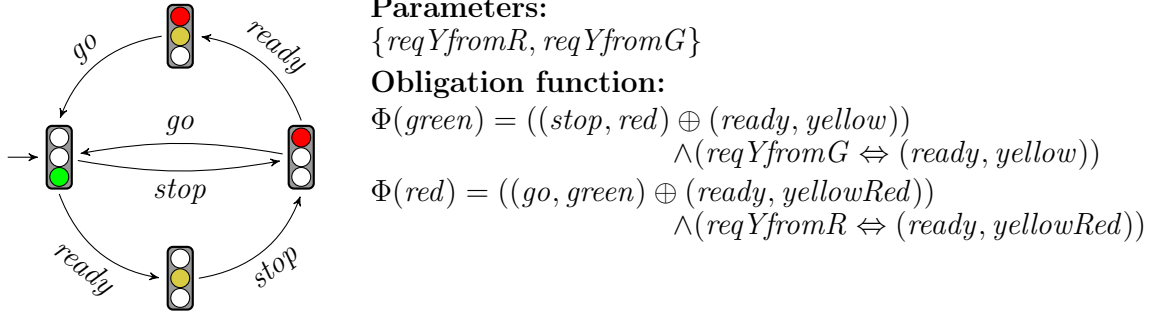
**Figure 3:** OTS Specification  $S_3$ .

Inspired by the recent notion of transition systems with obligations (OTS) [BK11], we can model the traffic light using specification as a transition system with arbitrary\* obligation formulae. These formulae are Boolean propositions over the outgoing transitions from each state, whose satisfying assignments yield the allowed combinations of outgoing transitions. A possible specification called  $S_3$  is given in Figure 3 and it uses the operation of exclusive-or. We will follow an agreement that whenever the obligation

\*In the transition systems with obligations only positive Boolean formulae are allowed.



function for some node is not listed in the system description then it is implicitly understood as requiring all the available outgoing transitions to be present. Due to the use of exclusive-or in the obligation function, the transition systems  $I_1$  and  $I_2$  are not valid implementation any more. Nevertheless, the implementation  $I_3$  in Figure 2c cannot be avoided in this formalism either.



**Figure 4:** PMTS specification  $S_4$ .

Finally, the problem with the alternating implementation  $I_3$  is that we cannot enforce in any of the above mentioned formalisms a uniform (persistent) implementation of the same transitions in all its states. In order to overcome this problem, we propose the so-called parametric MTS where we can, moreover, choose persistently whether the transition to *yellow* is present or not via the use of parameters. The PMTS specification with two parameters  $reqYfromR$  and  $reqYfromG$  is shown in Figure 4. Fixing a priori the (Boolean) values of the parameters makes the choices permanent in the whole implementation, hence we eliminate also the last problematic implementation  $I_3$ .

## 2.2 Definition of Parametric Modal Transition System

We shall now formally capture the intuition behind parametric MTS introduced above. First, we recall the standard propositional logic.

A Boolean formula over a set  $X$  of atomic propositions is given by the following abstract syntax

$$\varphi ::= \mathbf{tt} \mid x \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

where  $x$  ranges over  $X$ . The set of all Boolean formulae over the set  $X$  is denoted by  $\mathcal{B}(X)$ . Let  $\nu \subseteq X$  be a truth assignment, i.e. a set of variables with value true, then the satisfaction relation  $\nu \models \varphi$  is given by  $\nu \models \mathbf{tt}$ ,  $\nu \models x$  iff  $x \in \nu$ , and the satisfaction of the remaining Boolean connectives is defined in the standard way. We also use the standard derived operators like exclusive-or  $\varphi \oplus \psi = (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$ , implication  $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$  and equivalence  $\varphi \Leftrightarrow \psi = (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$ .

We can now proceed with the definition of parametric MTS.

**Definition 1.** A parametric MTS (PMTS) over an action alphabet  $\Sigma$  is a tuple  $(S, T, P, \Phi)$  such that

## A. Parametric Modal Transition Systems

- $S$  is a set of states,
- $T \subseteq S \times \Sigma \times S$  is a transition relation,
- $P$  is a finite set of parameters, and
- $\Phi : S \rightarrow \mathcal{B}((\Sigma \times S) \cup P)$  is a satisfiable obligation function over the atomic propositions containing outgoing transitions and parameters.

We implicitly assume that whenever  $(a, t) \in \Phi(s)$  then  $(s, a, t) \in T$ . By  $T(s) = \{(a, t) \mid (s, a, t) \in T\}$  we denote the set of all outgoing transitions of  $s$ .

We recall the agreement that whenever the obligation function for some node is not listed in the system description then it is implicitly understood as  $\Phi(s) = \bigwedge T(s)$ , with the empty conjunction being **tt**.

We call a PMTS *positive* if, for all  $s \in S$ , any negation occurring in  $\Phi(s)$  is applied only to a parameter. A PMTS is called *parameter-free* if  $P = \emptyset$ . We can now instantiate the previously studied specification formalisms as subclasses of PMTS.

**Definition 2.** A PMTS is called

- transition system with obligation (OTS) if it is parameter-free and positive,
- disjunctive modal transition system (DMTS) if it is an OTS and  $\Phi(s)$  is in the conjunctive normal form for all  $s \in S$ ,
- modal transition system (MTS) if it is a DMTS and  $\Phi(s)$  is a conjunction of positive literals (transitions) for all  $s \in S$ , and
- implementation (or simply a labelled transition system) if it is an MTS and  $\Phi(s) = \bigwedge T(s)$  for all  $s \in S$ .

Note that positive PMTS, despite the absence of a general negation and the impossibility to define for example exclusive-or, can still express useful requirements like  $\Phi(s) = p \Rightarrow (a, t) \wedge \neg p \Rightarrow (b, u)$  requiring in a state  $s$  a conditional presence of certain transitions. Even more interestingly, we can enforce binding of actions in different states, thus ensuring certain functionality. Take a simple two state-example:  $\Phi(s) = p \Rightarrow (request, t)$  and  $\Phi(t) = p \Rightarrow (response, s)$ . We shall further study OTS with formulae in the disjunctive normal form that are dual to DMTS and whose complexity of parallel composition is lower [BK11] while still being as expressive as DMTS.

### 2.3 Modal Refinement

A fundamental advantage of MTS-based formalisms is the presence of *modal refinement* that allows for a step-wise system design (see e.g. [Ant+08]). We shall now provide such a refinement notion for our general PMTS model so that it will specialize to the well-studied refinement notions on its subclasses. In the definition, the parameters are fixed first (persistence) followed by all valid choices modulo the fixed parameters that now behave as constants.

First we set the following notation. Let  $(S, T, P, \Phi)$  be a PMTS and  $\nu \subseteq P$  be a truth assignment. For  $s \in S$ , we denote by  $\text{Tran}_\nu(s) = \{E \subseteq T(s) \mid E \cup \nu \models \Phi(s)\}$  the set of all admissible sets of transitions from  $s$  under the fixed truth values of the parameters.

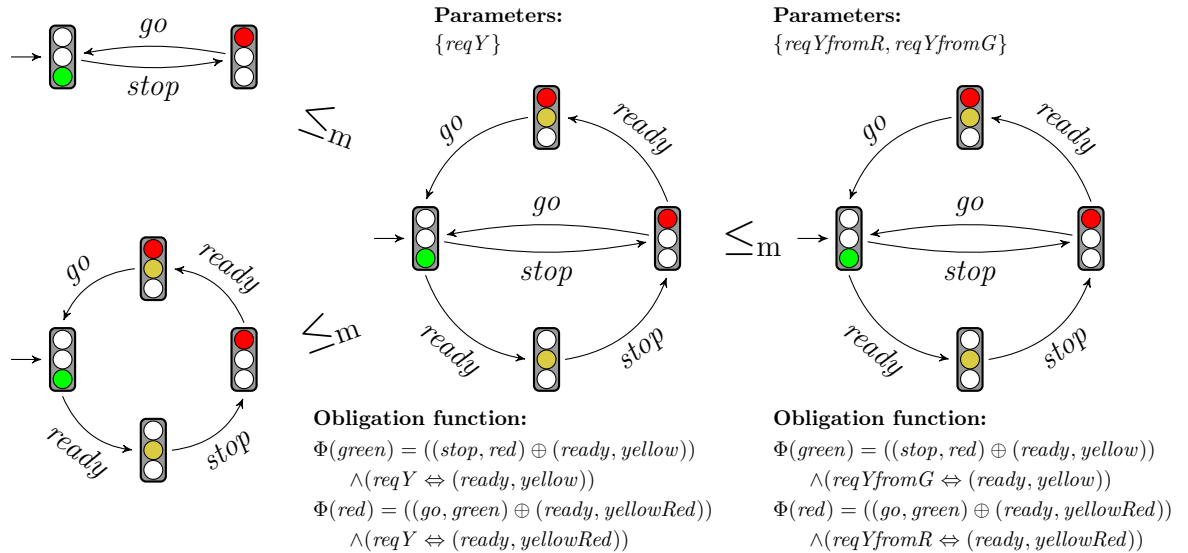
We can now define the notion of modal refinement between PMTS.

**Definition 3** (Modal Refinement). *Let  $(S_1, T_1, P_1, \Phi_1)$  and  $(S_2, T_2, P_2, \Phi_2)$  be two PMTSs. A binary relation  $R \subseteq S_1 \times S_2$  is a modal refinement if for each  $\mu \subseteq P_1$  there exists  $\nu \subseteq P_2$  such that for every  $(s, t) \in R$  holds*

$$\forall M \in \text{Tran}_\mu(s) \exists N \in \text{Tran}_\nu(t) : \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \wedge \\ \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R.$$

*We say that  $s$  modally refines  $t$ , denoted by  $s \leq_m t$ , if there exists a modal refinement  $R$  such that  $(s, t) \in R$ .*

*Example 1.* Consider the rightmost PMTS in Figure 5. It has two parameters  $\text{reqYfromG}$  and  $\text{reqYfromR}$  whose values can be set independently and it can be refined by the system in the middle of the figure having only one parameter  $\text{reqY}$ . This single parameter simply



**Figure 5:** Example of modal refinement.

## A. Parametric Modal Transition Systems

binds the two original parameters to the same value. The PMTS in the middle can be further refined into the implementations where either *yellow* is always used in both cases, or never at all. Notice that there are in principle infinitely many implementations of the system in the middle, however, they are all bisimilar to either of the two implementations depicted in the left of Figure 5.

In the next section, we shall investigate the complexity of positive subclasses of PMTS. For this reason we prove the following lemma showing how the definition of modal refinement can be simplified in this particular case.

We shall first realize that in positive PMTS and for any truth assignment  $\nu$ ,  $\text{Tran}_\nu(s)$  is *upward closed*, meaning that if  $M \in \text{Tran}_\nu(s)$  and  $M \subseteq M' \subseteq T(s)$  then  $M' \in \text{Tran}_\nu(s)$ .

**Lemma 4.** *Consider Definition 3 where the right-hand side PMTS is positive. Now the condition in Definition 3 can be equivalently rewritten as a conjunction of conditions (1) and (2)*

$$\forall M \in \text{Tran}_\mu(s) : \forall (a, s') \in M : \exists (a, t') \in T(t) : (s', t') \in R \quad (1)$$

$$\forall M \in \text{Tran}_\mu(s) : \text{match}_t(M) \in \text{Tran}_\nu(t) \quad (2)$$

where  $\text{match}_t(M)$  denotes the set  $\{(a, t') \in T(t) \mid \exists (a, s') \in M : (s', t') \in R\}$ . If the left-hand side PMTS is moreover positive too, Condition (1) is equivalent to

$$\forall (a, s') \in T(s) : \exists (a, t') \in T(t) : (s', t') \in R. \quad (3)$$

*Proof.* We shall first argue that the condition of modal refinement is equivalent to the conjunction of Conditions (4) and (5).

$$\forall M \in \text{Tran}_\mu(s) : \exists N \in \text{Tran}_\nu(t) : \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \quad (4)$$

$$\forall M \in \text{Tran}_\mu(s) : \exists N \in \text{Tran}_\nu(t) : \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R \quad (5)$$

Let  $\mu, \nu, R, s$  and  $t$  be fixed. Definition 3 trivially implies both Conditions (4) and (5). We now prove that (4) and (5) imply the condition in Definition 3.

Let  $M \in \text{Tran}_\mu(s)$  be arbitrary. There is some  $N_1 \in \text{Tran}_\nu(t)$  satisfying (4) and some  $N_2 \in \text{Tran}_\nu(t)$  satisfying (5). Let now  $N'_1 = \{(a, t') \in N_1 \mid \exists (a, s') \in M : (s', t') \in R\}$ . Consider  $N = N'_1 \cup N_2$ . Clearly, as  $\text{Tran}_\nu(t)$  is upward closed,  $N \in \text{Tran}_\nu(t)$ . Moreover, due to Condition (4) we have some  $(a, t') \in N_1$  such that  $(s', t') \in R$ . Clearly,  $(a, t') \in N'_1$  and thus also in  $N$ .

Now let  $(a, t') \in N$  be arbitrary. If  $(a, t') \in N_2$ , due to Condition (5) we have some  $(a, s') \in M$  such that  $(s', t') \in R$ . If  $(a, t') \notin N_2$  then  $(a, t') \in N'_1$ . The existence of  $(a, s') \in M$  such that  $(s', t') \in R$  is then guaranteed by the definition of  $N'_1$ .

Let us now proceed with proving the claims of the lemma. Condition (4) is trivially equivalent to (1) since  $\text{Tran}_\nu(t)$  is upward closed. Condition (5) is equivalent to (2). Indeed, (2) clearly implies (5) and we show that also (5) implies (2). Let  $M$  be arbitrary. We then have some  $N$  satisfying (5). Clearly,  $N \subseteq \text{match}_t(M)$ . Since  $\text{Tran}_\nu(t)$  is upward

closed,  $N \in \text{Tran}_\nu(t)$  implies  $\text{match}_t(M) \in \text{Tran}_\nu(t)$ . Due to the upward closeness of both  $\text{Tran}_\mu(s)$  and  $\text{Tran}_\nu(t)$  in the case of a positive left-hand side, the equivalence of (1) and (3) follows. ■

**Theorem 5.** *Modal refinement as defined on PMTS coincides with the standard modal refinement notions on MTS, DMTS and OTS. On implementations it coincides with bisimulation.*

*Proof.* The fact that Definition 3 coincides with modal refinement on OTS as defined in [BK11] is a straightforward corollary of Lemma 4 and its proof. Indeed, the two conditions given in [BK11] are exactly conditions (3) and (5). As the definition of modal refinement on OTS coincides with modal refinement on DMTS (as shown in [BK11]) and thus also on MTS, the proof is done.

However, for the reader's convenience, we present a direct proof that Definition 3 coincides with modal refinement on MTS. Assume a parameter-free PMTS  $(S, T, P, \Phi)$  where  $\Phi(s)$  is a conjunction of transitions for all  $s \in S$ , in other words it is a standard MTS where the must transitions are listed in the conjunction and the may transitions are simply present in the underlying transition system but not a part of the conjunction. Observe that every transition  $(s, a, t) \in T$  is contained in some  $M \in \text{Tran}_\emptyset(s)$ . Further, each must transition  $(s, a, t) \in T$  is contained in all  $M \in \text{Tran}_\emptyset(s)$ . Therefore, the first conjunct in Definition 3 requires that for all may transition from  $s$  there be a corresponding one from  $t$  with the successors in the refinement relation. Similarly, the second conjunct now requires that for all must transitions from  $t$  there be a corresponding must transition from  $s$ . This is exactly the standard notion of modal refinement as introduced in [LT88]. ■

### 3 Complexity of Modal Refinement Checking

We shall now investigate the complexity of refinement checking on PMTS and its relevant subclasses. Without explicitly mentioning it, we assume that all considered PMTS are now finite and the decision problems are hence well defined. The complexity bounds include classes from the polynomial hierarchy (see e.g. [Pap94]) where for example  $\Sigma_0^P = \Pi_0^P = P$ ,  $\Pi_1^P = \text{coNP}$  and  $\Sigma_1^P = \text{NP}$ .

	Boolean	Positive	pCNF	pDNF	MTS
Boolean	$\Pi_2^P$ -complete	coNP-complete	$\in \text{coNP}$ P-hard	coNP-complete	$\in \text{coNP}$ P-hard
Positive	$\Pi_2^P$ -complete	coNP-complete	P-complete	coNP-complete	P-complete
pCNF	$\Pi_2^P$ -complete	coNP-complete	P-complete	coNP-complete	P-complete
pDNF	$\Pi_2^P$ -complete	P-complete	P-complete	P-complete	P-complete
MTS	$\Pi_2^P$ -complete	P-complete	P-complete	P-complete	P-complete
Impl	NP-complete	P-complete	P-complete	P-complete	P-complete

**Table 1:** Complexity of modal refinement checking of parameter-free systems.

### 3.1 Parameter-Free Systems

Since even the parameter-free systems have interesting expressive power and the complexity of refinement on OTS has not been studied before, we first focus on parameter-free systems. Moreover, the results of this subsection are then applied to parametric systems in the next subsection. The results are summarized in Table 1. The rows in the table correspond to the restrictions on the left-hand side PMTS while the columns correspond to the restrictions on the right-hand side PMTS. Boolean denotes the general system with arbitrary negation. Positive denotes the positive systems, in this case exactly OTS. We use pCNF and pDNF to denote positive systems with formulae in conjunctive and disjunctive normal forms, respectively. In this case, pCNF coincides with DMTS. The special case of satisfaction relation, where the refining system is an implementation is denoted by Impl. We do not include Impl to the columns as it makes sense that an implementation is refined only to an implementation and here modal refinement corresponds to bisimilarity that is P-complete [BGS92] (see also [SJ05]). The P-hardness is hence the obvious lower bound for all the problems mentioned in the table.

We start with the simplest NP-completeness result.

**Proposition 6.** *Modal refinement between an implementation and a parameter-free PMTS is NP-complete.*

*Proof.* The containment part is straightforward. First we guess the relation  $R$ . As  $s$  is an implementation then the set  $\text{Tran}_\emptyset(s)$  is a singleton. We thus only need to further guess  $N \in \text{Tran}_\nu(t)$  and then in polynomial time verify the two conjuncts in Definition 3.

The hardness part is by a simple reduction from SAT. Let  $\varphi(x_1, \dots, x_n)$  be an given Boolean formula (instance of SAT). We construct two PMTSs  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  such that

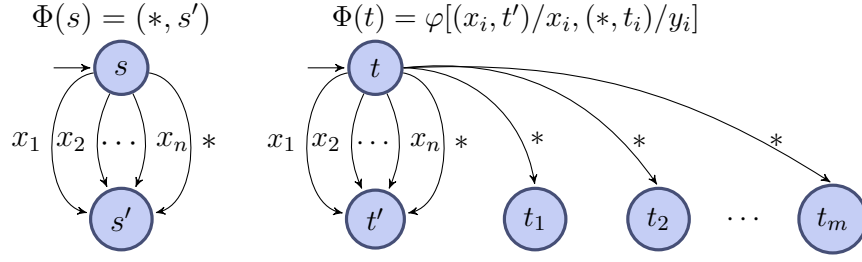
1.  $S = \{s, s'\}, T = (s, a, s'), P = \emptyset, \Phi(s) = (a, s')$  and  $\Phi(s') = \mathbf{tt}$  and
2.  $S' = \{t, t_1, \dots, t_n\}, T = \{(t, a, t_i) \mid 1 \leq i \leq n.\}, P' = \emptyset, \Phi(t) = \varphi[(a, t_i)/x_i]$  and  $\Phi(t_i) = \mathbf{tt}$  for all  $i, 1 \leq i \leq n$ .

Clearly,  $\varphi$  is satisfiable if and only if  $s \leq_m t$ . ■

Next we show that modal refinement is  $\Pi_2^P$ -complete. The following proposition introduces a gadget used also later on in other hardness results. We will refer to it as the *\*-construction*.

**Proposition 7.** *Modal refinement between two parameter-free PMTS is  $\Pi_2^P$ -hard even if the left-hand side is an MTS.*

*Proof.* The proof is by polynomial time reduction from the validity of the quantified Boolean formula  $\psi \equiv \forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_m : \varphi(x_1, \dots, x_n, y_1, \dots, y_m)$  to the refinement checking problem  $s \leq_m t$  where  $s$  and  $t$  are given as follows.



Assume that  $\psi$  is true. Let  $M \in \text{Tran}_\emptyset(s)$  (clearly  $(*, s') \in M$ ) and we want to argue that there is  $N \in \text{Tran}_\emptyset(t)$  with  $(*, t') \in N$  such that for all  $(x_i, s') \in M$  there is  $(x_i, t') \in N$  (clearly the states  $s'$ ,  $t'$  and  $t_i$  are in modal refinement) and for all  $(x_i, t') \in N$  there is  $(x_i, s') \in M$ . Such an  $N$  can be found by simply including  $(x_i, t')$  whenever  $(x_i, s') \in M$  and by adding also  $(*, t')$  into  $N$ . As  $\psi$  is true, we include into  $N$  also all  $(*, t_i)$  whenever  $y_i$  is set to true in  $\psi$ . Hence we get  $s \leq_m t$ .

On the other hand if  $\psi$  is false then we pick  $M \in \text{Tran}_\emptyset(s)$  such that  $M$  corresponds to the values of  $x_i$ 's such that there are no values of  $y_1, \dots, y_m$  that make  $\psi$  true. This means that from  $t$  there will be no transitions as  $\text{Tran}_\emptyset(t) = \emptyset$  assuming that  $(x_i, t')$  have to be set to true whenever  $(x_i, s') \in M$ , otherwise the refinement between  $s$  and  $t$  will fail. However, now  $(*, s') \in M$  cannot be matched from  $t$  and hence  $s \not\leq_m t$ . ■

**Proposition 8.** *Modal refinement between two parameter-free PMTS is in  $\Pi_2^P$ .*

*Proof.* The containment follows directly from Definition 3 (note that the parameters are empty) and the fact that the last conjunction in Definition 3 is polynomially verifiable once the sets  $M$  and  $N$  were fixed. The relation  $R$  could in principle be guessed before it is verified, however, this would increase the complexity bound to  $\Sigma_3^P$ . Instead, we will initially include all pairs (polynomially many) into  $R$  and for each pair ask whether for every  $M$  there is  $N$  such that the two conjuncts are satisfied. If it fails, we remove the pair and continue until we reach (after polynomially many steps) the greatest fixed point. The complexity in this way remains in  $\Pi_2^P$ . We shall use this standard method also in further proofs and refer to it as a co-inductive computation of  $R$ . ■

### 3.1.1 Positive Right-Hand Side.

We have now solved all the cases where the right-hand side is arbitrary. We now look at the cases where the right-hand side is positive. In the proofs that follow we shall use the alternative characterization of refinement from Lemma 4. The following proposition determines the subclasses on which modal refinement can be decided in polynomial time.



**Proposition 9.** *Modal refinement on parameter-free PMTS is in P, provided that both sides are positive and either the left-hand side is in pDNF or the right-hand side is in pCNF.*

*Proof.* Due to Lemma 4, the refinement is equivalent to the conjunction of (3) and (2). Clearly, (3) can be checked in P. We show that Condition (2) can be verified in P too. Recall that (2) says that

$$\forall M \in \text{Tran}_\mu(s) : \text{match}_t(M) \in \text{Tran}_\nu(t)$$

where  $\text{match}_t(M) = \{(a, t') \in T(t) \mid \exists (a, s') \in M : (s', t') \in R\}$ .

First assume that the left-hand side is in pDNF. If for some  $M$  the Condition (2) is satisfied then it is also satisfied for all  $M' \supseteq M$ , as  $\text{Tran}_\mu(s)$  is upwards closed. It is thus sufficient to verify the condition for all minimal elements (wrt. inclusion) of  $\text{Tran}_\mu(s)$ . In this case it corresponds to the clauses of  $\Phi(s)$ . Thus we get a polynomial time algorithm as shown in Algorithm 1.

---

**Algorithm 1:** Test for Condition (2) of modal refinement (pDNF)

---

**Input** : states  $s$  and  $t$  such that  $\Phi(s)$  is in positive DNF and  $\Phi(t)$  is positive, relation  $R$

**Output:** *true* if  $s, t$  satisfy the refinement condition, *false* otherwise

**foreach** clause  $(a_1, s_1) \wedge \dots \wedge (a_k, s_k)$  in  $\Phi(s)$  **do**

$N \leftarrow \{(a, t') \in T(t) \mid \exists i : a_i = a \wedge (s_i, t') \in R\};$   
**if**  $N \notin \text{Tran}_\nu(t)$  **then return** false;  
**;**

**return** true;

---

Second, assume that the right-hand side is in pCNF. Note that Condition (2) can be equivalently stated as

$$\forall M : \text{match}_t(M) \notin \text{Tran}_\nu(t) \Rightarrow M \notin \text{Tran}_\mu(s) \quad (6)$$

As  $\Phi(t)$  is in conjunctive normal form then  $N \in \text{Tran}_\nu(t)$  is equivalent to saying that  $N$  has nonempty intersection with each clause of  $\Phi(t)$ . We may thus enumerate all maximal  $N \notin \text{Tran}_\nu(t)$ . Having a maximal  $N \notin \text{Tran}_\nu(t)$ , we can easily construct  $M$  such that  $N = \text{match}_t(M)$ . This leads to the polynomial time Algorithm 2.

The statement of the proposition thus follows. ■

**Proposition 10.** *Modal refinement on parameter-free PMTS is in coNP, if the right-hand side is positive.*



---

**Algorithm 2:** Test for Condition (2) of modal refinement (pCNF)
 

---

**Input** : states  $s$  and  $t$  such that  $\Phi(s)$  is positive and  $\Phi(t)$  is in positive CNF,  
 relation  $R$   
**Output**: *true* if  $s, t$  satisfy the refinement condition, *false* otherwise  
**foreach** clause  $(a_1, t_1) \vee \dots \vee (a_k, t_k)$  in  $\Phi(t)$  **do**  
      $M \leftarrow T(s) \setminus \{(a, s') \in T(s) \mid \exists i : a_i = a \wedge (s', t_i) \in R\};$   
     **if**  $M \in \text{Tran}_\mu(s)$  **then return** false;  
     ;  
**return** true;

---

*Proof.* Due to Lemma 4 we can solve the two refinement conditions separately. Furthermore, both Condition (1) and (2) of Lemma 4 can be checked in coNP. The guessing of  $R$  is done co-inductively as described in the proof of Proposition 8. ■

**Proposition 11.** *Modal refinement on parameter-free systems is coNP-hard, even if the left-hand side is in positive CNF and the right-hand side is in positive DNF.*

*Proof.* We reduce SAT into non-refinement. Let  $\varphi(x_1, \dots, x_n)$  be a formula in CNF. We modify  $\varphi$  into an equivalent formula  $\varphi'$  as follows: add new variables  $\tilde{x}_1, \dots, \tilde{x}_n$  and for all  $i$  change all occurrences of  $\neg x_i$  into  $\tilde{x}_i$  and add new clauses  $(x_i \vee \tilde{x}_i)$  and  $(\neg x_i \vee \neg \tilde{x}_i)$ .

Observe now that all clauses contain either all positive literals or all negative literals. Let  $\psi^+$  denote a CNF formula that contains all positive clauses of  $\varphi'$  and  $\psi^-$  denote a CNF formula that contains all negative clauses of  $\varphi'$ . As  $\varphi' = \psi^+ \wedge \psi^-$  it is clear that  $\varphi'$  is satisfiable if and only if  $(\psi^+ \Rightarrow \neg \psi^-)$  is not valid.

Now we construct two PMTSs  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  over  $\Sigma = \{x_1, \dots, x_n, \tilde{x}_1, \dots, \tilde{x}_n\}$  as follows:

1.  $S = \{s, s'\}$ ,  $T = \{(s, x_i, s'), (s, \tilde{x}_i, s') \mid 1 \leq i \leq n\}$ ,  
 $P = \emptyset$ ,  $\Phi(s) = \psi^+[(x_i, s')/x_i, (\tilde{x}_i, s')/\tilde{x}_i]$  and  $\Phi(s') = \mathbf{tt}$ , and
2.  $S' = \{t, t'\}$ ,  $T' = \{(t, x_i, t'), (t, \tilde{x}_i, t') \mid 1 \leq i \leq n\}$ ,  
 $P' = \emptyset$ ,  $\Phi(t) = \neg \psi^-[(x_i, t')/x_i, (\tilde{x}_i, t')/\tilde{x}_i]$  and  $\Phi(t') = \mathbf{tt}$ .

Note that by pushing the negation of  $\psi^-$  inside, this formula can be written as pDNF. It is easy to see that now  $s \leq_m t$  if and only if  $(\psi^+ \Rightarrow \neg \psi^-)$  is valid. Therefore,  $s \not\leq_m t$  if and only if  $\varphi$  is satisfiable. ■

Note that the exact complexity of modal refinement with the right-hand side being in positive CNF or MTS and the left-hand side Boolean remains open.

### 3.2 Systems with Parameters

In the sequel we investigate the complexity of refinement checking in the general case of PMTS with parameters. The complexities are summarized in Table 2. We start with an observation of how the results on parameter-free systems can be applied to the parametric case.

**Proposition 12.** *The complexity upper bounds from Table 1 carry over to Table 2, as follows. If the modal refinement in the parameter-free case is in NP, coNP or  $\Pi_2^P$ , then the modal refinement with parameters is in  $\Pi_2^P$ ,  $\Pi_3^P$  and  $\Pi_4^P$ , respectively. Moreover, if the left-hand side is an MTS, the complexity upper bounds shift from NP and  $\Pi_2^P$  to NP and  $\Sigma_3^P$ , respectively.*

*Proof.* In the first case, we first universally choose  $\mu$ , we then existentially choose  $\nu$  and modify the formulae  $\Phi(s)$  and  $\Phi(t)$  by evaluating the parameters. This does not change the normal form/positiveness of the formulae. We then perform the algorithm for the parameter-free refinement. For the second case note that implementations and MTS have no parameters and we may simply choose (existentially)  $\nu$  and run the algorithm for the parameter-free refinement. ■

We now focus on the respective lower bounds.

**Proposition 13.** *Modal refinement between an implementation and a right-hand side in positive CNF or in DNF is NP-hard.*

*Proof.* The proof is by reduction from SAT. Let  $\varphi(x_1, \dots, x_n)$  be a formula in CNF and let  $\varphi_1, \varphi_2, \dots, \varphi_k$  be the clauses of  $\varphi$ . We construct two PMTSs  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  over the action alphabet  $\Sigma = \{a_1, \dots, a_k\}$  as follows:

1.  $S = \{s, s'\}$ ,  $T = \{(s, a_i, s') \mid 1 \leq i \leq k\}$ ,  
 $P = \emptyset$ ,  $\Phi(s) = \bigwedge_{1 \leq i \leq k} (a_i, s')$  and  $\Phi(s') = \mathbf{tt}$  and

	Boolean	positive	pCNF	pDNF
Boolean	$\Pi_4^P$ -complete	$\Pi_3^P$ -complete	$\in \Pi_3^P$ $\Pi_2^P$ -hard	$\Pi_3^P$ -complete
positive	$\Pi_4^P$ -complete	$\Pi_3^P$ -complete	$\Pi_2^P$ -complete	$\Pi_3^P$ -complete
pCNF	$\Pi_4^P$ -complete	$\Pi_3^P$ -complete	$\Pi_2^P$ -complete	$\Pi_3^P$ -complete
pDNF	$\Pi_4^P$ -complete	$\Pi_2^P$ -complete	$\Pi_2^P$ -complete	$\Pi_2^P$ -complete
MTS	$\Sigma_3^P$ -complete	NP-complete	NP-complete	NP-complete
Impl	NP-complete	NP-complete	NP-complete	NP-complete

**Table 2:** Complexity of modal refinement checking with parameters.

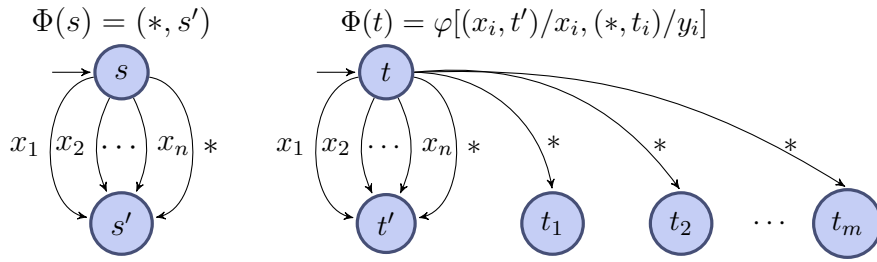
2.  $S' = \{t\} \cup \{t_i \mid 1 \leq i \leq k\}$ ,  $T' = \{(t, a_i, t_i) \mid 1 \leq i \leq k\}$ ,  
 $P' = \{x_1, \dots, x_n\}$ ,  $\Phi'(t) = \bigwedge_{1 \leq i \leq k} (a_i, t_i)$  and  $\Phi'(t_i) = \varphi_i$  for all  $1 \leq i \leq k$ .

Notice that each  $\varphi_i$  in  $\Phi'(t_i)$  is in positive form as we negate only the parameters  $x_i$  and every clause  $\varphi_i$  is trivially in DNF. Now we easily get that  $s \leq_m t$  if and only if  $\varphi$  is satisfiable. ■

**Proposition 14.** *Modal refinement is  $\Sigma_3^P$ -hard even if the left-hand side is MTS.*

*Proof.* The proof is done using the construction of the proof of Proposition 7 with parameters added on the right-hand side.

We will make a reduction from the validity of the quantified Boolean formula  $\psi \equiv \exists z_1, \dots, z_k : \forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_m : \varphi(z_1, \dots, z_k, x_1, \dots, x_n, y_1, \dots, y_m)$  to the refinement checking problem  $s \leq_m t$  where  $s$  and  $t$  are given as follows. Moreover, the right-hand side system has  $\{z_1, \dots, z_k\}$  as its set of parameters.



Assume that  $\psi$  is true. Then there exists a valuation  $\nu$  on  $\{z_1, \dots, z_k\}$  such that  $\forall x : \exists y : \varphi(x, y)$  is true. Using now the same argument as that in the proof of Proposition 7, we get that  $s \leq_m t$ .

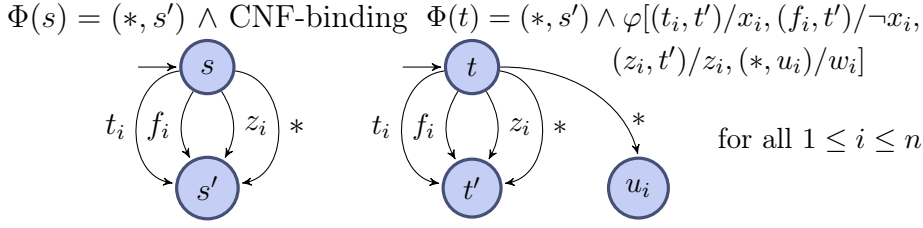
On the other hand let  $\psi$  be false and let  $\nu$  be an arbitrary valuation on  $\{z_1, \dots, z_k\}$ . We then may again use the reasoning in the proof of Proposition 7 to get that  $s \not\leq_m t$ . ■

The following proof introduces a gadget used also later on in other hardness results. We refer to it as *CNF-binding*. Further, we use the  $*$ -construction here.

**Proposition 15.** *Modal refinement is  $\Pi_4^P$ -hard even if the left-hand side is in positive CNF.*

*Sketch.* Consider a  $\Pi_4^P$ -hard QSAT instance, a formula  $\psi = \forall x \exists y \forall z \exists w : \varphi(x, y, z, w)$  with  $\varphi$  in CNF and  $x, y, z, w$  vectors of length  $n$ . We construct two system  $s$  and  $t$  and use the variables  $\{x_1, \dots, x_n\}$  as parameters for the left-hand side system  $s$ , and  $\{y_1, \dots, y_n\}$  as parameters for the right-hand side system  $t$ .

## A. Parametric Modal Transition Systems



On the left we require  $\Phi(s) = (*, s') \wedge \bigwedge_{1 \leq i \leq n} ((x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s')))$  and call the latter conjunct *CNF-binding*. Thus the value of each parameter  $x_i$  is “saved” into transitions of the system. Note that although both  $t_i$  and  $f_i$  may be present, a “minimal” implementation contains exactly one of them. On the right-hand side the transitions look similar but we require  $\Phi(t) = (*, t) \wedge \varphi'$  where  $\varphi'$  is created from  $\varphi$  by changing every positive literal  $x_i$  into  $(t_i, t')$ , every negative literal  $\neg x_i$  into  $(f_i, t')$ , every  $z_i$  into  $(z_i, t')$ , and every  $w_i$  into  $(*, u_i)$ .

We show that  $\psi$  is true iff  $s \leq_m t$ . Assume first that  $\psi$  is true. Therefore, for every choice of parameters  $x_i$  there is a choice of parameters  $y_i$  so that  $\forall z \exists w : \varphi(x, y, z, w)$  is true and, moreover,  $t_i$  or  $f_i$  is present on the left whenever  $x_i$  or  $\neg x_i$  is true, respectively (and possibly even if it is false). We set exactly all these transitions  $t_i$  and  $f_i$  on the right, too. Further, for every choice of transitions  $z_i$  on the left there are  $w_i$ 's so that  $\varphi(x, y, z, w)$  holds. On the right, we implement a transition  $(z_i, t')$  for each  $z_i$  set to true and  $(*, u_i)$  for each  $w_i$  set to true. Now  $\varphi'$  is satisfied as it has only positive occurrences of  $(t_i, t')$  and  $(f_i, t')$  and hence the extra  $t_i$ 's and  $f_i$ 's do not matter. Now for every implementation of  $s$  we obtained an implementation of  $t$ . Moreover, their transitions match. Indeed,  $t_i$ 's and  $f_i$ 's were set the same as on the left, similarly for  $z_i$ 's. As for the  $*$ -transition, we use the same argumentation as in the original  $*$ -construction. On the left, there is always one. On the right, there can be more of them due to  $w_i$ 's but at least one is also guaranteed by  $\Phi(t)$ .

Let now  $s \leq_m t$ . Then for every choice of  $x_i$ 's—and thus also for every choice of *exactly* one transition of  $t_i, f_i$  for each  $i$ —there are  $y_i$ 's so that every choice of transitions  $z_i$  can be matched on the right so that  $\varphi'$  is true with some transitions  $(*, u_i)$ . Since choices of  $t_i/f_i$  correspond exactly to choices of  $x_i$  it only remains to set  $w_i$  true for each transition  $(*, u_i)$  on the right, thus making  $\varphi$  true. ■

We now prove Proposition 16, followed by Proposition 17. The reason for this ordering is that the setting of Proposition 16 is simpler and the proof involves just one method, namely that of the CNF-binding, no  $*$ -construction is used and no additional actions are needed.

**Proposition 16.** *Modal refinement is  $\Pi_2^P$ -hard even if both sides are in positive CNF.*

*Proof.* Recall that positive means that there may be negations, but only limited to parameter literals. The proof is done by reduction from the validity of

### 3. Complexity of Modal Refinement Checking

$\forall x_1, \dots, x_n \exists y_1, \dots, y_m : \varphi(x_1, \dots, x_n, y_1, \dots, y_m)$ , where  $\varphi$  is in CNF. The idea is that the left-hand side has only  $x_i$  as parameters while the right-hand side has  $y_i$  as parameters. To ensure that the valuation of  $x_i$  is the same on both sides, we bind them through transitions.

Let  $\Sigma = \{t_1, \dots, t_n, f_1, \dots, f_n\}$  be the set of actions. The systems  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  are built as follows:

1.  $S = \{s, s'\}$ ,  
 $T = \{(s, t_i, s'), (s, f_i, s') \mid 1 \leq i \leq n\}$ ,  
 $P = \{x_1, \dots, x_n\}$ ,  
 $\Phi(s) = \bigwedge_{1 \leq i \leq n} ((x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s')))$  (note that this may be written in positive CNF),  
 $\Phi(s') = \mathbf{tt}$ ;
2.  $S' = \{t, t'\}$ ,  
 $T' = \{(t, t_i, t'), (t, f_i, t') \mid 1 \leq i \leq n\}$ ,  
 $P' = \{y_1, \dots, y_m\}$ ,  
 $\Phi'(t) = \varphi[(t_i, t')/x_i, (f_i, t')/\neg x_i]$ ,  
 $\Phi'(t') = \mathbf{tt}$ .

We now claim that  $\forall x \exists y : \varphi$  holds if and only if  $s \leq_m t$ . We show the two implications separately.

Let first  $\forall x \exists y : \varphi$  hold. Let  $\mu \subseteq P_1$  be arbitrary. As this is a truth valuation on the  $x_i$  variables, we know that there exists a valuation on the  $y_i$  variables such that  $\varphi$  holds. Let  $\nu \subseteq P_2$  be such a valuation. Let further  $M \in \text{Tran}_\mu(s)$  be arbitrary. Clearly, if  $x_i \in \mu$  then  $(t_i, s') \in M$  and if  $x_i \notin \mu$  then  $(f_i, s') \in M$ .

We set

$$N = \{(x, t') \mid (x, s') \in M\}$$

. Clearly, such  $N$  satisfies both conjuncts of the refinement definition. We need to show that  $N \in \text{Tran}_\nu(t)$ . We thus need to show that  $N$  satisfies all the clauses in  $\Phi'(t) = \varphi[(t_i, t')/x_i, (f_i, t')/\neg x_i]$ .

We use the fact that  $\varphi$  holds, given the current valuation  $\mu$  on  $x_i$  and  $\nu$  on  $y_i$ . Let  $(\ell_1 \vee \ell_2 \vee \dots \vee \ell_k)$  be an arbitrary clause of  $\varphi$ . Clearly, at least one literal was satisfied. If that literal was  $y_i$  or  $\neg y_i$  then the same literal appears in the modified clause of  $\Phi'(t)$  and we are done. If that literal was  $x_i$  then it has been changed into  $(t_i, t')$ , but as  $x_i \in \mu$ , we have that  $(t_i, t') \in N$ . Similarly, if that literal was  $\neg x_i$  then it has been changed into  $(f_i, t')$ , but as  $x_i \notin \mu$ , we have that  $(f_i, t') \in N$ . Thus  $s \leq_m t$ .

For the other implication let  $\exists x \forall y : \neg \varphi$  hold. We show that  $s \not\leq_m t$ . Let  $\mu$  be the valuation of  $x_i$  such that  $\exists x \forall y : \neg \varphi$  holds. Let  $\nu$  be arbitrary. This corresponds to a valuation on  $y_i$ .

We now set

$$M = \{(t_i, s') \mid x_i \in \mu\} \cup \{(f_i, s') \mid x_i \notin \mu\}.$$

Clearly,  $M \in \text{Tran}_\mu(s)$ . Let further  $N \in \text{Tran}_\nu(t)$ . (If  $\text{Tran}_\nu(t) = \emptyset$ , we are done.)

## A. Parametric Modal Transition Systems

We know that given the current  $x$  and  $y$  valuation,  $\varphi$  does not hold. This means that there exists at least one clause of  $\varphi$  that is false. Let  $(\ell_1 \vee \ell_2 \vee \dots \vee \ell_k)$  be such clause. All  $\ell_j$  are false, given current valuation  $\mu$  and  $\nu$ . However, the modified clause of  $\Phi'(t)$  corresponding to this one is satisfied by  $N$  (valuation of  $(t_i, t')$  and  $(f_i, f')$ ) as  $N \in \text{Tran}_\nu(t)$ .

Therefore, for some  $i$ , either  $(t_i, t') \in N$  while  $x_i \notin \mu$  or  $(f_i, t') \in N$  while  $x_i \in \mu$ . In both cases  $N$  does not satisfy the second conjunct part of the modal refinement definition. Therefore  $s \not\leq_m t$ .  $\blacksquare$

The next proposition again reuses the idea of CNF-binding in the very same fashion as above. Moreover, it handles more actions, more precisely those that appear as  $z_i$ 's in Proposition 15. Thus, the proof is the same, omitting the  $*$ -construction. Therefore, we only provide the reduction without repeating the formal arguments that it indeed works.

**Proposition 17.** *Modal refinement is  $\Pi_3^P$ -hard for the left-hand side in positive CNF and the right-hand side in positive DNF.*

*Proof.* The proof is done by reduction from the validity of the quantified Boolean formula  $\forall x_1, \dots, x_k \exists y_1, \dots, y_l \forall z_1, \dots, z_m : \varphi$  with  $\varphi$  in DNF.

Let the action alphabet be  $\Sigma = \{t_1, \dots, t_k, f_1, \dots, f_k, z_1, \dots, z_m\}$ .

The two systems  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  are built as follows:

1.  $S = \{s, s'\}$ ,  $T = \{(s, t_i, s'), (s, f_i, s') \mid 1 \leq i \leq k\} \cup \{(s, z_j, s') \mid 1 \leq j \leq m\}$ ,  
 $P = \{x_1, \dots, x_k\}$ ,  $\Phi(s) = \bigwedge_{1 \leq i \leq n} ((x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s')))$ , and  $\Phi(s') = \mathbf{tt}$ ;
2.  $S' = \{t, t'\}$ ,  $T' = \{(t, t_i, t'), (t, f_i, t') \mid 1 \leq i \leq k\} \cup \{(t, z_j, t') \mid 1 \leq j \leq m\}$ ,  
 $P' = \{y_1, \dots, y_k\}$ ,  $\Phi'(t) = \varphi[(t_i, t')/x_i, (f_i, t')/\neg x_i, (z_i, t')/z_i]$ ,  $\Phi'(t') = \mathbf{tt}$ .

Now  $\forall x \exists y \forall z : \varphi(x, y, z)$  holds if and only if  $s \leq_m t$ .  $\blacksquare$

We now modify the idea of CNF-binding to **DNF-binding** where instead of  $(x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s'))$  we use  $(x_i \wedge (t_i, s')) \vee (\neg x_i \wedge (f_i, s'))$  to bind parameters of left-hand side with transitions of right-hand side. The binding works slightly differently, as with DNF we are unable to make a conjunction of such formulae for all  $i$ . We thus employ a new special action  $\bullet$ . The left-hand side then first requires a  $\bullet$ -transition into  $n$  different states  $s_i$ , each requiring the above formula for its respective  $i$ .

**Proposition 18.** *Modal refinement is  $\Pi_2^P$ -hard even if left-hand side is in positive DNF and right-hand side is in positive CNF.*

*Proof.* The proof is done by reduction from the validity of the quantified Boolean formula  $\forall x_1, \dots, x_n \exists y_1, \dots, y_m : \varphi(x_1, \dots, x_n, y_1, \dots, y_m)$ , where  $\varphi$  is in CNF.

Let the action alphabet be  $\Sigma = \{t_1, \dots, t_n, f_1, \dots, f_n, \bullet\}$ . The two systems  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  are built as follows:

### 3. Complexity of Modal Refinement Checking

1.  $S = \{s, s'\} \cup \{s_i \mid 1 \leq i \leq n\}$ ,  $T = \{(s, \bullet, s_i), (s_i, t_i, s'), (s_i, f_i, s') \mid 1 \leq i \leq n\}$ ,  
 $P = \{x_1, \dots, x_n\}$ ,  $\Phi(s) = \bigwedge_i (\bullet, s_i)$ ,  $\Phi(s_i) = (x_i \wedge (t_i, s')) \vee (\neg x_i \wedge (f_i, s'))$ ,  $\Phi(s') = \mathbf{tt}$ ;
2.  $S' = \{t, t'\} \cup \{u_i, v_i \mid 1 \leq i \leq n\}$ ,  
 $T' = \{(t, \bullet, u_i), (t, \bullet, v_i), (u_i, t_i, t'), (u_i, f_i, t'), (v_i, f_i, t'), (v_i, t_i, t') \mid 1 \leq i \leq n\}$ ,  
 $P' = \{y_1, \dots, y_n\}$ ,  $\Phi'(t) = \varphi[(\bullet, u_i)/x_i, (\bullet, v_i)/\neg x_i]$ ,  
 $\Phi'(u_i) = (t_i, t')$ ,  $\Phi'(v_i) = (f_i, t')$ ,  $\Phi'(t') = \mathbf{tt}$ .

Now  $\forall x \exists y : \varphi(x, y)$  holds if and only if  $s \leq_m t$ . The reasoning behind this fact is similar to the proof of Proposition 16. ■

The proof of the next proposition is only a slight alteration of previous proof where the  $\bullet$ -construction is performed in two steps.

**Proposition 19.** *Modal refinement is  $\Pi_2^P$ -hard even if left-hand side is in positive DNF and right-hand side is in positive DNF.*

*Proof.* The proof is done by reduction from the validity of the quantified Boolean formula  $\forall x_1, \dots, x_n \exists y_1, \dots, y_m : \varphi(x_1, \dots, x_n, y_1, \dots, y_m)$ , where  $\varphi$  is in CNF. Let  $\varphi_1, \dots, \varphi_k$  denote the clauses of  $\varphi$ .

Let the action alphabet be  $\Sigma = \{t_1, \dots, t_n, f_1, \dots, f_n, \bullet\}$ . The two systems  $(S, T, P, \Phi)$  and  $(S', T', P', \Phi')$  are built as follows:

1.  $S = \{s, s', s''\} \cup \{s_i \mid 1 \leq i \leq n\}$ ,  
 $T = \{(s, \bullet, s')\} \cup \{(s', \bullet, s_i), (s_i, t_i, s''), (s_i, f_i, s'') \mid 1 \leq i \leq n\}$ ,  
 $P = \{x_1, \dots, x_n\}$ ,  $\Phi(s) = (\bullet, s')$ ,  $\Phi(s') = \bigwedge_i (\bullet, s_i)$ ,  
 $\Phi(s_i) = (x_i \wedge (t_i, s'')) \vee (\neg x_i \wedge (f_i, s''))$ ,  $\Phi(s'') = \mathbf{tt}$ ;
2.  $S' = \{t, t'\} \cup \{u_i, v_i \mid 1 \leq i \leq n\} \cup \{w_j \mid 1 \leq j \leq k\}$ ,  
 $T' = \{(t, \bullet, w_j) \mid 1 \leq j \leq k\} \cup \{(w_j, \bullet, u_i), (w_j, \bullet, v_i) \mid 1 \leq i \leq n, 1 \leq j \leq k\}$   
 $\cup \{(u_i, t_i, t'), (u_i, f_i, t'), (v_i, f_i, t'), (v_i, t_i, t') \mid 1 \leq i \leq n\}$ ,  
 $P' = \{y_1, \dots, y_n\}$ ,  $\Phi'(t) = \bigwedge_j w_j$ ,  $\Phi'(w_j) = \varphi'_j$  where  $\varphi'_j$  is created from  $\varphi_j$  by changing all positive literals  $x_i$  into  $(\bullet, u_i)$  and all negative literals  $\neg x_i$  into  $(\bullet, v_i)$ .  
 $\Phi'(u_i) = (t_i, t')$ ,  $\Phi'(v_i) = (f_i, t')$ ,  $\Phi'(t') = \mathbf{tt}$ .

Now  $\forall x \exists y : \varphi(x, y)$  holds if and only if  $s \leq_m t$ . ■

The proof of the last proposition is a combination of DNF-binding (including the  $\bullet$ -construction) with the previously used  $*$ -construction.

**Proposition 20.** *Modal refinement is  $\Pi_4^P$ -hard even if the left-hand side is in positive DNF.*

## A. Parametric Modal Transition Systems

*Proof.* The proof is done by reduction from the validity of the quantified Boolean formula  $\forall x \exists y \forall z \exists w : \varphi(x, y, z, w)$  where  $x, y, z, w$  are all  $n$ -dimensional binary vectors and  $\varphi$  is in CNF.

We let  $\Sigma = \{t_1, \dots, t_n, f_1, \dots, f_n, z_1, \dots, z_n, *, \bullet\}$  and we create the two systems  $(S, T, P, \Phi)$ ,  $(S', T', P', \Phi')$  over the action alphabet  $\Sigma$  as follows:

1.  $S = \{s, s'\} \cup \{s_i \mid 1 \leq i \leq n\}$ ,  
 $T = \{(s, \bullet, s_i), (s_i, t_i, s'), (s_i, f_i, s'), (s, z_i, s') \mid 1 \leq i \leq n\} \cup \{(s, *, s')\}$ ,  
 $P = \{x_1, \dots, x_n\}$ ,  $\Phi(s) = (*, s') \wedge \bigwedge_i (\bullet, s_i)$ ,  
 $\Phi(s_i) = (x_i \wedge (t_i, s')) \vee (\neg x_i \wedge (f_i, s'))$  for all  $1 \leq i \leq n$ ,  $\Phi(s') = \mathbf{tt}$ ;
2.  $S' = \{t, t'\} \cup \{u_i, v_i, w_i \mid 1 \leq i \leq n\}$ ,  
 $T' = \{(t, z_i, t'), (t, \bullet, u_i), (t, \bullet, v_i), (t, *, w_i), (u_i, t_i, t'), (u_i, f_i, t'), (v_i, t_i, t'), (v_i, f_i, t') \mid 1 \leq i \leq n\} \cup \{(t, *, t')\}$ ,  
 $P' = \{y_1, \dots, y_n\}$ ,  $\Phi'(t) = (*, t') \wedge \varphi[(\bullet, u_i)/x_i, (\bullet, v_i)/\neg x_i, (z_i, t')/z_i, (*, w_i)/w_i]$ ,  
for all  $1 \leq i \leq n$ :  $\Phi'(u_i) = (t_i, t')$ ,  $\Phi'(v_i) = (f_i, t')$ ,  $\Phi'(w_i) = \Phi'(t') = \mathbf{tt}$ .

It can be verified, using similar arguments as before, that  $s \leq_m t$  if and only if  $\forall x \exists y \forall z \exists w : \varphi(x, y, z, w)$ . ■

Although the complexity may seem discouraging in many cases, there is an important remark to make. The refinement checking may be exponential, but only in the outdegree of each state and the number of parameters, while it is polynomial in the number of states. As one may expect the outdegree and the number of parameters to be much smaller than the number of states, this means that the refinement checking may still be done in a rather efficient way. This claim is furthermore supported by the existence of efficient SAT solvers that may be employed to check the inner conditions in the modal refinement.

## 4 Conclusion and Future Work

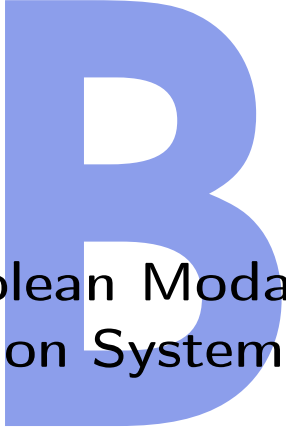
We have introduced an extension of modal transition systems called PMTS for parametric systems. The formalism is general enough to capture several features missing in the other extensions, while at the same time it offers an easy to understand semantics and a natural notion of modal refinement that specializes to the well-known refinements already studied on the subclasses of PMTS. Finally, we provided a comprehensive overview of complexity of refinement checking on PMTS and its subclasses.

We believe that our formalism is a step towards a more applicable notion of specification theories based on MTS.

In the future work we will study logical characterizations of the refinement relation, investigate compositional properties and focus on introducing quantitative aspects into the model in order to further increase its applicability.

**Acknowledgments.** We would like to thank to Sebastian Bauer for suggesting the traffic light example and for allowing us to use his figure environments.





# A Characterizing Logic for Boolean Modal Transition Systems

Mikael H. MØLLER      Kim G. LARSEN

*Aalborg University, Department of Computer Science, Denmark*

Xinxin LIU

*Institute of Software Chinese Academy of Sciences, China*

**Abstract** Modal transition systems (MTS) is a specification formalism with support for stepwise refinement. It is a well-studied formalism, and several different extensions and generalizations have been developed. The original MTS has been shown to be in close relation with Hennessy-Milner Logic (HML). We will in this paper define a logic for Boolean MTS (BMTS), and present the characteristic formula for such specifications, using a version of HML with recursion. We prove that our logic characterizes the modal refinement of BMTS. Further we apply our logic on the more general Parametric MTS (PMTS), where we show how to select the parameters of a PMTS under certain requirements expressed in the logic.

## 1 Introduction

Modal Transition Systems (MTS) [LT88; Ant+08] was developed as a flexible and easy-to-use compositional modeling framework for reactive systems. One of its core features is the support for stepwise refinement of specifications.

An MTS is basically a labeled transition system (LTS), but with two types of transitions: *may* transitions represent the only allowed behavior of a system specification and its implementations, and *must* transitions represent required behavior. This means that every must transition has to be implemented in all implementations of the specification, whereas the may transitions are optional. An MTS system specification yields a set of different implementations, where they all fulfill the requirements of the specification. A modal refinement relation on MTS describes whether a specification or implementation fulfills the modal requirements of another MTS specification.

Since the MTS was first introduced several extensions and generalizations have been presented, e.g. disjunctive MTS (DMTS)[LX90], 1-selecting MTS (1MTS) [FS08], transition systems with obligations (OTS)[BK11] and parametric MTS (PMTS)[Ben+11a].

MTS has been shown to be closely related to logics, in fact MTS may be seen as a fragment of a temporal logic [BL92]. Already in [Lar89] Larsen presented a logical characterization of MTS in Hennessy Milner Logic (HML), and in [Xin92] a similar characterization of DMTS was given. However it has been left as an open question whether the OTS and PMTS follow this trend.

OTS is a generalization of MTS, and PMTS is a further generalization of OTS. Instead of having the two may and must transition relations, both OTS and PMTS only have one transition relation, like a normal LTS. Instead the modalities of the transitions are expressed in an obligation formula for each state. The obligation function is a Boolean formula over outgoing transitions, and any set of outgoing transitions that satisfy the formula is a valid combination. In OTS, only positive Boolean obligation formulas over transitions are allowed, where in PMTS arbitrary boolean formulas over transitions and a set of prefixed parameters are allowed.

OTS and PMTS extend MTS in the sense that the (must and may) modalities assigned to transitions may be dependent. With this construction, OTS and PMTS support more complex modalities than just may and must.

In this paper we will initially work with Boolean MTS (BMTS), an OTS with arbitrary Boolean obligation formulas. This model was also presented as parameter-free PMTS in [Ben+11a].

We define an interpretation of the Hennessy Milner Logic with recursion [Lar88] to reason about BMTS specifications. We show that the logic characterizes the modal refinement relation on BMTSs. We apply the approach from [SI94], where the formula is actually a mutually recursive equation system, with an equation for each state of the specification.

Finally we apply the logic on the more general PMTS. In [Ben+11a] PMTS and the modal refinement relation on such specifications were defined. The modal refinement can be used to check whether a certain valuation of the parameters will refine a specification. However it is not known how one should select these parameters in order to refine a

specification.

As a PMTS with instantiated parameters is a BMTS, we will use our logic to solve this problem. We show that one can compute all the possible ways of instantiating the parameters, such that the resulting BMTS will satisfy the formula. This is done by expressing the problem as a mutually recursive equation system, and showing that it always has a unique maximal fix-point.

In relation to this work, the formalism of Feature Transition Systems (FTS), presented in [Cla+10], is somewhat related to PMTS. A FTS is just a LTS, with an additional label on each transition, namely the feature this transition is a part of. The formalism is used in Software Product Lines, and each feature can be either optional or required, like may and must in the family of MTSs. This notion of features is very similar to the parameters of PMTS, which can be used to model features in the same way, but is a more general concept. In addition there are also no modalities on the transitions in FTS. Finally, very related to our last result, [Cla+10] also gives a model checking algorithm, that given a requirement in LTL, lists the products (combinations of features) that fulfill the requirement.

## 2 Boolean Modal Transition Systems

We will now formally introduce the syntax and semantics of the computational model Boolean MTS (BMTS).

A *Boolean formula* over a set  $X$  of atomic propositions is given by the following abstract syntax

$$\varphi ::= \mathbf{tt} \mid x \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

where  $x$  ranges over  $X$ . The set of all Boolean formulas over the set  $X$  is denoted by  $\mathcal{B}(X)$ . The satisfaction relation is defined as usual. We say that a formula is *satisfiable* if there is at least one

**Definition 1.** A Boolean MTS (BMTS) over an action alphabet  $\Sigma$  is a tuple  $(S, T, \Phi, s_0)$  where

- $S$  is a set of states,
- $T \subseteq S \times \Sigma \times S$  is a transition relation,
- $\Phi : S \rightarrow \mathcal{B}(\Sigma \times S)$  is a satisfiable obligation function over the atomic propositions containing outgoing transitions, and
- $s_0 \in S$  is the initial state.

We implicitly assume that whenever  $(a, t)$  occurs in  $\Phi(s)$  then  $(s, a, t) \in T$ . By  $T(s) = \{(a, t) \mid (s, a, t) \in T\}$  we denote the set of all outgoing transitions of  $s$ .

## B. A Characterizing Logic for Boolean Modal Transition Systems

Note that this definition is almost identical to the definition of OTS. In OTS the obligation function is restricted to only return positive Boolean formulas. If we restrict the obligation function to return formulas in conjunctive normal form, then the definition would be equivalent to the one of Disjunctive MTS, furthermore if only conjunctions of outgoing transitions were allowed we have the definition of regular MTS. The BMTS formalism is also known as parameter-free PMTS as defined in [Ben+11a].

An *implementation* of a BMTS specification is a BTMS where the obligation function for each state is a conjunction of all outgoing transitions, i.e. all transitions are must transitions.

The obligation formulas might be satisfied by several different sets of outgoing transitions, thus we define  $\text{Tran}(s) = \{M \mid M \subseteq \Sigma \times S : M \models \Phi(s)\}$ . Intuitively  $\text{Tran}(s)$  is the set of all possible sets of outgoing transitions from  $s$ .

Clearly with these BMTSs, we can express many different kinds of modalities on transitions. We will still use the notion may and must, even though it is not as clear anymore.

A must transition is a transition that is required in any implementation. Thus a transition  $(s, a, s')$  is a must transition if  $(a, s') \in M$  for all  $M \in \text{Tran}(s)$ . We draw must transitions as full lined arrows.

A may transition is a transition that can be in an implementation. Thus a transition  $(s, a, s')$  is a may transition if  $(a, s') \in M$  for some  $M \in \text{Tran}(s)$ . We draw may transitions as dashed lined arrows.

Before we define the refinement relation, let us have a look at a small example.

*Example 2.* Assume a specification of a simple vending machine. The vending machine must be able to receive a Coin. After receiving a Coin, the vending machine may deliver one out of three products, either Snack, Water or Beer. Finally the vending machine must offer at least two of the products.

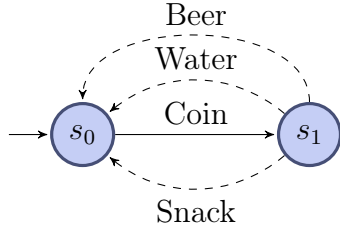
A specification of such vending machine is given as  $\mathcal{S}_1$  in Figure 1a. The system starts in state  $s_0$ , where it must receive a Coin, *must* as the only way to satisfy  $\Phi(s_0)$  is having the Coin transition. After receiving a Coin, the vending machine may deliver one of three products, and return to  $s_0$ .

The requirement that the vending machine must offer at least two products is expressed in the obligation function  $\Phi(s_1)$ , which is a disjunction over possible outgoing transitions with two products. Having all three products is also fine, as this also satisfies the formula. Note that this requirement is something that one cannot express in a regular MTS.

If we had to refine the requirement, so that the vending machine must always offer Snack, we can do this by modifying  $\Phi(s_1)$ , as done in  $\mathcal{S}_2$  in Figure 1b. We say that  $\mathcal{S}_2$  is a modal refinement of  $\mathcal{S}_1$ , as it does not violate any requirement of  $\mathcal{S}_1$ .

A BMTS specification yields a set of implementations, two implementations of  $\mathcal{S}_1$  are shown in Figure 1c and Figure 1d.

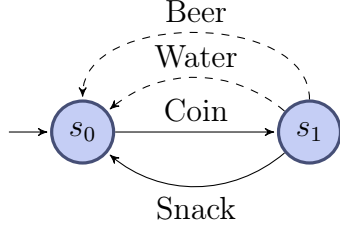
We will now formally define the notion of modal refinement between BMTSs. The definition is a modified version of the one presented for PMTS in [Ben+11a].


 (a) BMTS specification  $\mathcal{S}_1$ .

**Obligation function:**

$$\Phi(s_0) = (\text{Coin}, s_1)$$

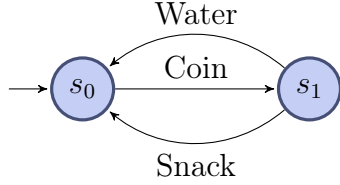
$$\begin{aligned} \Phi(s_1) = & ((\text{Snack}, s_0) \wedge (\text{Water}, s_0)) \\ & \vee ((\text{Snack}, s_0) \wedge (\text{Beer}, s_0)) \\ & \vee ((\text{Water}, s_0) \wedge (\text{Beer}, s_0)) \end{aligned}$$


 (b) BMTS specification  $\mathcal{S}_2$ .

**Obligation function:**

$$\Phi(s_0) = (\text{Coin}, s_1)$$

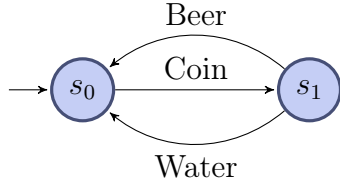
$$\begin{aligned} \Phi(s_1) = & ((\text{Snack}, s_0) \wedge (\text{Water}, s_0)) \\ & \vee ((\text{Snack}, s_0) \wedge (\text{Beer}, s_0)) \end{aligned}$$


 (c) BMTS implementation  $\mathcal{I}_1$ .

**Obligation function:**

$$\Phi(s_0) = (\text{Coin}, s_1)$$

$$\Phi(s_1) = ((\text{Snack}, s_0) \wedge (\text{Water}, s_0))$$


 (d) BMTS implementation  $\mathcal{I}_2$ .

**Obligation function:**

$$\Phi(s_0) = (\text{Coin}, s_1)$$

$$\Phi(s_1) = ((\text{Water}, s_0) \wedge (\text{Beer}, s_0))$$

**Figure 1:** BMTS example of a simple vending machine.

**Definition 2** (Modal Refinement). *Let  $(S_1, T_1, \Phi_1, s_0)$  and  $(S_2, T_2, \Phi_2, t_0)$  be two BMTSs. A binary relation  $R \subseteq S_1 \times S_2$  is a modal refinement if  $\forall (s, t) \in R$  it holds that*

$$\begin{aligned} \forall M \in \text{Tran}(s) : \exists N \in \text{Tran}(t) : \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \ \wedge \\ \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R . \end{aligned}$$

*We say that  $s$  modally refines  $t$ , denoted by  $s \leq_m t$ , if there exists a modal refinement  $R$  such that  $(s, t) \in R$ .*

Note that this definition of modal refinement coincide with the modal refinement for

MTS, DMTS and OTS, if we restrict our models as described after the definition of BMTS.

*Example 3.* If we again look at the specifications  $\mathcal{S}_1$  and  $\mathcal{S}_2$  and implementations  $\mathcal{I}_1$  and  $\mathcal{I}_2$ . It should be clear that  $\mathcal{S}_2 \leq_m \mathcal{S}_1$ ,  $\mathcal{I}_1 \leq_m \mathcal{S}_1$ ,  $\mathcal{I}_2 \leq_m \mathcal{S}_1$ , and  $\mathcal{I}_1 \leq_m \mathcal{S}_2$ . But as Snack is not implemented in  $\mathcal{I}_2$ , we have that  $\mathcal{I}_2 \not\leq_m \mathcal{S}_2$ .

### 3 Characterizing Logic

In this section we define a new logic for reasoning about BMTS specifications. The logic is based on Hennessy Milner Logic (HML) with recursion [Lar88]. Originally [HM85] HML was defined on LTS and has been proven to characterize bisimulation.

The syntax is identical to HML with recursion, we actually only extend the semantics to capture the modalities of BMTS. We follow the approach used in [SI94], for more see [Ace+07]. The main idea is to present a way to construct the characteristic formula for a BMTS, and then prove that the satisfaction of the logic coincide with modal refinement checking.

We build the characteristic formula as a mutually recursive equation system with an equation for each state of the BMTS.

Let  $\mathcal{X} = \{X_1, \dots, X_n\}$  be a set of variables, then the set  $\mathcal{M}_{\mathcal{X}}$  of formulas over a set of actions  $\Sigma$  is given by the following abstract syntax.

$$F, G ::= X \mid \mathbf{tt} \mid \mathbf{ff} \mid F \wedge G \mid F \vee G \mid \langle a \rangle F \mid [a]F ,$$

where  $a \in \Sigma$  and  $X \in \mathcal{X}$ .

A mutually recursive equation system is a set of equations like

$$\begin{aligned} X_1 &= F_{X_1} \\ X_2 &= F_{X_2} \\ &\vdots \\ X_n &= F_{X_n} , \end{aligned}$$

where  $X_1, \dots, X_n$  are unique variables and  $F_{X_1}, \dots, F_{X_n}$  are logic formulas that each may contain several of the variables. The equation system is formally given by a declaration, a function  $D : \mathcal{X} \rightarrow \mathcal{M}_{\mathcal{X}}$ , assigning a formula to each variable in  $\mathcal{X}$ .

Given a BMTS  $(S, T, \Phi, s_0)$  over a set of actions  $\Sigma$  we define the denotational semantics as a function  $\mathcal{O}_F : (2^S)^n \rightarrow 2^S$  inductively for  $F \in \mathcal{M}_{\mathcal{X}}$  by

$$\begin{aligned} \mathcal{O}_{X_i}(S_1, \dots, S_n) &= S_i \\ \mathcal{O}_{\mathbf{tt}}(S_1, \dots, S_n) &= S \\ \mathcal{O}_{\mathbf{ff}}(S_1, \dots, S_n) &= \emptyset \\ \mathcal{O}_{F \wedge G}(S_1, \dots, S_n) &= \mathcal{O}_F(S_1, \dots, S_n) \cap \mathcal{O}_G(S_1, \dots, S_n) \\ \mathcal{O}_{F \vee G}(S_1, \dots, S_n) &= \mathcal{O}_F(S_1, \dots, S_n) \cup \mathcal{O}_G(S_1, \dots, S_n) \\ \mathcal{O}_{\langle a \rangle F}(S_1, \dots, S_n) &= \langle \cdot a \cdot \rangle \mathcal{O}_F(S_1, \dots, S_n) \\ \mathcal{O}_{[a]F}(S_1, \dots, S_n) &= [\cdot a \cdot] \mathcal{O}_F(S_1, \dots, S_n) , \end{aligned}$$

where  $\langle \cdot a \cdot \rangle$  and  $[\cdot a \cdot]$  are defined as follows

$$\begin{aligned}\langle \cdot a \cdot \rangle S' &= \{s \in S \mid \forall M \in \text{Tran}(s), \exists (a, s'') \in M : s'' \in S'\} \\ [\cdot a \cdot] S' &= \{s \in S \mid \forall M \in \text{Tran}(s), \forall (a, s') \in M : s' \in S'\}.\end{aligned}$$

The modal logic presented here is basically HML with recursion, with  $[\cdot a \cdot]$  and  $\langle \cdot a \cdot \rangle$  defined to match must and may modalities in BMTSs.

We define  $\langle \cdot a \cdot \rangle$  so it corresponds to a must transition, recall that must means that there has to be an  $a$  transition no matter how one satisfies the obligation formula. Note that it is enough to quantify over  $\forall M \in \text{Tran}(s)$ , as  $\Phi(s)$  is satisfiable, so there exists at least one  $M \in \text{Tran}(s)$ .

We define  $[\cdot a \cdot]$  so it corresponds to may transitions, i.e. a transition that can be there in any of the possible configurations of a state. Also note that the definition of  $[\cdot a \cdot]$  and  $\langle \cdot a \cdot \rangle$  are closely related to the two requirements in modal refinement in Definition 2.

The return value of  $\mathcal{O}_F(S_1, \dots, S_n)$  is the set of states satisfying  $F$  assuming that  $S_i$  is the set of states satisfying  $X_i$  for  $1 \leq i \leq n$ .

As we are dealing with a mutually recursive equation system declared by a function  $D$ , the solution of such can be a maximal fix-point. Let  $\llbracket D \rrbracket$  be the *semantic function*  $\llbracket D \rrbracket : \mathcal{D} \rightarrow \mathcal{D}$  defined by

$$\llbracket D \rrbracket(S_1, \dots, S_n) = (\mathcal{O}_{F_{X_1}}(S_1, \dots, S_n), \dots, \mathcal{O}_{F_{X_n}}(S_1, \dots, S_n)),$$

where  $S_i \subseteq S$  for  $1 \leq i \leq n$ .

As  $\llbracket D \rrbracket$  is monotonic, Tarski's fixed-point theorem gives us that there exists a unique maximal fix-point. We define the satisfaction relation for every  $i$  as

$$s \models_{\max} D(X_i) \text{ iff } s \in S_i, \text{ where } (S_1, \dots, S_n) = \bigcap \{O \mid O \supseteq \llbracket D \rrbracket(O)\}.$$

We only consider the maximal fix-points, so from now on we will omit the *max* subscript. We shall sometimes abuse the notation a bit, by applying the satisfaction relation directly on a BTMS and a declaration, like  $\mathcal{S} \models D_{\mathcal{T}}$ . This just means that  $s_0 \models D_{\mathcal{T}}(X_{t_0})$ , where  $s_0$  and  $t_0$  are the initial states of  $\mathcal{S}$  and  $\mathcal{T}$ .

Now given a BTMS  $\mathcal{T}$  we will define a declaration  $D_{\mathcal{T}}$  such that for all other BMTSs  $\mathcal{S}$  we know that  $\mathcal{S}$  is in modal refinement of  $\mathcal{T}$  if and only if  $\mathcal{S}$  satisfies  $D_{\mathcal{T}}$ .

**Theorem 3.** *Given two BMTSs  $\mathcal{T} = (S_1, T_1, \Phi_1, t_0)$ ,  $\mathcal{S} = (S_2, T_2, \Phi_2, s_0)$ . Let  $D_{\mathcal{T}}$  be constructed as follows for each state  $t \in S_1$*

$$D_{\mathcal{T}}(X_t) = \bigvee_{N \in \text{Tran}(t)} \left[ \left( \bigwedge_{(a, t') \in N} \langle a \rangle X_{t'} \right) \wedge \bigwedge_{a \in \Sigma} [a] \left( \bigvee_{(a, t') \in N} (X_{t'}) \right) \right],$$

*then*

$$\mathcal{S} \models D_{\mathcal{T}} \text{ iff } S \leq_m T.$$

## B. A Characterizing Logic for Boolean Modal Transition Systems

We note that (3) generalizes the characteristic formula construction in [SI94] and [Ace+07] with the disjunction in front.

Before we prove Theorem 3, let us again have a look at the example from last section.

*Example 4.* The logical characterization of  $\mathcal{S}_1$  looks like this,

$$\begin{aligned}
D_{\mathcal{S}_1}(X_{s_1}) &= (\langle \text{Coin} \rangle X_{s_2}) \wedge ([\text{Coin}] X_{s_2} \wedge [\text{Snack}] \mathbf{ff} \wedge [\text{Water}] \mathbf{ff} \wedge [\text{Beer}] \mathbf{ff}) \\
D_{\mathcal{S}_1}(X_{s_2}) &= \left( (\langle \text{Snack} \rangle X_{s_1} \wedge \langle \text{Water} \rangle X_{s_1} \wedge \langle \text{Beer} \rangle X_{s_1}) \wedge \right. \\
&\quad \left. ([\text{Coin}] \mathbf{ff} \wedge [\text{Snack}] X_{s_1} \wedge [\text{Water}] X_{s_1} \wedge [\text{Beer}] X_{s_1}) \right) \vee \\
&\quad \left( (\langle \text{Snack} \rangle X_{s_1} \wedge \langle \text{Water} \rangle X_{s_1}) \wedge \right. \\
&\quad \left. ([\text{Coin}] \mathbf{ff} \wedge [\text{Snack}] X_{s_1} \wedge [\text{Water}] X_{s_1} \wedge [\text{Beer}] \mathbf{ff}) \right) \vee \\
&\quad \left( (\langle \text{Snack} \rangle X_{s_1} \wedge \langle \text{Beer} \rangle X_{s_1}) \wedge \right. \\
&\quad \left. ([\text{Coin}] \mathbf{ff} \wedge [\text{Snack}] X_{s_1} \wedge [\text{Water}] \mathbf{ff} \wedge [\text{Beer}] X_{s_1}) \right) \vee \\
&\quad \left( (\langle \text{Water} \rangle X_{s_1} \wedge \langle \text{Beer} \rangle X_{s_1}) \wedge \right. \\
&\quad \left. ([\text{Coin}] \mathbf{ff} \wedge [\text{Snack}] \mathbf{ff} \wedge [\text{Water}] X_{s_1} \wedge [\text{Beer}] X_{s_1}) \right).
\end{aligned}$$

We will prove Theorem 3 by the following two lemmas.

**Lemma 4.** *Let  $D_{\mathcal{T}}(X_t)$  be defined as in (3), then for each state  $t \in S_1$  we have*

$$s \leq_m t \text{ implies } s \models D_{\mathcal{T}}(X_t),$$

*for all  $s \in S_2$ .*

*Proof.* Assume that  $s \leq_m t$ . We will show that  $s \in \llbracket D_{\mathcal{T}}(X_t) \rrbracket$ , the set of all states that satisfy  $D_{\mathcal{T}}(X_t)$ . Further let  $[t]_{\leq_m} = \{s \mid s \leq_m t\}$  be the set of all states in modal refinement of  $t$ .

As the satisfaction relation is defined as a maximal fix-point, we need to show that

$$s \in \bigcup_{N \in \text{Tran}(t)} \left[ \left( \bigcap_{(a,t') \in N} \langle \cdot a \cdot \rangle [t']_{\leq_m} \right) \cap \bigcap_{a \in \Sigma} [\cdot a \cdot] \left( \bigcup_{(a,t') \in M} [t']_{\leq_m} \right) \right].$$



Assume  $M \in \text{Tran}(s)$ . By definition of modal refinement we know that there exists  $N \in \text{Tran}(t)$ , such that the two requirements for modal refinement are fulfilled. Let us now use this  $N$ , and prove the two following cases:

1.  $s \in \left( \bigcap_{(a,t') \in N} \langle \cdot a \cdot \rangle [t']_{\leq m} \right)$
2.  $s \in \bigcap_{a \in \Sigma} [\cdot a \cdot] \left( \bigcup_{(a,t') \in O} [t']_{\leq m} \right)$

Proof of 1. Assume  $(a, t') \in N$ . By modal refinement we know that for all  $(a, t') \in N$  there exists  $(a, s') \in M$  such that  $s' \leq_m t'$ . Thus we have that  $s \in \left( \bigcap_{(a,t') \in N} \langle \cdot a \cdot \rangle [t']_{\leq m} \right)$ .

Proof of 2. Assume  $(a, s') \in M$ . By modal refinement we get that there exists  $(a, t') \in N$  st.  $s' \in [t']_{\leq m}$ . This gives us  $s \in \bigcap_{a \in \Sigma} [\cdot a \cdot] \left( \bigcup_{(a,t') \in N} [t']_{\leq m} \right)$ . ■

**Lemma 5.** *Let  $D_{\mathcal{T}}(X_t)$  be defined as in (3), then for each state  $t \in S_1$  we have*

$$s \models D_{\mathcal{T}}(X_t) \text{ implies } s \leq_m t,$$

*for all  $s \in S_2$ .*

This lemma can be proved in the same manner as Lemma 4. By these two lemmas we have proved Theorem 3.

Unfortunately the size of the formula is exponential in the given BMTS. In the worst case there can be exponentially many  $N \in \text{Tran}(s)$ , so it is exponential in the number of outgoing transitions. We could however not expect it to be smaller, as modal refinement checking of BMTS was shown to be  $\Pi_2^P$ -complete in [Ben+11a].

## 4 Parametric Modal Transition System

In this section we will apply our logic from last section on a more general class of MTS. In [Ben+11a] Parametric MTS and modal refinement for such specifications were defined. A PMTS is a BMTS with an additional set of parameters that can be used in the obligation formulas. The parameters of a PMTS can not be altered when they have been instantiated, thus a PMTS represents a set of BMTSs, one for each truth assignment of the parameters.

The modal refinement can be used to check whether a given truth assignment of the parameters is a refinement of a specification. However it has not been investigated how one should choose these parameters in order to satisfy a given requirement. To do this we use our logic to specify the system requirements and define a new semantic of the logic on PMTS.

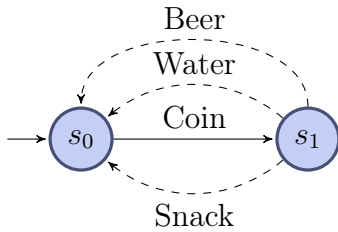
First, we formally define the parametric MTS formalism.

**Definition 6.** A parametric MTS (PMTS) over an action alphabet  $\Sigma$  is a tuple  $(S, T, P, \Phi, s_0)$  such that,

- $S$  is a set of states,
- $T \subseteq S \times \Sigma \times S$  is a transition relation,
- $P$  is a finite set of parameters,
- $\Phi : S \rightarrow \mathcal{B}((\Sigma \times S) \cup P)$  is a satisfiable obligation function over the atomic propositions containing outgoing transitions and parameters, and
- $s_0 \in S$  is the initial state.

One can see a PMTS as a function of the parameters, that for a given valuation returns a BMTS. We shall sometimes use the notation  $s(\nu)$  for a state  $s$  in a BMTS that is derived from a PMTS with the parameter valuation  $\nu \subseteq P$ .

Like with BMTS, the obligation formulas might be satisfied by several different sets of outgoing transitions. Again we introduce the handy notation of the set of all admissible sets of transitions from  $s$  under a fixed valuation of the parameters. Let  $(S, T, P, \Phi, s_0)$  be a PMTS and  $\nu \subseteq P$  be a truth assignment. For  $s \in S$ , we define  $\text{Tran}_\nu(s) = \{N \subseteq T(s) \mid N \cup \nu \models \Phi(s)\}$ .



**Parameters:**

$\{\text{MustSnack}\}$

**Obligation function:**

$\Phi(s_0) = (\text{Coin}, s_1)$

$\Phi(s_1) = (\text{MustSnack} \Leftrightarrow (\text{Snack}, s_0)) \wedge$   
 $((\text{Snack}, s_0) \wedge (\text{Water}, s_0))$   
 $\vee ((\text{Snack}, s_0) \wedge (\text{Beer}, s_0))$   
 $\vee ((\text{Water}, s_0) \wedge (\text{Beer}, s_0))$

**Figure 2:** PMTS Specification  $\mathcal{S}_3$  of a vending machine.

*Example 5.* In Figure 2,  $\mathcal{S}_3$  is a PMTS specification of the running example of the vending machine. We have added the parameter `MustSnack`, which is used to model a persistent choice of having `Snack` as an option or not. We have modeled this by the bi-implication in  $\Phi(s_1)$ . In this way we only allow implementations that always can do `Snack` or never does `Snack`. Note that this is something we could not model in BMTS, as there is no notion of persistency in BMTS.

We will now define the semantics, on PMTS, of our logic defined in last section. As

a PMTS can be seen as a set of BMTSs, the PMTS should only satisfy a formula if there is a truth assignment of the parameters such that the resulting BMTS satisfies the formula. Formally this means that given a PMTS  $(S, T, P, \Phi, s_0)$ ,

$$s \models_P F \text{ iff } \exists \nu \in P \text{ such that } s(\nu) \models F,$$

where  $s$  is a state in the given PMTS.

In order to decide whether such truth assignment exists, we will again use a mutually recursive equation system.

We will evaluate the expression  $s \models_P F$  as the set of truth assignments that yields a BMTS that satisfy the formula  $F$ , that is  $\llbracket s \models_P F \rrbracket = \{\nu \mid s(\nu) \models F\}$ . If the set is empty, then  $s$  does not satisfy the formula.

Given a PMTS  $(S, T, P, \Phi, s_0)$ , a state  $s \in S$  and a formula  $F$  of our logic defined in last section, we build an equation system of the following rules.

$$\begin{aligned} \llbracket s \models_P X \rrbracket &= \llbracket s \models_P F_X \rrbracket, \text{ where } X = F_X \\ \llbracket s \models_P \mathbf{tt} \rrbracket &= 2^P \\ \llbracket s \models_P \mathbf{ff} \rrbracket &= \emptyset \\ \llbracket s \models_P F \wedge G \rrbracket &= \llbracket s \models_P F \rrbracket \cap \llbracket s \models_P G \rrbracket \\ \llbracket s \models_P F \vee G \rrbracket &= \llbracket s \models_P F \rrbracket \cup \llbracket s \models_P G \rrbracket \\ \llbracket s \models_P \langle a \rangle F \rrbracket &= \{\nu \in P \mid \forall M \in \text{Tran}_\nu(s) \exists (a, s') \in M . \nu \in \llbracket s' \models_P F \rrbracket\} \\ \llbracket s \models_P [a]F \rrbracket &= \{\nu \in P \mid \forall M \in \text{Tran}_\nu(s) \forall (a, s') \in M . \nu \in \llbracket s' \models_P F \rrbracket\} \end{aligned}$$

It is important to note that as the PMTS model and the formula are both finite, we will only get a finite number of equations.

The equations are each reasoning about subsets of parameters for which a state  $s$  satisfies a formula  $F$ . Thus the equation system is interpreted over  $n$ -vectors of subsets of parameters, where  $n$  is the number of equations. With the partial order  $\sqsubseteq$  defined component wise as

$$(Y_1, Y_2, \dots, Y_n) \sqsubseteq (Y'_1, Y'_2, \dots, Y'_n) \text{ if } \forall 1 \leq i \leq n \ Y_i \subseteq Y'_i,$$

we have the complete lattice  $((2^P)^n, \sqsubseteq)$ .

From an equation system build with the rules above we can define a semantic function  $f : (2^P)^n \rightarrow (2^P)^n$ , where the input is candidate sets of truth assignments for each equation, and the result is sets of truth assignments given that these candidate sets were right.

One can easily show that  $f$  is monotonic on the complete lattice, thus we know from Tarski's fixed-point theroem, that  $f$  has a unique maximal fixed-point.

We can now prove that from the maximal fix-point of  $f$ , we get the set of truth assignments that will result in a BMTS that satisfies the formula.

**Theorem 7.** *Let  $s$  be a state of a PMTS and  $F$  a formula of our logic. Assume for  $k \in N$  that*

$$f^k(2^P, 2^P, \dots, 2^P) = (Y_1, Y_2, \dots, Y_n)$$

*is the maximal fix-point for the semantic function  $f$ , then it holds*

$$s(\nu) \models F \text{ iff } \nu \in Y_i,$$

*where  $i$  is the index of the equation  $s \models_P F$  in the equation system.*

*Proof.* This theorem is proved by inspecting each of the rules for building the equation system.

Actually we only need to prove that  $\text{Tran}(s(\nu)) = \text{Tran}_\nu(s)$ . This follows since it by this fact is trivial to prove that the rules for building the equation system, are directly derived from the semantics of the logic on BMTS.

Note that  $\text{Tran}(s(\nu))$  is the set of all admissible sets of outgoing transitions from state  $s$  in the BMTS given by a PMTS with parameters assigned to  $\nu$ . Furthermore  $\text{Tran}_\nu(s)$  is the set of all admissible sets of outgoing transitions from state  $s$  in a PMTS given that the parameters are evaluated as  $\nu$ . ■

## 5 Conclusion and Future Work

We have introduced a new logic for Boolean modal transition systems (BMTS), based on Hennessy-Milner Logic with recursion. The logic characterizes the modal refinement relation on BMTS, which we proved by giving the characteristic formula for a BMTS and showing that the satisfaction relation coincided with modal refinement.

Secondly we applied the logic on parametric MTS (PMTS) as a tool to select the right truth assignment for the parameters of a PMTS, in order to fulfill a requirement. We gave a new semantic to the logic on PMTS, and showed that we can solve this problem via a maximal fix-point of a mutually recursive equation system.

By providing these results, we think that the OTS, BMTS and PMTS are all justified formalisms of the MTS specification family.

In the future we would like to investigate whether there exists an efficient algorithm for selecting the parameters of a PMTS given a requirement. One could try using BDDs or similar data structures to represent the constraints on the parameters.

**Acknowledgments** We would like to thank Jiří Srba for our initial discussions and his helpful comments. We would also like to thank Line Juhl for proofreading the paper.



# Dual-Priced Modal Transition Systems with Time Durations

Nikola BENEŠ

*Masaryk University, Department of Computer Science, Czech Republic*

Jan KŘETÍNSKÝ

*Institut für Informatik, Technische Universität München, Germany*

Kim G. LARSEN      Mikael H. MØLLER      Jiří SRBA

*Aalborg University, Department of Computer Science, Denmark*

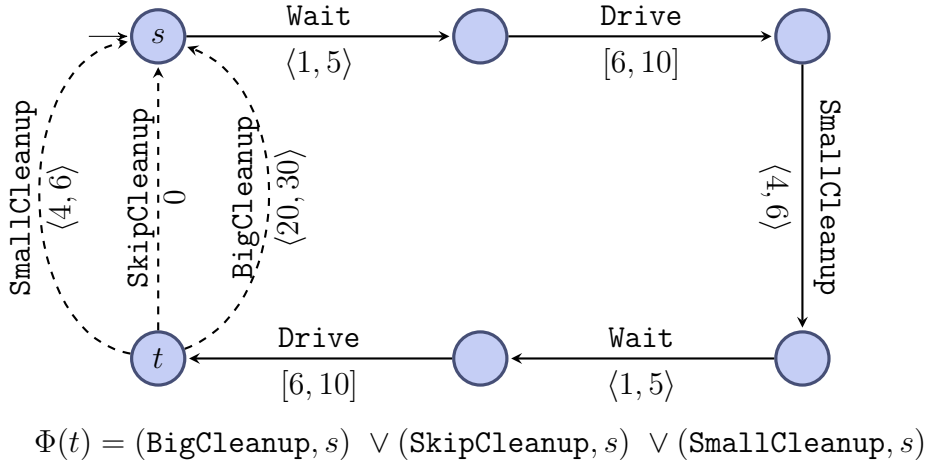
**Abstract** Modal transition systems are a well-established specification formalism for a high-level modelling of component-based software systems. We present a novel extension of the formalism called modal transition systems with durations where time durations are modelled as controllable or uncontrollable intervals. We further equip the model with two kinds of quantitative aspects: each action has its own running cost per time unit, and actions may require several hardware components of different costs. We ask the question, given a fixed budget for the hardware components, what is the implementation with the cheapest long-run average reward. We give an algorithm for computing such optimal implementations via a reduction to a new extension of mean payoff games with time durations and analyse the complexity of the algorithm.

## 1 Introduction and Motivating Example

Modal Transition Systems (MTS) is a specification formalism [LT88; Ant+08] that aims at providing a flexible and easy-to-use compositional development methodology for reactive systems. The formalism can be viewed as a fragment of a temporal logic [Ace+11; BL92] that at the same time offers a behavioural compositional semantics with an intuitive notion of process refinement. The formalism of MTS is essentially a labelled transition system that distinguishes two types of labelled transitions: *must* transitions which are required in any refinement of the system, and *may* transitions that are allowed to appear in a refined system but are not required. The refinement of an MTS now essentially consists of iteratively resolving the presence or absence of may transitions in the refined process.

In a recent line of work [JLS12; Bau+11], the MTS framework has been extended to allow for the specification of additional constraints on quantitative aspects (e.g. time, power or memory), which are highly relevant in the area of embedded systems. In this paper we continue the pursuit of quantitative extensions of MTS by presenting a novel extension of MTS with time durations being modelled as controllable or uncontrollable intervals. We further equip the model with two kinds of quantitative aspects: each action has its own running cost per time unit, and actions may require several hardware components of different costs. Thus, we ask the question, given a fixed budget for the investment into the hardware components, what is the implementation with the cheapest long-run average reward.

Before we give a formal definition of modal transition systems with durations (MTSD) and the dual-price scheme, and provide algorithms for computing optimal implementations, we present a small motivating example.

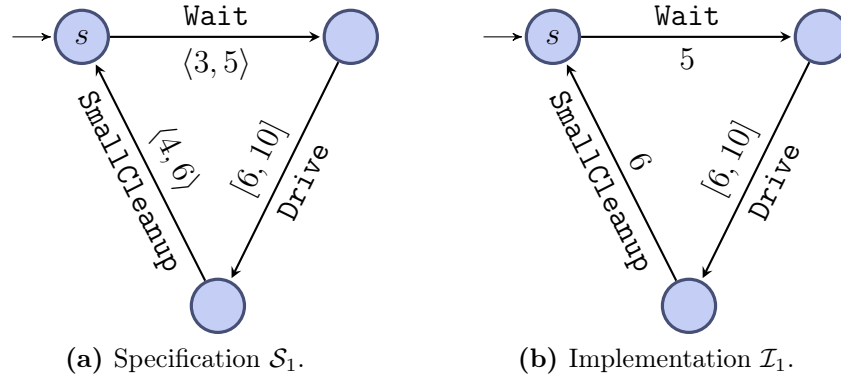


**Figure 1:** Example of Dual-Priced Modal Transition Systems with Time Durations, specification  $\mathcal{S}$ .

Consider the specification  $\mathcal{S}$  in Figure 1 describing the work of a shuttle bus driver. He drives a bus between a hotel and the airport. First, the driver has to *Wait* for the passengers at the hotel. This can take one to five minutes. Since this behaviour is

required to be present in all the implementations of this specification, it is drawn as a solid arrow and called a *must* transition. Then the driver has to **Drive** the bus to the airport (this takes six to ten minutes) where he has to do a **SmallCleanup**, then **Wait** before he can **Drive** the bus back to the hotel. When he returns he can do either a **SmallCleanup**, **BigCleanup** or **SkipCleanup** of the bus before he continues. Here we do not require a particular option to be realised in the implementations, hence we only draw the transitions as dashed arrows. As these transitions may or may not be present in the implementations, they are called *may* transitions. However, here the intention is to require at least one option to be realised. Hence, we specify this using a propositional formula  $\Phi$  assigned to the state  $t$  over its outgoing transitions as described in [BK11; Ben+11a]. After performing one of the actions, the driver starts over again. Note that next time the choice in  $t$  may differ.

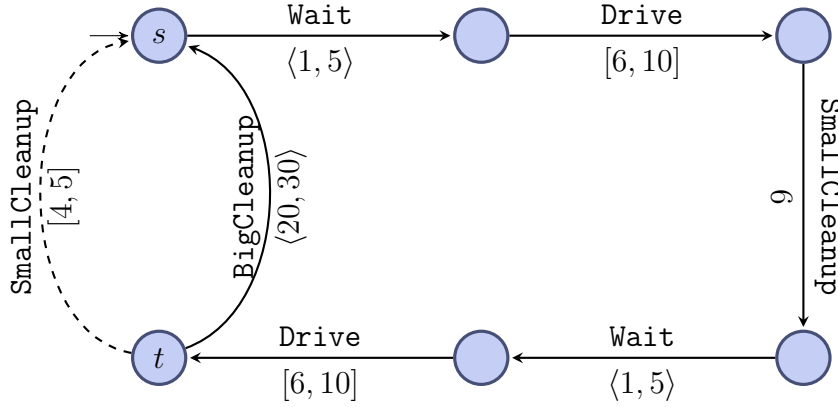
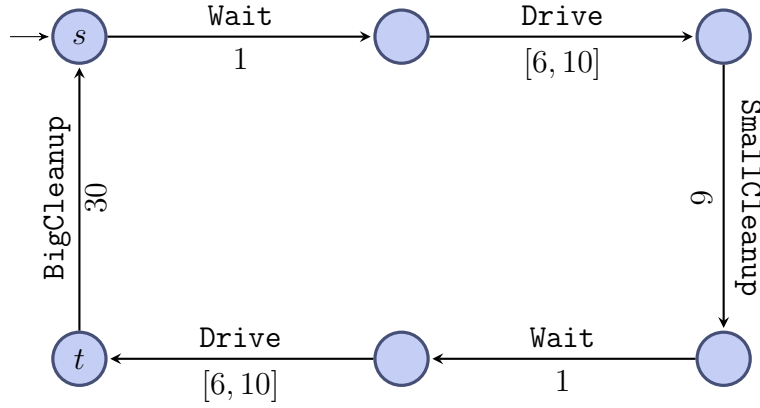
Observe that there are three types of durations on the transitions. First, there are *controllable* intervals, written in angle brackets. The meaning of e.g.  $\langle 1, 5 \rangle$  is that in the implementation we can instruct the driver to wait for a fixed number of minutes in the range. Second, there are *uncontrollable* intervals, written in square brackets. The interval  $[6, 10]$  on the **Drive** transition means that in the implementation we cannot fix any particular time and the time can vary, say, depending on the traffic and it is chosen nondeterministically by the environment. Third, the degenerated case of a single number, e.g. 0, denotes that the time taken is always constant and given by this number. In particular, a zero duration means that the transition happens instantaneously.



**Figure 2:** Examples of Dual-Priced Modal Transition Systems with Time Durations.

The system  $\mathcal{S}_1$  is another specification, a *refinement* of  $\mathcal{S}$ , where we additionally specify that the driver must do a **SmallCleanup** after each **Drive**. Note that the **Wait** interval has been narrowed. The system  $\mathcal{I}_1$  is an implementation of  $\mathcal{S}_1$  (and actually also of  $\mathcal{S}$ ) where all controllable time intervals have already been fully resolved to their final single values: the driver must **Wait** for 5 minutes and do the **SmallCleanup** for 6 minutes. Note that uncontrollable intervals remain unresolved in the implementations and the time is chosen by the environment each time the action is performed. This reflects the inherent uncontrollable uncertainty of the timing, e.g. of a traffic.

The system  $\mathcal{S}_2$  is yet another specification and again a refinement of  $\mathcal{S}$ , where the


 Figure 3: Specification  $\mathcal{S}_2$ .

 Figure 4: Implementation  $\mathcal{I}_2$ .

driver can always do a **BigCleanup** in  $t$  and possibly there is also an alternative allowed here of a **SmallCleanup**. Notice that both **SmallCleanup** intervals have been restricted and changed to uncontrollable. This means that we give up the control over the duration of this action and if this transition is implemented, its duration will be every time chosen nondeterministically in that range. Finally,  $\mathcal{I}_2$  is then an implementation of  $\mathcal{S}_2$  and  $\mathcal{S}$ .

Furthermore, we develop a way to model cost of resources. Each action is assigned a *running price* it costs per time unit, e.g. **Drive** costs 10 each time unit it is being performed as it can be seen in the left table of Figure 5. In addition, in order to perform an action, some hardware may be needed, e.g. a **VacuumCleaner** for the **BigCleanup** and its price is 100 as can be seen on the right. This *investment price* is paid once only.

Let us now consider the problem of finding an optimal implementation, so that we spend the least possible amount of money (e.g. the pay to the driver) per time unit while conforming to the specification  $\mathcal{S}$ . We call this problem *the cheapest implementation problem*. The optimal implementation is to buy a vacuum cleaner if one can afford an investment of 100 and do the **BigCleanup** every time as long as possible and **Wait** as shortly as possible. (Note that **BigCleanup** is more costly per time unit than **SmallCleanup** but lasts longer.) This is precisely implemented in  $\mathcal{I}_2$  and the (worst-



$$H = \{\text{VacuumCleaner}, \text{Sponge}\}$$

$a \in \Sigma$	$r(a)$	$h \in H$	$i(h)$
Wait	8	VacuumCleaner	100
Drive	10	Sponge	5
SmallCleanup	6		
BigCleanup	7		
SkipCleanup	0		

$$\Psi(a) = \begin{cases} \text{VacuumCleaner} & \text{if } a = \text{BigCleanup} \\ \text{Sponge} \vee \text{VacuumCleaner} & \text{if } a = \text{SmallCleanup} \\ \text{true} & \text{otherwise} \end{cases}$$

Figure 5: Price Scheme.

case) average cost per time unit is  $\approx 7.97$ . If one cannot afford the vacuum cleaner but only a sponge, the optimal worst case long run average is then a bit worse and is implemented by doing the **SmallCleanup** as long as possible and **Wait** now as *long* as possible. This is depicted in  $\mathcal{I}_1$  and the respective average cost per time unit is  $\approx 8.10$ .

The most related work is [Cha+03] where prices are introduced into a class of interface theories and long-run average objectives are discussed. Our work omits the issue of distinguishing input and output actions. Nevertheless, compared to [Cha+03], this paper deals with the time durations, the one-shot hardware investment and, most importantly, refinement of specifications. Further, timed automata have also been extended with prices [BLR04] and the long-run average reward has been computed in [BBL08]. However, priced timed automata lack the hardware and any notion of refinement, too.

The paper is organized as follows. We introduce the MTS with the time durations in Section 2 and the dual-price scheme in Section 3. Section 4 presents the main results on the complexity of the cheapest implementation problem. First, we state the complexity of this problem in general and in an important special case and prove the hardness part. The algorithms proving the complexity upper bounds are presented only after introducing an extension of mean payoff games with time durations. These are needed to establish the results but are also interesting on their own as discussed in Section 4.1. We conclude and give some more account on related and future work in Section 5.

## 2 Modal Transition Systems with Durations

In order to define MTS with durations, we first introduce the notion of *controllable* and *uncontrollable* duration intervals. A controllable interval is a pair  $\langle m, n \rangle$  where  $m, n \in \mathbb{N}_0$  and  $m \leq n$ . Similarly, an uncontrollable interval is a pair  $[m, n]$  where  $m, n \in \mathbb{N}_0$  and  $m \leq n$ . We denote the set of all controllable intervals by  $\mathfrak{I}_c$ , the set of all uncontrollable intervals by  $\mathfrak{I}_u$ , and the set of all intervals by  $\mathfrak{I} = \mathfrak{I}_c \cup \mathfrak{I}_u$ . We also write only  $m$  to

denote the singleton interval  $[m, m]$ . Singleton controllable intervals need not be handled separately as there is no semantic difference to the uncontrollable counterpart.

We can now formally define modal transition systems with durations. In what follows,  $\mathcal{B}(X)$  denotes the set of propositional logic formulae over the set  $X$  of atomic propositions, where we assume the standard connectives  $\wedge, \vee, \neg$ .

**Definition 1** (MTSD). *A Modal Transition System with Durations (MTSD) is a tuple  $\mathcal{S} = (S, T, D, \Phi, s_0)$  such that*

- *$S$  is a set of states with the initial state  $s_0$ ,*
- *$T \subseteq S \times \Sigma \times S$  is a set of transitions,*
- *$D : T \rightarrow \mathcal{I}$  is a duration interval function, and*
- *$\Phi : S \rightarrow \mathcal{B}(\Sigma \times S)$  is an obligation function.*

*We assume that whenever the atomic proposition  $(a, t)$  occurs in the Boolean formula  $\Phi(s)$  then also  $(s, a, t) \in T$ .*

*Moreover, we require that there is no cycle of transitions that allows for zero accumulated duration, i.e. there is no path  $s_1 a_1 s_2 a_2 \cdots s_n$  where  $(s_i, a_i, s_{i+1}) \in T$  and  $s_n = s_1$  such that for all  $i$ , the interval  $D((s_i, a_i, s_{i+1}))$  is of the form either  $\langle 0, m \rangle$  or  $[0, m]$  for some  $m$ .*

Note that instead of the basic may and must modalities known from the classical modal transition systems (see e.g. [Ant+08]), we use arbitrary boolean formulae over the outgoing transitions of each state in the system as introduced in [Ben+11a]. This provides a higher generality as the formalism is capable to describe, apart from standard modal transition systems, also more expressive formalisms like disjunctive modal transition systems [LX90] and transition systems with obligations [BK11]. See [Ben+11a] for a more thorough discussion of this formalism.

In the rest of the paper, we adapt the following convention when drawing MTSDs. Whenever a state  $s$  is connected with a solid arrow labelled by  $a$  to a state  $s'$ , this means that in any satisfying assignment of the Boolean formula  $\Phi(s)$ , the atomic proposition  $(a, s')$  is always set to true (the transition *must* be present in any refinement of the system). Should this not be the case, we use a dashed arrow instead (meaning that the corresponding transition *may* be present in a refinement of the system but it can be also left out). For example the solid edges in Figure 1 correspond to an implicitly assumed  $\Phi(s) = (a, s')$  where  $(s, a, s')$  is the (only) outgoing edge from  $s$ ; in this case we do not explicitly write the obligation function. The three dashed transitions in the figure are optional, though at least one of them has to be preserved during any refinement (a feature that can be modelled for example in disjunctive MTS [LX90]).

*Remark 1.* The standard notion of modal transition systems (see e.g. [Ant+08]) is obtained under the restriction that the formulae  $\Phi(s)$  in any state  $s \in S$  have the form

$(a_1, s_1) \wedge \dots \wedge (a_n, s_n)$  where  $(s, a_1, s_1), \dots, (s, a_n, s_n) \in T$ . The edges mentioned in such formulae are exactly all must transitions; may transitions are not listed in the formula and hence can be arbitrarily set to true or false.

Let by  $T(s) = \{(a, t) \mid (s, a, t) \in T\}$  denote the set of all outgoing transitions from the state  $s \in S$ . A modal transition system with durations is called an *implementation* if  $\Phi(s) = \bigwedge T(s)$  for all  $s \in S$  (every allowed transition is also required), and  $D(s, a, s') \in \mathcal{I}_u$  for all  $(s, a, s') \in T$ , i.e. all intervals are uncontrollable, often singletons. Figure 2b shows an example of an implementation, while Figure 2a is not yet an implementation as it still contains the controllable intervals  $\langle 3, 5 \rangle$  and  $\langle 4, 6 \rangle$ .

We now define a notion of modal refinement. In order to do that, we first need to define refinement of intervals as a binary relation  $\leq \subseteq \mathcal{I} \times \mathcal{I}$  such that

- $\langle m', n' \rangle \leq \langle m, n \rangle$  whenever  $m' \geq m$  and  $n' \leq n$ , and
- $[m', n'] \leq \langle m, n \rangle$  whenever  $m' \geq m$  and  $n' \leq n$ .

Thus controllable intervals can be refined by narrowing them, at most until they become singleton intervals, or until they are changed to uncontrollable intervals. Let us denote the collection of all possible sets of outgoing transitions from a state  $s$  by  $\text{Tran}(s) := \{E \subseteq T(s) \mid E \models \Phi(s)\}$  where  $\models$  is the classical satisfaction relation on propositional formulae assuming that  $E$  lists all true propositions.

**Definition 2** (Modal Refinement). *Let  $\mathcal{S}_1 = (S_1, T_1, D_1, \Phi_1, s_1)$  and  $\mathcal{S}_2 = (S_2, T_2, D_2, \Phi_2, s_2)$  be two MTSDs. A binary relation  $R \subseteq S_1 \times S_2$  is a modal refinement if for every  $(s, t) \in R$  the following holds:*

$$\forall M \in \text{Tran}(s) : \exists N \in \text{Tran}(t) :$$

$$\forall (a, s') \in M : \exists (a, t') \in N : D_1(s, a, s') \leq D_2(t, a, t') \wedge (s', t') \in R \text{ and}$$

$$\forall (a, t') \in N : \exists (a, s') \in M : D_1(s, a, s') \leq D_2(t, a, t') \wedge (s', t') \in R .$$

*We say that  $s \in S_1$  modally refines  $s' \in S_2$ , denoted by  $s \leq_m s'$ , if there exists a modal refinement  $R$  such that  $(s, s') \in R$ . We also write  $\mathcal{S}_1 \leq_m \mathcal{S}_2$  if  $s_1 \leq_m s_2$ .*

Intuitively, the pair  $(s, t)$  can be in the relation  $R$  if for any satisfiable instantiation of outgoing edges from  $s$  there is a satisfiable instantiation of outgoing edges from  $t$  so that they can be mutually matched, possibly with  $s$  having more refined intervals, and the resulting states are again in the relation  $R$ .

Observe that in our running example the following systems are in modal refinement:  $\mathcal{I}_1 \leq_m \mathcal{S}_1 \leq_m \mathcal{S}$  and thus also  $\mathcal{I}_1 \leq_m \mathcal{S}$ , and similarly  $\mathcal{I}_2 \leq_m \mathcal{S}_2 \leq_m \mathcal{S}$  and thus also  $\mathcal{I}_2 \leq_m \mathcal{S}$ .

The reader can verify that on the standard modal transition systems (see Remark 1) the modal refinement relation corresponds to the classical modal refinement as introduced in [LT88].

### 3 Dual-Price Scheme

In this section, we formally introduce a dual-price scheme on top of MTSD in order to model the *investment cost* (cost of hardware necessary to perform the implemented actions) and the *running cost* (weighted long-run average of running costs of actions). We therefore consider only deadlock-free implementations (every state has at least one outgoing transition) so that the long-run average reward is well defined.

**Definition 3** (Dual-Price Scheme). *A dual-price scheme over an alphabet  $\Sigma$  is a tuple  $\mathcal{P} = (r, H, \Psi, i)$  where*

- $r : \Sigma \rightarrow \mathbb{Z}$  is a running cost function of actions per time unit,
- $H$  is a finite set of available hardware,
- $\Psi : \Sigma \rightarrow \mathcal{B}(H)$  is a hardware requirement function, and
- $i : H \rightarrow \mathbb{N}_0$  is a hardware investment cost function.

Hence every action is assigned its unit cost and every action can have different hardware requirements (specified as a Boolean combination of hardware components) on which it can be executed. This allows for much more variability than a possible alternative of a simple investment cost  $\Sigma \rightarrow \mathbb{N}_0$ . Further, observe that the running cost may be negative, meaning that execution of such an action actually gains rather than spends resources.

Let  $\mathcal{I}$  be an implementation with an initial state  $s_0$ . A set  $G \subseteq H$  of hardware is *sufficient* for an implementation  $\mathcal{I}$ , written  $G \models \mathcal{I}$ , if  $G \models \Psi(a)$  for every action  $a$  reachable from  $s_0$ . The *investment cost* of  $\mathcal{I}$  is then defined as

$$\text{ic}(\mathcal{I}) = \min_{G \models \mathcal{I}} \sum_{g \in G} i(g) .$$

Further, a *run* of  $\mathcal{I}$  is an infinite sequence  $s_0 a_0 t_0 s_1 a_1 t_1 \dots$  with  $(s_i, a_i, s_{i+1}) \in T$  and  $t_i \in D(s_i, a_i, s_{i+1})$ . Hence, in such a run, a concrete time duration in each uncontrollable interval is selected. We denote the set of all runs of  $\mathcal{I}$  by  $\mathcal{R}(\mathcal{I})$ . The *running cost* of an implementation  $\mathcal{I}$  is the worst-case long-run average

$$\text{rc}(\mathcal{I}) = \sup_{s_0 a_0 t_0 s_1 a_1 t_1 \dots \in \mathcal{R}(\mathcal{I})} \limsup_{n \rightarrow \infty} \frac{\sum_{i=0}^n r(a_i) \cdot t_i}{\sum_{i=0}^n t_i} .$$

Our *cheapest-implementation problem* is now defined as follows: given an MTSD specification  $\mathcal{S}$  together with a dual-price scheme over the same alphabet, and given an upper-bound  $\text{max}_{\text{ic}}$  for the investment cost, find an implementation  $\mathcal{I}$  of  $\mathcal{S}$  (i.e.  $\mathcal{I} \leq_{\text{m}} \mathcal{S}$ ) such that  $\text{ic}(\mathcal{I}) \leq \text{max}_{\text{ic}}$  and for every implementation  $\mathcal{I}'$  of  $\mathcal{S}$  with  $\text{ic}(\mathcal{I}') \leq \text{max}_{\text{ic}}$ , we have  $\text{rc}(\mathcal{I}) \leq \text{rc}(\mathcal{I}')$ .

Further, we introduce the respective decision problem, the *implementation problem*, as follows: given an MTSD specification  $\mathcal{S}$  together with a dual-price scheme, and given an upper-bound  $max_{ic}$  for the investment cost and an upper bound  $max_{rc}$  on the running cost, decide whether there is an implementation  $\mathcal{I}$  of  $\mathcal{S}$  such that both  $ic(\mathcal{I}) \leq max_{ic}$  and  $rc(\mathcal{I}) \leq max_{rc}$ .

*Example 6.* Figure 5 depicts a dual-price scheme over the same alphabet  $\Sigma = \{\text{Wait}, \text{Drive}, \text{SmallCleanup}, \text{BigCleanup}, \text{SkipCleanup}\}$  as of our motivating specification  $\mathcal{S}$ . The running cost of the implementation  $\mathcal{I}_2$  is  $(1 \cdot 8 + 10 \cdot 10 + 6 \cdot 6 + 1 \cdot 8 + 10 \cdot 10 + 30 \cdot 7) / (1 + 10 + 6 + 1 + 10 + 30) \approx 7.97$  as the maximum value is achieved when **Drive** (with running cost 10) takes 10 minutes. On the one hand, this is optimal for  $\mathcal{S}$  and a maximum investment cost at least 100. On the other hand, if the maximum investment cost is 99 or less then the optimal implementation is depicted in  $\mathcal{I}_1$  and its cost is  $(5 \cdot 8 + 10 \cdot 10 + 6 \cdot 5) / (5 + 10 + 6) \approx 8.10$ .

*Remark 2.* Note that the definition of the dual-price scheme only relies on having durations on the labelled transition systems. Hence, one could easily apply this in various other settings like in the special case of traditional MTS (with may and must transitions instead of the obligation function) or in the more general case of parametric MTS (see [Ben+11a]) when equipped with durations as described above.

## 4 Complexity Results

In this section, we give an overview of the complexity of our problem both in general and in an important special case. We start with establishing the hardness results. The matching upper bounds and the outline of their proofs follow. When referring to the size of MTSDs and the dual-price scheme, we implicitly assume binary encoding of numbers. We start by observing that the implementation problem is NP-hard even if no hardware is involved.

**Proposition 4.** *The implementation problem is NP-hard even for the hardware requirement function  $\Psi$  that is constantly true for all actions.*

*Proof.* We shall reduce the satisfiability problem of Boolean formulae (SAT) to our problem. Let  $\varphi$  be a Boolean formula over the variables  $x_1, \dots, x_n$ . We define a MTSD  $\mathcal{S}$  over the set of actions  $\Sigma = \{x_1, \dots, x_n, *\}$  such that the running cost is  $r(x_j) = 1$  for all  $1 \leq j \leq n$  and  $r(*) = 2$  and the duration of all actions is 1. The specification  $\mathcal{S}$  has one state  $s$  and a self-loop under all elements of  $\Sigma$  with the obligation function  $\Phi(s) = \varphi \vee (*, s)$ . The reason for adding the action  $*$  is to make sure that in case  $\varphi$  is not satisfiable then we can still have a deadlock-free, but more running-cost-expensive implementation. Now we set the hardware to  $H = \emptyset$  and the hardware requirement function  $\Psi(a)$  constantly true for all  $a \in \Sigma$ . It is easy to observe that the formula  $\varphi$  is satisfiable iff  $\mathcal{S}$  has an implementation  $\mathcal{I}$  with  $rc(\mathcal{I}) \leq 1$  (and  $ic(\mathcal{I}) = 0$ ). ■

Note that in the proof we required  $\Phi$  to be a general Boolean formula. If, for instance, we considered  $\Phi$  in *positive form* (i.e. only containing  $\wedge$  and  $\vee$  operators and not  $\neg$ ), the hardness would not hold. Thus on the one hand, one source of hardness is the complexity of  $\Phi$ . On the other hand, even if  $\Phi$  corresponds to the simplest case of an implementation ( $\Phi$  is a conjunction of atomic propositions), the problem remains hard due to the hardware.

**Proposition 5.** *The implementation problem is NP-hard even for specifications that are already implementations.*

*Proof.* We reduce the NP-complete problem of vertex cover to our problem. Let  $(V, E)$  where  $E \subseteq V \times V$  be a graph and  $k \in \mathbb{N}$  be an integer. We ask whether there is a subset of vertices  $V_k \subseteq V$  of cardinality  $k$  such that for every  $(v_1, v_2) \in E$  at least  $v_1 \in V_k$  or  $v_2 \in V_k$ . Let us construct an MTSD specification  $\mathcal{S}$  with hardware  $H = V$  and the investment function  $i(v) = 1$  for all  $v \in H$ , such that  $\mathcal{S}$  has only one state  $s$  and a self-loop under a single action  $a$  that is required ( $\Phi(s) = (a, s)$ ) and where the hardware requirement function is  $\Psi(a) = \bigwedge_{(u,v) \in E} (u \vee v)$ . There is now a vertex cover in  $(V, E)$  of size  $k$  iff  $\mathcal{S}$  has an implementation  $\mathcal{I}$  with  $\text{ic}(\mathcal{I}) \leq k$ . Setting e.g.  $D(s, a, s) = 1$  and the running cost  $r(a) = 0$  establishes NP-hardness of the implementation problem where we ask for the existence of an implementation of  $\mathcal{S}$  with maximum running cost 0 and maximum investment cost  $k$ .

Alternatively, we may introduce a self-loop with a new action name  $a_{(u,v)}$  for every edge  $(u, v)$  in the graph such that  $\Psi(a_{(u,v)}) = u \vee v$ , showing NP-hardness even for the case where the hardware requirement function is a simple disjunction of hardware components. ■

In the subsequent sections, we obtain the following matching upper bound which yields the following theorem.

**Theorem 6.** *The implementation problem is NP-complete.*

By analysing the proof of Proposition 5, it is clear that we have to restrict the hardware requirement function before we can obtain a more efficient algorithm for the implementation problem. We do so by assuming a constant number of hardware components (not part of the input). If we at the same time require the obligation function in positive form, we obtain a simpler problem as stated in the following theorem.

**Theorem 7.** *The implementation problem with positive obligation function and a constant number of hardware components is polynomially equivalent to mean payoff games and thus it is in  $NP \cap coNP$  and solvable in pseudo-polynomial time.*

The subsequent sections are devoted to proving Theorems 6 and 7. The algorithm to solve the implementation problem first reduces the dual-priced MTSD into a mean

payoff game extended with time durations and then solves this game. This new extension of mean payoff games and an algorithm to solve them is presented in Section 4.1. The translation follows in Section 4.2. Since this translation is exponential in general, Section 4.3 then shows how to translate in polynomial time with only local exponential blow-ups where negations occur. Section 4.4 then concludes and establishes the complexity bounds.

## 4.1 Weighted Mean Payoff Games

We extend the standard model of mean payoff games (MPG) [EM79] with time durations. Not only is this extension needed for our algorithm, but it is also useful for modelling by itself. Consider, for instance, energy consumption of 2kW for 10 hours and 10kW for 2 hour, both followed by 10 hours of inactivity. Obviously, although both consumptions are 20kWh per cycle, the average consumption differs: 1kW in the former case and 20/12kW in the latter one. We also allow zero durations in order to model e.g. discrete changes of states, an essential part of our algorithm. Another extension of MPGs with dual-cost was studied in [Blo+09].

**Definition 8.** A weighted mean payoff game is  $G = (V, V_{min}, V_{max}, E, r, d)$  where  $V$  is a set of vertices partitioned into  $V_{min}$  and  $V_{max}$ ,  $E \subseteq V \times V$  is a set of edges,  $r : E \rightarrow \mathbb{Z}$  is a rate function,  $d : E \rightarrow \mathbb{N}_0$  is a duration function.

It is assumed that there are no deadlocks (vertices with out-degree 0) and that there are no zero-duration cycles. The game is played by two players,  $min$  and  $max$ . The play is an infinite path such that each player picks successors in his/her vertices. The value of a play  $v_0v_1v_2 \dots$  is defined as:

$$\nu(v_0v_1v_2 \dots) = \limsup_{n \rightarrow \infty} \frac{\sum_{i=0}^n r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^n d(v_i, v_{i+1})}. \quad (*)$$

Player  $min$  tries to minimize this value, while  $max$  aims at the opposite. Let  $v(s)$  denote the infimum of the values  $min$  can guarantee if the play begins in the vertex  $s$ , no matter what the player  $max$  does.

Note that the standard MPGs where edges are assigned only integer weights can be seen as weighted MPGs with rates equal to weights and durations equal to 1 on all edges.

We now show how to solve weighted MPGs by reduction to standard MPGs. We first focus on the problem whether  $v(s) \geq 0$  for a given vertex  $s$ . As the durations are nonnegative and there are no zero-duration cycles, the denominator of the fraction in (\*) will be positive starting from some  $n$ . Therefore, the following holds for every play  $v_0v_1v_2 \dots$  and every (large enough)  $n$ :

$$\frac{\sum_{i=0}^n r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^n d(v_i, v_{i+1})} \geq 0 \Leftrightarrow \frac{1}{n} \sum_{i=0}^n r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1}) \geq 0.$$



We may thus solve the question whether  $v(s) \geq 0$  by transforming the weighted MPG into a standard MPG, leaving the set of vertices and edges the same and taking  $w(u, v) = r(u, v) \cdot d(u, v)$  as the edge weight function. Although the value  $v(s)$  may change in this reduction, its (non)negativeness does not.

Further, we may transform any problem of the form  $v(s) \geq \lambda$  for any fixed constant  $\lambda$  into the above problem. Let us modify the weighted MPG as follows. Let  $r'(u, v) = r(u, v) - \lambda$  and leave everything else the same. The value of a play  $v_0 v_1 v_2 \dots$  is thus changed as follows.

$$\nu'(v_0 v_1 v_2 \dots) = \limsup_{n \rightarrow \infty} \frac{\sum_{i=0}^n (r(v_i, v_{i+1}) - \lambda) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^n d(v_i, v_{i+1})} = \nu(v_0 v_1 v_2 \dots) - \lambda$$

It is now clear that  $v(s) \geq \lambda$  in the original game if and only if  $v'(s) \geq 0$  in the modified game.

Furthermore, there is a one-to-one correspondence between the strategies in the original weighted MPG and the constructed MPG. Due to the two equivalences above, this correspondence preserves optimality. Therefore, there are optimal positional strategies in weighted MPGs since the same holds for standard MPGs [EM79]. (A strategy is positional if its decision does not depend on the current history of the play but only on the current vertex, i.e. can be described as a function  $V \rightarrow V$ .)

## 4.2 Translating Dual-priced MTSD into Weighted MPG

We first focus on the implementation problem without considering the hardware ( $H = \emptyset$ ). We show how the implementation problem can be solved by reduction to the weighted MPGs. The first translation we present is exponential, however, we provide methods for making it smaller in the subsequent section.

We are given an MTSD  $\mathcal{S} = (S, T, D, \Phi, s_0)$  and a dual-price scheme  $(r, H, \Psi, i)$  and assume that there is no state  $s$  with  $\emptyset \in \text{Tran}(s)$ . Let us define the following auxiliary vertices that will be used to simulate the more complicated transitions of MTSD in the simpler setting of weighted MPG (by convention all singleton intervals are treated as uncontrollable).

$$\begin{aligned} T_u &= \{(s, a, t) \mid (s, a, t) \in T; D(s, a, t) \in \mathcal{I}_u\} \\ T_c &= \{(s, a, t) \mid (s, a, t) \in T; D(s, a, t) \in \mathcal{I}_c\} \\ T_* &= \{(s, a, j, t) \mid (s, a, t) \in T; j \in D(s, a, t)\} \end{aligned}$$

We construct the weighted mean-payoff game with  $V_{\min} = S \cup T_c \cup T_*$ ,  $V_{\max} = 2^T \cup T_u$  and  $E$  defined as follows:

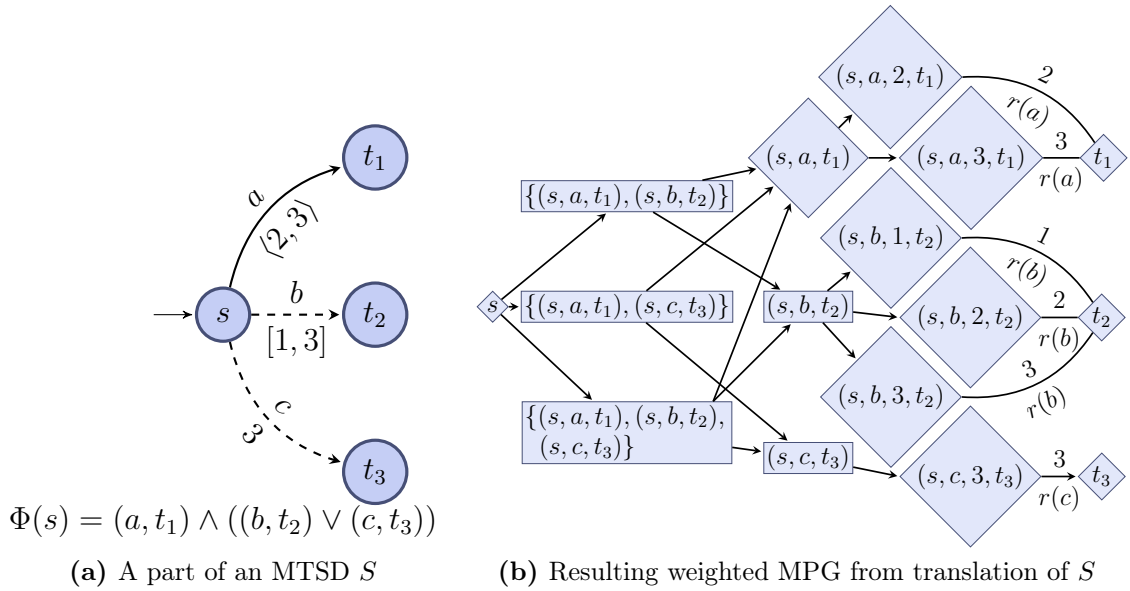
$$\begin{aligned} (s, X) &\in E && \text{iff } \exists V \in \text{Tran}(s) : X = \{(s, a, t) \mid (a, t) \in V\} \\ (X, (s, a, t)) &\in E && \text{iff } (s, a, t) \in X \\ ((s, a, t), (s, a, j, t)) &\in E && \text{iff } j \in D(s, a, t) \\ ((s, a, j, t), t) &\in E && \text{(always)} \end{aligned}$$

Further,  $r((s, a, j, t), t) = r(a)$ ,  $d((s, a, j, t), t) = j$  and  $r(-, -) = d(-, -) = 0$  otherwise.



*Example 7.* In Figure 6 we show an example of how this translation to weighted MPG works. For simplicity we only translate a part of the MTSD  $\mathcal{S}$  shown in Figure 6a. The resulting weighted MPG is shown in Figure 6b. The diamond shaped states belong to *min* and the squared states belong to *max*. In the vertex  $s$ , *min* chooses which outgoing transition are implemented. Only the choices satisfying  $\Phi(s)$  are present in the game. Afterwards, *max* decides which transition to take. The chosen transition is then assigned by one of the players a time that it is going to take.

Notice that  $(s, a, t_1)$  is the only transition controlled by *min*, because it has a controllable interval  $\langle 2, 3 \rangle$ . The remaining transitions with uncontrollable intervals are operated by *max* who chooses the time from these intervals. All the “auxiliary” transitions are displayed without any labels meaning their duration (and rate) is zero. Thus, only the transitions corresponding to “real” transitions in MTSDs are taken into account in the value of every play.



**Figure 6:** Translating MTSD to weighted MPG.

We now show how a strategy for player *min* in the constructed weighted MPG may be translated into an implementation of the original MTSD. In Section 4.1, we have shown that there are optimal *positional* strategies. Hence, we may safely restrict this translation to positional strategies. Let  $\sigma$  be such a positional strategy. We build the implementation as  $\mathcal{I} = (S', T', D', \Phi', s_0)$  where

- $S' = \{s_1 \mid s \in S\}$
- $(s_1, a, t_1) \in T'$  if  $(s, a, t) \in X$  where  $\sigma(s) = X$
- $D'(s_1, a, t_1) = D(s, a, t)$  if  $D(s, a, t) \in \mathcal{I}_u$
- $D'(s_1, a, t_1) = j$  if  $D(s, a, t) \in \mathcal{I}_c$  and  $\sigma((s, a, t)) = (s, a, j, t)$

### C. Dual-Priced Modal Transition Systems with Time Durations

- $\Phi'(s_1) = \bigwedge_{(s_1, a, t_1) \in T'} (a, t_1)$

The set of states remains the same as in the original MTSD; we change every state  $s$  into  $s_1$  to be able to reason about the states of the original MTSD and of the implementation separately. The following two lemmas state that the construction is sound.

**Lemma 9.** *The constructed implementation  $\mathcal{I}$  is an implementation of the original MTSD.*

*Proof.* We show that  $R = \{(s_1, s) \mid s \in S\}$  is a modal refinement relation. Let  $(s_1, s) \in R$  and let  $M \in \text{Tran}(s_1)$ . Clearly  $M = \{(a, t_1) \mid (s_1, a, t_1) \in T'\}$ . We take  $N = \{(a, t) \mid (a, t_1) \in M\}$ . The fact that  $N \in \text{Tran}(s)$  is clear from the construction. We now need to show that  $D'(s_1, a, t_1) \leq D(s, a, t)$ . If  $D(s, a, t) \in \mathcal{I}_u$  then  $D'(s_1, a, t_1) = D(s, a, t)$  and the statement holds. If  $D(s, a, t) \in \mathcal{I}_c$  then  $D'(s_1, a, t_1) = j$  where  $j \in D(s, a, t)$  due to the construction. Thus  $R$  is a modal refinement relation where  $s_1 \leq_m s$  for all  $s \in S$  and therefore  $\mathcal{I} \leq_m \mathcal{S}$ . ■

**Lemma 10.** *We have  $\text{rc}(\mathcal{I}) \leq \lambda$  if and only if the strategy  $\sigma$  ensures a value of at most  $\lambda$ .*

*Proof.* Every run of the implementation corresponds to a play (where player *min* plays according to strategy  $\sigma$ ) and vice versa. Hence, the worst case of the long run average over all runs in  $\mathcal{I}$  is the same as over all plays according to  $\sigma$ . ■

We conclude the proof of correctness of the reduction by showing its completeness.

**Lemma 11.** *For every implementation  $\mathcal{I}$  of  $\mathcal{S}$ , there exists a strategy  $\sigma$  for player *min* such that  $\sigma$  ensures value of at most  $\text{rc}(\mathcal{I})$ .*

*Proof.* We show how to transform an arbitrary implementation  $\mathcal{I}$  into a strategy  $\sigma : V^* \rightarrow V$  (depending on the whole prefix of a play where we currently are) for player *min* that guarantees the same or smaller value. (Although this is not a positional strategy, we know there is also a positional strategy ensuring the same or smaller value.) The idea of the construction is that for each history  $\sigma$  mimics the behaviour of the implementation at its respective state. In other words, the decision of  $\sigma$  in some history is based on mapping the history to the implementation and doing what the implementation does. However, there is a small catch: the implementation may implement more possible decisions (its branching may be even uncountable). Nonetheless, we may take any of the decisions, as the resulting strategy then corresponds to a pruning of the original implementation and the strategy's worst case long run average is thus either the same or even smaller.

Let  $i_0$  be the initial state of  $\mathcal{I}$  and  $s_0$  the initial state of  $\mathcal{S}$ ; we have  $i_0 \leq_m s_0$ . The corresponding vertex of the constructed weighted MPG also bears the name  $s_0$ . We first define a mapping  $\mu : V^* \rightarrow I^*$  from paths in the weighted MPG of the form

$s_0(s_0, N_0)(s_0, a_0, s_1)(s_0, a_0, j_0, s_1)s_1 \cdots s_n$  to sequences of states of  $\mathcal{I}$  inductively as follows.

$$\begin{aligned}\mu(s_0) &= i_0 \\ \mu(s_0 \cdots s_n(s_n, N_n)(s_n, a_n, s_{n+1})(s_n, a_n, j_n, s_{n+1})s_{n+1}) &= \mu(s_0 \cdots s_n)i_{n+1}\end{aligned}$$

where  $i_{n+1}$  is an arbitrary state satisfying  $(i_n, a_n, i_{n+1}) \in T$  and  $i_{n+1} \leq_m s_{n+1}$ . The image of  $\mu$  now defines the desired pruning of  $\mathcal{I}$  that is still an implementation of  $\mathcal{S}$ , has at most the same worst case long run average, and can now be canonically mapped to a deterministic strategy  $\sigma$  as follows.

Let  $\pi = s_0(s_0, N_0)(s_0, a_0, s_1)(s_0, a_0, j_0, s_1)s_1 \cdots x \in V^*$  be a prefix of a play with  $x \in V_{min}$ . Denote  $s_n$  the last element of  $\mathcal{S}$  in  $\pi$  and  $i_0 \cdots i_n = \mu(s_0 \cdots s_n)$ . We define  $\sigma(\pi)$  as follows.

- If  $x \in S$ , then  $\pi = s_0 \cdots s_n$  and due to  $i_n \leq_m s_n$  there exists  $N$  from the definition of modal refinement. We set  $\sigma_s(\pi) = (s_n, N)$ .
- If  $x \in T_c$ , then  $\pi = s_0 \cdots s_n(s_n, N_n)(s_n, a_n, s_{n+1})$  and there exists a unique  $i_{n+1}$  such that  $i_0 \cdots i_n i_{n+1} = \mu(s_0 \cdots s_n s_{n+1})$ . Here we use the fact that  $\mu$  does not depend on which vertices of the form  $(s, a, j, t)$  were visited. We set  $\sigma(\pi) = (s_n, a_n, D(i_n, a_n, i_{n+1}), s_{n+1})$ .
- If  $x \in T_*$ , then  $x = (s_n, a_n, j_n, s_{n+1})$  and there exists only one outgoing edge. We set  $\sigma(\pi) = s_{n+1}$ .

Now for each play that player *min* plays according to  $\sigma$ , there is a run in the original implementation that has the same long-run average. Hence the supremum over all plays is at most the supremum over all runs.  $\blacksquare$

### 4.3 Optimizations

We now simplify the construction. The first simplification is summarized by the observation that the strategies of both players only need to choose the extremal points of the interval in vertices of the form  $(s, a, t)$ .

**Lemma 12.** *There are optimal positional strategies  $\sigma', \rho'$  for min and max, respectively, such that the choice in vertices of the form  $(s, a, t)$  is always one of the two extremal points of the interval  $D(s, a, t)$ .*

*Proof.* Let  $\sigma$  and  $\rho$  be optimal positional strategies for *min* and *max*. We transform them into  $\sigma'$  and  $\pi'$ . The proof is done by induction on the number of vertices of the form  $(s, a, t)$  from which one of the strategies chooses a non-extremal point of the interval  $D(s, a, t)$ .

If there are no such vertices, we are obviously done. Suppose further that there is at least one such vertex, say  $(s, a, t)$ . Without loss of generalization, let this vertex be player

### C. Dual-Priced Modal Transition Systems with Time Durations

$\min$ 's vertex. (The other case is handled similarly.) We thus have  $D(s, a, t) = \langle m, n \rangle$  and  $\sigma((s, a, t)) = (s, a, j, t)$  with  $m < j < n$ .

We investigate three cases, depending on the relationship of the rate  $r(a)$  and the value  $v(s)$ .

- $r(a) = v(s)$ . Then the duration of the transition  $(s, a, t)$  does not matter and we may freely change  $\sigma$  into  $\sigma'$  by defining  $\sigma'((s, a, t)) = (s, a, n, t)$  and  $v(s)$  remains the same.
- $r(a) > v(s)$ . Clearly, changing  $\sigma$  into  $\sigma'$  by defining  $\sigma'((s, a, t)) = (s, a, m, t)$  may only decrease  $v(s)$  or not change it at all. (As  $\sigma$  is optimal strategy,  $v(s)$  remains the same using  $\sigma'$ .)
- $r(a) < v(s)$ . Using similar argument as in the previous case, we change  $\sigma$  into  $\sigma'$  by defining  $\sigma'(s, a, t) = (s, a, n, t)$  and  $\sigma'$  remains optimal.

Repeated use of this argument concludes the proof. ■

We may thus simplify the construction according to the previous lemma so that there are at most two outgoing edges for each state of the form  $(s, a, t)$  are as follows:  $((s, a, t), (s, a, j, t)) \in E$  iff  $j$  is an extremal point of  $D(s, a, t)$ .

We can also optimize the expansion of  $\text{Tran}(s)$ . So far, we have built an exponentially larger weighted MPG graph as the size of  $\text{Tran}(s)$  is exponential in the out-degree of  $s$ . However, we can do better if we restrict ourselves to the class of MTSD where all  $\Phi(s)$  are positive boolean formulae, i.e. the only connectives are  $\wedge$  and  $\vee$ . Instead of enumerating all valuations, we can use the syntactic tree of the formula to build a weighted MPG of polynomial size.

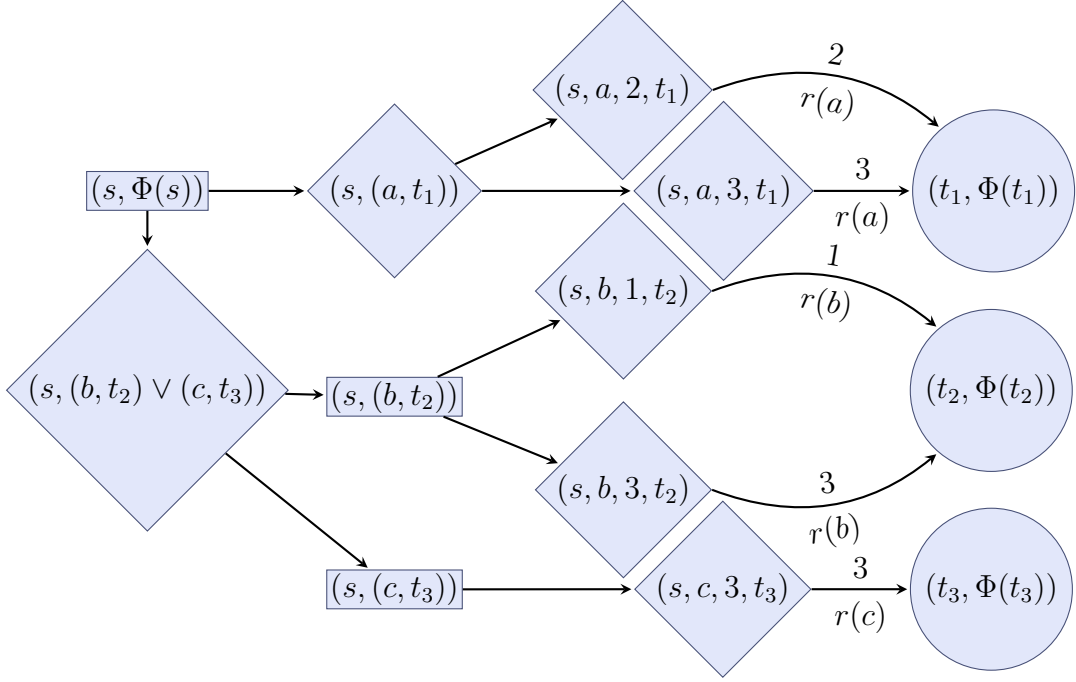
Let  $sf(\varphi)$  denote the set of all sub-formulae of  $\varphi$  (including  $\varphi$ ). Let further  $S_* = \{(s, \varphi) \mid s \in S; \varphi \in sf(\Phi(s))\}$ . The weighted MPG is constructed with

- $V_{\min} = \{(s, \varphi) \in S_* \mid \varphi = \varphi_1 \vee \varphi_2 \text{ or } (\varphi = (a, t) \text{ and } D(s, a, t) \in \mathcal{I}_c)\} \cup T_*$
- $V_{\max} = \{(s, \varphi) \in S_* \mid \varphi = \varphi_1 \wedge \varphi_2 \text{ or } (\varphi = (a, t) \text{ and } D(s, a, t) \in \mathcal{I}_u)\}$
- $E$  is defined as follows:

$$\begin{aligned}
 ((s, \varphi_1 \wedge \varphi_2), (s, \varphi_i)) &\in E & i &\in \{1, 2\} \\
 ((s, \varphi_1 \vee \varphi_2), (s, \varphi_i)) &\in E & i &\in \{1, 2\} \\
 ((s, (a, t)), (s, a, j, t)) &\in E & &\text{iff } j \text{ is an extremal point of } D(s, a, t) \\
 ((s, a, j, t), (t, \Phi(t))) &\in E & &\text{(always)}
 \end{aligned}$$

- $r((s, a, j, t), (t, \Phi(t))) = r(a)$  and  $r(-, -) = 0$  otherwise
- $d((s, a, j, t), (t, \Phi(t))) = j$  and  $d(-, -) = 0$  otherwise.

*Example 8.* In Figure 7 we show the result of translating the part of an MTSD from Figure 6a. This weighted MPG is similar to the one in Figure 6b, but instead of having a vertex for each satisfying set of outgoing transitions, we now have the syntactic tree of the obligation formula for each state. Further the vertex  $(s, b, 2, t_2)$  is left out, due to Lemma 12. Note that the vertices  $(t_1, \Phi(t_1))$ ,  $(t_2, \Phi(t_2))$  and  $(t_3, \Phi(t_3))$  are drawn as circles, because the player of these states depends on the obligation formula and the outgoing transitions.



**Figure 7:** Result of the improved translation of  $S$  in Figure 6a.

*Remark 3.* Observe that one can perform this optimization even in the general case. Indeed, for those  $s$  where  $\Phi(s)$  is positive we locally perform this transformation; for  $s$  with  $\Phi(s)$  containing negations we stick to the original expansion. Thus, the exponential (in out-degree) blow-up occurs only locally.

**Lemma 13.** *Both optimized translations are correct and on MTSDs where the obligation function is positive they run in polynomial time.*

*Proof.* A strategy for the player  $\min$  of a weighted MPG from a translation utilizing the optimizations of Section 4.3 can be translated into an implementation of the original MTSD as follows. Let  $\sigma$  be the strategy. We first define an auxiliary function on the

vertices of the MPG as follows:

$$\begin{aligned} f(s, (a, t)) &= \{(a, t)\} \\ f(s, \varphi \vee \psi) &= f(\sigma(s, \varphi \vee \psi)) \\ f(s, \varphi \wedge \psi) &= f(s, \varphi) \cup f(s, \psi) \end{aligned}$$

We then build the implementation as  $(S', T', D', \Phi')$  where

- $S' = \{s_1 \mid s \in S\}$
- $(s_1, a, t_1) \in T'$  if  $(a, t) \in f(s, \Phi(s))$
- $D'(s_1, a, t_1) = D(s, a, t)$  if  $D(s, a, t) \in \mathcal{I}_u$
- $D'(s_1, a, t_1) = j$  if  $D(s, a, t) \in \mathcal{I}_c$  and  $\sigma((s, (a, t))) = (s, a, j, t)$
- $\Phi'(s) = \bigwedge_{(s_1, a, t_1) \in T'} (a, t_1)$ .

The fact that the constructed implementation is indeed an implementation of the given MTSD and that the running cost of the implementation starting from state  $s_1$  is the same as  $v((s, \Phi(s)))$  is straightforward and can be proved as in the previous case.

Similarly, one can derive a strategy from an implementation so that its value does not get worse. We may safely assume that the implementation only implements durations as the extremal points of the controllable intervals. (Indeed, due to the original translation, Lemma 12, and the transformation back of Lemma 11, one can obtain an implementation with only extremal points that have the same or smaller cost.) The new transformation (strategy  $\sigma$  and mapping  $\mu$ ) is the same as the previous one of Lemma 11, the difference is that for history  $\pi$  ending in  $(s, \varphi_1 \vee \varphi_2)$  we set  $\sigma(\pi) = (s, \varphi_j)$  for an arbitrary  $j \in \{1, 2\}$  such that  $T(i_n) \models \varphi_j$  where  $\mu(\pi') = i_0 \cdots i_n$  and  $\pi'$  is the longest prefix of  $\pi$  ending with the vertex  $(s, \Phi(s))$ .

The polynomial running time is clear from the construction. ■

## 4.4 The Algorithm and its Complexity

The algorithm for our problem, given a specification  $\mathcal{S}$ , works as follows.

1. Nondeterministically choose hardware with the total price at most  $max_{ic}$ .
2. Create the weighted MPG out of  $\mathcal{S}$ .
3. Solve the weighted MPG using the reduction to MPG and any standard algorithm for MPG that finds an optimal strategy for player *min* and computes the value  $v(s_0)$ .
4. Transform the strategy to an implementation  $\mathcal{I}$ .
5. In the case of the cheapest-implementation problem return  $\mathcal{I}$ ;  
in the case of the implementation (decision) problem return  $v(s_0) \leq max_{rc}$ .

We can now prove the following result, finishing the proof of Theorem 6.

**Proposition 14.** *The implementation problem is in NP.*

*Proof.* We first nondeterministically guess the hardware assignment. Due to Section 4.2, we know that the desired implementation has the same states as the original MTSD and its transitions are a subset of the transitions of the original MTSD as the corresponding optimal strategies are positional. The first optimization (Section 4.3) guarantees that durations can be chosen as the extremal points of the intervals. Thus we can nondeterministically guess an optimal implementation and its durations, and verify that it satisfies the price inequality. ■

**Proposition 15.** *The implementation problem for MTSD with positive obligation function and a constant number of hardware components is in  $NP \cap coNP$  and solvable in pseudo-polynomial time.*

*Proof.* With the constant number of hardware components, we get a constant number of possible hardware configurations and we can check each configuration separately one by one. Further, by the first and the second optimization in Section 4.3, the MPG graph is of size  $\mathcal{O}(|T| + |\Phi|)$ . Therefore, we polynomially reduce the implementation problem to the problem of solving constantly many mean payoff games. The result follows by the existence of pseudo-polynomial algorithms for MPGs [ZP96]. ■

Further, our problem is at least as hard as solving MPGs that are clearly a special case of our problem. Hence, Theorem 7 follows.

## 5 Conclusion and Future Work

We have introduced a new extension of modal transition systems. The extension consists in introducing (1) variable time durations of actions and (2) pricing of actions, where we combine one-shot investment price for the hardware and cost for running it per each time unit it is active. We believe that this formalism is appropriate to modelling many types of embedded systems, where safety comes along with economical requirements.

We have solved the problem of finding the cheapest implementation w.r.t. the running cost given a maximum hardware investment we can afford, and we established the complexity of the decision problem in the general setting and in a practically relevant subcase revealing a close connection with mean payoff games.

As for the future work, apart from implementing the algorithm, one may consider two types of extensions. First, one can extend the formalism to cover the distinction between input, output and internal actions as it is usual in interface theories [Cha+03], and include even more time features, such as clocks in priced timed automata [BLR04; BBL08]. Second, one may extend the criteria for synthesis of the cheapest implementation by an additional requirement that the partial sums stay within given bounds

### *C. Dual-Priced Modal Transition Systems with Time Durations*

as done in [Bou+08], or requiring the satisfaction of a temporal property as suggested in [Cha+03; CD10].





# Channel Properties of Asynchronously Composed Petri Nets

Serge HADDAD

*LSV, ENS Cachan & CNRS & INRIA, France*

Rolf HENNICKER

*Institut für Informatik, Ludwig-Maximilians-Universität München, Germany*

Mikael H. MØLLER

*Aalborg University, Department of Computer Science, Denmark*

**Abstract** We consider asynchronously composed I/O-Petri nets (AIOPNs) with built-in communication channels. They are equipped with a compositional semantics in terms of asynchronous I/O-transition systems (AIOTSSs) admitting infinite state spaces. We study various channel properties that deal with the production and consumption of messages exchanged via the communication channels and establish useful relationships between them. In order to support incremental design we show that the channel properties considered in this work are preserved by asynchronous composition, i.e. they are compositional. As a crucial result we prove that the channel properties are decidable for AIOPNs.

## 1 Introduction

**(A)synchronous composition.** The design of hardware and software systems is often component-based which has well-known advantages: management of complexity, reusability, separation of concerns, collaborative design, etc. One critical feature of such systems is the protocol supporting the communication between components and, in particular, the way they synchronise. Synchronous composition ensures that both parts are aware that communication has taken place and then simplifies the validation of the system. However in a large scale distributed environment synchronous composition may lead to redhibitory inefficiency during execution and thus asynchronous composition should be adopted. The FIFO requirement of communication channels is often not appropriate in this context. This is illustrated by the concept of a software bus where applications push and pop messages in mailboxes. Also on the modelling level FIFO ordering is often not assumed, like for the composition of UML state machines which relies on event pools without specific requirements.

**Compositions of Petri nets.** In the context of Petri nets, composition has been studied both from theoretical and practical points of view. The process algebra approach has been investigated by several works leading to the Petri net algebra [BDK01]. Such an approach is closely related to synchronous composition. In [Sou91] and [SM91] asynchronous composition of nets are performed via a set of places or, more generally, via a subnet modelling some medium. Then structural restrictions on the subnets are proposed in order to preserve global properties like liveness or deadlock-freeness. In [Rei09] a general composition operator is proposed and its associativity is established. A closely related concept to composition is the one of open Petri nets which has been used in different contexts like the analysis of web services [SW09]. Numerous compositional approaches have been proposed for the modelling of complex applications but most of them are based on high-level Petri nets; see [GB05] for a detailed survey.

**Channel properties.** With the development of component-based applications, one is interested in verifying behavioural properties of the communication and, in the asynchronous case, in verifying the properties related to communication channels. Channel properties naturally occur when reasoning about distributed mechanisms, algorithms and applications (e.g. management of sockets in UNIX, maintaining unicity of a token in a ring based algorithm, recovery points with empty channels for fault management, guarantee of email reading, etc.).

**Our contributions.** In this work we are interested in general channel properties and not in specific system properties related to particular applications. The FIFO requirement for channels potentially can decrease the performance of large scale distributed systems. Thus we restrict ourselves to *unordered* channels which can be naturally modelled by places of Petri nets. We propose asynchronously composed Petri nets (AIOPNs) by (1) explicitly representing channels for internal communication inside the net and (2) defining communication capabilities to the outside in terms of (open) input and output labels with appropriate transitions. Then we define an asynchronous composition operator which introduces new channels for the communication between the composed nets. AIOPNs are equipped with a semantics in terms of asynchronously composed I/O-

transition systems (AIOTS). We show that this semantics is fully compositional, i.e. it commutes with asynchronous composition.

In our study two kinds of channel properties are considered which are related to consumption requirements and to the termination of communication. Consumption properties deal with requirements that messages sent to a communication channel should also be consumed. They can be classified w.r.t. two criteria. The first criterion is the nature of the requirement: consuming messages, decreasing the number of messages on a channel, and emptying channels. The second criterion expresses the way the requirement is achieved: possibly immediately, possibly after some delay, or necessarily in each weakly fair run. Communication termination deals with (immediate or delayed) closing of communication channels when the receiver is not ready to consume any more. We establish useful relations between the channel properties and prove that all channel properties considered here are compositional, i.e. preserved by asynchronous composition, which is an important prerequisite for incremental design.

From a verification point of view, we study the decidability of properties in the framework of AIOPN. Thanks to several complementary works on decidability for Petri net problems, we show that all channel properties considered in this work are decidable, though with a high computational complexity.

**Related work.** To the best of our knowledge, no work has considered channels explicitly defined for communication inside composite components. However there have been several works where channels are associated with asynchronous composition. They can be roughly classified depending whether their main feature is an algorithmic or a semantic one.

From an algorithmic point of view, in the seminal work of [BZ83], the authors discuss several properties like *channel boundedness* and *specified receptions* and propose methods to analyse them. In [CF05], a two-component based system is studied using a particular (decidable) channel property, the *half-duplex property*: at any time at most one channel is not empty.

From a semantic point view, in [HJK10] “connection-safe” component assemblies have been studied incorporating both synchronous and asynchronous communication. More recently in [BBO12] *synchronisability*, a property of asynchronous systems, is introduced such that when it holds the system can be safely abstracted by a synchronous one. In the framework of Petri nets, E. Kindler has defined Petri net components where the interface is composed by places and composition consists in merging places with same identities [Kin97]. He proposed a partial order semantics for such components and proved that the semantic is fully compositional. Furthermore, for a restricted linear temporal logic he established that properties of this logic are preserved by composition; however such a logic cannot express some of the channel properties we introduce here due to their branching kind.

The Petri net based formalism of open nets is the closest formalism to ours. Several works [LMW07],[SW09], and [SV12] address both the semantic point of view and the algorithmic one but only when the nets are assumed to be bounded which is not required here. We postpone to Section 2.2 a more detailed comparison with our formalism.

**Organisation.** In Section 2, we introduce AIOPNs and their asynchronous composition.

Then, we provide a compositional semantic for AIOPNs in Section 3 in terms of AIOTSSs. In Section 4, we define the channel properties and study their relationships and their preservation under asynchronous composition. In Section 5, we establish that all channel properties are decidable for AIOPNs. Finally, in Section 6, we conclude and give some perspectives for future work.

## 2 Asynchronous I/O-Petri Nets

### 2.1 Basic Notions

We recall some basic notions of labelled Petri nets and define their transition semantics.

**Definition 1** (Labelled Petri Net). A labelled Petri net is a tuple  $\mathcal{N} = (P, T, \Sigma, W^-, W^+, \lambda, m^0)$ , such that

- $P$  is a finite set of places,
- $T$  is a finite set of transitions with  $P \cap T = \emptyset$ ,
- $\Sigma$  is a finite alphabet,
- $W^-$  (resp.  $W^+$ ) is a matrix indexed by  $P \times T$  with values in  $\mathbb{N}$ ; it is called the backward (resp. forward) incidence matrix,
- $\lambda : T \rightarrow \Sigma$  is a transition labelling function, and
- $m^0$  is a vector indexed by  $P$  and called the initial marking.

The labelling function  $\lambda$  is extended as usual to sequences of transitions. The input (output resp.) vector  $W^-(t)$  ( $W^+(t)$  resp.) of a transition  $t$  is the column vector of matrix  $W^-$  ( $W^+$  resp.) with index  $t$ . Given two vectors  $\vec{v}$  and  $\vec{v}'$ , one writes  $\vec{v} \geq \vec{v}'$  if  $\vec{v}$  is componentwise greater or equal than  $\vec{v}'$ . A marking is a vector indexed by  $P$ . A transition  $t \in T$  is *firable* from a marking  $m$ , denoted by  $m \xrightarrow{t}$ , if  $m \geq W^-(t)$ . The firing of  $t$  from  $m$  leads to the marking  $m'$ , denoted by  $m \xrightarrow{t} m'$ , and defined by  $m' = m - W^-(t) + W^+(t)$ . If  $\lambda(t) = a$  we write  $m \xrightarrow{a} m'$ . The firing of a transition is extended as usual to firing sequences  $m \xrightarrow{\sigma} m'$  with  $\sigma \in T^*$ . A marking  $m$  is reachable if there exists a firing sequence  $\sigma \in T^*$  such that  $m^0 \xrightarrow{\sigma} m$ .

Our approach is based on a state transition system semantics for Petri nets.

**Definition 2** (Labelled Transition System). A labelled transition system (LTS) is a tuple  $\mathcal{S} = (\Sigma, Q, q^0, \longrightarrow)$ , such that

- $\Sigma$  is a finite set of labels,

- $Q$  is a (possibly infinite) set of states,
- $q^0 \in Q$  is the initial state, and
- $\longrightarrow \subseteq Q \times \Sigma \times Q$  is a labelled transition relation.

We will write  $q \xrightarrow{a} q'$  for  $(q, a, q') \in \longrightarrow$ , and we write  $q \xrightarrow{a}$  if there exists  $q' \in Q$  such that  $q \xrightarrow{a} q'$ . Let  $q_1 \in Q$ . A *trace* of  $\mathcal{S}$  starting in  $q_1$  is a finite or infinite sequence  $\rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} \dots$ . For  $a \in \Sigma$  we write  $a \in \rho$ , if there exists  $a_i$  in the sequence  $\rho$  such that  $a_i = a$ , and  $\#_\rho(a)$  denotes the (possibly infinite) number of occurrences of  $a$  in  $\rho$ . For  $q \in Q$  we write  $q \in \rho$ , if there exists  $q_i$  in the sequence  $\rho$  such that  $q_i = q$ . For  $\sigma = a_1 a_2 \dots a_n \in \Sigma^*$  and  $q, q' \in Q$  we write  $q \xrightarrow{\sigma} q'$  if there exists a (finite) trace  $q \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots \xrightarrow{a_n} q'$ . Often we need to reason about the successor states reachable from a given state  $q \in Q$  with a subset of labels  $\bar{\Sigma} \subseteq \Sigma$ . We define  $\text{Post}(q, \bar{\Sigma}) = \{q' \in Q \mid \exists a \in \bar{\Sigma} . q \xrightarrow{a} q'\}$  and we write  $\text{Post}(q)$  for  $\text{Post}(q, \Sigma)$ . Further we define  $\text{Post}^*(q, \bar{\Sigma}) = \{q' \in Q \mid \exists \sigma \in \bar{\Sigma}^* . q \xrightarrow{\sigma} q'\}$  and we write  $\text{Post}^*(q)$  for  $\text{Post}^*(q, \Sigma)$ .

**Definition 3** (Associated Labelled Transition System). *The semantics of a labelled Petri net  $\mathcal{N} = (P, T, \Sigma, W^-, W^+, \lambda, m^0)$  is given by its associated labelled transition system  $\text{lts}(\mathcal{N}) = (\Sigma, Q, q^0, \longrightarrow)$  which represents the reachability graph of the net and is defined by*

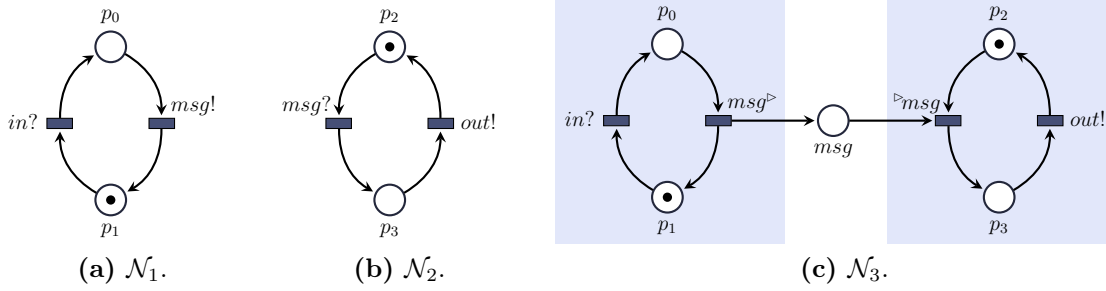
- $Q \subseteq \mathbb{N}^P$  is the set of reachable markings of  $\mathcal{N}$ ,
- $\longrightarrow = \{(m, a, m') \mid a \in \Sigma \text{ and } m \xrightarrow{a} m'\}$ , and
- $q^0 = m^0$ .

## 2.2 Asynchronous I/O-Petri Nets and Their Composition

In this paper we consider systems which may be open for communication with other systems and may be composed to larger systems. Both the behaviour of primitive components *and* of larger systems obtained by composition can be described by asynchronous I/O-Petri nets introduced in the following. We assume that communication is asynchronous and takes place via unbounded and unordered channels such that for each message type to be exchanged within a system there is exactly one communication channel. The open actions are modelled by distinguished input and output labels while communication inside the system via the channels is modelled by communication labels. Given a finite set  $C$  of channels, an *I/O-alphabet over  $C$*  is the disjoint union  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  of pairwise disjoint sets  $\text{in}$  of input labels,  $\text{out}$  of output labels and  $\text{com}$  of communication labels, such that  $\Sigma \cap C = \emptyset$ ,  $\text{com} = \{\triangleright a, a^\triangleright \mid a \in C\}$  and  $\text{in}$  and  $\text{out}$  do not contain labels of the form  $\triangleright x$  or  $x^\triangleright$ . For each channel  $a \in C$ , the communica-

### D. Channel Properties of Asynchronously Composed Petri Nets

tion label  $\triangleright a$  represents consumption of a message from the channel  $a$  and  $a^\triangleright$  represents putting a message on  $a$ . Each channel is modelled as a place and the transitions for communication actions are modelled by putting or removing tokens from the channel places. Three examples of AIOPNs are shown in Fig. 1. The nets  $\mathcal{N}_1$  and  $\mathcal{N}_2$  model primitive components (without channels) which repeatedly input and output messages. The net  $\mathcal{N}_3$  in Fig. 1c models a simple producer/consumer system with one channel  $msg$  obtained by composition of the two primitive components; see below. Here and in the following drawings input labels are indicated by “?” and output labels by “!”.



**Figure 1:** Asynchronous I/O-Petri nets.

**Definition 4** (Asynchronous I/O-Petri net). *An asynchronous I/O-Petri net (AIOPN) is a tuple  $\mathcal{N} = (C, P, T, \Sigma, W^-, W^+, \lambda, m^0)$ , such that*

- $(P, T, \Sigma, W^-, W^+, \lambda, m^0)$  is a labelled Petri net,
- $C$  is a finite set of channels,
- $C \subseteq P$ , i.e. each channel is a place,
- $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is an I/O-alphabet over  $C$ ,
- for all  $a \in C$  and  $t \in T$ ,  

$$W^-(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = \triangleright a, \\ 0 & \text{otherwise} \end{cases} \quad W^+(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = a^\triangleright, \\ 0 & \text{otherwise} \end{cases}$$
- for all  $a \in C$ ,  $m^0(a) = 0$ .

Two I/O-alphabets are composable if there are no name conflicts between labels and channels and, following [AH05], if shared labels are either input labels of one alphabet and output labels of the other or conversely. For the composition each shared label  $a$  gives rise to a new communication channel, also called  $a$ , and hence to new communication labels  $a^\triangleright$  for putting and  $\triangleright a$  for removing messages. The input and output labels of the alphabet composition are the non-shared input and output labels of the underlying alphabets.

**Definition 5** (Alphabet composition). *Let  $\Sigma_S = \text{in}_S \uplus \text{out}_S \uplus \text{com}_S$  and  $\Sigma_T = \text{in}_T \uplus \text{out}_T \uplus \text{com}_T$  be two I/O-alphabets over channels  $C_S$  and  $C_T$  resp.  $\Sigma_S$  and  $\Sigma_T$  are composable if  $(\Sigma_S \cup \Sigma_T) \cap (C_S \cup C_T) = \emptyset$  and  $\Sigma_S \cap \Sigma_T = (\text{in}_S \cap \text{out}_T) \cup (\text{in}_T \cap \text{out}_S)$ . The composition of  $\Sigma_S$  and  $\Sigma_T$  is the I/O-alphabet  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  over the composed set of channels  $C = C_S \uplus C_T \uplus C_{ST}$ , with new channels  $C_{ST} = \Sigma_S \cap \Sigma_T$ , such that*

- $\text{in} = (\text{in}_S \setminus \text{out}_T) \uplus (\text{in}_T \setminus \text{out}_S)$ ,
- $\text{out} = (\text{out}_S \setminus \text{in}_T) \uplus (\text{out}_T \setminus \text{in}_S)$ , and
- $\text{com} = \{a^\triangleright, \triangleright a \mid a \in C\}^a$

---

<sup>a</sup> $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is indeed a disjoint union, since for all  $a \in C_{ST}$  the communication labels  $a^\triangleright, \triangleright a$  are new names due to the general assumption that input and output labels are not of the form  $x^\triangleright, \triangleright x$ .

Two AIOPNs can be (asynchronously) composed, if their underlying I/O-alphabets are composable. The composition is constructed by taking the disjoint union of the underlying nets and adding a new channel place for each shared label. Every transition with shared output label  $a$  becomes a transition with the communication label  $a^\triangleright$  that produces a token on the (new) channel place  $a$  and, similarly, any transition with shared input label  $a$  becomes a transition with the communication label  $\triangleright a$  that consumes a token from the (new) channel place  $a$ . For instance, the AIOPN  $\mathcal{N}_3$  in Fig. 1c is the result of the asynchronous composition of the two AIOPNs  $\mathcal{N}_1$  and  $\mathcal{N}_2$  in Fig. 1a and Fig. 1b resp. The newly introduced channel place is the place *msg*.

Our approach looks very similar to open Petri nets, see e.g. [LMW07], which use interface places for communication. But there are two important differences: First, we explicitly distinguish channel places thus being able to reason on the communication behaviour between composed components; see Sect. 4. The second difference is quite important from the software engineer's point of view. We do not use interface places to indicate communication abilities of a component but we use distinguished input and output labels instead. We believe that this has an important advantage to achieve separation of concerns: The designer of a component has not to take care whether the component will be used in a synchronous or in an asynchronous environment later on; this should be the decision of the system architect. Indeed open Petri nets already rely on asynchronous composition while our formalism would also support synchronous composition, see [Pet81], and mixed architectures. Since synchronous composition relies on matching of transitions rather than communication channels we have not elaborated this case here. The difference between AIOPNs and modal I/O-Petri nets introduced in [EHH12] is that AIOPNs comprise distinguished channel places but they do not support modalities for refinement (yet).

**Definition 6** (Asynchronous composition of AIOPNs). Let  $\mathcal{N} = (C_{\mathcal{N}}, P_{\mathcal{N}}, T_{\mathcal{N}}, \Sigma_{\mathcal{N}}, W_{\mathcal{N}}^-, W_{\mathcal{N}}^+, \lambda_{\mathcal{N}}, m_{\mathcal{N}}^0)$  and  $\mathcal{M} = (C_{\mathcal{M}}, P_{\mathcal{M}}, T_{\mathcal{M}}, \Sigma_{\mathcal{M}}, W_{\mathcal{M}}^-, W_{\mathcal{M}}^+, \lambda_{\mathcal{M}}, m_{\mathcal{M}}^0)$  be two AIOPNs.  $\mathcal{N}$  and  $\mathcal{M}$  are composable if  $\Sigma_{\mathcal{N}}$  and  $\Sigma_{\mathcal{M}}$  are composable and if  $P_{\mathcal{N}} \cap P_{\mathcal{M}} = \emptyset$ ,  $(P_{\mathcal{N}} \cup P_{\mathcal{M}}) \cap (\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}) = \emptyset$ , and  $T_{\mathcal{N}} \cap T_{\mathcal{M}} = \emptyset$ . In this case their asynchronous composition is the AIOPN  $\mathcal{N} \otimes_{pn} \mathcal{M} = (C, P, T, \Sigma, W^-, W^+, \lambda, m^0)$  defined as follows:

- $C = C_{\mathcal{N}} \uplus C_{\mathcal{M}} \uplus C_{\mathcal{NM}}$ , with  $C_{\mathcal{NM}} = \Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}$ ,
- $P = P_{\mathcal{N}} \uplus P_{\mathcal{M}} \uplus C_{\mathcal{NM}}$ ,
- $T = T_{\mathcal{N}} \uplus T_{\mathcal{M}}$ ,
- $\Sigma$  is the alphabet composition of  $\Sigma_{\mathcal{S}}$  and  $\Sigma_{\mathcal{T}}$ ,
- $W^-$  (resp.  $W^+$ ) is the backward (forward) incidence matrix defined by:

for all  $p \in P_{\mathcal{N}} \cup P_{\mathcal{M}}$  and  $t \in T$

$$W^-(p, t) = \begin{cases} W_{\mathcal{N}}^-(p, t) & \text{if } p \in P_{\mathcal{N}}, t \in T_{\mathcal{N}} \\ W_{\mathcal{M}}^-(p, t) & \text{if } p \in P_{\mathcal{M}}, t \in T_{\mathcal{M}} \\ 0 & \text{otherwise} \end{cases}$$

$$W^+(p, t) = \begin{cases} W_{\mathcal{N}}^+(p, t) & \text{if } p \in P_{\mathcal{N}}, t \in T_{\mathcal{N}} \\ W_{\mathcal{M}}^+(p, t) & \text{if } p \in P_{\mathcal{M}}, t \in T_{\mathcal{M}} \\ 0 & \text{otherwise} \end{cases}$$

for all  $a \in C_{\mathcal{NM}}$  and  $t \in T$

$$W^-(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = \triangleright a \\ 0 & \text{otherwise} \end{cases} \quad W^+(a, t) = \begin{cases} 1 & \text{if } \lambda(t) = a^\triangleright \\ 0 & \text{otherwise} \end{cases}$$

- $\lambda : T \rightarrow \Sigma$  is defined, for all  $t \in T$ , by

$$\lambda(t) = \begin{cases} \lambda_{\mathcal{N}}(t) & \text{if } t \in T_{\mathcal{N}}, \lambda_{\mathcal{N}}(t) \notin \Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}} \\ \lambda_{\mathcal{M}}(t) & \text{if } t \in T_{\mathcal{M}}, \lambda_{\mathcal{M}}(t) \notin \Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}} \\ \triangleright \lambda_{\mathcal{N}}(t) & \text{if } t \in T_{\mathcal{N}}, \lambda_{\mathcal{N}}(t) \in in_{\mathcal{N}} \cap out_{\mathcal{M}} \\ \triangleright \lambda_{\mathcal{M}}(t) & \text{if } t \in T_{\mathcal{M}}, \lambda_{\mathcal{M}}(t) \in in_{\mathcal{M}} \cap out_{\mathcal{N}} \\ \lambda_{\mathcal{N}}(t)^\triangleright & \text{if } t \in T_{\mathcal{N}}, \lambda_{\mathcal{N}}(t) \in in_{\mathcal{M}} \cap out_{\mathcal{N}} \\ \lambda_{\mathcal{M}}(t)^\triangleright & \text{if } t \in T_{\mathcal{M}}, \lambda_{\mathcal{M}}(t) \in in_{\mathcal{N}} \cap out_{\mathcal{M}} \end{cases}$$

- $m^0$  is defined, for all  $p \in P$ , such that

$$m^0(p) = \begin{cases} m_{\mathcal{N}}^0(p) & \text{if } p \in P_{\mathcal{N}} \\ m_{\mathcal{M}}^0(p) & \text{if } p \in P_{\mathcal{M}} \\ 0 & \text{otherwise} \end{cases}$$



### 3 Compositional Semantics

We extend the transition system semantics of labelled Petri nets defined in Sect. 2.1 to AIOPNs. For this purpose we introduce asynchronous I/O-transition systems which are labelled transition systems extended by channels and a *channel valuation* function  $\text{val} : Q \rightarrow \mathbb{N}^C$ . The channel valuation function determines for each state  $q \in Q$  how many messages are actually pending on each channel  $a \in C$ . For  $q \in Q$  and  $a \in C$  we use the notation  $\text{val}(q)[a++]$  to denote the updated map

$$\text{val}(q)[a++](x) = \begin{cases} \text{val}(q)(a) + 1 & \text{if } x = a, \\ \text{val}(q)(x) & \text{otherwise.} \end{cases}$$

The updated map  $\text{val}(q)[a--]$  is defined similarly. Instead of  $\text{val}(q)(a)$  we will often write  $\text{val}(q, a)$ .

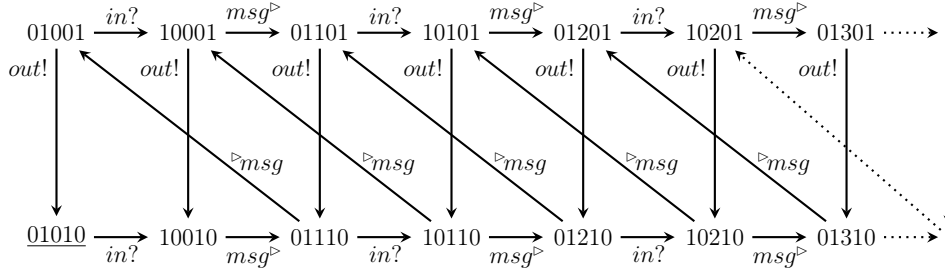
**Definition 7** (Asynchronous I/O-transition system). *An asynchronous I/O-transition system (AIOTS) is a tuple  $\mathcal{S} = (C, \Sigma, Q, q^0, \rightarrow, \text{val})$ , such that*

- $(\Sigma, Q, q^0, \rightarrow)$  is a labelled transition system,
- $C$  is a finite set of channels,
- $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is an I/O-alphabet over  $C$ ,
- $\text{val} : Q \rightarrow \mathbb{N}^C$  is a function, such that for all  $a \in C, q, q' \in Q$ :
  - $\text{val}(q^0, a) = 0$ ,
  - $q \xrightarrow{a^>} q' \implies \text{val}(q') = \text{val}(q)[a++]$ ,
  - $q \xrightarrow{^>a} q' \implies \text{val}(q, a) > 0$  and  $\text{val}(q') = \text{val}(q)[a--]$ , and
  - for all  $x \in (\text{in} \cup \text{out}), q \xrightarrow{x} q' \implies \text{val}(q') = \text{val}(q)$ .

The first condition for  $\text{val}$  assumes that initially all communication channels are empty. The second condition states that transitions with labels  $a^>$  and  $^>a$  have the desired effect of putting one message on a channel (consuming one message from a channel resp.). The last condition requires that the input and output actions of an open system do not change the valuation of any channel. Sometimes we need to reason about the number of messages on a subset  $B \subseteq C$  of the channels in a state  $q \in Q$ . We define  $\text{val}(q, B) = \sum_{a \in B} \text{val}(q, a)$ .

The semantics of an asynchronous I/O-Petri net  $\mathcal{N}$  is given by its *associated* asynchronous I/O-transition system  $\text{aiots}(\mathcal{N})$ . It is based on the transition system semantics of a labelled Petri net (see Sect. 2.1) such that markings become states, but additionally we define the valuation of a channel in a current state  $m$  by the number of tokens on the channel under the marking  $m$ .

### D. Channel Properties of Asynchronously Composed Petri Nets



**Figure 2:** Part of the associated AIOTS for  $\mathcal{N}_3$  in Fig. 1c.

**Definition 8** (Associated asynchronous I/O-transition system). *Let  $\mathcal{N} = (C, P, T, \Sigma, W^-, W^+, \lambda, m^0)$  be an AIOPN. The AIOTS associated with  $\mathcal{N}$  is given by  $\text{aiots}(\mathcal{N}) = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$ , such that*

- $(\Sigma, Q, q^0, \longrightarrow) = \text{lts}(P, T, \Sigma, W^-, W^+, \lambda, m^0)$ ,
- for all  $a \in C$  and  $m \in Q$ ,  $\text{val}(m, a) = m(a)$ .

*Example 9.* The transition systems associated with the AIOPNs  $\mathcal{N}_1$  and  $\mathcal{N}_2$  in Fig. 1a and 1b have two reachable states and the transitions between them correspond directly to their Petri net representations. The situation is different for the AIOPN  $\mathcal{N}_3$  in Fig. 1c. It has infinitely many reachable markings and hence its associated AIOTS has infinitely many reachable states. Fig. 2 shows an excerpt of it. The states indicate the number of tokens in each place in the order  $p_0, p_1, \text{msg}, p_2, p_3$ . The initial state is underlined.

Like AIOPNs also two AIOTSs can be asynchronously composed, if their underlying I/O-alphabets are composable. The composition is constructed by introducing a new communication channel for each shared input/output action and by appropriate transitions for the corresponding communication actions that modify the valuation of the new channels (see items 3 and 4 in Def. 9). Since the states of the composition must record the number of messages on the new channels  $C_{\mathcal{ST}}$ , the state space of the composition adds to the Cartesian product of the underlying state spaces the set  $\mathbb{N}^{C_{\mathcal{ST}}}$  of valuations of the new channels. For a valuation  $\vec{v} : C_{\mathcal{ST}} \mapsto \mathbb{N}$  and channel  $a \in C_{\mathcal{ST}}$  we use the notation  $\vec{v}[a++]$  ( $\vec{v}[a--]$  resp.) to denote the updated map which increments (decrements) the value of  $a$  by 1 and leaves the values of all other channels unchanged.

**Definition 9** (Asynchronous composition of AIOTS).

*Let  $\mathcal{S} = (C_{\mathcal{S}}, \Sigma_{\mathcal{S}}, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \longrightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}})$  and  $\mathcal{T} = (C_{\mathcal{T}}, \Sigma_{\mathcal{T}}, Q_{\mathcal{T}}, q_{\mathcal{T}}^0, \longrightarrow_{\mathcal{T}}, \text{val}_{\mathcal{T}})$  be two AIOTSs.  $\mathcal{S}$  and  $\mathcal{T}$  are composable if  $\Sigma_{\mathcal{S}}$  and  $\Sigma_{\mathcal{T}}$  are composable. In this case their asynchronous composition is the AIOTS  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$  defined as follows:*

- $C = C_{\mathcal{S}} \uplus C_{\mathcal{T}} \uplus C_{\mathcal{ST}}$ , with  $C_{\mathcal{ST}} = \Sigma_{\mathcal{S}} \cap \Sigma_{\mathcal{T}}$ ,

- $\Sigma$  is the alphabet composition of  $\Sigma_S$  and  $\Sigma_T$ ,
- $Q \subseteq Q_S \times Q_T \times \mathbb{N}^{C_{ST}}$ ,
- $q^0 = (q_S^0, q_T^0, \vec{0}) \in Q$ , with  $\vec{0}$  being the zero-map,
- $Q$  and  $\longrightarrow$  are inductively defined as follows for  $(q_S, q_T, \vec{v}) \in Q$ :
  - 1: For all  $a \in (\Sigma_S \setminus C_{ST})$ , if  $q_S \xrightarrow{a}_S q'_S$  then  
 $(q_S, q_T, \vec{v}) \xrightarrow{a} (q'_S, q_T, \vec{v})$  and  $(q'_S, q_T, \vec{v}) \in Q$ .
  - 2: For all  $a \in (\Sigma_T \setminus C_{ST})$ , if  $q_T \xrightarrow{a}_T q'_T$  then  
 $(q_S, q_T, \vec{v}) \xrightarrow{a} (q_S, q'_T, \vec{v})$  and  $(q_S, q'_T, \vec{v}) \in Q$ .
  - 3: For all  $a \in \text{in}_S \cap \text{out}_T$ ,
    - 3.1: if  $q_S \xrightarrow{a}_S q'_S$  and  $\vec{v}(a) > 0$  then  
 $(q_S, q_T, \vec{v}) \xrightarrow{\vec{v}_a} (q'_S, q_T, \vec{v}[a--])$  and  $(q'_S, q_T, \vec{v}[a--]) \in Q$ ,
    - 3.2: if  $q_T \xrightarrow{a}_T q'_T$  then  
 $(q_S, q_T, \vec{v}) \xrightarrow{a^>} (q_S, q'_T, \vec{v}[a++])$  and  $(q_S, q'_T, \vec{v}[a++]) \in Q$ .
  - 4: For all  $a \in \text{in}_T \cap \text{out}_S$ ,
    - 4.1: if  $q_S \xrightarrow{a}_S q'_S$  then  $(q_S, q_T, \vec{v}) \xrightarrow{a^>} (q'_S, q_T, \vec{v}[a++])$   
 and  $(q'_S, q_T, \vec{v}[a++]) \in Q$ ,
    - 4.2: if  $q_T \xrightarrow{a}_T q'_T$  and  $\vec{v}(a) > 0$  then  
 $(q_S, q_T, \vec{v}) \xrightarrow{\vec{v}_a} (q_S, q'_T, \vec{v}[a--])$  and  $(q_S, q'_T, \vec{v}[a--]) \in Q$ .
- For all  $(q_S, q_T, \vec{v}) \in Q$  and  $a \in C$ ,
 
$$\text{val}((q_S, q_T, \vec{v}), a) = \begin{cases} \text{val}_S(q_S, a) & \text{if } a \in C_S \\ \text{val}_T(q_T, a) & \text{if } a \in C_T \\ \vec{v}(a) & \text{if } a \in C_{ST} \end{cases}$$

For the rules (1), (3.1) and (4.1), we say that the resulting transition in the composition is triggered by  $\mathcal{S}$ . Let  $\rho$  be a trace of  $\mathcal{S} \otimes \mathcal{T}$  starting from a state  $q = (q_S, q_T, \vec{v}) \in Q$ . The projection of  $\rho$  to  $\mathcal{S}$ , denoted by  $\rho|_S$ , is the sequence of transitions of  $\mathcal{S}$ , starting from  $q_S$ , which have triggered corresponding transitions in  $\rho$ .

We can now consider the class of all asynchronous I/O-transition systems that are generated by asynchronous I/O-Petri nets and prove that this class is closed under asynchronous composition. This is a consequence of the fact that the generation of AIOTs from AIOPNs commutes with asynchronous composition. The proof of this theorem is technical, but straightforward.

**Theorem 10.** *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two composable AIOPNs. Then it holds that  $\text{aiots}(\mathcal{N} \otimes_{pn} \mathcal{M}) = \text{aiots}(\mathcal{N}) \otimes \text{aiots}(\mathcal{M})$  (up to bijection between state spaces).*

*Proof.* We give a sketch of the proof. The full proof is technical but straightforward. Let  $\text{aiots}(\mathcal{N} \otimes_{pn} \mathcal{M}) = (C_A, \Sigma_A, Q_A, q_A^0, \longrightarrow_A, \text{val}_A)$  and  $\text{aiots}(\mathcal{N}) \otimes \text{aiots}(\mathcal{M}) = (C_B, \Sigma_B, Q_B, q_B^0, \longrightarrow_B, \text{val}_B)$ . It is clear by Def. 6, Def. 8 and Def. 9 that

$$C_A = C_{\mathcal{N}} \uplus C_{\mathcal{M}} \uplus (\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}) = C_B.$$

and that  $\Sigma_A = \Sigma_B$ . We will now consider the two state spaces. By Def. 8 a state in  $Q_A$  is a reachable marking of  $\mathcal{N} \otimes_{pn} \mathcal{M}$ , by Def. 6 we get that the set of places of  $\mathcal{N} \otimes_{pn} \mathcal{M}$  is  $P_{\mathcal{N}} \uplus P_{\mathcal{M}} \uplus \{\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}\}$ . Thus, it follows that

$$Q_A \subseteq \mathbb{N}^{P_{\mathcal{N}} \uplus P_{\mathcal{M}} \uplus \{\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}\}}.$$

By Def. 9 a state in  $Q_B$  is a reachable state consisting of three parts, a state of  $\text{aiots}(\mathcal{N}) \subseteq \mathbb{N}^{P_{\mathcal{N}}}$ , a state of  $\text{aiots}(\mathcal{M}) \subseteq \mathbb{N}^{P_{\mathcal{M}}}$ , and the content of the new channels in their composition. Hence,

$$Q_B \subseteq \mathbb{N}^{P_{\mathcal{N}}} \times \mathbb{N}^{P_{\mathcal{M}}} \times \mathbb{N}^{\{\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}\}}.$$

Obviously, the two supersets  $\mathbb{N}^{P_{\mathcal{N}} \uplus P_{\mathcal{M}} \uplus \{\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}\}}$  and  $\mathbb{N}^{P_{\mathcal{N}}} \times \mathbb{N}^{P_{\mathcal{M}}} \times \mathbb{N}^{\{\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}\}}$  are set-theoretically isomorphic. It remains to prove that the transition relations  $\longrightarrow_A$  and  $\longrightarrow_B$  are equal for isomorphic states which implies also that  $Q_A$  and  $Q_B$  are isomorphic. Finally one must prove that the valuation functions coincide on isomorphic states. These two facts are straightforward but technical to prove on the basis of Def. 6 and Def. 9. ■

## 4 Channel Properties and Their Compositionality

In this section we consider various properties concerning the asynchronous communication via channels. We give a classification of the properties, show their relationships and prove that they are compositional w.r.t. asynchronous composition, a prerequisite for incremental design.

### 4.1 Channel Properties

We consider two classes of channel properties. The first class deals with the requirements that messages sent to a communication channel should also be consumed; the second class concerns the termination of communication in the sense that if consumption from a channel has been stopped then also production on this channel will stop. The channel properties will be defined by considering the semantics of AIOPNs, i.e. they will be formulated for AIOTSSs.

Some of the properties, precisely the “necessarily properties” of type (c) in Def. 11 below, rely on the consideration of system runs. In principle a system run is a maximal

execution trace; it can be infinite but also finite if no further actions are enabled. It is important to remember, that we deal with open systems whose possible behaviours are also determined by the environment. Hence, the definition of a system run must take into account the possibility that the system may stop in a state where the environment does not serve any offered input of the system while at the same time the system has no enabled autonomous action, i.e. an action which is not an input from the environment. Such states will be called pure input states. They correspond to markings that “stop except for inputs” in [SV12]. Note that all possible communication actions inside the system can be autonomously executed. The same holds for output actions to the environment, since we are working with asynchronous communication such that messages can always be sent, even if they are never accepted by the environment. Formally, system runs are defined as follows.

Let  $\mathcal{S} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$  be an AIOTS with  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$ . A state  $q \in Q$  is called a *pure input state* if  $\text{Post}(q, \Sigma \setminus \text{in}) = \emptyset$ , i.e. only inputs are enabled. A pure input state is a potential deadlock, as the environment of  $\mathcal{S}$  might not serve any inputs for  $\mathcal{S}$ . Let  $q_1 \in Q$ . A *run* of  $\mathcal{S}$  starting in  $q_1$  is a trace of  $\mathcal{S}$  starting in  $q_1$ , that is either infinite or finite such that its last state is a pure input state. We denote the set of all runs of  $\mathcal{S}$  starting from  $q_1$  as  $\text{run}_{\mathcal{S}}(q_1)$ .

In the following we also assume that system runs are only executed in a runtime infrastructure which follows a weakly fair scheduling policy. In our context this means that any autonomous action  $a$ , that is always enabled from a certain state on, will infinitely often be executed. Formally, a run  $\rho \in \text{run}_{\mathcal{S}}(q_1)$  with  $q_1 \in Q, \rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots$ , is called *weakly fair* if it is finite or if it is infinite and for all  $a \in (\Sigma \setminus \text{in})$  the following holds:

$$(\exists k \geq 1 . \forall i \geq k . q_i \xrightarrow{a}) \implies (\forall k \geq 1 . \exists i \geq k . a_i = a).$$

We denote the set of all weakly fair runs of  $\mathcal{S}$  starting from  $q_1$  by  $\text{wfrun}_{\mathcal{S}}(q_1)$ . It should be noted that for our results it is sufficient to use weak fairness instead of strong fairness.\*

*Example 10.* Let  $\mathcal{S} = \text{aiots}(\mathcal{N}_3)$  be the associated AIOTS of the Petri net  $\mathcal{N}_3$  in Fig. 1c on page 106. An excerpt of  $\mathcal{S}$  has been shown in Fig. 2. The following are three traces of  $\mathcal{S}$  starting in the initial state 01010:

$$\begin{aligned} \rho_0 &= 01010, \\ \rho_1 &= 01010 \xrightarrow{\text{in}^?} 10010 \xrightarrow{\text{msg}^>} 01110 \xrightarrow{\triangleright \text{msg}} 01001, \\ \rho_2 &= 01010 \xrightarrow{\text{in}^?} 10010 \xrightarrow{\text{msg}^>} 01110 \xrightarrow{\triangleright \text{msg}} 01001 \xrightarrow{\text{out}!} 01010. \end{aligned}$$

The traces  $\rho_0$  and  $\rho_2$  are runs of  $\mathcal{S}$  while  $\rho_1$  is not a run, since 01001 is not a pure input state. Now consider the infinite trace indicated at the bottom line in Fig. 2, that is an infinite alternation of  $\text{in}^?$  and  $\text{msg}^>$ . The trace is a run, since it is infinite. But the run is not weakly fair, since from the second state on  $\triangleright \text{msg}$  is always enabled but never taken.

---

\*For a discussion of the different fairness properties see, e.g., [Ber+01].

#### D. Channel Properties of Asynchronously Composed Petri Nets

Our first class of channel properties deals with the consumption of previously produced messages. We consider four groups of such properties (P1) - (P4) with different strength. In each case we consider three variants which all are parametrised w.r.t. a subset  $B$  of the communication channels.

Let us discuss the consuming properties (P1) of Def. 11 below for an AIOTS  $\mathcal{S}$  and a subset  $B$  of its channels. Property (P1.a) requires, for each channel  $a \in B$ , that if in an arbitrary reachable state  $q$  of  $\mathcal{S}$  there is a message available on  $a$ , then  $\mathcal{S}$  can consume the message possibly after the execution of some autonomous actions. Let us comment on the role of the environment for the formulation of this property. First, we consider arbitrary reachable states  $q \in \text{Post}^*(q^0)$  with  $q^0$  being the initial state of  $\mathcal{S}$ . This means that we take into account the worst environment which can let  $\mathcal{S}$  go everywhere by providing (non-deterministically) all inputs that  $\mathcal{S}$  can accept. Then, at some point at which a message is available on channel  $a$ , the environment can stop to provide further inputs and waits whether  $\mathcal{S}$  can *autonomously* reach a state  $q' \in \text{Post}^*(q, \Sigma \setminus \text{in})$  in which it can consume from  $a$ , i.e. execute  $\triangleright a$ . To allow autonomous actions before consumption is inspired by the property of “output compatibility” studied for synchronously composed transition systems in [HK11]. Property (P1.b) does not allow autonomous actions before consumption. It requires that  $\mathcal{S}$  can immediately consume the message in state  $q$ , similar to the property of specified reception in [BZ83]. Property (P1.c) requires that the message will definitely be consumed on each weakly fair run of  $\mathcal{S}$  starting from  $q$  and, due to the definition of a system run, that this will happen in any environment whatever inputs are provided.

As an example consider the AIOTS  $\mathcal{S} = \text{aiots}(\mathcal{N}_3)$  associated with the AIOPN  $\mathcal{N}_3$  in Fig. 1c and its reachable state 01101 such that one message is on channel  $msg$ . In this state  $\mathcal{S}$  can autonomously perform the output  $out!$  reaching state 01110 and then it can consume the message by performing  $\triangleright msg$ . Since also in all other reachable states in which the channel is not empty  $\mathcal{S}$  can autonomously reach a state in which it can consume from the channel,  $\mathcal{S}$  satisfies property (P1.a) (for its only channel  $msg$ ). However,  $\mathcal{S}$  is not strongly consuming (P1.b). For instance in state 01101,  $\mathcal{S}$  cannot immediately consume the message. On the other hand,  $\mathcal{S}$  is necessarily consuming (P1.c). Whenever in a reachable state  $q$  the channel is not empty an autonomous action, either  $\triangleright msg$  or  $out!$ , is enabled. Hence  $q$  is not a pure input state and, due to the weak fairness condition, eventually  $\triangleright msg$  or  $out!$  must be performed in any weakly fair run starting from  $q$ . If  $\triangleright msg$  is performed we are done. If  $out!$  is performed we reach a state where  $\triangleright msg$  is enabled and with the same reasoning eventually  $\triangleright msg$  will be performed. This can be easily detected by considering Fig. 2.

The other groups of properties (P2) - (P4) express successively stronger (or equivalent) requirements on the kind of consumption. For instance, (P3) requires that the consumption will lead to a state in which the channel is empty. Again we distinguish if this can be achieved after some autonomous actions (P3.a), can be achieved immediately (P3.b), or must be achieved in any weakly fair run (P3.c).

**Definition 11** (Consumption properties). *Let  $\mathcal{S} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$  be an AIOTS with I/O-alphabet  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  and let  $B \subseteq C$  be a subset of its channels.*

*P1: (Consuming)*

a)  $\mathcal{S}$  is  $B$ -consuming, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \exists q' \in \text{Post}^*(q, \Sigma \setminus \text{in}) . q' \xrightarrow{\triangleright_a} .$$

b)  $\mathcal{S}$  is strongly  $B$ -consuming, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies q \xrightarrow{\triangleright_a} .$$

c)  $\mathcal{S}$  is necessarily  $B$ -consuming, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \forall \rho \in \text{wfrun}_{\mathcal{S}}(q) . \triangleright_a \in \rho .$$

*P2: (Decreasing)*

a)  $\mathcal{S}$  is  $B$ -decreasing, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \exists q' \in \text{Post}^*(q, \Sigma \setminus \text{in}) . \text{val}(q', a) < \text{val}(q, a) .$$

b)  $\mathcal{S}$  is strongly  $B$ -decreasing, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \exists q' \in \text{Post}(q, \Sigma \setminus \text{in}) . \text{val}(q', a) < \text{val}(q, a) .$$

c)  $\mathcal{S}$  is necessarily  $B$ -decreasing, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \forall \rho \in \text{wfrun}_{\mathcal{S}}(q), \exists q' \in \rho . \text{val}(q', a) < \text{val}(q, a) .$$

*P3: (Emptying)*

a)  $\mathcal{S}$  is  $B$ -emptying, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \exists q' \in \text{Post}^*(q, \Sigma \setminus \text{in}) . \text{val}(q', a) = 0 .$$

b)  $\mathcal{S}$  is strongly  $B$ -emptying, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \exists q' \in \text{Post}(q, \Sigma \setminus \text{in}) . \text{val}(q', a) = 0 .$$

c)  $\mathcal{S}$  is  $B$ -necessarily emptying, if for all  $a \in B$  and all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, a) > 0 \implies \forall \rho \in \text{wfrun}_{\mathcal{S}}(q), \exists q' \in \rho . \text{val}(q', a) = 0 .$$

P4: (Wholly emptying)

a)  $\mathcal{S}$  is  $B$ -wholly emptying, if for all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, B) > 0 \implies \exists q' \in \text{Post}^*(q, \Sigma \setminus \text{in}) . \text{val}(q', B) = 0 .$$

b)  $\mathcal{S}$  is strongly  $B$ -wholly emptying, if for all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, B) > 0 \implies \exists q' \in \text{Post}(q, \Sigma \setminus \text{in}) . \text{val}(q', B) = 0 .$$

c)  $\mathcal{S}$  is  $B$ -necessarily wholly emptying, if for all  $q \in \text{Post}^*(q^0)$ ,

$$\text{val}(q, B) > 0 \implies \forall \rho \in \text{wfrun}_{\mathcal{S}}(q), \exists q' \in \rho . \text{val}(q', B) = 0 .$$

Note if the initial state of  $\mathcal{S}$  is reachable from all other reachable states, i.e. the initial state is a *home state*, then  $\mathcal{S}$  is  $B$ -wholly emptying.

The next class of channel properties concerns the termination of communication. We consider two variants: (P5.a) requires that in any weakly fair run, in which consumption from a channel  $a$  has stopped, only finitely many subsequent productions are possible, i.e. the channel is closed after a while. Property (P5.b) expresses that the channel is immediately closed.

**Definition 12** (Communication stopping). *Let  $\mathcal{S}$  be an AIOTS and  $B \subseteq C$  be a subset of its channels.*

P5: (Communication stopping)

a)  $\mathcal{S}$  is  $B$ -communication stopping, if for all  $q \in \text{Post}^*(q^0)$ ,

$$\rho \in \text{wfrun}_{\mathcal{S}}(q) \text{ and } a \in B, \#_{\rho}(^{\triangleright}a) = 0 \implies \#_{\rho}(a^{\triangleright}) < \infty .$$

b)  $\mathcal{S}$  is strongly  $B$ -communication stopping, if for all  $q \in \text{Post}^*(q^0)$ ,

$$\rho \in \text{wfrun}_{\mathcal{S}}(q) \text{ and } a \in B, \#_{\rho}(^{\triangleright}a) = 0 \implies \#_{\rho}(a^{\triangleright}) = 0 .$$

We say that an AIOTS  $\mathcal{S}$  has a channel property  $P$ , if  $\mathcal{S}$  has property  $P$  with respect to the set  $C$  of all channels of  $\mathcal{S}$ .

**Definition 13** (Channel properties of AIOPNs). *Let  $P$  be an arbitrary channel property as defined above. An AIOPN  $\mathcal{N}$  has property  $P$  w.r.t. some subset  $B$  of its channels if its generated AIOTS  $\text{aiots}(\mathcal{N})$  has property  $P$  w.r.t.  $B$ ;  $\mathcal{N}$  has property  $P$  if  $\text{aiots}(\mathcal{N})$  has property  $P$ .*



**Relevance of channel properties.** The generic properties that we have defined fit well with properties related to specific distributed mechanisms, algorithms and applications. For instance:

- When sending an email the user must be confident that its mail will be eventually read. Such a property can be formalized as the necessarily consuming property.
- Most distributed applications can be designed with an underlying token circulation between the processes of the applications. This requires that at any time there is at most one token in all channels and that this token can be immediately handled. Such a property can be formalized as the strong wholly emptying property.
- Recovery points are useful for applications prone to faults. While algorithms for building recovery points can handle non-empty channels, the existence (and identification) of states with empty channels eases this task. Such a property can be formalized as the necessarily wholly emptying property.
- In UNIX, one often requires that a process should not write in a socket when no reader of the socket is still present (and this could raise a signal). Such a property can be formalized as the strong communication stopping property.

## 4.2 Relationships Between Channel Properties

Table 1 shows relationships between the channel properties and pointers to examples of AIOPNs from Fig. 1 and Fig. 3 which have the indicated properties.

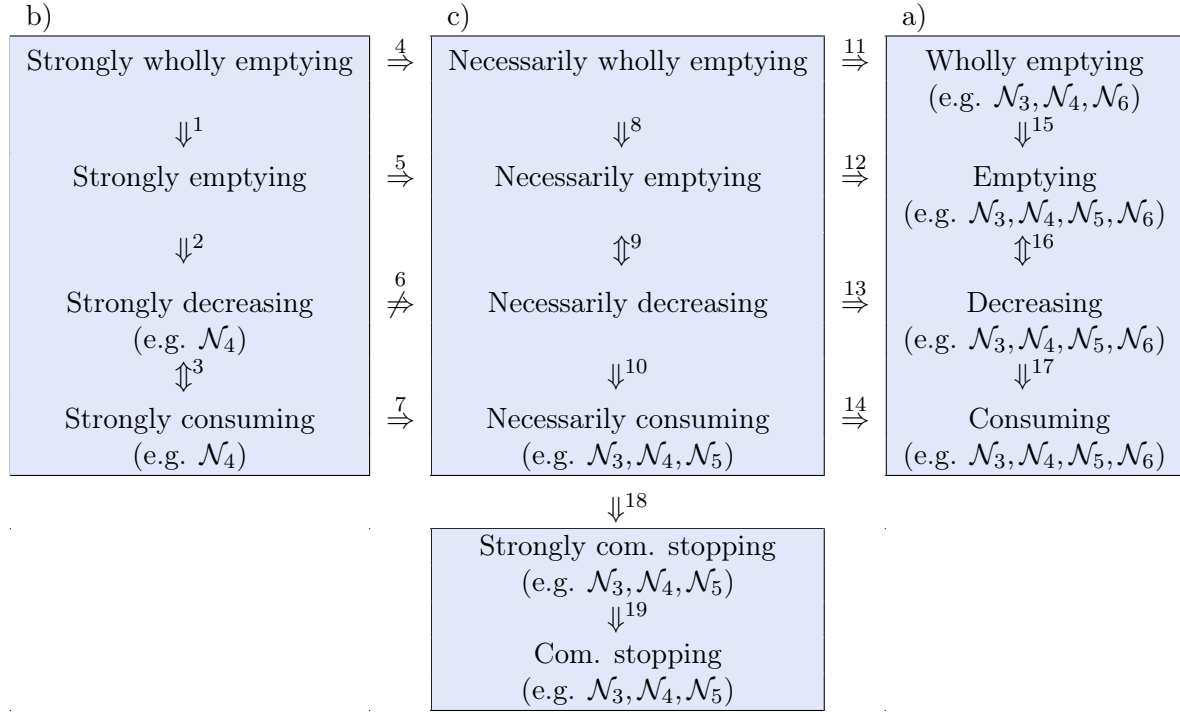
All the downward implications inside the boxes are direct consequences of the definitions. It is trivial to see that downward implication 3 is an equivalence, since *immediate* consumption leads to a decreasing valuation. Downward implications 9 and 16 are equivalences, since repeated decreasing of messages on a channel will eventually lead to an empty channel. The implications 4, 5 and 7 will be proved in Prop. 14, the implications 11-14 in Prop. 15, and implication 18 in Prop. 16. Additionally we have that all properties in box b) of Tab. 1 imply the strongest property in box a), since if  $\mathcal{S}$  is strongly  $B$ -consuming we can by repeated consumption empty all channels in  $B$ .

In the following propositions we assume given an AIOTS  $\mathcal{S} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$  and a subset  $B \subseteq C$ .

**Proposition 14.** *If  $\mathcal{S}$  is strongly  $B$ -wholly emptying (strongly  $B$ -emptying, strongly  $B$ -consuming resp.), then  $\mathcal{S}$  is necessarily  $B$ -wholly emptying (necessarily  $B$ -emptying, necessarily  $B$ -consuming resp.).*

*Proof.* We only prove that strongly consuming implies necessarily consuming. The other implications are simple extensions of this proof. Let  $q \in \text{Post}^*(q^0)$ ,  $a \in B$  such that  $\text{val}(q, a) > 0$ , and let  $\rho \in \text{run}_{\mathcal{S}}(q)$  be a weakly fair run. Assume for the contrary that  $\triangleright a \notin \rho$ . By definition of AIOTS we get that for all  $q' \in \rho$ ,  $\text{val}(q', a) \geq \text{val}(q, a)$ . By

## D. Channel Properties of Asynchronously Composed Petri Nets



**Table 1:** Relationships between channel properties and examples.

assumption  $\mathcal{S}$  is strongly  $B$ -consuming, which implies that  $q' \xrightarrow{\triangleright_a}$  for all  $q' \in \rho$ . This is a contradiction to  $\rho$  being a weakly fair run. ■

**Proposition 15.** *If  $\mathcal{S}$  is necessarily  $B$ -wholly emptying (necessarily  $B$ -emptying, necessarily decreasing, necessarily  $B$ -consuming resp.), then  $\mathcal{S}$  is  $B$ -wholly emptying ( $B$ -emptying,  $B$ -decreasing,  $B$ -consuming resp.).*

*Proof.* The proof relies on the fact that for each  $q \in \text{Post}^*(q^0)$  there exists a weakly fair run  $\rho \in \text{wfruns}_{\mathcal{S}}(q)$ , such that for all  $a \in \text{in}$ ,  $a \notin \rho$ . This run can be constructed by choosing, in a weakly fair manner in each reached state, some enabled non-input action. If no such action is enabled in the last visited state, the last state is a pure input state and we are done. Otherwise the resulting infinite run has the required property.

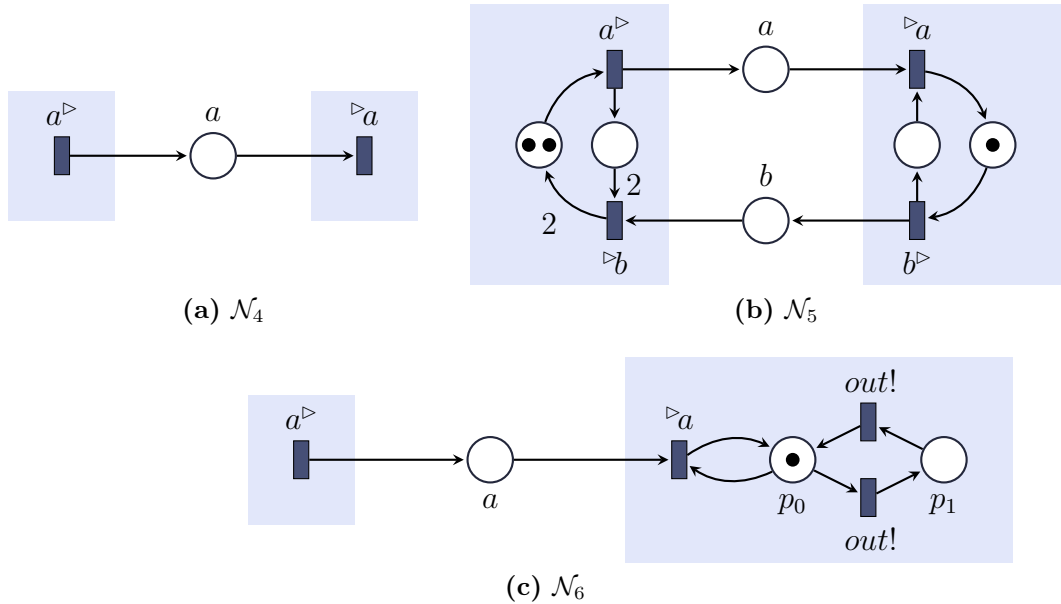
With this fact, we can prove the implication 14 in Tab. 1 as follows: Let  $q \in \text{Post}^*(q^0)$ ,  $a \in B$  such that  $\text{val}(q, a) > 0$ . By necessarily consuming, for all  $\rho \in \text{wfruns}_{\mathcal{S}}(q)$  we have  $\triangleright_a \in \rho$ . Since we know there exists a weakly fair run  $\rho$  without input actions, we get that there exists  $q' \in \text{Post}^*(q, \Sigma \setminus \text{in})$  such that  $q' \xrightarrow{\triangleright_a}$ . The other implications are proven in the same way. ■

**Proposition 16.** *If  $\mathcal{S}$  is necessarily  $B$ -consuming then  $\mathcal{S}$  is strongly  $B$ -communication stopping.*

*Proof.* Assume the contrary, that  $\mathcal{S}$  is not strongly  $B$ -communication stopping. This means that there exists  $q \in \text{Post}^*(q^0)$ ,  $a \in B$  and a weakly fair run  $\rho \in \text{wfrun}_{\mathcal{S}}(q)$  such that  $\sharp_{\rho}(\triangleright a) = 0$  and  $\sharp_{\rho}(a^{\triangleright}) > 0$ . Then there exists  $q' \in \rho$  reached after  $q$ , such that  $\text{val}(q', a) > 0$ . Let  $\rho'$  be the suffix of  $\rho$  starting from  $q'$  which is weakly fair as well. Since  $\sharp_{\rho'}(\triangleright a) = 0$ ,  $\mathcal{S}$  is not necessarily consuming. ■

Let us now discuss some counterexamples. As discussed in Sect. 4.1, a counterexample for the converse of implication 7 is the AIOPN  $\mathcal{N}_3$  in Fig. 1c on page 106. An obvious counterexample for the converse of the implications 2, 10, 11, 12, 13 is given by the AIOPN  $\mathcal{N}_4$  shown in Fig. 3a.  $\mathcal{N}_4$  is also a counterexample for implication 6. The AIOPN  $\mathcal{N}_5$  in Fig. 3b with channels  $a$  and  $b$  is a counterexample for the converse of implication 15. The net can empty each single channel  $a$  and  $b$  but it can never have both channels empty at the same time (after the first message has been produced on a channel). A counterexample for the converse of implication 14 is shown by the net  $\mathcal{N}_6$  in Fig. 3c. The net can put a token on the channel  $a$ , but afterwards the transition  $\triangleright a$  is not necessarily always enabled which means there exists a weakly fair run such that there is always a token in  $a$  and  $\triangleright a$  is never fired.

Counterexamples for the converse of implication 17 rely on the idea to produce twice while consuming once. A counterexample for the converse of implication 18 is provided by a net that first produces a finite number  $n$  of messages on a channel, then it consumes less than  $n$  of these messages and then it stops. Counterexamples for the remaining converse implications are straightforward to construct.



**Figure 3:** Examples of AIOPNs.

### 4.3 Compositionality of Channel Properties

Modular verification of systems is an important goal in any development method. In our context this concerns the question whether channel properties are preserved in arbitrary environments or, more precisely, whether they are preserved under asynchronous composition. In this section we show that indeed all channel properties defined above are compositional. This can be utilised to get a method for incremental design.

In order to relate channel properties of asynchronous compositions to channel properties of their constituent parts we need the next two lemmas. The first one shows that autonomous executions of constituent parts (not involving inputs) can be lifted to executions of compositions. This is the essence to prove compositionality of the properties of type (a) and type (b) in Def. 11.

**Lemma 17.** *Let  $\mathcal{S} = (C_{\mathcal{S}}, \Sigma_{\mathcal{S}}, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \longrightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}})$ ,  $\mathcal{T} = (C_{\mathcal{T}}, \Sigma_{\mathcal{T}}, Q_{\mathcal{T}}, q_{\mathcal{T}}^0, \longrightarrow_{\mathcal{T}}, \text{val}_{\mathcal{T}})$  be two composable AIOTSSs, and let  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$ . For all  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in \text{Post}^*(q^0)$  and  $\sigma \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})^*$  it holds that*

$$q_{\mathcal{S}} \xrightarrow{\sigma}_{\mathcal{S}} q'_{\mathcal{S}} \implies \exists \vec{v}' . (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xrightarrow{\bar{\sigma}} (q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}'),$$

*with  $\bar{\sigma} \in (\Sigma \setminus \text{in})^*$  obtained from  $\sigma$  by replacing any occurrence of a shared label  $a \in \text{out}_{\mathcal{S}} \cap \text{in}_{\mathcal{T}}$  by the communication label  $a^{\triangleright}$ .*

*Proof.* Obviously it is sufficient to show the claim for an arbitrary  $a \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$ . The general result then follows by induction on the length of  $\sigma$ .

1.  $a \in \text{out}_{\mathcal{S}} \cap \text{in}_{\mathcal{T}}$ : By rule (3.1) in Def. 9 it follows that there exists  $\vec{v}'$  such that  $q_{\mathcal{S}} \xrightarrow{a}_{\mathcal{S}} q'_{\mathcal{S}} \implies (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xrightarrow{a^{\triangleright}} (q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}')$ .
2.  $a \notin \text{out}_{\mathcal{S}} \cap \text{in}_{\mathcal{T}}$ : There are two subcases, either  $a \in \text{com}_{\mathcal{S}}$ , or  $a \in \text{out}_{\mathcal{S}} \setminus \text{in}_{\mathcal{T}}$ . In both cases we get from rule (1) in Def. 9,  $q_{\mathcal{S}} \xrightarrow{a}_{\mathcal{S}} q'_{\mathcal{S}} \implies (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xrightarrow{a} (q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v})$ .

■

The next lemma is crucial to prove compositionality of the “necessarily” properties of type (c) in Def. 11 and the communication stopping properties in Def. 12. It shows that projections of weakly fair runs are weakly fair runs again. This result can only be achieved in the asynchronous context.

**Lemma 18.** *Let  $\mathcal{S}, \mathcal{T}$  be two composable AIOTSSs, and  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$ . Let  $q = (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in Q$  and  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}(q)$  be a weakly fair run. Then  $\rho|_{\mathcal{S}} \in \text{wfrun}_{\mathcal{S}}(q_{\mathcal{S}})$ , is a weakly fair run.*

*Proof.* Let  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}(q)$ . First we show that  $\rho|_{\mathcal{S}}$  is weakly fair as well. If  $\rho|_{\mathcal{S}}$  is finite it is weakly fair. If  $\rho|_{\mathcal{S}}$  is infinite, then assume that an autonomous action  $a \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$

is always enabled in  $\rho|_S$  from a certain state on. Due to the asynchronous composition we get that  $a$  is also always enabled in  $\rho$  from a certain state on. Hence  $a$  will be infinitely often executed in  $\rho$  and therefore also in  $\rho|_S$ .

It remains to prove that  $\rho|_S \in \text{runs}_S(q_S)$ . There are two main cases. Either  $\rho$  is finite or infinite. Assume  $\rho$  is finite with its last state being  $q' = (q'_S, q'_T, \vec{v}')$ . Clearly  $q'_S$  must be the last state of  $\rho|_S$ . By definition of a run,  $q'$  is a pure input state. Then  $q'_S$  must be a pure input state of  $S$ , hence  $\rho|_S \in \text{runs}_S(q_S)$ .

Now assume that  $\rho$  is infinite and weakly fair. In this case there are two subcases. Either  $\rho|_S$  is finite or infinite. If  $\rho|_S$  is infinite, then  $\rho|_S \in \text{runs}_S(q_S)$ .

Assume  $\rho|_S$  is finite such that its last state is  $q'_S$ , with  $(q'_S, q'_T, \vec{v}') \in \rho$ . Since  $\rho$  is weakly fair we can prove that  $q'_S$  is a pure input state as follows. Assume the contrary, that  $q'_S$  is not a pure input state, i.e. there exists  $a \in (\Sigma \setminus \text{in})$  such that  $q'_S \xrightarrow{a}$ . As  $\rho$  is infinite we get that  $a$  is enabled always from  $(q'_S, q'_T, \vec{v}') \in \rho$ , which is a contradiction, since  $\rho$  is fair and  $a$  is not occurring infinitely often after  $(q'_S, q'_T, \vec{v}') \in \rho$  as  $\rho|_S$  is finite. Now, knowing that  $q'_S$  is a pure input state, we get that  $\rho|_S \in \text{runs}_S(q_S)$ . ■

**Proposition 19** (Compositionality of Channel Properties). *Let  $S$  and  $T$  be two composable AIOTs such that  $C_S$  is the set of channels of  $S$ . Let  $B \subseteq C_S$  and let  $P$  be an arbitrary channel property as defined in Sec. 4.1. If  $S$  has property  $P$  with respect to the channels  $B$ , then  $S \otimes T$  has property  $P$  with respect to the channels  $B$ . This holds analogously for asynchronous I/O-Petri nets (due to the compositional semantics of AIOPNs; see Thm. 10).*

*Proof.* Let  $S = (C_S, \Sigma_S, Q_S, q_S^0, \longrightarrow_S, \text{val}_S)$ ,  $T = (C_T, \Sigma_T, Q_T, q_T^0, \longrightarrow_T, \text{val}_T)$ ,  $S \otimes T = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$  with  $\Sigma_S = \text{in}_S \uplus \text{out}_S \uplus \text{com}_S$  and  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$ . We split this proof into the following three parts: “non-necessarily” channel properties of types (a) and (b) in Def. 11, necessarily channel properties of type (c) in Def. 11, and communication stopping properties in Def. 12.

**Non-necessarily channel properties:** Before we prove each case we state two simple consequences of Lem. 17: For all  $(q_S, q_T, \vec{v}) \in \text{Post}^*(q^0)$  and  $q'_S \in Q_S$ ,

$$(1) \quad q'_S \in \text{Post}^*(q_S, \Sigma_S \setminus \text{in}_S) \Rightarrow \exists \vec{v}'. (q'_S, q_T, \vec{v}') \in \text{Post}^*((q_S, q_T, \vec{v}), \Sigma \setminus \text{in})$$

$$(2) \quad q'_S \in \text{Post}(q_S, \Sigma_S \setminus \text{in}_S) \Rightarrow \exists \vec{v}'. (q'_S, q_T, \vec{v}') \in \text{Post}((q_S, q_T, \vec{v}), \Sigma \setminus \text{in})$$

For the proof of the P1 - P3 properties let  $(q_S, q_T, \vec{v}) \in \text{Post}^*(q^0)$  and  $a \in B$  such that  $\text{val}((q_S, q_T, \vec{v}), a) > 0$ .

P1.a: Assume that  $S$  is  $B$ -consuming. Obviously  $q_S \in \text{Post}^*(q_S^0)$ . By assumption there exists  $q'_S \in \text{Post}^*(q_S, \Sigma_S \setminus \text{in}_S)$  such that  $q'_S \xrightarrow{a}_S$ . By (1) above there exists  $\vec{v}'$  such that  $(q'_S, q_T, \vec{v}') \in \text{Post}^*((q_S, q_T, \vec{v}), \Sigma \setminus \text{in})$ , and by definition of  $\otimes$  we get  $(q'_S, q_T, \vec{v}') \xrightarrow{a}$ .

## D. Channel Properties of Asynchronously Composed Petri Nets

P1.b: If  $\mathcal{S}$  is strongly  $B$ -consuming, we get by assumption that  $q_{\mathcal{S}} \xrightarrow{\triangleright a}$ . By definition of  $\otimes$  this means that  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xrightarrow{\triangleright a}$ .

P2.a: Assume that  $\mathcal{S}$  is  $B$ -decreasing. Obviously  $q_{\mathcal{S}} \in \text{Post}^*(q_{\mathcal{S}}^0)$ . By assumption there exists  $q'_{\mathcal{S}} \in \text{Post}^*(q_{\mathcal{S}}, \Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$  such that  $\text{val}_{\mathcal{S}}(q'_{\mathcal{S}}, a) < \text{val}_{\mathcal{S}}(q_{\mathcal{S}}, a)$ . By (1) there exists  $v'$  such that  $(q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}') \in \text{Post}^*((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), \Sigma \setminus \text{in})$ . Finally as  $a \in C_{\mathcal{S}}$  and  $\text{val}_{\mathcal{S}}(q'_{\mathcal{S}}, a) < \text{val}_{\mathcal{S}}(q_{\mathcal{S}}, a)$  we get, by definition of  $\text{val}$ , that  $\text{val}((q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}'), a) < \text{val}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), a)$ .

P2.b: If  $\mathcal{S}$  is strongly  $B$ -decreasing the claim follows by the same reasoning using (2) from above.

P3.a: The proof is analogous to the case P2.a.

P3.b: The proof is analogous to the case P2.b.

For the proof of P4 let  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in \text{Post}^*(q^0)$  such that  $\text{val}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), B) > 0$ .

P4.a: Assume that  $\mathcal{S}$  is  $B$ -wholly emptying. Obviously  $q_{\mathcal{S}} \in \text{Post}^*(q_{\mathcal{S}}^0)$ . By assumption there exists  $q'_{\mathcal{S}} \in \text{Post}^*(q_{\mathcal{S}}, \Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$  such that  $\text{val}_{\mathcal{S}}(q'_{\mathcal{S}}, B) = 0$ . By (1) there exists  $\vec{v}'$  such that  $(q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}') \in \text{Post}^*((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), \Sigma \setminus \text{in})$ . Finally as  $B \subseteq C_{\mathcal{S}}$  we get, by definition of  $\text{val}$ , that  $\text{val}((q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}'), B) = \text{val}_{\mathcal{S}}(q'_{\mathcal{S}}, B) = 0$ .

P4.b: If  $\mathcal{S}$  is strongly  $B$ -wholly emptying the claim follows by the same reasoning using (2) from above.

**Necessarily channel properties:** We will only prove the claim for property (P1.c), as it can be proven analogously for the other necessarily properties.

Assume that  $\mathcal{S}$  is necessarily  $B$ -consuming. Let  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$ ,  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in \text{Post}^*(q^0)$  and  $a \in B$ , such that  $\text{val}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), a) > 0$ . Let  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}))$  be weakly fair. By Lem. 18 we get that  $\rho|_{\mathcal{S}}$  is a weakly fair run of  $\mathcal{S}$ . By assumption we know  $\triangleright a \in \rho|_{\mathcal{S}}$  and hence it follows that  $\triangleright a \in \rho$ .

**Communication stopping properties:** We will only prove the claim for property (P5.a), it can be proven analogously for (P5.b).

Assume that  $\mathcal{S}$  is  $B$ -communication stopping. Let  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \longrightarrow, \text{val})$ ,  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in \text{Post}^*(q^0)$ ,  $a \in B$  and  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}))$ , such that  $\sharp_{\rho}(\triangleright a) = 0$ . By Lem. 18 we get that  $\rho|_{\mathcal{S}} \in \text{wfrun}_{\mathcal{S}}(q_{\mathcal{S}})$ . By assumption we know that  $\sharp_{\rho|_{\mathcal{S}}}(\triangleright a) < \infty$ . Since  $\triangleright a$  is triggered by  $\mathcal{S}$  in  $\mathcal{S} \otimes \mathcal{T}$ , we get that  $\sharp_{\rho}(\triangleright a) < \infty$ .  $\blacksquare$

Proposition 19 leads to the desired modular verification result for all properties except wholly emptying (P4): In order to check that a composition  $\mathcal{N} \otimes_{pn} \mathcal{M}$  of two AIOPNs has a channel property  $P$ , i.e.  $P$  holds for all channels of the composition, it is sufficient to know that  $\mathcal{N}$  and  $\mathcal{M}$  have property  $P$  and to prove that  $\mathcal{N} \otimes_{pn} \mathcal{M}$  has property  $P$  with respect to the new channels introduced by the asynchronous composition.

**Theorem 20** (Incremental Design). *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two composable AIOPNs with shared actions  $\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}$  and let  $P$  be an arbitrary channel property but  $(P_4)$ . If both  $\mathcal{N}$  and  $\mathcal{M}$  have property  $P$  and if  $\mathcal{N} \otimes_{pn} \mathcal{M}$  has property  $P$  with respect to the new channels  $\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}$ , then  $\mathcal{N} \otimes_{pn} \mathcal{M}$  has property  $P$ .*

## 5 Decidability of Channel Properties

We begin this section by recalling some information related to semi-linear sets and decision procedures in Petri nets that we use in our proofs.

Let  $E \subseteq \mathbb{N}^k$ ,  $E$  is a *linear set* if there exists a finite set of vectors of  $\mathbb{N}^k$   $\{v_0, \dots, v_n\}$  such that  $E = \{v_0 + \sum_{1 \leq i \leq n} \lambda_i v_i \mid \forall i \lambda_i \in \mathbb{N}\}$ . A *semi-linear set* [GS66] is a finite union of linear sets; a representation of it is given by the family of finite sets of vectors defining the corresponding linear sets. Semi-linear sets are *effectively* closed w.r.t. union, intersection and complementation. This means that one can compute a representation of the union, intersection and complementation starting from a representation of the original semi-linear sets.  $E$  is an *upward closed set* if  $\forall v \in E. v' \geq v \Rightarrow v' \in E$ . An upward closed set has a finite set of minimal vectors denoted  $\min(E)$ . An upward closed set is a semi-linear set which has a representation that can be derived from the equation  $E = \min(E) + \mathbb{N}^k$  if  $\min(E)$  is computable.

Given a Petri net  $\mathcal{N}$  and a marking  $m$ , the *reachability* problem consists in deciding whether  $m$  is reachable from  $m_0$  in  $\mathcal{N}$ . This problem is decidable [May81] but none of the associated algorithms are primitive recursive. Furthermore this procedure can be adapted to semi-linear sets when markings are identified to vectors of  $\mathbb{N}^{|P|}$ . Based on reachability analysis, the authors of [EJ89] design an algorithm that decides whether a marking  $m$  is a *home state*, i.e.  $m$  is reachable from any reachable marking. A more general problem is in fact decidable: given a subset of places  $P'$  and a (sub)marking  $m$  on this subset, is it possible from any reachable marking to reach a marking that coincides on  $P'$  with  $m$ ?

In [Rac78], the *coverability* is shown to be EXPSPACE-complete. The coverability problem consists in determining, given a net and a target marking, whether one can reach a marking greater or equal than the target. In [VJ85] given a Petri net, several procedures have been designed to compute the minimal set of markings of several interesting upward closed sets. In particular, given an upward closed set *Target*, by a backward analysis one can compute the (representation of) upward closed set from which *Target* is reachable. Using the results of [Rac78], this algorithm performs in EXPSPACE.

While in Petri nets, strong fairness is undecidable [Car87], weak fairness is decidable and more generally, the existence of an infinite sequence fulfilling a formula of the following fragment of LTL is decidable [Jan90]. The literals are (1) comparisons between places markings and values, (2) transition firings and (3) their negations. Formulas are inductively defined as literals, conjunction or disjunction of formulas and  $GF\varphi$  where  $GF$  is the infinitely often operator and  $\varphi$  is a formula.



## D. Channel Properties of Asynchronously Composed Petri Nets

The next theorem establishes the decidability of the strong properties of type (b) of Def. 11. Observe that their proofs are closely related and that they rely on the decidability of reachability and coverability problems.

**Theorem 21.** *The following problems are decidable for AIOPNs: Is an AIOPN  $\mathcal{N}$  strongly  $B$ -consuming, strongly  $B$ -decreasing, strongly  $B$ -emptying, strongly  $B$ -wholly emptying?*

*Proof. Strongly  $B$ -consuming (and strongly  $B$ -decreasing).* Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is strongly  $B$ -consuming as follows. Let  $E$  be the set of markings defined by:

$$E = \{m \mid \exists a \in B \ m(a) > 0 \text{ and } \forall t \in T \text{ with } \lambda(t) = {}^{\triangleright}a \ m \not\geq W^-(t)\}$$

$\mathcal{N}$  is strongly  $B$ -consuming iff  $E$  is not reachable. Since  $E$  is a semi-linear set, its reachability is decidable.

**Strongly  $B$ -emptying.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is strongly  $B$ -emptying as follows.

First by coverability analysis, one decides whether the following upward closed subset  $E$  is reachable:

$$E = \{m \mid \exists a \in B \ m(a) > 1\}.$$

If  $E$  is reachable then  $\mathcal{N}$  is not strongly  $B$ -emptying. Otherwise one checks whether  $\mathcal{N}$  is strongly  $B$ -consuming in order to decide.

**Strongly  $B$ -wholly emptying.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is strongly  $B$ -wholly emptying as follows.

First by coverability analysis, one decides whether the following upward closed subset  $E$  is reachable:

$$E = \{m \mid m(B) > 1\}.$$

If  $E$  is reachable then  $\mathcal{N}$  is not strongly  $B$ -wholly emptying. Otherwise one checks whether  $\mathcal{N}$  is strongly  $B$ -consuming in order to decide. ■

The next theorem establishes the decidability of the properties of type (a) of Def. 11. Observe that their proofs rely on (1) the effectiveness of backward analysis for upward closed marking sets (2) the decidability of reachability and home space problems and (3) appropriate transformations of the net.

**Theorem 22.** *The following problems are decidable for AIOPNs: Is an AIOPN  $\mathcal{N}$   $B$ -consuming,  $B$ -decreasing,  $B$ -emptying,  $B$ -wholly emptying?*



*Proof.*

**$B$ -consuming.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is  $B$ -consuming as follows.

Let  $a \in B$  and  $E_a$  be the upward closed set of markings defined by:

$$E_a = \{m \mid \exists t \in T \text{ with } \lambda(t) = \triangleright a \text{ and } m \geq W^-(t)\}$$

$E_a$  is the set of markings from which one can immediately consume some message  $a$ . Let  $F_a$  be the upward closed set of markings defined by:

$$F_a = \{m \mid \exists m' \in E_a \exists \sigma \in T^*. \lambda(\sigma) \in (\Sigma \setminus \text{in})^* \wedge m \xrightarrow{\sigma} m'\}$$

$F_a$  is the set of markings from which one can later (without the help of the environment) consume some message  $a$ . One computes  $F_a$  by backward analysis. Let  $G$  be defined by:

$$G = \{m \mid \exists a \in B. m(a) > 0 \wedge m \notin F_a\}$$

$G$  is a semi-linear set corresponding to the markings from which some message  $a \in B$  will never be consumed. Then  $\mathcal{N}$  is not  $B$ -consuming iff  $G$  is reachable.

**$B$ -emptying (and  $B$ -decreasing).** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is  $B$ -emptying as follows. First one builds a net  $\mathcal{N}'$ :

- $P' = P \uplus \{\text{run}\}$
- $T' = T \uplus \{\text{stop}\}$
- $\forall p \in P \forall t \in T \ W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m'^0(p) = m^0(p)$
- $W'^-(\text{run}, \text{stop}) = 1, W'^+(\text{run}, \text{stop}) = 0, m'^0(\text{run}) = 1$
- $\forall p \in P \ W'^-(p, \text{stop}) = W'^+(p, \text{stop}) = 0$
- $\forall t \in T \text{ such that } \lambda(t) \in \text{in} \ W'^-(\text{run}, t) = W'^+(\text{run}, t) = 1$
- $\forall t \in T \text{ such that } \lambda(t) \notin \text{in} \ W'^-(\text{run}, t) = W'^+(\text{run}, t) = 0$

$\mathcal{N}'$  behaves as  $\mathcal{N}$  as long as  $\text{stop}$  is not fired. When  $\text{stop}$  is fired only transitions not labelled by inputs are fireable. Thus  $\mathcal{N}$  is  $B$ -emptying iff for all  $a \in B$  the set of markings  $Z_a = \{m \mid m(a) = 0\}$  is a home space for  $\mathcal{N}'$ .

**$B$ -wholly emptying.** Using the same construction  $\mathcal{N}$  is  $B$ -weakly wholly emptying if  $Z_B = \{m \mid m(B) = 0\}$  is a home space for  $\mathcal{N}'$ . ■

The next theorems, establish the decidability of the necessarily properties of type (c) of Def. 11 and the communication stopping properties.

**Theorem 23.** *The following problems are decidable for AIOPNs: Is an AIOPN  $\mathcal{N}$  necessarily  $B$ -consuming, necessarily  $B$ -decreasing, necessarily  $B$ -emptying, necessarily  $B$ -wholly emptying?*

*Proof. Necessarily  $B$ -consuming.* Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is necessarily  $B$ -consuming as follows.

First one checks whether  $\mathcal{N}$  is  $B$ -consuming, a necessary condition for being necessarily  $B$ -consuming. If  $\mathcal{N}$  is  $B$ -consuming, then one checks whether for some  $a \in B$ , there exists a reachable marking  $m$  fulfilling  $m(a) > 0$  from which an infinite weakly fair run is fireable without occurrence of transitions labelled by  $\triangleright a$ . To perform this test, one builds a net  $\mathcal{N}'$  as follows.

- $P' = P \uplus \{run\}$
- $T' = T \uplus \{stop\}$
- $\forall p \in P \forall t \in T \ W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m'^0(p) = m^0(p)$
- $W'^-(run, stop) = 1, W'^+(run, stop) = 0, m'^0(run) = 1$
- $W'^-(a, stop) = W'^+(a, stop) = 1$
- $\forall p \in P \setminus \{a\} \ W'^-(p, stop) = W'^+(p, stop) = 0$
- $\forall t \in T \text{ such that } \lambda(t) = \triangleright a \ W'^-(run, t) = W'^+(run, t) = 1$
- $\forall t \in T \text{ such that } \lambda(t) \neq \triangleright a \ W'^-(run, t) = W'^+(run, t) = 0$

Transition *stop* can be fired only if  $m(a) > 0$ . When *stop* is fired only transitions not labelled by  $\triangleright a$  are fireable. Then one checks whether there exists an infinite weakly fair run that fulfils formula  $GFrun = 0 \wedge GF\text{-fireable}(\triangleright a)$  in  $\mathcal{N}'$ . Since  $\text{fireable}(\triangleright a)$  is expressible by a boolean combination of comparisons of place markings with constants, the formula belongs to the logic of [Jan90]. The first term witnesses the firing of *stop* and the second term ensures that the sequence is also weakly fair in  $\mathcal{N}$ . Then  $\mathcal{N}$  is necessarily  $B$ -consuming iff there is no such run.

**Necessarily  $B$ -emptying (and  $B$ -decreasing).** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is necessarily  $B$ -emptying as follows.

First one checks whether  $\mathcal{N}$  is  $B$ -emptying, a necessary condition for being necessarily  $B$ -emptying. If  $\mathcal{N}$  is  $B$ -emptying, then one checks whether for some  $a \in B$ , there exists a reachable marking  $m$  from which an infinite weakly fair run is fireable such that for every marking  $m'$  visited,  $m'(a) > 0$ . To perform this test, one builds a net  $\mathcal{N}'$  as follows.

- $P' = P \uplus \{run\}$
- $T' = T \uplus \{stop\}$  with  $\lambda(stop) = stop$
- $\forall p \in P \forall t \in T \ W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m'^0(p) = m^0(p)$

- $W'^-(run, stop) = 1, W'^+(run, stop) = 0, m^0(run) = 1,$
- $W'^-(a, stop) = 1, W'^+(a, stop) = 0$
- $\forall p \in P \setminus \{a\} W'^-(p, stop) = W'^+(p, stop) = 0$
- $\forall t \in T W'^-(run, t) = W'^+(run, t) = 0$

Transition *stop* can be fired only if  $m(a) > 0$  and it consumes one token of  $a$ . Transition *stop* can be fired only once due to place *run*. Then one checks whether there exists an infinite weakly fair run that fulfils formula  $G\text{Frun} = 0$  (witnessing the firing of *stop*) in  $\mathcal{N}'$ . Then  $\mathcal{N}$  is necessarily  $B$ -emptying iff there is no such run. Indeed there is an infinite weakly fair run in  $\mathcal{N}'$  after the firing of *stop* iff from some marking  $m$  in  $\mathcal{N}$ , there is an infinite weakly fair run where the marking of  $a$  is never zero from some state.

**Necessarily  $B$ -wholly emptying.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is necessarily  $B$ -wholly emptying as follows.

First one checks whether  $\mathcal{N}$  is  $B$ -wholly emptying, a necessary condition for being necessarily  $B$ -wholly emptying. If  $\mathcal{N}$  is  $B$ -wholly emptying, then one checks whether there exists a reachable marking  $m$  from which an infinite weakly fair run is fireable such that for every marking  $m'$  visited,  $m'(B) > 0$ . To perform this test, one builds a net  $\mathcal{N}'$  as follows.

- $P' = P \uplus \{run, B\}$
- $T' = T \uplus \{stop\}$  with  $\lambda(stop) = stop$
- $\forall p \in P \forall t \in T W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m^0(p) = m^0(p)$
- $\forall t \in T W'^-(B, t) = \sum_{a \in B} W^-(a, t), W'^+(B, t) = \sum_{a \in B} W^+(a, t), m^0(B) = 0$
- $W'^-(run, stop) = 1, W'^+(run, stop) = 0, m^0(run) = 1,$
- $W'^-(B, stop) = 1, W'^+(B, stop) = 0$
- $\forall p \in P W'^-(p, stop) = W'^+(p, stop) = 0$
- $\forall t \in T W'^-(run, t) = W'^+(run, t) = 0$

In  $\mathcal{N}'$  there is an additional place  $B$  containing the sum of tokens of places  $a \in B$  whose management is straightforward. As in the previous constructions, there is a control transition *stop* that modifies the behaviour of  $\mathcal{N}$ . As long as *stop* is not fired,  $\mathcal{N}'$  behaves as  $\mathcal{N}$ . In order to fire *stop* (which can be done only once),  $B$  is decreased. Then one checks whether there exists an infinite weakly fair run that fulfils formula  $G\text{Frun} = 0$  (witnessing the firing of *stop*) in  $\mathcal{N}'$ .  $\mathcal{N}$  is necessarily  $B$ -wholly emptying iff there is no such run. ■

**Theorem 24.** *The following problems are decidable for AIOPNs: Is an AIOPN  $\mathcal{N}$   $B$ -communication stopping, strongly  $B$ -communication stopping?*

*Proof.*  **$B$ -communication stopping.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is  $B$ -communication stopping as follows. Observe that this property is expressed by the event-based LTL formula  $\varphi = \bigwedge_{a \in B} GFa^\triangleright \Rightarrow GF^\triangleright a$ . By definition, this property is satisfied for finite runs. However  $\neg\varphi$  is not a formula of the fragment of [Jan90]. In order to check whether there exists a weakly fair infinite run satisfying the formula  $\neg\varphi$ , one builds net  $\mathcal{N}_a$  as follows.

- $P' = P \uplus \{run\}$
- $T' = T \uplus \{stop\}$
- $\forall p \in P \forall t \in T \ W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m'^0(p) = m^0(p)$
- $W'^-(run, stop) = 1, W'^+(run, stop) = 0, m'^0(run) = 1$
- $\forall p \in P \ W'^-(p, stop) = W'^+(p, stop) = 0$
- $\forall t \in T \text{ such that } \lambda(t) = \triangleright a \ W'^-(run, t) = W'^+(run, t) = 1$
- $\forall t \in T \text{ such that } \lambda(t) \neq \triangleright a \ W'^-(run, t) = W'^+(run, t) = 0$

$\mathcal{N}_a$  behaves as  $\mathcal{N}$  as long as  $stop$  is not fired. Transition  $stop$  can be fired only once. When  $stop$  is fired only transitions not labelled by  $\triangleright a$  are fireable. Then one checks whether there exists an infinite weakly fair run that fulfils formula  $GFr_{run} = 0 \wedge GFa^\triangleright$  in  $\mathcal{N}'$ . Then  $\mathcal{N}$  is  $B$ -communication stopping iff there is no such run in any  $\mathcal{N}_a$ .

**Strongly  $B$ -communication stopping.** Given an AIOPN  $\mathcal{N}$  and  $B$  a subset of its channels, one decides whether  $\mathcal{N}$  is strongly  $B$ -communication stopping as follows.

For all  $a \in B$ , one builds a net  $\mathcal{N}_a$  starting from  $\mathcal{N}$ . Since the formal specification of  $\mathcal{N}_a$  is cumbersome, one describes it in words.

- $\mathcal{N}_a$  has two additional places  $run$  and  $wit$ , with  $m'^0(run) = 1, m'^0(wit) = 0$ .
- Place  $run$  loops around all transitions labelled by  $\triangleright a$ .
- Every transition  $t$  labelled by  $a^\triangleright$  has a copy  $t'$  with the same inputs and outputs, plus an additional input  $run$  and an additional output  $wit$ .

$\mathcal{N}_a$  behaves as  $\mathcal{N}$  until a transition  $t'$  is fired. This firing is possible only once. Once a transition  $t'$  is fired place  $wit$  is marked and transitions labelled by  $\triangleright a$  are no more fireable. Define the semi-linear set  $E_a$  by:

$$E_a = \{m \mid m(wit) = 1 \wedge \forall t \in T. \lambda(t) \notin \text{in} \implies m \not\geq W^-(t)\}$$

Observe that a marking of  $E_a$  is a pure input state of the original net (i.e. when restricted to  $P$ ) that has been reached by a finite run when an occurrence of  $a^\triangleright$  has not been followed later by an occurrence of  $\triangleright a$ .

So  $\mathcal{N}$  is strongly  $B$ -communication stopping iff for all  $a \in B$ , in  $\mathcal{N}_a$  there does not exist an infinite weakly fair run fulfilling  $G\text{Fair} = 0$  (witnessing a *bad* weakly fair infinite run in  $\mathcal{N}$ ) and  $E_a$  is not reachable (witnessing a *bad* finite run in  $\mathcal{N}$ ). ■

## 6 Conclusion and Future Work

We have introduced asynchronously composed I/O-Petri nets and we have studied various properties of their communication channels based on a transition system semantics. Useful links between the channel properties are established. We have shown that the channel properties are compositional thus supporting incremental design. Moreover we have shown that the channel properties for AIOPNs are decidable. This work can be extended in at least three directions. The first direction would introduce new operations on AIOPNs, like hiding, to design component systems in a hierarchical way by encapsulating subsystems. The second direction concerns more general communication schemes like broadcasting. Finally, we want to establish conditions for the preservation of channel properties along the “vertical axis” namely by refinement, in particular within the framework of modal Petri nets as considered in [EHH12].

**Acknowledgement** We would like to thank the reviewers of the submitted version of this paper for many useful comments which led to a major restructuring of the paper.





# Specification of Asynchronous Component Systems with Modal I/O-Petri Nets

Serge HADDAD

*LSV, ENS Cachan & CNRS & INRIA, France*

Rolf HENNICKER

*Institut für Informatik, Ludwig-Maximilians-Universität München, Germany*

Mikael H. MØLLER

*Aalborg University, Department of Computer Science, Denmark*

**Abstract** Modal transition systems are an elegant way to formalise the design process of a system through refinement and composition. Here we propose to adapt this methodology to asynchronous composition via Petri nets. The Petri nets that we consider have distinguished labels for inputs, outputs, internal communications and silent actions and “must” and “may” modalities for transitions. The input/output labels show the interaction capabilities of a net to the outside used to build larger nets by asynchronous composition via communication channels. The modalities express constraints for Petri net refinement taking into account observational abstraction from silent transitions. Modal I/O-Petri nets are equipped with a modal transition system semantics. We show that refinement is preserved by asynchronous composition and by hiding of communication channels. We study compatibility properties which express communication requirements for composed systems and we show that these properties are decidable, they are preserved in larger contexts and also by modal refinement. On this basis we propose a methodology for the specification of distributed systems in terms of modal I/O-Petri nets which supports incremental design, encapsulation of components, step-wise refinement and independent implementability.

# 1 Introduction

Component-based design is an important field in software engineering. Crucial tasks in the design process concern the stepwise refinement of specifications towards implementations and the formation of component assemblies by composition. Many approaches and formalisms have been proposed for rigorous component-based design supporting different communication styles and different notions of refinement. Among them particular attention has been attracted by modal transition systems introduced by Larsen and Thomsen in 1988 [LT88] which use distinguished may- and must-transitions to specify allowed and obligatory behaviour and thus provide a flexible basis for refinement. While refinement concerns the vertical dimension of system development, composition concerns the horizontal dimension in which larger systems are built from smaller ones such that communication requirements must be respected. Communication properties are important when reasoning about distributed mechanisms, algorithms and applications (e.g. management of sockets in UNIX, maintaining unicity of a token in a ring based algorithm, guarantee of email reading, etc.).

Petri nets are a natural model for the design of concurrent and distributed systems. They have received a great attention w.r.t. the composition and refinement issues including communication properties. Composition of nets has been addressed via several paradigms. The process algebra approach has been investigated by several works leading to the Petri net algebra [BDK01]. Such an approach is closely related to synchronous composition. In [Sou91] and [SM91] asynchronous composition of nets is performed via a set of places or, more generally, via a subnet modelling some medium. Then structural restrictions on the subnets are proposed in order to preserve global properties like liveness or deadlock-freeness. In [Rei09] a general composition operator is proposed and its associativity is established. A closely related concept to composition is the one of open Petri nets which has been used in different contexts like the analysis of web services [SW09]. In parallel, very early works have been done for defining and analyzing refinement of nets; see [BGV91] for a survey. Looking at more recent works, [SV07] (which is the closest to our contribution) studies the refinement in the context of circuit design. In [SV07] the authors introduce the notion of a correct implementation and show that it is compositional. Several works also use an abstraction/refinement paradigm to propose efficient verification methods; see e.g. [GRB07].

In our contribution we want to combine the advantages of modal transition systems with the ability of Petri nets to specify infinite state systems, with their decidability potential and with their way how asynchronous composition can be achieved. A natural candidate are *modal Petri nets* introduced in [EHH12] (and later in [BK12] as a special case of Modal Process Rewrite Systems) which studies modal refinement and decidability results. Surprisingly, to the best of our knowledge, no other approaches to modal Petri nets exist yet. On the other hand, for asynchronous communication, we have recently introduced in [HHM13a] *asynchronous I/O-Petri nets*, for which we have analysed several communication properties from the compositionality and decidability point of view. Hence it is an obvious goal to combine, adjust and extend the results achieved in [EHH12] and [HHM13a] to a rigorous design methodology that supports

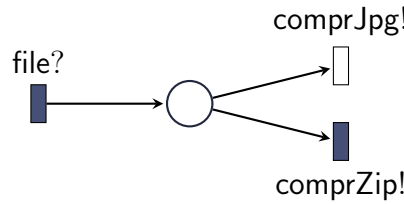


the vertical and the horizontal dimension of software development in a uniform and compatible way.

We summarise our proposal by means of an illustrating example in Sect. 2. Sect. 3 presents the underlying formal definitions of modal asynchronous I/O–Petri nets (MAIOPNs) and their semantics in terms of modal asynchronous I/O–transition systems (MAIOTSs). In Sect. 4 we consider modal refinement and, in Sect. 5, we study generic communication properties for components which interact via unordered and unbounded communication channels. We focus on the *message consuming* and the *necessarily message consuming* properties which are important requirements to ensure that previously sent messages can be consumed by the communication partner or must necessarily be consumed on all observationally weakly fair runs. The latter is a significant new concept which is the basis for the preservation of the necessarily consuming property when components are (1) put in larger contexts or (2) are refined by modal refinement. We show that the principle of incremental design in [HHM13a] extends to the generalised channel properties and that the decidability results carry over to the new integrated framework. We also study compositionality of modal refinement w.r.t. asynchronous composition and show that our framework supports the principle of independent implementability in the sense of [AH05]. Our design methodology is complemented by an operator for channel hiding which supports encapsulation of component assemblies and the construction of black-box behaviours.

## 2 Illustrating Example

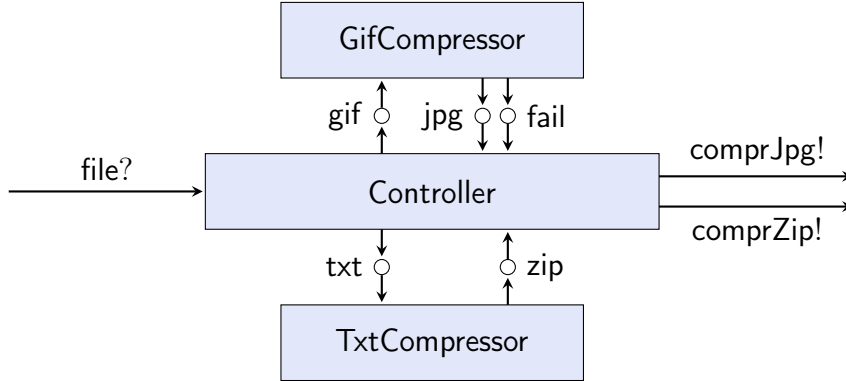
We introduce an illustrating example to motivate our notions of modal asynchronous I/O–Petri nets (MAIOPNs), their composition, hiding and refinement. For this purpose we consider a top down approach to the design of a simple compressing system (inspired by [BCD02]) which is able to receive files for compression and outputs either zip- or jpg-files. We start with an interface specification of the system modelled by the CompressorInterface in Fig. 1.



**Figure 1:** CompressorInterface.

The interface specification is presented by a labelled Petri net with distinguished input and output labels and with modalities on the transitions. The label `file` suffixed with “?” indicates an input action and the labels `comprJpg`, `comprZip` suffixed with “!” indicate output actions. Following the idea of modal transition systems introduced by Larsen and

Thomsen in [LT88] transitions are equipped with “must” or “may” modalities. A must-transition, drawn black, indicates that this transition is required for any refinement while a may-transition, drawn white, may also be removed or turned into a must-transition. Models contain only must-transitions represent implementations. In the example it is required that input files must always be received and that the option to produce zipped text files is always available while a refinement may or may not support the production of compressed jpg-files for graphical data. Our interface specification models an infinite state system since an unbounded number of files can be received.



**Figure 2:** CompressorAssembly (architecture).

In the next step we propose an architecture for the realisation of the compressing system as shown in Fig. 2. It is given by an assembly of three connected components, a **Controller** component which delegates the compression tasks, a **GifCompressor** component which actually performs the compression of gif-files into jpg-files and a **TxtCompressor** component which produces zip-files from text files. The single components are connected by unbounded and unordered channels **gif**, **jpg**, ... for asynchronous communication.

The behaviour of the single components is modelled by the MAIOPNs shown in Figs. 3a, 3b and 3c. The behaviour of the **CompressorAssembly** is given by the asynchronous composition of the single Petri nets shown in Fig. 4. For each pair of shared input and output actions a new place is introduced, called communication channel. Transitions with a shared output label  $a$  (of a given component) are connected to the new place  $a$  and the transition label is renamed to  $a^\triangleright$  in the composition. Similarly the place  $a$  is connected to transitions with the corresponding input label  $a$  which is then renamed to  $\triangleright a$  in the composition. The result of our composition is very similar to the composition of open Petri nets, see e.g. [LMW07], which relies on matching of interface places. But our approach is methodologically different since we introduce the communication places only when the composition is constructed. In that way our basic components are not biased to asynchronous composition but could be used for synchronous composition as well. In this work we focus on asynchronous composition and we are particularly interested in the analysis of generic communication properties ensuring that messages pending on communication channels are eventually consumed. Therefore our notion of modal asynchronous I/O-Petri net will comprise an explicit discrimination of channel places and, additionally to input/output labels we use, for each channel  $a$ , distinguished

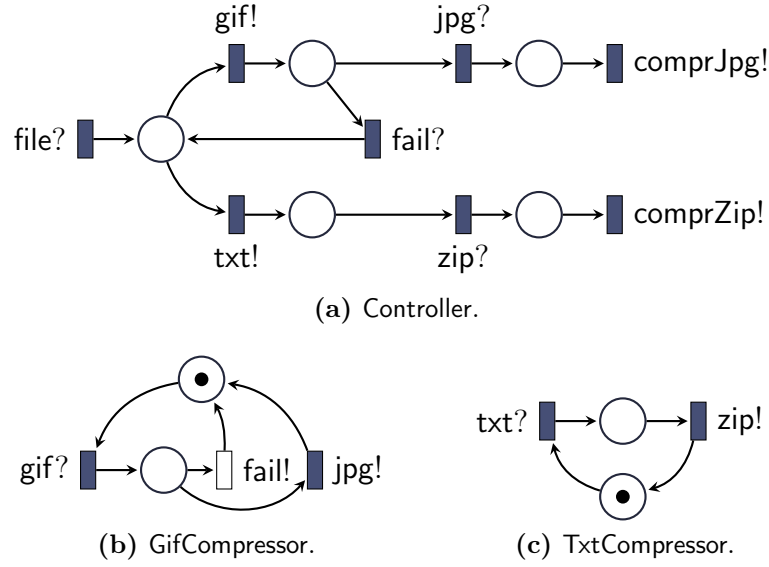


Figure 3: MAIOPN components.

communication labels  $a^\triangleright$  for putting messages on the channel and  $\triangleright a$  for consuming messages from the channel. If the set of channels is empty a MAIOPN models an interface or a primitive component from which larger systems can be constructed by asynchronous composition.

In our example the behaviour of the **CompressorAssembly** is given by the asynchronous composition  $\text{Controller} \otimes_{pn} \text{TxtCompressor} \otimes_{pn} \text{GifCompressor}$  shown in Fig. 4. It models a highly parallel system such that compressing of files can be executed concurrently and new files can be obtained at the same time. Each single compressing tool, however, is working sequentially. Its behaviour should be clear from the specifications. The **GifCompressor** in Fig. 3b has an optional behaviour modelled by a may-transition to indicate a compressing failure and then the controller will submit the file again.

After the assembly has been established we are interested in whether communication works properly in the sense that pending messages on communication channels will be consumed. We will distinguish between two variants of consumption requirements (see Sect. 5) expressing that for non-empty channels there must be a possibility for consumption or, more strongly, that consumption must always happen on each (observationally weakly fair) run. The fairness assumption is essential to support incremental design (Thm. 11); for instance we can first check that  $\text{Controller} \otimes_{pn} \text{TxtCompressor}$  has the desired communication properties for the channels  $\{\text{txt}, \text{zip}\}$ , then we check that the full assembly has the desired communication properties for its channel subset  $\{\text{gif}, \text{jpg}, \text{fail}\}$  and from this we can automatically derive that the assembly has the properties for *all* its channels.

It remains to show that the **CompressorAssembly** is indeed a realisation of the **CompressorInterface**. For this purpose we consider the black-box behaviour of the assembly obtained by hiding the communication channels. This is done by applying our hiding operator to the **CompressorAssembly** denoted by  $\text{CompressorAssembly} \setminus_{pn} \{\text{gif}, \text{jpg}, \text{fail}, \text{txt},$

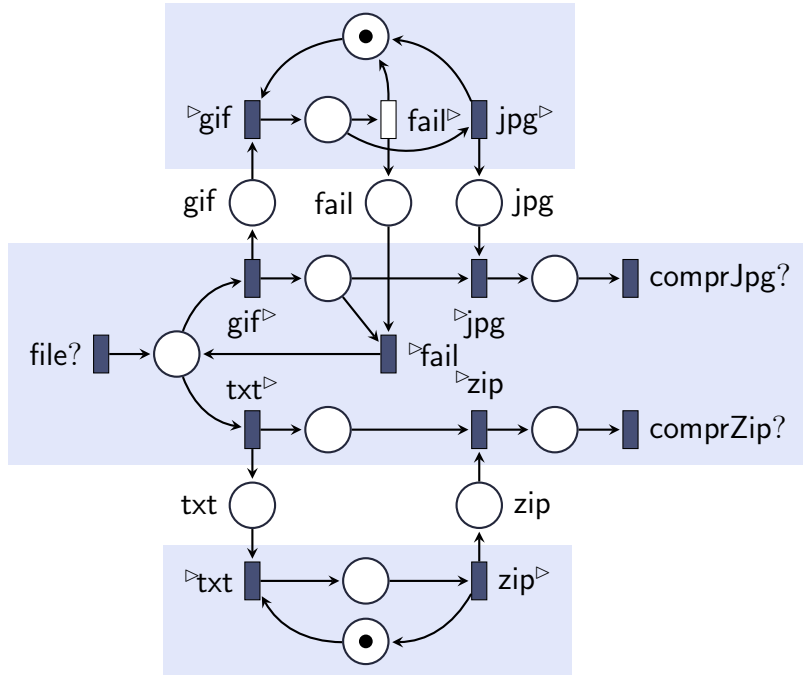


Figure 4: CompressorAssembly.

$\text{zip}\}$ ; see Fig. 5. Hiding moves all communication labels  $a^\triangleright$  and  $\triangleright a$  for the hidden channels  $a$  to the invisible action  $\tau$ . In this way producing and consuming messages from hidden channels become silent transitions. Now we have to establish a refinement relation between  $\text{CompressorAssembly} \setminus_{pn} \{\text{gif}, \text{jpg}, \text{fail}, \text{txt}, \text{zip}\}$  and the  $\text{CompressorInterface}$  by taking into account the modalities on the transitions such that must-transitions of the abstract specification must be available in the refinement and all transitions of the refinement must be allowed by corresponding may-transitions of the abstract specification. In our example the assembly has implemented the optional jpg compression of the interface by a must-transition. Obviously we must also deal with silent transitions which, in our example, occur in  $\text{CompressorAssembly} \setminus_{pn} \{\text{gif}, \text{jpg}, \text{fail}, \text{txt}, \text{zip}\}$ . For this purpose we use a modal refinement relation, denoted by  $\leq_m^*$ , which supports observational abstraction. In our case study this is expressed by the proof obligation (1) in Fig. 5.

Fig. 5 illustrates that after the assembly is proven to be a correct realisation of the interface one can still further refine the assembly by component-wise refinement of its constituent parts. For instance, we can locally refine the  $\text{GifCompressor}$  by resolving the may-modality for producing failures. There are basically two possibilities: Either the failure option is removed or it is turned into a must-transition. The component  $\text{GifCompressorRef}$  in Fig. 5 represents such a refinement of  $\text{GifCompressor}$  indicated by (2). We will show in Thm. 3.1 that refinement is compositional, i.e. we obtain automatically that the assembly  $\text{CompressorAssemblyRef}$  obtained by composition with the new gif compressor is a refinement of  $\text{CompressorAssembly}$  indicated by (3) in Fig. 5. As a crucial result we will also show in Thm. 14 that refinement preserves communication

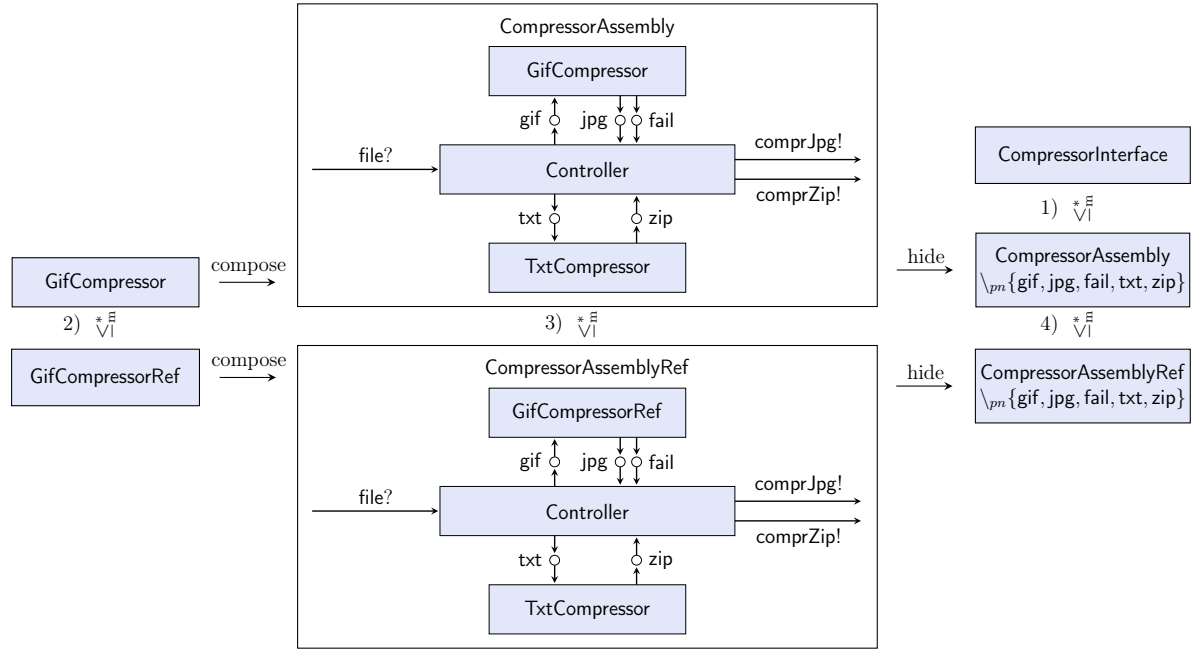


Figure 5: System development methodology.

properties which then can be automatically derived for **CompressorAssemblyRef**. Finally, we must be sure that the refined assembly provides a realisation of the original interface specification **CompressorInterface**. This can be again automatically achieved since hiding preserves refinement (Thm. 3.2) which leads to (4) in Fig. 5 and since refinement is transitive.

In the next sections we will formally elaborate the notions discussed above. We hope that their intended meaning is already sufficiently explained such that we can keep the presentation short. An exception concerns the consideration of the message consuming properties in Sect. 5 which must still be carefully studied to ensure incremental design and preservation by refinement.

### 3 Modal Asynchronous I/O-Petri Nets

In this section we formalise the syntax of MAIOPNs and we define their transition system semantics. First we recall some basic definitions for modal Petri nets and modal transition systems.

#### 3.1 Modal Petri Nets and Modal Transition Systems

In the following we consider labelled Petri nets such that transitions are equipped with labels of an alphabet  $\Sigma$  or with the symbol  $\tau$  that models silent transitions. We write  $\Sigma_\tau$  for  $\Sigma \uplus \{\tau\}$ . We assume the reader to be familiar with the basic notions of labelled

Petri nets consisting of a finite set  $P$  of places, a finite set  $T$  of transitions, a set of arcs between places and transitions (transitions and places resp.), formalised by incidence matrices  $W^-$  and  $W^+$ , an initial marking  $m^0$  and a labelling function  $\lambda : T \rightarrow \Sigma_\tau$ . In [EHH12] we have introduced *modal Petri nets*  $\mathcal{N} = (\mathcal{N}, T^\square)$  such that  $T^\square \subseteq T$  is a distinguished subset of must-transitions. We write  $m \xrightarrow{t} m'$  if a transition  $t \in T$  is firable from a marking  $m$  leading to a marking  $m'$ . If  $t \in T^\square$  we write  $m \xrightarrow{t} m'$ . If  $\lambda(t) = a$  we write  $m \xrightarrow{a} m'$  and for  $t \in T^\square$  we write  $m \xrightarrow{a} m'$ . The notation is extended as usual to firing sequences  $m \xrightarrow{\sigma} m'$  with  $\sigma \in T^*$  and to  $m \xrightarrow{\sigma} m'$  with  $\sigma \in (T^\square)^*$ . A marking  $m$  is reachable if there exists a firing sequence  $\sigma \in T^*$  such that  $m^0 \xrightarrow{\sigma} m$ .

*Modal transition systems* have been introduced in [LT88]. We will use them to provide semantics for modal Petri nets. A modal transition system is a tuple  $\mathcal{S} = (\Sigma, Q, q^0, \xrightarrow{\cdot}, \rightarrow)$ , such that  $\Sigma$  is a set of labels,  $Q$  is a set of states,  $q^0 \in Q$  is the initial state,  $\xrightarrow{\cdot} \subseteq Q \times \Sigma_\tau \times Q$  is a may-transition relation, and  $\rightarrow \subseteq \xrightarrow{\cdot}$  is a must-transition relation. We explicitly allow the state space  $Q$  to be infinite which is needed to give interpretations to modal Petri nets that model infinite state systems.

The semantics of a modal Petri net  $\mathcal{N} = (\mathcal{N}, T^\square)$  is given by the modal transition system  $\text{mts}(\mathcal{N}) = (\Sigma, Q, q^0, \xrightarrow{\cdot}, \rightarrow)$  representing the reachability graph of the net by taking into account modalities:

- $Q \subseteq \mathbb{N}^P$  is the set of reachable markings of  $\mathcal{N}$ ,
- $\xrightarrow{\cdot} = \{(m, a, m') \mid a \in \Sigma_\tau \text{ and } m \xrightarrow{a} m'\}$ ,
- $\rightarrow = \{(m, a, m') \mid a \in \Sigma_\tau \text{ and } m \xrightarrow{a} m'\}$ , and
- $q^0 = m^0$ .

In the sequel we will use the following notations for modal transition systems. We write may-transitions as  $q \xrightarrow{a} q'$  for  $(q, a, q') \in \xrightarrow{\cdot}$  and similarly must-transitions as  $q \xrightarrow{a} q'$ . The notation is extended in the usual way to finite sequences  $\sigma \in \Sigma_\tau^*$  by the notations  $q \xrightarrow{\sigma} q'$  and  $q \xrightarrow{\sigma} q'$ . The set of reachable states of  $\mathcal{S}$  is given by  $\text{Reach}(\mathcal{S}) = \{q \mid \exists \sigma \in \Sigma_\tau^* . q^0 \xrightarrow{\sigma} q\}$ . For a sequence  $\sigma \in \Sigma_\tau^*$  the observable projection  $\text{obs}(\sigma) \in \Sigma^*$  is obtained from  $\sigma$  by removing all occurrences of  $\tau$ . Let  $a \in \Sigma$  be a visible action and  $q, q' \in Q$ . We write  $q \xRightarrow{a} q'$  if  $q \xrightarrow{a} q'$  with  $\text{obs}(\sigma) = a$  and we call  $q \xRightarrow{a} q'$  a weak may-transition. Similarly we write  $q \xRightarrow{a} q'$  if all transitions are must-transitions and call  $q \xRightarrow{a} q'$  a weak must-transition. The notation is extended as expected to finite sequences  $\sigma \in \Sigma^*$  of visible actions by the notations  $q \xRightarrow{\sigma} q'$  and  $q \xRightarrow{\sigma} q'$ . In particular,  $q \xRightarrow{\epsilon} q'$  means that there is a (possibly empty) sequence of silent may-transitions from  $q$  to  $q'$  and similarly  $q \xRightarrow{\epsilon} q'$  expresses a finite sequence of silent must-transitions.

### 3.2 Modal Asynchronous I/O-Petri Nets, Composition and Hiding

In this paper we consider systems which may be open for communication with other systems and may be composed to larger systems. The open actions are modelled by input labels (for the reception of messages from the environment) and output labels (for sending messages to the environment) while communication inside an asynchronously composed system takes place by removing or putting messages to distinguished communication channels. Given a finite set  $C$  of channels, an *I/O-alphabet over  $C$*  is the disjoint union  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  of pairwise disjoint sets of input labels, output labels and communication labels, such that  $\text{com} = \{\triangleright a, a^\triangleright \mid a \in C\}$ . For each channel  $a \in C$ , the label  $\triangleright a$  represents consumption of a message from  $a$  and  $a^\triangleright$  represents putting a message on  $a$ . A *modal asynchronous I/O-Petri net* (MAIOPN)  $\mathcal{N} = (C, P, T, T^\square, \Sigma, W^-, W^+, \lambda, m^0)$  is a modal Petri net such that  $C \subseteq P$  is a set of channel places which are initially empty,  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is an I/O-alphabet over  $C$ , and for all  $a \in C$  and  $t \in T$ , there exists an (unweighted) arc from  $a$  to  $t$  iff  $\lambda(t) = \triangleright a$  and there exists an (unweighted) arc from  $t$  to  $a$  iff  $\lambda(t) = a^\triangleright$ .

The *asynchronous composition* of MAIOPNs works as for asynchronous I/O-Petri nets considered in [HHM13a] by introducing new channel places and appropriate transitions for shared input/output labels. The non-shared input and output labels remain open in the composition. Moreover, we require that the must-transitions of the composition are the union of the must-transitions of the single components. The asynchronous composition of two MAIOPNs  $\mathcal{N}$  and  $\mathcal{M}$  is denoted by  $\mathcal{N} \otimes_{pn} \mathcal{M}$ . It is commutative and also associative under appropriate syntactic restrictions on the underlying alphabets. An example of a composition of three MAIOPNs is given in Fig. 4.

We introduce a *hiding operator* on MAIOPNs which allows us to hide communication channels. In particular, it allows us to compute the black-box behaviour of an assembly when all channels are hidden. Let  $\mathcal{N}$  be a MAIOPN with I/O-alphabet  $\Sigma_{\mathcal{N}} = \text{in}_{\mathcal{N}} \uplus \text{out}_{\mathcal{N}} \uplus \text{com}_{\mathcal{N}}$ , and let  $H \subseteq C_{\mathcal{N}}$  be a subset of its channels. The *channel hiding of  $H$  in  $\mathcal{N}$*  is the MAIOPN  $\mathcal{N} \setminus_{pn} H$  with channels  $C = C_{\mathcal{N}} \setminus H$ , with I/O-alphabet  $\Sigma = \text{in}_{\mathcal{N}} \uplus \text{out}_{\mathcal{N}} \uplus \text{com}$  such that  $\text{com} = \{\triangleright a, a^\triangleright \mid a \in C\}$ , and with the labelling function:

$$\lambda(t) = \begin{cases} \tau & \text{if } \exists a \in H . \lambda_{\mathcal{N}}(t) = \triangleright a \text{ or } \lambda_{\mathcal{N}}(t) = a^\triangleright, \\ \lambda_{\mathcal{N}}(a) & \text{otherwise.} \end{cases}$$

### 3.3 Semantics: Modal Asynchronous I/O-Transition Systems

We extend the transition system semantics of modal Petri nets defined in Sect. 3.1 to MAIOPNs. For this purpose we introduce *modal asynchronous I/O-transition system* (MAIOTS)  $\mathcal{S} = (C, \Sigma, Q, q^0, \dashrightarrow, \rightarrow, \text{val})$  which are modal transition systems such that  $C$  is a finite set of channels,  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  is an I/O-alphabet over  $C$ , and  $\text{val} : Q \rightarrow \mathbb{N}^C$  is a channel valuation function which determines for each state  $q \in Q$  how many messages are actually pending on each channel  $a \in C$ . Instead of  $\text{val}(q)(a)$  we write  $\text{val}(q, a)$ . We require that initially all channels are empty, i.e.  $\text{val}(q^0, a) = 0$  for

all  $a \in C$ , that for each  $a \in C$  putting  $a^\triangleright$  and consuming  $\triangleright a$  messages from  $a$  has the expected behaviour, and that the open input/output actions have no effect on a channel, i.e.

$$\begin{aligned} q \xrightarrow{a^\triangleright} q' &\implies \text{val}(q') = \text{val}(q)[a++]^*, \\ q \xrightarrow{\triangleright a} q' &\implies \text{val}(q, a) > 0 \text{ and } \text{val}(q') = \text{val}(q)[a--], \text{ and} \\ \text{for all } x \in (\text{in} \cup \text{out}), q \xrightarrow{x} q' &\implies \text{val}(q') = \text{val}(q). \end{aligned}$$

The semantics  $\text{maiots}(\mathcal{N})$  of a modal asynchronous I/O-Petri net  $\mathcal{N}$  is given by the transition system semantics of modal Petri nets such that the reachable markings are the states. Additionally we define the associated channel valuation function  $\text{val} : Q \rightarrow \mathbb{N}^C$  such that the valuation of a channel  $a$  in a current state  $m$  is given by the number of tokens on  $a$  under the marking  $m$ , i.e.  $\text{val}(m, a) = m(a)$ . For instance, the semantics of the **CompressorInterface** in Sect. 2 and of the **Controller** leads to infinite state transition systems; the transition systems associated with the two compressor components have two reachable states and the transitions between them correspond directly to their Petri net representations in Fig. 3b and Fig. 3c.

The *asynchronous composition*  $\mathcal{S} \otimes \mathcal{T}$  of two MAIOTSs  $\mathcal{S}$  and  $\mathcal{T}$  works as for asynchronous I/O-transition systems in [HHM13a] taking additionally care that must-transitions of  $\mathcal{S}$  and  $\mathcal{T}$  induce must-transitions of  $\mathcal{S} \otimes \mathcal{T}$ . The composition introduces new channels  $C_{\mathcal{S}\mathcal{T}} = \Sigma_{\mathcal{S}} \cap \Sigma_{\mathcal{T}}$  for shared input/output labels. Every transition with a shared output label  $a$  becomes a transition with the communication label  $a^\triangleright$ , and similarly transitions with input labels  $a$  become transitions with label  $\triangleright a$ . The state space of the composition is (the reachable part of) the Cartesian product of the underlying state spaces of  $\mathcal{S}$  and  $\mathcal{T}$  together with the set  $\mathbb{N}^{C_{\mathcal{S}\mathcal{T}}}$  of valuations for the new channels such that transitions labelled by  $a^\triangleright$  and  $\triangleright a$  have the expected effect on the new channels; for details see [HHM13a]. The asynchronous composition of two MAIOTSs is commutative and also associative under appropriate syntactic restrictions on the underlying alphabets.

We also introduce a *hiding operator* on MAIOTSs that hides communication channels and moves all corresponding communication labels to  $\tau$ . Let  $\mathcal{S} = (C_{\mathcal{S}}, \Sigma_{\mathcal{S}}, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \dashrightarrow_{\mathcal{S}}, \rightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}})$  be a MAIOTS with I/O-alphabet  $\Sigma_{\mathcal{S}} = \text{in}_{\mathcal{S}} \uplus \text{out}_{\mathcal{S}} \uplus \text{com}_{\mathcal{S}}$ , and let  $H \subseteq C_{\mathcal{S}}$  be a subset of its channels. The *channel hiding of  $H$  in  $\mathcal{S}$*  is the MAIOTS  $\mathcal{S} \setminus H = (C, \Sigma, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \dashrightarrow, \rightarrow, \text{val})$ , such that  $C = C_{\mathcal{S}} \setminus H$ ,  $\Sigma = \text{in}_{\mathcal{S}} \uplus \text{out}_{\mathcal{S}} \uplus \text{com}$  with  $\text{com} = \{\triangleright a, a^\triangleright \mid a \in C\}$ ,  $\text{val}(q, a) = \text{val}_{\mathcal{S}}(q, a)$  for all  $q \in Q, a \in C$ , and the may-transition relation is given by:

$$\begin{aligned} \dashrightarrow = & \{(q, a, q') \mid a \in (\Sigma_{\tau} \setminus \{\triangleright a, a^\triangleright \mid a \in H\}) \text{ and } q \dashrightarrow_{\mathcal{S}}^a q'\} \\ & \cup \{(q, \tau, q') \mid \exists a \in H \text{ such that either } q \dashrightarrow_{\mathcal{S}}^{a^\triangleright} q' \text{ or } q \dashrightarrow_{\mathcal{S}}^{\triangleright a} q'\}, \end{aligned}$$

---

\* $\text{val}(q)[a++]$  ( $\text{val}(q)[a--]$  resp.) denotes the update of  $\text{val}$  which increments (decrements) the value of  $a$  and leaves the values of all other channels unchanged.



$$\begin{aligned} \dashrightarrow = & \{(q, a, q') \mid a \in (\Sigma_\tau \setminus \{\triangleright a, a^\triangleright \mid a \in H\}) \text{ and } q \dashrightarrow_S^a q'\} \cup \\ & \{(q, \tau, q') \mid \exists a \in H \text{ such that either } q \dashrightarrow_S^{a^\triangleright} q' \text{ or } q \dashrightarrow_S^{\triangleright a} q'\}, \end{aligned}$$

The must-transition relation  $\longrightarrow$  is defined analogously.

For the results developed in the next sections it is important that our transition system semantics is compositional and compatible with hiding as stated in the next theorem.

**Theorem 1.** *Let  $\mathcal{N}$  and  $\mathcal{M}$  be two composable MAIOPNs and let  $H \subseteq C_{\mathcal{N}}$  be a subset of the channels of  $\mathcal{N}$ . The following holds:*

1.  $\text{maiots}(\mathcal{N} \otimes_{pn} \mathcal{M}) = \text{maiots}(\mathcal{N}) \otimes \text{maiots}(\mathcal{M})$  (up to isomorphic state spaces),
2.  $\text{maiots}(\mathcal{N} \setminus_{pn} H) = \text{maiots}(\mathcal{N}) \setminus H$ .

*Proof.* The proof of part 1.1 is omitted as it is almost identical to the proof in [HHM13b].

**Proof of 1.2** Let  $\text{maiots}(\mathcal{N} \setminus_{pn} H) = A = (C_A, \Sigma_A, Q_A, q_A^0, \dashrightarrow_A, \longrightarrow_A, \text{val}_A)$  and  $\text{maiots}(\mathcal{N}) \setminus H = B = (C_B, \Sigma_B, Q_B, q_B^0, \dashrightarrow_B, \longrightarrow_B, \text{val}_B)$ . The following equalities follow directly from the definition of channel hiding on MAIOPNs, channel hiding on MAIOTSs and the semantics of a MAIOPN:  $C_A = C_B$ ,  $\Sigma_A = \Sigma_B$ ,  $Q_A = Q_B$ ,  $q_A^0 = q_B^0$ , and  $\text{val}_A = \text{val}_B$ . It remains to prove that the two transition relations coincide.

From the semantics of a MAIOPN it follows that

$$\dashrightarrow_A = \{(m, a, m') \mid a \in (\Sigma_A)_\tau . m \dashrightarrow_{(\mathcal{N} \setminus_{pn} H)}^a m'\},$$

where  $\dashrightarrow_{(\mathcal{N} \setminus_{pn} H)}$  is the may-transition relation of  $\mathcal{N} \setminus_{pn} H$ .

From the definition of channel hiding on MAIOPNs we see that communication labels on hidden channels are renamed to  $\tau$ . Thus,

$$\begin{aligned} \dashrightarrow_B = & \{(m, \tau, m') \mid \exists a \in H . m \dashrightarrow_{\mathcal{N}}^{\triangleright a} m' \vee m \dashrightarrow_{\mathcal{N}}^{a^\triangleright} m'\} \\ & \cup \{(m, a, m') \mid a \in (\Sigma_\tau \setminus \{\triangleright a, a^\triangleright \mid a \in H\}) . m \dashrightarrow_{\mathcal{N}}^a m'\}, \end{aligned}$$

where  $\Sigma$  is the I/O-alphabet of  $\mathcal{N}$  and  $\dashrightarrow_{\mathcal{N}}$  is the may-transition relation of  $\mathcal{N}$ . It then follows from the definition of channel hiding on MAIOTS that  $\dashrightarrow_A = \dashrightarrow_B$ . Similarly one can prove that  $\longrightarrow_A = \longrightarrow_B$ .  $\blacksquare$

## 4 Modal Refinement

The refinement relation between MAIOPNs will be defined by considering their semantics, i.e. MAIOTSs. For this purpose we adapt the weak modal refinement relation for modal transition systems introduced in [HL89] which is based on a simulation relation in both directions. It says that must-transitions of an abstract specification must be

preserved by the refinement while may-transitions of a concrete specification must be allowed by the abstract one. In any case silent transitions labelled with  $\tau$  can be inserted, similarly to weak bisimulation, but respecting modalities. Observational abstraction from silent transitions is indeed important in many examples; e.g. when relating the encapsulated behaviour of an assembly to a requirements specification as discussed in Sect. 2. If all transitions of the abstract specification are must-transitions, modal refinement coincides with weak bisimulation. Obviously, the modal refinement relation defined as follows is reflexive and transitive.

**Definition 2** (Modal refinement). *Let  $\mathcal{S} = (C, \Sigma, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \dashrightarrow_{\mathcal{S}}, \rightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}})$  and  $\mathcal{T} = (C, \Sigma, Q_{\mathcal{T}}, q_{\mathcal{T}}^0, \dashrightarrow_{\mathcal{T}}, \rightarrow_{\mathcal{T}}, \text{val}_{\mathcal{T}})$  be two MAIOTSs with the same I/O-alphabet  $\Sigma$  over channels  $C$ . A relation  $R \subseteq Q_{\mathcal{S}} \times Q_{\mathcal{T}}$  is a modal refinement relation between  $\mathcal{S}$  and  $\mathcal{T}$  if for all  $(q_{\mathcal{S}}, q_{\mathcal{T}}) \in R$  and  $a \in \Sigma$ :*

- 1:  $q_{\mathcal{T}} \xrightarrow{a}_{\mathcal{T}} q'_{\mathcal{T}} \implies \exists q'_{\mathcal{S}} \in Q_{\mathcal{S}} . q_{\mathcal{S}} \xRightarrow{a}_{\mathcal{S}} q'_{\mathcal{S}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R,$
- 2:  $q_{\mathcal{T}} \xrightarrow{\tau}_{\mathcal{T}} q'_{\mathcal{T}} \implies \exists q'_{\mathcal{S}} \in Q_{\mathcal{S}} . q_{\mathcal{S}} \xRightarrow{\epsilon}_{\mathcal{S}} q'_{\mathcal{S}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R,$
- 3:  $q_{\mathcal{S}} \dashrightarrow_{\mathcal{S}} q'_{\mathcal{S}} \implies \exists q'_{\mathcal{T}} \in Q_{\mathcal{T}} . q_{\mathcal{T}} \dashrightarrow_{\mathcal{T}} q'_{\mathcal{T}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R,$
- 4:  $q_{\mathcal{S}} \dashrightarrow_{\mathcal{S}} q'_{\mathcal{S}} \implies \exists q'_{\mathcal{T}} \in Q_{\mathcal{T}} . q_{\mathcal{T}} \xRightarrow{\epsilon}_{\mathcal{T}} q'_{\mathcal{T}} \wedge (q'_{\mathcal{S}}, q'_{\mathcal{T}}) \in R.$

*We say that  $\mathcal{S}$  is a modal refinement of  $\mathcal{T}$ , written  $\mathcal{S} \leq_m^* \mathcal{T}$ , if there exists a modal refinement relation  $R$  between  $\mathcal{S}$  and  $\mathcal{T}$  such that  $(q_{\mathcal{S}}^0, q_{\mathcal{T}}^0) \in R$ . We write  $q_{\mathcal{S}} \leq_m^* q_{\mathcal{T}}$  when  $(q_{\mathcal{S}}, q_{\mathcal{T}}) \in R$  for a modal refinement relation  $R$ .*

The next theorem shows that modal refinement is preserved by asynchronous composition and by channel hiding. The compositionality result stems in principle from [HL89]. The second result is similarly to a result in [HK11] for hiding in synchronous systems.

**Theorem 3.** *Let  $\mathcal{S}, \mathcal{T}, \mathcal{E}$  and  $\mathcal{F}$  be MAIOTSs such that  $C$  is the set of channels of  $\mathcal{S}$  and of  $\mathcal{T}$  and let  $H \subseteq C$ .*

1. *If  $\mathcal{S} \leq_m^* \mathcal{T}$ ,  $\mathcal{E} \leq_m^* \mathcal{F}$  and  $\mathcal{S}$  and  $\mathcal{E}$  are composable, then  $\mathcal{T}$  and  $\mathcal{F}$  are composable and  $\mathcal{S} \otimes \mathcal{E} \leq_m^* \mathcal{T} \otimes \mathcal{F}$ .*
2. *If  $\mathcal{S} \leq_m^* \mathcal{T}$  then  $\mathcal{S} \setminus H \leq_m^* \mathcal{T} \setminus H$ .*

*Proof of 3.1.* We need to prove that there exists a modal refinement relation between  $\mathcal{S} \otimes \mathcal{E}$  and  $\mathcal{T} \otimes \mathcal{F}$  containing the initial states. We do this by defining a relation  $R$ , and proving that this particular relation is a modal refinement relation.

First let  $R_1$  be the modal refinement relation between  $\mathcal{S}$  and  $\mathcal{T}$ , and  $R_2$  the modal refinement relation between  $\mathcal{E}$  and  $\mathcal{F}$ . We can now define  $R$  as follows:

$$R = \{((q_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}), (q_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}})) \mid (q_{\mathcal{S}}, q_{\mathcal{T}}) \in R_1 \wedge (q_{\mathcal{E}}, q_{\mathcal{F}}) \in R_2 \wedge \vec{v}_{\mathcal{SE}} = \vec{v}_{\mathcal{TF}}\}.$$

Obviously  $R$  contains the initial states of  $\mathcal{S} \otimes \mathcal{E}$  and  $\mathcal{T} \otimes \mathcal{F}$ .

We will now prove that  $R$  is a modal refinement relation between  $\mathcal{S} \otimes \mathcal{E}$  and  $\mathcal{T} \otimes \mathcal{F}$ . We do this by going through each of the four rules of Def. 2. Let  $\Sigma$  be the alphabet composition of  $\Sigma_{\mathcal{T}} = \text{in}_{\mathcal{T}} \uplus \text{out}_{\mathcal{T}} \uplus \text{com}_{\mathcal{T}}$  and  $\Sigma_{\mathcal{F}} = \text{in}_{\mathcal{F}} \uplus \text{out}_{\mathcal{F}} \uplus \text{com}_{\mathcal{F}}$  and let  $C_{\mathcal{T}\mathcal{F}} = \Sigma_{\mathcal{T}} \cap \Sigma_{\mathcal{F}}$ . Assume that  $((q_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}), (q_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}})) \in R$ .

1. Let  $a \in \Sigma$  such that  $(q_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}}) \xrightarrow{a} (q'_{\mathcal{T}}, q'_{\mathcal{F}}, \vec{v}'_{\mathcal{TF}})$ . By definition of asynchronous composition [HHM13a] it follows that this transition could be triggered in six different ways.
  - (1) Assume that  $a \in (\Sigma_{\mathcal{T}} \setminus C_{\mathcal{T}\mathcal{F}})$ ,  $q_{\mathcal{T}} \xrightarrow{a}_{\mathcal{T}} q'_{\mathcal{T}}$ ,  $q_{\mathcal{F}} = q'_{\mathcal{F}}$  and  $\vec{v}_{\mathcal{TF}} = \vec{v}'_{\mathcal{TF}}$ . By definition of  $R$  we know that  $q_{\mathcal{S}} \leq_m^* q_{\mathcal{T}}$ , which by (1) in Def. 2 gives us that  $q_{\mathcal{S}} \xRightarrow{a} q'_{\mathcal{S}}$  such that  $q'_{\mathcal{S}} \leq_m^* q'_{\mathcal{T}}$ . By definition of asynchronous composition it follows that  $(q_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}) \xRightarrow{a} (q'_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}})$ , and by definition of  $R$ ,  $((q'_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}), (q'_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}})) \in R$ .
  - (2) Assume that  $a \in (\Sigma_{\mathcal{F}} \setminus C_{\mathcal{T}\mathcal{F}})$ ,  $q_{\mathcal{F}} \xrightarrow{a}_{\mathcal{F}} q'_{\mathcal{F}}$ ,  $q_{\mathcal{T}} = q'_{\mathcal{T}}$ , and  $\vec{v}_{\mathcal{TF}} = \vec{v}'_{\mathcal{TF}}$ . This case can be proven analogously.
  - (3.1) Assume that  $a \in \text{in}_{\mathcal{T}} \cap \text{out}_{\mathcal{F}}$ ,  $\vec{v}_{\mathcal{TF}}(a) > 0$ ,  $q_{\mathcal{T}} \xrightarrow{a}_{\mathcal{T}} q'_{\mathcal{T}}$ ,  $q_{\mathcal{F}} = q'_{\mathcal{F}}$  and  $\vec{v}'_{\mathcal{TF}} = \vec{v}_{\mathcal{TF}}[a - -]$ . By definition of  $R$  we know that  $q_{\mathcal{S}} \leq_m^* q_{\mathcal{T}}$ , which by (1) in Def. 2 gives us that  $q_{\mathcal{S}} \xRightarrow{a} q'_{\mathcal{S}}$  such that  $q'_{\mathcal{S}} \leq_m^* q'_{\mathcal{T}}$ . By definition of  $R$  we know that  $\vec{v}_{\mathcal{SE}}(a) = \vec{v}_{\mathcal{TF}} > 0$ . By definition of asynchronous composition it follows that  $(q_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}) \xRightarrow{a} (q'_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}[a - -])$  taking into account that  $\tau$ -transitions do not change the valuation of channels. Finally by definition of  $R$ ,  $((q'_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}[a - -]), (q'_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}}[a - -])) \in R$ .
  - (3.2) Assume that  $a \in \text{in}_{\mathcal{T}} \cap \text{out}_{\mathcal{F}}$ ,  $q_{\mathcal{T}} = q'_{\mathcal{T}}$ ,  $q_{\mathcal{F}} \xrightarrow{a}_{\mathcal{F}} q'_{\mathcal{F}}$  and  $\vec{v}'_{\mathcal{TF}} = \vec{v}_{\mathcal{TF}}[a + +]$ . By definition of  $R$  we know that  $q_{\mathcal{E}} \leq_m^* q_{\mathcal{F}}$ , which by (1) in Def. 2 gives us that  $q_{\mathcal{E}} \xRightarrow{a} q'_{\mathcal{E}}$  such that  $q'_{\mathcal{E}} \leq_m^* q'_{\mathcal{F}}$ . By definition of asynchronous composition we get that  $(q_{\mathcal{S}}, q_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}) \xRightarrow{a} (q_{\mathcal{S}}, q'_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}[a + +])$ . Since silent transitions do not change the channel valuation. Finally by definition of  $R$ ,  $((q_{\mathcal{S}}, q'_{\mathcal{E}}, \vec{v}_{\mathcal{SE}}[a + +]), (q_{\mathcal{T}}, q'_{\mathcal{F}}, \vec{v}_{\mathcal{TF}}[a + +])) \in R$ .
  - (4.1) Assume that  $a \in \text{in}_{\mathcal{F}} \cap \text{out}_{\mathcal{T}}$ ,  $\vec{v}_{\mathcal{TF}}(a) > 0$ ,  $q_{\mathcal{T}} = q'_{\mathcal{T}}$ ,  $q_{\mathcal{F}} \xrightarrow{a}_{\mathcal{F}} q'_{\mathcal{F}}$  and  $\vec{v}'_{\mathcal{TF}} = \vec{v}_{\mathcal{TF}}[a - -]$ . This case can be proven analogously to (3.1).
  - (4.2) Assume that  $a \in \text{in}_{\mathcal{F}} \cap \text{out}_{\mathcal{T}}$ ,  $q_{\mathcal{T}} \xrightarrow{a}_{\mathcal{T}} q'_{\mathcal{T}}$ ,  $q_{\mathcal{F}} = q'_{\mathcal{F}}$  and  $\vec{v}'_{\mathcal{TF}} = \vec{v}_{\mathcal{TF}}[a + +]$ . This case can be proven analogously to (3.2).
2. Assume  $(q_{\mathcal{T}}, q_{\mathcal{F}}, \vec{v}_{\mathcal{TF}}) \xrightarrow{\tau} (q'_{\mathcal{T}}, q'_{\mathcal{F}}, \vec{v}'_{\mathcal{TF}})$ . By definition of asynchronous composition it follows that this transition could be triggered in two different ways, either by  $q_{\mathcal{T}} \xrightarrow{\tau}_{\mathcal{T}} q'_{\mathcal{T}}$  or  $q_{\mathcal{F}} \xrightarrow{\tau}_{\mathcal{F}} q'_{\mathcal{F}}$ . Both cases can be proven analogously to case (1) and (2) above, using (2) of Def. 2.

3. Let  $a \in \Sigma$  such that  $(q_S, q_E, \vec{v}_{SE}) \xrightarrow{-a} (q'_S, q'_E, \vec{v}'_{SE})$ . By definition of asynchronous composition it follows that this transition could be triggered in six different ways. Each of the six cases can be proven analogously to case (1), (2), (3.1), (3.2), (4.1), and (4.2) above, using (3) of Def. 2.
4. Assume  $(q_S, q_E, \vec{v}_{SE}) \xrightarrow{-\tau} (q'_S, q'_E, \vec{v}'_{SE})$ . By definition of asynchronous composition it follows that this transition could be triggered in two different ways, either by  $q_T \xrightarrow{-\tau} q'_T$  or  $q_F \xrightarrow{-\tau} q'_F$ . Both cases can be proven analogously to case (1) and (2) above using (4) of Def. 2.

■

*Proof of 3.2.* First let  $\mathcal{S} \setminus H = (C_{\setminus H}, \Sigma_{\setminus H}, Q_S, q_S^0, \xrightarrow{-\cdot}_{(\mathcal{S} \setminus H)}, \xrightarrow{\cdot}_{(\mathcal{S} \setminus H)}, \text{val}_{(\mathcal{S} \setminus H)})$  and  $\mathcal{T} \setminus H = (C_{\setminus H}, \Sigma_{\setminus H}, Q_T, q_T^0, \xrightarrow{-\cdot}_{(\mathcal{T} \setminus H)}, \xrightarrow{\cdot}_{(\mathcal{T} \setminus H)}, \text{val}_{(\mathcal{T} \setminus H)})$ .

As  $\mathcal{S} \leq_m^* \mathcal{T}$  we know that there exists a modal refinement relation  $R \subseteq Q_S \times Q_T$ . We will prove that  $R$  is also a modal refinement relation for  $\mathcal{S} \setminus H$  and  $\mathcal{T} \setminus H$ .  $R$  is a good candidate as the set of states does not change when hiding a subset of the channels. We will prove this by going through the four requirements of Def. 2. However, we will only provide the proof of the two first cases, as the last two can be proven in the same manner. Let  $(q_S, q_T) \in R$ .

1. Let  $a \in \Sigma_{\setminus H}$ . Assume that  $q_T \xrightarrow{a}_{(\mathcal{T} \setminus H)} q'_T$ . By definition of hiding it follows that  $q_T \xrightarrow{a}_T q'_T$ . By modal refinement we get that there exists  $q'_S \in Q_S$  .  $q_S \xrightarrow{a}_T q'_S \wedge (q'_S, q'_T) \in R$ . Again by hiding we get that  $q_S \xrightarrow{a}_{(\mathcal{S} \setminus H)} q'_S$ .
2. Assume that  $q_T \xrightarrow{\tau}_{(\mathcal{T} \setminus H)} q'_T$ . By definition of hiding there can be two cases. Either  $\tau$  stems from  $\mathcal{T}$  directly, or it stems from channel hiding.
  - Assume  $q_T \xrightarrow{\tau}_T q'_T$ . Then the argumentation follows as as in item 1.
  - Assume that  $a \in H$  and  $q_T \xrightarrow{a^\triangleright}_T q'_T$ . By modal refinement we get that there exists  $q'_S \in Q_S$  .  $q_S \xrightarrow{a^\triangleright}_T q'_S \wedge (q'_S, q'_T) \in R$ . Now by hiding and the fact that  $a \in H$  we get that  $q_S \xrightarrow{a}_{(\mathcal{S} \setminus H)} q'_S$ .

■

The refinement definition and results are propagated to modal asynchronous I/O-Petri nets in the obvious way: A MAIOPN  $\mathcal{M}$  is a *modal refinement* of a MAIOPN  $\mathcal{N}$ , also denoted by  $\mathcal{M} \leq_m^* \mathcal{N}$ , if  $\text{maiot}(\mathcal{M}) \leq_m^* \text{maiot}(\mathcal{N})$ . The counterparts of Thm. 3.1 and 3.2 for MAIOPNs are consequences of the semantic compatibility results in Thm. 1.1. Examples for modal refinements of MAIOPNs and applications of the theorem are pointed out in Sect. 2.

The decidability status of the modal refinement problem for MAIOPNs depends on the kind of Petri nets one considers. Observing that there is a simple reduction from the bisimilarity problem to the modal refinement problem and that the former problem is undecidable for Petri nets [Jan95], one gets that the latter problem is also undecidable;

for an evolved discussion see [BK12]. However when one restricts Petri nets to be modally weakly deterministic, the modal refinement problem becomes decidable. The modal weak determinism of Petri nets is a behavioural property which can also be decided (see [EHH12] for both proofs). In addition, determinism is a desirable feature for a specification (when possible). For instance modal language specification is an alternative to modal transition systems that presents such a behaviour [Rac07].

## 5 Message Consuming Systems

In this section we consider generic properties concerning the asynchronous communication via channels inspired by the various channel properties studied in [HHM13a]. We focus on the *message consuming* and the *necessarily message consuming* properties and generalise them to take into account modalities and observational abstraction w.r.t. silent transitions. Our goal is that the properties scale up to larger contexts (to support incremental design) and that they are preserved by modal refinement. Moreover we consider their decidability. For the definitions we rely on the semantics of MAIOPNs, i.e. on MAIOTSs.

### 5.1 Concepts and Definitions

Let us first discuss the message consuming property (a) of Def. 4 for a MAIOTS  $\mathcal{S}$  and a subset  $B$  of its channels. It requires, for each channel  $a \in B$ , that if in an arbitrary reachable state  $q$  of  $\mathcal{S}$  there is a message on  $a$ , then  $\mathcal{S}$  must be able to consume it, possibly after some delay caused by the execution of autonomous must-transitions. All transitions that do not depend on the environment, i.e. are not related to input labels, are considered to be autonomous. Our approach follows a “pessimistic” assumption taking into account arbitrary environments that can let the system go where it wants and can also stop to provide inputs at any moment. It is important that we require must-transitions since the consuming property should be preserved by modal refinement. It is inspired by the notion of “output compatibility” studied for synchronously composed transition systems in [HK11].

A central role when components run in parallel is played by fairness assumptions; see, e.g., [Ber+01]. First we must define what we mean by a run and then we will explain our fairness notion. A run is a finite or infinite sequence of state transitions which satisfies a maximality condition. In principle a run can only stop when a deadlock is reached. However we must be careful since (1) we are dealing with open systems whose executions depend on the input from the environment, (2) we must take into account that transitions with a may-modality can be skipped in a refinement, and (3) we must be aware that also silent must-transitions without successive mandatory visible actions can be omitted in a refinement; cf. Def. 2, rule (2). In particular divergence in an abstract state could be implemented by a deadlock. If one of the above conditions holds in a certain state it is called a *potential deadlock state*.

Formally, let  $\mathcal{S} = (C, \Sigma, Q, q^0, \dashrightarrow, \longrightarrow, \text{val})$  be a MAIOTS with  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$ . A

state  $q \in Q$  is a potential deadlock state if for all  $a \in (\Sigma \setminus \text{in})$ , there is no state  $q' \in Q$  such that  $q \xrightarrow{a} q'$ . A *run* of  $\mathcal{S}$  starting in a state  $q_1 \in Q$  is a finite or infinite sequence  $\rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} \dots$  with  $a_i \in \Sigma_\tau$  and  $q_i \in Q$  such that, if the sequence is finite, its last state is a potential deadlock state. We assume that system runs are executed in a runtime infrastructure which follows a fair scheduling policy. In our context this means that any visible autonomous action  $a$ , that is always enabled by a weak must-transition from a certain state on, will infinitely often be executed. Formally, a run  $\rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots$  is called *observationally weakly fair* if it is finite or if it is infinite and then for all  $a \in (\Sigma \setminus \text{in})$  the following holds:

$$(\exists k \geq 1 . \forall i \geq k . \exists q' . q_i \xrightarrow{a} q') \implies (\forall k \geq 1 . \exists i \geq k . a_i = a).$$

We denote the set of all observationally weakly fair runs of  $\mathcal{S}$  starting from  $q_1$  by  $\text{wfrun}_{\mathcal{S}}(q_1)$ . For instance, for the MAIOPN  $\mathcal{M}$  in Fig. 6b on page 153 an infinite run which always executes  $a^\triangleright, \triangleright a, \dots$  is observationally weakly fair. A (diverging) run of  $\mathcal{M}$  which always executes  $\tau$  from a certain state on is not observationally weakly fair since  $\triangleright a$  is then always enabled by a weak must-transition but never taken.

Note that for our results it is sufficient to use a weak fairness property instead of strong fairness. We are now ready to define also the necessarily consuming property (b) which requires that whenever a message is pending on a communication channel then the message must eventually be consumed on all observationally weakly fair runs.

**Definition 4** (Message consuming). *Let  $\mathcal{S} = (C, \Sigma, Q, q^0, \xrightarrow{\cdot}, \xrightarrow{\cdot}, \text{val})$  be a MAIOTS with I/O-alphabet  $\Sigma = \text{in} \uplus \text{out} \uplus \text{com}$  and let  $B \subseteq C$  be a subset of its channels.*

- a)  $\mathcal{S}$  is message consuming w.r.t.  $B$  if for all  $a \in B$  and all  $q \in \text{Reach}(\mathcal{S})$ ,  
 $\text{val}(q, a) > 0 \implies \exists q', q'' \in Q . \exists \sigma \in (\Sigma \setminus \text{in})^* . q \xrightarrow{\sigma} q' \xrightarrow{\triangleright a} q''$ .
- b)  $\mathcal{S}$  is necessarily message consuming w.r.t.  $B$  if for all  $a \in B, q \in \text{Reach}(\mathcal{S})$ ,  
 $\text{val}(q, a) > 0 \implies \forall \rho \in \text{wfrun}_{\mathcal{S}}(q) . \triangleright a \in \rho .^a$

$\mathcal{S}$  is (necessarily) message consuming if  $\mathcal{S}$  is (necessarily) message consuming w.r.t.  $C$ .

---

<sup>a</sup> i.e. there is a transition in  $\rho$  labelled by  $\triangleright a$ .

In the special case, in which all may-transitions are must-transitions and no silent transitions occur observationally weakly fair runs coincide with weakly fair runs and the two consuming properties coincide with the corresponding properties in [HHM13a].

**Proposition 5.** *Let  $\mathcal{S}$  be a MAIOTS. If  $\mathcal{S}$  is necessarily message consuming w.r.t  $B$  then  $\mathcal{S}$  is message consuming w.r.t  $B$ .*

*Proof.* This is an adaption of the proof in [HHM13b]. The proof relies on the fact that for each  $q \in \text{Reach}(\mathcal{S})$  there exists an observationally weakly fair run  $\rho \in \text{owfruns}_{\mathcal{S}}(q)$ , such that  $\rho$  is a must-trace,  $a \notin \rho$  for all  $a \in \text{in}$ , and there is no infinite  $\tau$  sequence in  $\rho$ . This run can be constructed by choosing, in an observationally weakly fair manner in each reached state, some enabled weak non-input must-action. If no such action is enabled in the last visited state, the last state is a deadlock state and we are done. Otherwise the resulting infinite run has the required property.

With this fact, we can prove the proposition as follows: Let  $q \in \text{Reach}(\mathcal{S})$ ,  $a \in B$  such that  $\text{val}(q, a) > 0$ . By necessarily message consuming, for all  $\rho \in \text{owfruns}_{\mathcal{S}}(q)$  we have  $\triangleright a \in \rho$ . Since we know there exists a weakly fair run  $\rho$  without input actions, constructed as described above, we get that there exists  $\sigma \in (\Sigma \setminus \text{in})^*$  and  $q' \in Q$  such that  $q \xRightarrow{\sigma} q' \xrightarrow{\triangleright a}$ . Thus  $\mathcal{S}$  is message consuming w.r.t.  $B$ . ■

The definitions and the proposition are propagated to modal asynchronous I/O-Petri nets in the obvious way. For instance, a MAIOPN  $\mathcal{N}$  is (*necessarily*) *message consuming* if  $\text{maiots}(\mathcal{N})$  is (*necessarily*) message consuming. All MAIOPNs considered in Sect. 2 are necessarily message consuming.

As stated in the introduction, Petri nets are a useful model since (1) they model infinite state systems and (2) several relevant properties of transition systems are decidable. The following proposition whose proof is an adaption of the one in [HHM13a] establishes that one can decide both consuming properties.

Before establishing the proof of Proposition 6, we recall some results about Petri nets that we apply here. Let  $E \subseteq \mathbb{N}^k$ ,  $E$  is a *linear set* if there exists a finite set of vectors of  $\mathbb{N}^k$   $\{v_0, \dots, v_n\}$  such that  $E = \{v_0 + \sum_{1 \leq i \leq n} \lambda_i v_i \mid \forall i \lambda_i \in \mathbb{N}\}$ . A *semi-linear set* [GS66] is a finite union of linear sets; a representation of it is given by the family of finite sets of vectors defining the corresponding linear sets. Semi-linear sets are *effectively* closed w.r.t. union, intersection and complementation. This means that one can compute a representation of the union, intersection and complementation starting from a representation of the original semi-linear sets.  $E$  is an *upward closed set* if  $\forall v \in E. v' \geq v \Rightarrow v' \in E$ . An upward closed set has a finite set of minimal vectors denoted  $\text{min}(E)$ . An upward closed set is a semi-linear set which has a representation that can be derived from the equation  $E = \text{min}(E) + \mathbb{N}^k$  if  $\text{min}(E)$  is computable.

Given a Petri net  $\mathcal{N}$  and a marking  $m$ , the *reachability* problem consists in deciding whether  $m$  is reachable from  $m_0$  in  $\mathcal{N}$ . This problem is decidable [May81] but none of the associated algorithms are primitive recursive. Furthermore this procedure can be adapted to semi-linear sets when markings are identified to vectors of  $\mathbb{N}^{|P|}$ .

In [Rac78], the *coverability* is shown to be EXPSpace-complete. The coverability problem consists in determining, given a net and a target marking, whether one can reach a marking greater or equal than the target. In [VJ85] given a Petri net, several procedures have been designed to compute the minimal set of markings of several interesting upward

closed sets. In particular, given an upward closed set **Target**, by a backward analysis one can compute the (representation of) upward closed set from which **Target** is reachable. Using the results of [Rac78], this algorithm performs in EXPSpace.

While in Petri nets, strong fairness is undecidable [Car87], observationally weak fairness is decidable and more generally, the existence of an infinite sequence fulfilling a formula of the following fragment of LTL is decidable [Jan90]. The literals are (1) comparisons between places markings and values, (2) transition firings and (3) their negations. Formulas are inductively defined as literals, conjunction or disjunction of formulas and  $GF\varphi$  where  $GF$  is the infinitely often operator and  $\varphi$  is a formula.

**Proposition 6.** *Let  $\mathcal{N}$  be a MAIOPN and let  $B \subseteq C$  be a subset of its channels. The satisfaction by  $\mathcal{N}$  of the message consuming and the necessarily message consuming properties w.r.t.  $B$  are decidable.*

*Proof.* **Consuming property.**

Let  $a \in B$  and  $E_a$  be the upward closed set of markings defined by:

$$E_a = \{m \mid \exists t \in T^\square \text{ with } \lambda(t) = \triangleright a \text{ and } m \geq W^-(t)\},$$

$E_a$  is the set of markings from which (in all refinements) one can immediately consume some message  $a$ . Let  $F_a$  be the upward closed set of markings defined by:

$$F_a = \{m \mid \exists m' \in E_a \exists \sigma \in (T^\square)^*. \lambda(\sigma) \in (\Sigma_\tau \setminus \text{in})^* \wedge m \xrightarrow{\sigma} m'\},$$

$F_a$  is the set of markings from which one can later (in all refinements without the help of the environment) consume some message  $a$ . One computes  $F_a$  by backward analysis. Let  $G$  be defined by:

$$G = \{m \mid \exists a \in B. m(a) > 0 \wedge m \notin F_a\}.$$

$G$  is a semi-linear set corresponding to the markings from which some message  $a \in B$  will never be consumed. Then  $\mathcal{N}$  is not  $B$ -consuming iff  $G$  is reachable.

**Necessarily consuming property.**

First one checks whether  $\mathcal{N}$  is  $B$ -consuming, a necessary condition for being necessarily  $B$ -consuming. If  $\mathcal{N}$  is  $B$ -consuming, then one checks whether for some  $a \in B$ , there exists a reachable marking  $m$  fulfilling  $m(a) > 0$  from which an infinite weakly fair run is fireable without occurrence of transitions labelled by  $\triangleright a$ . To perform this test, one builds a net  $\mathcal{N}'$  as follows.

- $P' = P \uplus \{\text{run}\}$
- $T' = T \uplus \{\text{stop}\}$
- $\forall p \in P \forall t \in T \ W'^-(p, t) = W^-(p, t), W'^+(p, t) = W^+(p, t), m'^0(p) = m^0(p)$
- $W'^-(\text{run}, \text{stop}) = 1, W'^+(\text{run}, \text{stop}) = 0, m'^0(\text{run}) = 1$



- $W'^-(a, stop) = W'^+(a, stop) = 1$
- $\forall p \in P \setminus \{a\} \ W'^-(p, stop) = W'^+(p, stop) = 0$
- $\forall t \in T \text{ such that } \lambda(t) = \triangleright_a \ W'^-(run, t) = W'^+(run, t) = 1$
- $\forall t \in T \text{ such that } \lambda(t) \neq \triangleright_a \ W'^-(run, t) = W'^+(run, t) = 0$

$\mathcal{N}'$  behaves as  $\mathcal{N}$  as long as *stop* is not fired. Transition *stop* can be fired only if  $m(a) > 0$ . When *stop* is fired only transitions not labelled by  $\triangleright_a$  are fireable. Then one checks whether there exists an infinite observationally weakly fair run that fulfils formula  $G\text{Fair} = 0 \wedge GF\text{fireable}(\triangleright_a)$  in  $\mathcal{N}'$ . Since  $\text{fireable}(\triangleright_a)$  is expressible by a boolean combination of comparisons of place markings with constants, the formula belongs to the logic of [Jan90]. The first term witnesses the firing of *stop* and the second term ensures that the sequence is also weakly fair in  $\mathcal{N}$ . Then  $\mathcal{N}$  is necessarily *B*-consuming iff there is no such run. ■

## 5.2 Composition of Message Consuming Systems

In order to prove that message consuming properties are preserved when systems are put into larger contexts we need the next two lemmas. The first one shows that autonomous executions of constituent parts (not involving inputs) can be lifted to executions of compositions. This is the essence to get compositionality of the message consuming property (a).

**Lemma 7.** *Let  $\mathcal{S} = (C_{\mathcal{S}}, \Sigma_{\mathcal{S}}, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \dashrightarrow_{\mathcal{S}}, \rightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}}), \mathcal{T} = (C_{\mathcal{T}}, \Sigma_{\mathcal{T}}, Q_{\mathcal{T}}, q_{\mathcal{T}}^0, \dashrightarrow_{\mathcal{T}}, \rightarrow_{\mathcal{T}}, \text{val}_{\mathcal{T}})$  be two composable MAIOTSs, and let  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \dashrightarrow, \rightarrow, \text{val})$ . For all  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in Q$  and  $\sigma \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})^*$  it holds:*

$$q_{\mathcal{S}} \xRightarrow{\sigma}_{\mathcal{S}} q'_{\mathcal{S}} \implies \exists \vec{v}' . (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xRightarrow{\bar{\sigma}} (q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}'),$$

*with  $\bar{\sigma} \in (\Sigma \setminus \text{in})^*$  obtained from  $\sigma$  by replacing any occurrence of a shared label  $a \in \text{out}_{\mathcal{S}} \cap \text{in}_{\mathcal{T}}$  by the communication label  $a^{\triangleright}$ .*

The proof of this lemma is omitted as is almost identical to the proof given in [HHM13b].

The second lemma is crucial to prove compositionality of the necessarily consuming property (b). It shows that projections of observationally weakly fair runs to constituent parts of a composition are again observationally weakly fair runs. Formally, let  $\rho$  be a trace of  $\mathcal{S} \otimes \mathcal{T}$  starting from a state  $q = (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in Q$ . The *projection* of  $\rho$  to  $\mathcal{S}$ , denoted by  $\rho|_{\mathcal{S}}$ , is the sequence of transitions of  $\mathcal{S}$ , starting from  $q_{\mathcal{S}}$ , which have triggered corresponding transitions in  $\rho$ .

**Lemma 8.** *Let  $\mathcal{S}, \mathcal{T}$  be two composable MAIOTSs and  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \dashrightarrow, \rightarrow, \text{val})$ . Let  $q = (q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in Q$  and  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}(q)$ . Then  $\rho|_{\mathcal{S}} \in \text{wfrun}_{\mathcal{S}}(q_{\mathcal{S}})$  is an observationally weakly fair run of  $\mathcal{S}$ .*

The following proof is a generalisation of the proof of the corresponding lemma in [HHM13b].

*Proof.* Let  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}(q)$ . First we show that  $\rho|_{\mathcal{S}}$  is observationally weakly fair as well. If  $\rho|_{\mathcal{S}}$  is finite we are done. If  $\rho|_{\mathcal{S}}$  is infinite, then assume that from a certain state of  $\rho|_{\mathcal{S}}$  on a visible autonomous action  $a \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$  is always enabled by a weak must-transition. Due to the asynchronous composition we get that also in  $\rho$  from a certain state on the action triggered by  $a$  (i.e.  $a$  or  $a^{\triangleright}$ ) is always enabled by a weak must-transition. Hence the action triggered by  $a$  will be infinitely often executed in  $\rho$  and therefore  $a$  will be infinitely often executed in  $\rho|_{\mathcal{S}}$ .

It remains to prove that  $\rho|_{\mathcal{S}}$  is a run of  $\mathcal{S}$ . There are two main cases. Either  $\rho$  is finite or infinite. Assume  $\rho$  is finite with its last state being  $q' = (q'_{\mathcal{S}}, q'_{\mathcal{T}}, \vec{v}')$ . Clearly  $q'_{\mathcal{S}}$  must be the last state of  $\rho|_{\mathcal{S}}$ . By definition of a run,  $q'$  is a potential deadlock state of  $\mathcal{S} \otimes \mathcal{T}$ . Then  $q'_{\mathcal{S}}$  must be a potential deadlock state of  $\mathcal{S}$ , hence  $\rho|_{\mathcal{S}}$  is a run of  $\mathcal{S}$ .

Now assume that  $\rho$  is infinite and observationally weakly fair. In this case there are two subcases. Either  $\rho|_{\mathcal{S}}$  is finite or infinite. If  $\rho|_{\mathcal{S}}$  is infinite, then  $\rho|_{\mathcal{S}}$  is a run of  $\mathcal{S}$  and we are done. Assume  $\rho|_{\mathcal{S}}$  is finite such that its last state is  $q'_{\mathcal{S}}$ , with  $(q'_{\mathcal{S}}, q'_{\mathcal{T}}, \vec{v}') \in \rho$ . Since  $\rho$  is observationally weakly fair we can prove that  $q'_{\mathcal{S}}$  is a potential deadlock state of  $\mathcal{S}$  as follows: Assume the contrary, that  $q'_{\mathcal{S}}$  is not a potential deadlock state, i.e. there exists  $a \in (\Sigma_{\mathcal{S}} \setminus \text{in}_{\mathcal{S}})$  such that  $q \xrightarrow{a}$ . As  $\rho$  is infinite we get that from  $(q'_{\mathcal{S}}, q'_{\mathcal{T}}, \vec{v}') \in \rho$  on  $a$  (or  $a^{\triangleright}$ ) is always enabled by a weak must-transition. This is a contradiction, since  $\rho$  is observationally weakly fair and  $a$  ( $a^{\triangleright}$  resp.) does not occur infinitely often after  $(q'_{\mathcal{S}}, q'_{\mathcal{T}}, \vec{v}') \in \rho$  since  $\rho|_{\mathcal{S}}$  is finite. Now, knowing that  $q'_{\mathcal{S}}$  is a potential deadlock state of  $\mathcal{S}$ , we get that  $\rho|_{\mathcal{S}}$  is a run of  $\mathcal{S}$ .  $\blacksquare$

**Proposition 9.** *Let  $\mathcal{S}$  and  $\mathcal{T}$  be two composable MAIOTs such that  $C_{\mathcal{S}}$  is the set of channels of  $\mathcal{S}$ . Let  $B \subseteq C_{\mathcal{S}}$ . If  $\mathcal{S}$  is (necessarily) consuming w.r.t.  $B$ , then  $\mathcal{S} \otimes \mathcal{T}$  is (necessarily) consuming w.r.t.  $B$ .*

*Proof.* This proof follows the same shape as the proof of the corresponding proposition in [HHM13b] but using Lemma 7 and Lemma 8.

Let  $\mathcal{S} = (C_{\mathcal{S}}, \Sigma_{\mathcal{S}}, Q_{\mathcal{S}}, q_{\mathcal{S}}^0, \dashrightarrow_{\mathcal{S}}, \rightarrow_{\mathcal{S}}, \text{val}_{\mathcal{S}})$ ,  $\mathcal{T} = (C_{\mathcal{T}}, \Sigma_{\mathcal{T}}, Q_{\mathcal{T}}, q_{\mathcal{T}}^0, \dashrightarrow_{\mathcal{T}}, \rightarrow_{\mathcal{T}}, \text{val}_{\mathcal{T}})$  and  $\mathcal{S} \otimes \mathcal{T} = (C, \Sigma, Q, q^0, \dashrightarrow, \rightarrow, \text{val})$ . Let  $a \in B$  and  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \in \text{Reach}(\mathcal{S} \otimes \mathcal{T})$  such that  $\text{val}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}), a) > 0$ .

1. Assume that  $\mathcal{S}$  is  $B$ -consuming. Obviously  $q_{\mathcal{S}} \in \text{Reach}(\mathcal{S})$ . Then there exist  $q'_{\mathcal{S}}, q''_{\mathcal{S}} \in Q_{\mathcal{S}}$  and  $\sigma \in (\Sigma \setminus \text{in})^*$  such that  $q_{\mathcal{S}} \xrightarrow{\sigma}_{\mathcal{S}} q'_{\mathcal{S}} \xrightarrow{\triangleright_a}_{\mathcal{S}} q''_{\mathcal{S}}$ . As a direct consequence of Lem. 7, we get that there exist  $\vec{v}', \vec{v}''$  such that  $(q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}) \xrightarrow{\bar{\sigma}} (q'_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}') \xrightarrow{\triangleright_a} (q''_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}'')$ . Hence,  $\mathcal{S} \otimes \mathcal{T}$  is  $B$ -consuming.
2. Assume that  $\mathcal{S}$  is necessarily  $B$ -consuming. Let  $\rho \in \text{wfrun}_{\mathcal{S} \otimes \mathcal{T}}((q_{\mathcal{S}}, q_{\mathcal{T}}, \vec{v}))$  be an observationally weakly fair run. By Lem. 8 we get that  $\rho|_{\mathcal{S}}$  is an observationally weakly fair run of  $\mathcal{S}$ . By assumption  $\triangleright_a \in \rho|_{\mathcal{S}}$ , hence it follows that  $\triangleright_a \in \rho$ .

■

Proposition 9 leads to the desired modular verification result which allows us to check consuming properties in an incremental manner: To show that a composed system  $\mathcal{S} \otimes \mathcal{T}$  is (necessarily) message consuming it is sufficient to know that both constituent parts  $\mathcal{S}$  and  $\mathcal{T}$  have this property and to check that  $\mathcal{S} \otimes \mathcal{T}$  is (necessarily) message consuming w.r.t. the new channels established for the communication between  $\mathcal{S}$  and  $\mathcal{T}$ , i.e. that  $\mathcal{S}$  and  $\mathcal{T}$  are compatible.

**Definition 10** (Compatibility). *Two composable MAIOTSs  $\mathcal{S}$  and  $\mathcal{T}$  with shared actions  $\Sigma_{\mathcal{S}} \cap \Sigma_{\mathcal{T}}$  are (necessarily) message consuming compatible if  $\mathcal{S} \otimes \mathcal{T}$  is (necessarily) message consuming w.r.t.  $\Sigma_{\mathcal{N}} \cap \Sigma_{\mathcal{M}}$ .*

**Theorem 11** (Incremental Design). *Let  $\mathcal{S}$  and  $\mathcal{T}$  be (necessarily) message consuming compatible. If both  $\mathcal{S}$  and  $\mathcal{T}$  are (necessarily) message consuming, then  $\mathcal{S} \otimes \mathcal{T}$  is (necessarily) message consuming.*

All results holds analogously for asynchronous I/O-Petri nets due to the compositional semantics of MAIOPNs; see Thm. 1.1 An application of incremental design has been discussed in Sect. 2.

### 5.3 Refinement of Message Consuming Systems

An important issue concerns the preservation of the message consuming properties by refinement. We can show that this is indeed the case for modal refinement. For this purpose we need the following two lemmas. The first lemma shows that for any “concrete” reachable state there is a related “abstract” state with the same number of messages on each channel. To prove the preservation of the necessarily consuming property the essential point is to show that for any observationally weakly fair run of a concrete MAIOTS there is a corresponding observationally weakly fair run of the abstract MAIOTS with the same visible actions.

**Lemma 12.** *Let  $\mathcal{S}$  and  $\mathcal{T}$  be two MAIOTSs with channels  $C$  such that  $\mathcal{S} \leq_m^* \mathcal{T}$ . Then for all  $q_{\mathcal{S}} \in \text{Reach}(\mathcal{S})$  there exists  $q_{\mathcal{T}} \in \text{Reach}(\mathcal{T})$  such that  $q_{\mathcal{S}} \leq_m^* q_{\mathcal{T}}$  and  $\text{val}_{\mathcal{S}}(q_{\mathcal{S}}, a) = \text{val}_{\mathcal{T}}(q_{\mathcal{T}}, a)$ , for all  $a \in C$ .*

*Proof.* Let  $q_{\mathcal{S}} \in \text{Reach}(\mathcal{S})$ . This implies there exists  $\sigma \in \Sigma_{\tau}^*$  such that  $q_{\mathcal{S}}^0 \xrightarrow{\sigma} q_{\mathcal{S}}$ . As  $\mathcal{S} \leq_m^* \mathcal{T}$  we know that  $q_{\mathcal{S}}^0 \leq_m^* q_{\mathcal{T}}^0$ . Then by rule (3) and (4) in Def. 2 we get that there exists  $q_{\mathcal{T}} \in \text{Reach}(\mathcal{T})$  such that  $q_{\mathcal{T}}^0 \xrightarrow{\text{obs}(\sigma)} q_{\mathcal{T}}$ .

Now let  $a \in C$ . Note that  $q_{\mathcal{S}}$  and  $q_{\mathcal{T}}$  are both reached by the same sequence of observable actions  $\text{obs}(\sigma)$ . By the fact that silent  $\tau$ -transitions cannot modify the number of messages on channels we get that  $\text{val}(q_{\mathcal{S}}, a) = \text{val}(q_{\mathcal{T}}, a)$ . ■

The next lemma shows that for any observationally weakly fair run of a concrete MAIOTS specification there is a corresponding observationally weakly fair run of the abstract MAIOTS with the same visible actions. This fact is crucial to prove that the necessarily consuming property is preserved by modal refinement. Formally, for a run  $\rho$ , we denote by  $\text{obs}(\rho)$  the sequence of visible actions occurring in  $\rho$ .

**Lemma 13.** *Let  $\mathcal{S}$  and  $\mathcal{T}$  be two MAIOTSs such that  $\mathcal{S} \leq_m^* \mathcal{T}$ . Let  $q_S \in Q_S$  and  $q_T \in Q_T$  such that  $q_S \leq_m^* q_T$ . Then for all  $\rho_S \in \text{owfrun}_S(q_S)$  there exists  $\rho_T \in \text{owfrun}_T(q_T)$  such that  $\text{obs}(\rho_S) = \text{obs}(\rho_T)$ .*

*Proof.* Let  $\rho_S = q_S \xrightarrow{a_1}_S q_S^2 \xrightarrow{a_2}_S q_S^3 \xrightarrow{a_3}_S \dots \in \text{owfrun}_S(q_S)$  with  $a_i \in \Sigma_\tau$ . Then by (3) and (4) in Def. 2 there exists a trace  $\rho_T$  of  $\mathcal{T}$  and states  $q_T^i \in \rho_T$  for  $i \geq 2$  such that  $q_T \xrightarrow{\bar{a}_1}_T q_T^2 \xrightarrow{\bar{a}_2}_T q_T^3 \xrightarrow{\bar{a}_3}_T \dots$  with  $q_S^i \leq_m^* q_T^i$  and

$$\bar{a}_i = \begin{cases} a_i & \text{if } a_i \in \Sigma \\ \epsilon & \text{if } a_i = \tau \end{cases}$$

Obviously  $\text{obs}(\rho_S) = \text{obs}(\rho_T)$ . We have to show that  $\rho_T$  is an observationally weakly fair run of  $\mathcal{T}$ . We distinguish the following cases:

**Case 1:** Assume  $\rho_S$  is finite with last state  $q_S^n$ . Then  $\rho_T$  is finite with last state  $q_T^n$ , and therefore  $\rho_T$  is observationally weakly fair. It remains to show that  $\rho_T$  is a run of  $\mathcal{T}$ , i.e.  $q_T^n$  is a potential deadlock state. Assume that  $q_T^n$  is not a potential deadlock state. Then there exists  $a \in (\Sigma \setminus \text{in})$  such that  $q_T^n \xrightarrow{a}$ . From (1) and (2) of Def. 2 it follows that  $q_S^n \xrightarrow{a}$ . Then  $q_S^n$  is not a potential deadlock state of  $\mathcal{S}$  and hence  $\rho_S$  is not a run of  $\mathcal{S}$ , which is a contradiction.

**Case 2:** Assume  $\rho_S$  is infinite. Then  $\rho_T$  can be infinite, but also finite as explained in case (2.1).

2.1: Assume  $\rho_T$  is finite. Then  $\rho_T$  is observationally weakly fair and it remains to show that  $\rho_T$  is a run of  $\mathcal{T}$ . Since  $\rho_T$  is finite there exists  $n \geq 2$  such that for all  $j \geq n$  we have  $q_T^j = q_T^n$  and hence  $q_S^j \leq_m^* q_T^n$  and  $q_S^j \xrightarrow{\tau}_S q_S^{j+1}$  for all  $j \geq n$ .

We show that  $q_T^n$ , which is the last state of  $\rho_T$ , is a potential deadlock state, and then  $\rho_T$  is a run of  $\mathcal{T}$ . Assume that  $q_T^n$  is not at potential deadlock state of  $\mathcal{T}$ . Then there exists  $a \in (\Sigma \setminus \text{in})$  such that  $q_T^n \xrightarrow{a}$ . Since  $q_S^j \leq_m^* q_T^n$  for all  $j \geq n$  we get, by (1) and (2) in Def. 2, that  $q_S^j \xrightarrow{a}$  for all  $j \geq n$ . On the other hand  $q_S^j \xrightarrow{\tau}_S q_S^{j+1}$  for all  $j \geq n$ . Then  $\rho_S$  is not observational weakly fair which is a contradiction.

2.2: Assume  $\rho_T$  is infinite. Then  $\rho_T$  is a run of  $\mathcal{T}$  and it remains to prove that  $\rho_T$  is observationally weakly fair.

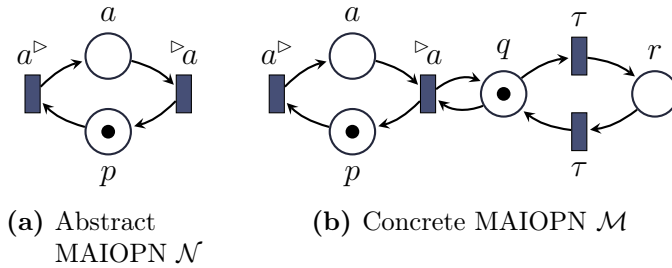
Let  $a \in (\Sigma \setminus \text{in})$  and  $k \geq 2$  such that for all states  $q_T^k \in \rho_T$  occurring after  $q_T^k$  in  $\rho_T$  we have  $q_T^k \xrightarrow{a}$ . For each state  $q_S^k \in \rho_S$  occurring after  $q_S^k$  in  $\rho_S$  there exists a state

$q'_\mathcal{T} \in \rho_\mathcal{T}$  as above such that  $q'_\mathcal{S} \leq_m^* q'_\mathcal{T}$ . Hence, by (1) and (2) in Def. 2,  $q'_\mathcal{S} \xRightarrow{a}$ . Since  $\rho_\mathcal{S}$  is observationally weakly fair  $a$  occurs infinitely often in  $\rho_\mathcal{S}$  and hence also in  $\rho_\mathcal{T}$  since  $\text{obs}(\rho_\mathcal{S}) = \text{obs}(\rho_\mathcal{T})$ . Therefore  $\rho_\mathcal{T}$  is observationally weakly fair. ■

**Theorem 14.** *Let  $\mathcal{S}$  and  $\mathcal{T}$  be two MAIOTs with channels  $C$  such that  $\mathcal{S} \leq_m^* \mathcal{T}$ . Let  $B \subseteq C$ . If  $\mathcal{T}$  is (necessarily) message consuming w.r.t.  $B$ , then  $\mathcal{S}$  is (necessarily) message consuming w.r.t.  $B$ . By definition, the theorem propagates to MAIOPNs.*

*Proof.* Let  $a \in B$  and  $q_\mathcal{S} \in \text{Reach}(\mathcal{S})$  such that  $\text{val}_\mathcal{S}(q_\mathcal{S}, a) > 0$ . By Lem. 12 it follows that there exists  $q_\mathcal{T} \in \text{Reach}(\mathcal{T})$  such that  $q_\mathcal{S} \leq_m^* q_\mathcal{T}$  and  $\text{val}_\mathcal{T}(q_\mathcal{T}, a) = \text{val}_\mathcal{S}(q_\mathcal{S}, a) > 0$ .

1. We assume that  $\mathcal{T}$  is message consuming w.r.t.  $B$ . Then there exist  $q'_\mathcal{T}, q''_\mathcal{T} \in Q_\mathcal{T}$  and  $\sigma \in (\Sigma \setminus \text{in})^*$  such that  $q_\mathcal{T} \xRightarrow{\sigma}_\mathcal{T} q'_\mathcal{T} \xrightarrow{\triangleright a}_\mathcal{T} q''_\mathcal{T}$ . From (1) and (2) in Def. 2 it follows that there exist  $q'_\mathcal{S}, q''_\mathcal{S} \in Q_\mathcal{S}$  such that  $q_\mathcal{S} \xRightarrow{\sigma}_\mathcal{S} q'_\mathcal{S} \xrightarrow{\triangleright a}_\mathcal{S} q''_\mathcal{S}$ . Hence  $\mathcal{S}$  is message consuming w.r.t.  $B$ .
2. We assume that  $\mathcal{T}$  is necessarily message consuming w.r.t.  $B$ .  
Let  $\rho_\mathcal{S} \in \text{owfrun}_\mathcal{S}(q_\mathcal{S})$ . By Lem. 13 we get that there exists  $\rho_\mathcal{T} \in \text{owfrun}_\mathcal{S}(q_\mathcal{T})$  with  $\text{obs}(\rho_\mathcal{T}) = \text{obs}(\rho_\mathcal{S})$ . By assumption we know that  $\triangleright a \in \rho_\mathcal{T}$ . Since  $\text{obs}(\rho_\mathcal{T}) = \text{obs}(\rho_\mathcal{S})$  we get that  $\triangleright a \in \rho_\mathcal{S}$ . ■



**Figure 6:** Necessarily consuming and modal refinement

*Example 11.* The nets in Fig. 6 show an abstract MAIOPN  $\mathcal{N}$  and a concrete MAIOPN  $\mathcal{M}$  with silent  $\tau$ -transitions. Both nets have a single channel place  $a$ . Obviously,  $\mathcal{M} \leq_m^* \mathcal{N}$  is a modal refinement. It is also clear that  $\mathcal{N}$  is necessarily message consuming. By Thm. 14,  $\mathcal{M}$  is necessarily message consuming as well. Indeed a diverging run of  $\mathcal{M}$  which always executes  $\tau$  from a certain state on is not observationally weakly fair (since  $\triangleright a$  is then always observationally enabled by must-transitions but never taken) and therefore needs not to be considered. This shows also why weakly fair runs are not

appropriate here since a diverging run of  $\mathcal{M}$  is weakly fair (it always visits a state in which  $\triangleright a$  is not immediately enabled) but would not consume.

As a consequence of Thm. 3.3 and Thm. 14 our theory supports the principle of independent implementability in the sense of [AH05]. This fact is applied in Sect. 2 to obtain the global refinement (3) in Fig. 5 from the local refinement (2).

**Corollary 15** (Independent Implementability). *Let  $\mathcal{S}, \mathcal{T}, \mathcal{E}$  and  $\mathcal{F}$  be MAIOTs. If  $\mathcal{T}$  and  $\mathcal{F}$  are (necessarily) message consuming compatible and  $\mathcal{S} \leq_m^* \mathcal{T}$  and  $\mathcal{E} \leq_m^* \mathcal{F}$ , then  $\mathcal{S}$  and  $\mathcal{E}$  are (necessarily) message consuming compatible and  $\mathcal{S} \otimes \mathcal{E} \leq_m^* \mathcal{T} \otimes \mathcal{F}$ . This holds analogously for MAIOPNs.*

## 6 Conclusion and Future Work

We have developed a fully integrated approach for the design of asynchronously composed component systems based on the formalism of MAIOPNs. Our approach ensures that the communication properties are preserved by asynchronous composition and by modal refinement, the basic ingredients of the design process. Several continuations of this work are possible. First, it would be interesting to see how our approach works in larger case studies and concrete applications.

The “Assume/Guarantee” approach is a standard way to substitute a component by a behavioural interface in order to make easier the compositional verification. We plan to investigate how this approach can be integrated in our framework. Finally, broadcasting is an appropriate communication mechanism in the asynchronous environment. So it would be interesting to investigate how our approach can be adapted to this communication operator.

# Bibliography

- [Ace+07] L. Aceto, A. Ingólfssdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007, pp. 1–300.
- [Ace+11] L. Aceto, I. Fábregas, D. de Frutos-Escrig, A. Ingólfssdóttir, and M. Palomino. “Graphical representation of covariant-contravariant modal formulae”. In: *EXPRESS*. Vol. 64. EPTCS. 2011, pp. 1–15.
- [AH05] L. d. Alfaro and T. A. Henzinger. “Interface-Based Design”. In: *Engineering Theories of Software Intensive Systems*. Ed. by M. Broy, J. Grünbauer, D. Harel, and T. Hoare. Vol. 195. NATO Science Series. Springer Netherlands, 2005, pp. 83–104. DOI: 10.1007/1-4020-3532-2\_3.
- [AJ96] P. A. Abdulla and B. Jonsson. “Verifying Programs with Unreliable Channels”. In: *Information and Computation* 127.2 (1996), pp. 91–101. DOI: 10.1006/inco.1996.0053.
- [Ant+08] A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. “20 Years of Modal and Mixed Specifications”. In: *Bulletin of the EATCS no. 95* (2008), pp. 94–129.
- [Bal+01] P. Baldan, A. Corradini, H. Ehrig, and R. Heckel. “Compositional Modelling of Reactive Systems Using Open Nets”. English. In: *CONCUR 2001 – Concurrency Theory*. Ed. by K. Larsen and M. Nielsen. Vol. 2154. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 502–518. DOI: 10.1007/3-540-44685-0\_34.
- [Bau+11] S. S. Bauer, U. Fahrenberg, L. Juhl, K. G. Larsen, A. Legay, and C. R. Thrane. “Quantitative Refinement for Weighted Modal Transition Systems”. In: *MFCS*. Vol. 6907. LNCS. Springer, 2011, pp. 60–71.
- [BBL08] P. Bouyer, E. Brinksma, and K. G. Larsen. “Optimal infinite scheduling for multi-priced timed automata”. In: *Formal Methods in System Design* 32.1 (2008), pp. 3–23.
- [BBO12] S. Basu, T. Bultan, and M. Ouederni. “Synchronizability for Verification of Asynchronously Communicating Systems”. In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by V. Kuncak and A. Rybalchenko. Vol. 7148. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 56–71. DOI: 10.1007/978-3-642-27940-9\_5.
- [BCD02] M. Bernardo, P. Ciancarini, and L. Donatiello. “Architecting families of software systems with process algebras”. In: *ACM Trans. Softw. Eng. Methodol.* 11.4 (Oct. 2002), pp. 386–426. DOI: 10.1145/606612.606614.

- [BCK11] N. Beneš, I. Čerá, and J. Křetínský. “Modal Transition Systems: Composition and LTL Model Checking”. In: *Automated Technology for Verification and Analysis*. Ed. by T. Bultan and P.-A. Hsiung. Vol. 6996. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 228–242. DOI: 10.1007/978-3-642-24372-1\_17.
- [BDK01] E. Best, R. R. Devillers, and M. Koutny. *Petri net algebra*. Springer, 2001, pp. I–XI, 1–378.
- [BE01] O. Burkart and J. Esparza. “More Infinite Results”. In: *Current Trends in Theoretical Computer Science*. 2001, pp. 480–503.
- [Ben+11a] N. Beneš, J. Křetínský, K. Larsen, M. Møller, and J. Srba. “Parametric Modal Transition Systems”. In: *Proceedings of ATVA’11*. Vol. 6996. LNCS. Springer-Verlag, 2011, pp. 275–289.
- [Ben+11b] N. Beneš, J. Křetínský, K. G. Larsen, M. H. H. Møller, and J. Srba. *Parametric Modal Transition Systems*. Technical report FIMU-RS-2011-03. Faculty of Informatics, Masaryk University, Brno, 2011.
- [Ber+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.
- [BGS92] J. L. Balcazar, J. Gabarró, and M. Santha. “Deciding bisimilarity is P-complete”. In: *Formal aspects of computing* 4.6 A (1992), pp. 638–648.
- [BGV91] W. Brauer, R. Gold, and W. Vogler. “A survey of behaviour and equivalence preserving refinements of petri nets”. In: *Advances in Petri Nets 1990*. Ed. by G. Rozenberg. Vol. 483. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, pp. 1–46. DOI: 10.1007/3-540-53863-1\_19.
- [BK11] N. Beneš and J. Křetínský. “Process Algebra for Modal Transition Systems”. In: *Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS’10) – Selected Papers*. Ed. by L. Matyska, M. Kozubek, T. Vojnar, P. Zemčík, and D. Antos. Vol. 16. OpenAccess Series in Informatics (OASICS). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011, pp. 9–18. DOI: 10.4230/OASICS.MEMICS.2010.9.
- [BK12] N. Beneš and J. Křetínský. “Modal Process Rewrite Systems”. In: *Theoretical Aspects of Computing – ICTAC 2012*. Ed. by A. Roychoudhury and M. D’Souza. Vol. 7521. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 120–135. DOI: 10.1007/978-3-642-32943-2\_9.
- [BL92] G. Boudol and K. G. Larsen. “Graphical Versus Logical Specifications”. In: *Theor. Comput. Sci.* 106.1 (1992), pp. 3–20.
- [Blo+09] R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann. “Synthesizing Robust Systems”. In: *Proc. of FMCAD09*. IEEE, 2009, pp. 85–92.



- [BLR04] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. “Priced Timed Automata: Algorithms and Applications”. In: *FMCO*. Vol. 3657. LNCS. Springer, 2004, pp. 162–182.
- [Bou+08] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. “Infinite Runs in Weighted Timed Automata with Energy Constraints”. In: *FORMATS*. Vol. 5215. LNCS. Springer, 2008, pp. 33–47.
- [BZ83] D. Brand and P. Zafropulo. “On Communicating Finite-State Machines”. In: *J. ACM* 30.2 (Apr. 1983), pp. 323–342. DOI: 10.1145/322374.322380.
- [Car87] H. Carstensen. “Decidability questions for fairness in Petri nets”. In: *STACS 87*. Ed. by F. Brandenburg, G. Vidal-Naquet, and M. Wirsing. Vol. 247. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, pp. 396–407. DOI: 10.1007/BFb0039622.
- [CD10] K. Chatterjee and L. Doyen. “Energy Parity Games”. In: *ICALP (2)*. Ed. by S. Abramsky, C. Gavoille, C. Kirchner, F. M. auf der Heide, and P. G. Spirakis. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 599–610.
- [CE82] E. M. Clarke and E. A. Emerson. “Design and synthesis of synchronization skeletons using branching time temporal logic”. In: *Logics of Programs*. Ed. by D. Kozen. Vol. 131. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1982, pp. 52–71. DOI: 10.1007/BFb0025774.
- [CF05] G. Cécé and A. Finkel. “Verification of programs with half-duplex communication”. In: *Inf. Comput.* 202.2 (Nov. 2005), pp. 166–190. DOI: 10.1016/j.ic.2005.05.006.
- [Cha+03] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. “Resource Interfaces”. In: *EMSOFT*. Ed. by R. Alur and I. Lee. Vol. 2855. Lecture Notes in Computer Science. Springer, 2003, pp. 117–133.
- [Cla+10] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. “Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines”. In: *32nd International Conference on Software Engineering, ICSE 2010, May 2-8, 2010, Cape Town, South Africa, Proceedings*. ACM, 2010, pp. 335–344.
- [CN01] P. C. Clements and L. Northrop. *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.
- [EC80] E. A. Emerson and E. M. Clarke. “Characterizing correctness properties of parallel programs using fixpoints”. In: *Automata, Languages and Programming*. Ed. by J. Bakker and J. Leeuwen. Vol. 85. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1980, pp. 169–181. DOI: 10.1007/3-540-10003-2\_69.

- [EHH12] D. Elhog-Benzina, S. Haddad, and R. Hennicker. “Refinement and Asynchronous Composition of Modal Petri Nets”. In: *Transactions on Petri Nets and Other Models of Concurrency V*. Ed. by K. Jensen, S. Donatelli, and J. Kleijn. Vol. 6900. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 96–120. DOI: 10.1007/978-3-642-29072-5\_4.
- [EJ89] D. Escrig and C. Johnen. *Decidability of Home Space Property*. Rapports de recherche // PARIS-SUD UNIV LRI. Université de Paris-Sud, Centre d’Orsay, Laboratoire de Recherche en Informatique, 1989.
- [EM79] A. Ehrenfeucht and J. Mycielski. “Positional strategies for mean payoff games”. In: *International Journal of Game Theory* 8 (2 1979), pp. 109–113. DOI: 10.1007/BF01768705.
- [EN94] J. Esparza and M. Nielsen. “Decidability Issues for Petri Nets - a survey”. In: *Bulletin of the EATCS* 52 (1994), pp. 244–262.
- [FS08] H. Fecher and H. Schmidt. “Comparing Disjunctive Modal Transition Systems with an One-Selecting Variant”. In: *J. of Logic and Alg. Program.* 77.1-2 (2008), pp. 20–39.
- [GB05] L. Gomes and J. P. Barros. “Structuring and composability issues in Petri nets modeling”. In: *IEEE Trans. Industrial Informatics* 1.2 (2005), pp. 112–123. DOI: 10.1109/TII.2005.844433.
- [GHJ01] P. Godefroid, M. Huth, and R. Jagadeesan. “Abstraction-Based Model Checking Using Modal Transition Systems”. In: *Proc. CONCUR’01*. Vol. 2154. LNCS. Springer, 2001, pp. 426–440.
- [GLS08] A. Gruler, M. Leucker, and K. D. Scheidemann. “Modeling and Model Checking Software Product Lines”. In: *FMOODS*. Ed. by G. Barthe and F. S. de Boer. Vol. 5051. Lecture Notes in Computer Science. Springer, 2008, pp. 113–131. DOI: 10.1007/978-3-540-68863-1\_8.
- [GRB07] P. Ganty, J.-F. Raskin, and L. Begin. “From Many Places to Few: Automatic Abstraction Refinement for Petri Nets”. In: *Petri Nets and Other Models of Concurrency – ICATPN 2007*. Ed. by J. Kleijn and A. Yakovlev. Vol. 4546. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 124–143. DOI: 10.1007/978-3-540-73094-1\_10.
- [GS66] S. Ginsburg and E. H. Spanier. “Semigroups, Presburger Formulas, and Languages”. In: *Pacific Journal of Mathematics* 16 (1966), pp. 285–296.
- [HHM13a] S. Haddad, R. Hennicker, and M. H. Møller. “Channel Properties of Asynchronously Composed Petri Nets”. In: *Application and Theory of Petri Nets and Concurrency*. Ed. by J.-M. Colom and J. Desel. Vol. 7927. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 369–388. DOI: 10.1007/978-3-642-38697-8\_20.

- [HHM13b] S. Haddad, R. Hennicker, and M. H. Møller. *Channel Properties of Asynchronously Composed Petri Nets*. Tech. rep. Research Report LSV-13-05. Laboratoire Spécification et Vérification, ENS Cachan, France, 2013.
- [HJK10] R. Hennicker, S. Janisch, and A. Knapp. “Refinement of Components in Connection-Safe Assemblies with Synchronous and Asynchronous Communication”. In: *Foundations of Computer Software. Future Trends and Techniques for Development*. Ed. by C. Choppy and O. Sokolsky. Vol. 6028. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 154–180. DOI: 10.1007/978-3-642-12566-9\_9.
- [HJS01] M. Huth, R. Jagadeesan, and D. A. Schmidt. “Modal Transition Systems: A Foundation for Three-Valued Program Analysis”. In: *Proc. of ESOP’01*. Vol. 2028. LNCS. Springer, 2001, pp. 155–169.
- [HK11] R. Hennicker and A. Knapp. “Modal Interface Theories for Communication-Safe Component Assemblies”. In: *Theoretical Aspects of Computing – ICTAC 2011*. Ed. by A. Cerone and P. Pihlajasaari. Vol. 6916. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 135–153. DOI: 10.1007/978-3-642-23283-1\_11.
- [HL89] H. Hüttel and K. G. Larsen. “The Use of Static Constructs in A Modal Process Logic”. In: *Logic at Botik*. Ed. by A. R. Meyer and M. A. Taitlin. Vol. 363. Lecture Notes in Computer Science. Springer, 1989, pp. 163–180.
- [HM85] M. Hennessy and R. Milner. “Algebraic Laws for Nondeterminism and Concurrency”. In: *J. ACM* 32.1 (1985), pp. 137–161.
- [Jan90] P. Jancar. “Decidability of a Temporal Logic Problem for Petri Nets”. In: *Theoretical Computer Science* 74.1 (1990), pp. 71–93. DOI: 10.1016/0304-3975(90)90006-4.
- [Jan95] P. Jancar. “Undecidability of Bisimilarity for Petri Nets and Some Related Problems”. In: *Theoretical Computer Science* 148.2 (1995), pp. 281–301. DOI: 10.1016/0304-3975(95)00037-W.
- [JLS12] L. Juhl, K. G. Larsen, and J. Srba. “Modal transition systems with weight intervals”. In: *The Journal of Logic and Algebraic Programming* 81.4 (2012), pp. 408–421. DOI: 10.1016/j.jlap.2012.03.008.
- [Kel76] R. M. Keller. “Formal Verification of Parallel Programs”. In: *Commun. ACM* 19.7 (July 1976), pp. 371–384. DOI: 10.1145/360248.360251.
- [Kin97] E. Kindler. “A Compositional Partial Order Semantics for Petri Net Components”. In: *ICATPN*. Ed. by P. Azéma and G. Balbo. Vol. 1248. Lecture Notes in Computer Science. Springer, 1997, pp. 235–252.
- [Lar88] K. G. Larsen. “Proof System for Hennessy-Milner Logic with Recursion”. In: *CAAP*. 1988, pp. 215–230. DOI: 10.1007/BFb0026106.
- [Lar89] K. G. Larsen. “Modal Specifications”. In: *Automatic Verification Methods for Finite State Systems*. 1989, pp. 232–246.

## Bibliography

- [LMW07] N. Lohmann, P. Massuthe, and K. Wolf. “Operating Guidelines for Finite-State Services”. In: *ICATPN*. Ed. by J. Kleijn and A. Yakovlev. Vol. 4546. Lecture Notes in Computer Science. Springer, 2007, pp. 321–341.
- [LNW07] K. G. Larsen, U. Nyman, and A. Wasowski. “On Modal Refinement and Consistency”. In: *Proc. of CONCUR’07*. Vol. 4703. LNCS. Springer, 2007, pp. 105–119.
- [LT88] K. G. Larsen and B. Thomsen. “A Modal Process Logic”. In: *Proceedings, Third Annual Symposium on Logic in Computer Science, 5-8 July 1988, Edinburgh, Scotland, UK*. IEEE Computer Society, 1988, pp. 203–210.
- [LX90] K. G. Larsen and L. Xinxin. “Equation Solving Using Modal Transition Systems”. In: *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 1990, pp. 108–117.
- [May81] E. W. Mayr. “An algorithm for the general Petri net reachability problem”. In: *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. STOC ’81. Milwaukee, Wisconsin, USA: ACM, 1981, pp. 238–246. DOI: 10.1145/800076.802477.
- [Mol96] F. Moller. “Infinite Results”. In: *Proceedings of the 7th International Conference on Concurrency Theory*. CONCUR ’96. London, UK, UK: Springer-Verlag, 1996, pp. 195–216.
- [NNN08] S. Nanz, F. Nielson, and H. R. Nielson. “Modal Abstractions of Concurrent Behaviour”. In: *Proc. of SAS’08*. Vol. 5079. LNCS. Springer, 2008, pp. 159–173.
- [Pap94] C. H. Papadimitriou. *Computational complexity*. Reading, MA, USA: Addison-Wesley Publishing Co., Inc., 1994.
- [Pet62] C. A. Petri. “Kommunikation mit Automaten”. PhD thesis. Darmstadt University of Technology, Germany, 1962.
- [Pet81] J. Peterson. *Petri net theory and the modeling of systems*. Foundations of Philosophy Series. Prentice-Hall, 1981.
- [Plo81] G. D. Plotkin. *A Structural Approach to Operational Semantics*. Technical Report DAIMI FN-19. Aarhus, Denmark: Computer Science Department, Aarhus University, 1981.
- [QS82] J. P. Queille and J. Sifakis. “Specification and verification of concurrent systems in CESAR”. In: *International Symposium on Programming*. Ed. by M. Dezani-Ciancaglini and U. Montanari. Vol. 137. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1982, pp. 337–351. DOI: 10.1007/3-540-11494-7\_22.
- [Rac+09] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. “Why Are Modalities Good for Interface Theories?” In: *ACSD*. IEEE, 2009, pp. 119–127. DOI: 10.1109/ACSD.2009.22.

- [Rac07] J.-B. Raclet. “Residual for Component Specifications”. In: *Proc. of the 4th International Workshop on Formal Aspects of Component Software*. 2007.
- [Rac78] C. Rackoff. “The Covering and Boundedness Problems for Vector Addition Systems”. In: *Theoretical Computer Science* 6 (1978), pp. 223–231. DOI: 10.1016/0304-3975(78)90036-1.
- [Rei09] W. Reisig. “Simple Composition of Nets”. In: *Proceedings of the 30th International Conference on Applications and Theory of Petri Nets*. PETRI NETS ’09. Paris, France: Springer-Verlag, 2009, pp. 23–42. DOI: 10.1007/978-3-642-02424-5\_4.
- [Sch02] P. Schnoebelen. “Verifying lossy channel systems has nonprimitive recursive complexity”. In: *Information Processing Letters* 5 (2002), pp. 251–261. DOI: 10.1016/S0020-0190(01)00337-4.
- [SI94] B. Steffen and A. Ingólfssdóttir. “Characteristic Formulae for Processes with Divergence”. In: *Inf. Comput.* 110.1 (1994), pp. 149–163.
- [SJ05] Z. Sawa and P. Jančar. “Behavioural Equivalences on Finite-State Systems are PTIME-hard”. In: *Computing and informatics* 24.5 (2005), pp. 513–528.
- [SM91] Y. Souissi and G. Memmi. “Composition of nets via a communication medium”. In: *Advances in Petri Nets 1990*. Ed. by G. Rozenberg. Vol. 483. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, pp. 457–470. DOI: 10.1007/3-540-53863-1\_34.
- [Sou91] Y. Souissi. “On liveness preservation by composition of nets via a set of places”. In: *Advances in Petri Nets 1991*. Ed. by G. Rozenberg. Vol. 524. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, pp. 277–295. DOI: 10.1007/BFb0019979.
- [SV07] M. Schäfer and W. Vogler. “Component refinement and CSC-solving for STG decomposition”. In: *Theor. Comput. Sci.* 388.1-3 (2007), pp. 243–266. DOI: 10.1016/j.tcs.2007.08.005.
- [SV12] C. Stahl and W. Vogler. “A trace-based service semantics guaranteeing deadlock freedom”. English. In: *Acta Informatica* 49.2 (2012), pp. 69–103. DOI: 10.1007/s00236-012-0151-5.
- [SW09] C. Stahl and K. Wolf. “Deciding service composition and substitutability using extended operating guidelines”. In: *Data Knowl. Eng.* 68.9 (Sept. 2009), pp. 819–833. DOI: 10.1016/j.datak.2009.02.012.
- [UC04] S. Uchitel and M. Chechik. “Merging partial behavioural models”. In: *Proc. of FSE’04*. ACM, 2004, pp. 43–52.
- [VJ85] R. Valk and M. Jantzen. “The residue of vector sets with applications to decidability problems in petri nets”. In: *Advances in Petri Nets 1984*. Ed. by G. Rozenberg. Vol. 188. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1985, pp. 234–258. DOI: 10.1007/3-540-15204-0\_14.

## *Bibliography*

- [Xin92] L. Xinxin. *Specification and Decomposition in Concurrency*. Report (Aalborg universitetscenter. Afdeling for matematik og datalogi). University of Aalborg, Institute for Electronic Systems, Department of Mathematics and Computer Science, 1992.
- [ZP96] U. Zwick and M. Paterson. “The Complexity of Mean Payoff Games on Graphs”. In: *Theoretical Computer Science* 158 (1996), pp. 343–359.