



Quantitative Message Sequence Graphs

Diploma Thesis in Computer Science

Hussein Hamid Baagil

Matriculation Number: 280303

September 30, 2012

Thesis Supervisor: Prof. Dr. Ir. Joost-Pieter Katoen

Second Examiner: Prof. Dr. Thomas Noll

Declaration

I hereby declare that I am the sole author of this thesis. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions.

Aachen, 30 September 2012

Hussein Hamid Baagil

Acknowledgements

First of all I would like to thank my supervisor Prof. Dr. Ir. Joost-Pieter Katoen for his patience, guidance and time that he took in discussing this work. I would also like to thank Prof. Dr. Thomas Noll, my second examiner.

Further, I would like to thank my mother for her understanding and making all this possible. Francesca for her patience and support. Thank you for those who helped me in correcting my work, especially Rob. And all my family and friends who have supported during this time.

Contents

1	Introduction	8
1.1	Preliminary	8
1.2	Thesis Contribution	9
1.2.1	Thesis Outline	10
2	Basic Concepts	11
2.1	Message Sequence Charts	11
2.2	Message Sequence Graphs	15
2.3	Propositional Dynamic Logic	17
3	Message Sequence Charts With Unreliable Channels	20
3.1	Introduction	20
3.2	Probabilistic Message Sequence Charts	21
3.3	Generating posets of a pMSC	29
3.3.1	The Algorithm	31
3.3.2	Complexity of generating a set of posets and the size of it	38
3.4	Probability measure on pMSC	41
4	Composing Probabilistic MSCs	46
4.1	Introduction	46
4.2	Probabilistic Message Sequence Graphs	47
4.3	Markov Chain	55
4.4	From pMSG to Markov Chain	58
5	Associating Logic to pMSCs	62
5.1	Introduction	62
5.2	Probabilistic Propositional Dynamic Logic	63

5.2.1	Syntax of PPDL	63
5.2.2	Semantics of PPDL	64
5.3	Model Checking	71
5.3.1	Model Checking of PDL over MSCs	72
6	An Automaton Model for pMSCs	76
6.1	Lossy Communication Finite-State Machines	77
6.1.1	Syntax of LCFM	78
6.1.2	Semantics of LCFM	80
6.2	Probabilistic Lossy Communication Finite-State Machine	82
6.2.1	Syntax of PLCFM	83
6.2.2	Semantics of PLCFM	84
6.3	An Approach in Realizing pMSCs	87
7	Conclusion	88
7.1	Summary	88
7.2	Future Work	89

List of Figures

2.1	Example of MSC M	14
3.1	Examples of pMSCs	22
3.2	The empty pMSC M_\emptyset	23
3.3	$full(M_1)$	26
3.4	$full(M_2)$	26
3.5	pMSC M_1 with a lost event e_6	28
3.6	30
3.7	An example of a pMSC M with properties 1 and 2.	39
3.8	An example of a pMSC contradicting property (1).	39
3.9	An example of a pMSC contradicting property (2).	40
4.1	pMSG G_1 over $\mathcal{M} = \{M_1, M_2, M_3, M_4, M_\emptyset\}$	49
4.2	Concatenation of M_2 and M_3	54
4.3	A Markov Chain for a simple weather model.	57
4.4	A Markov Chain \mathcal{MC}_1 from G_1 with respect to the scheduler \mathfrak{A}_1	61
4.5	A Markov Chain \mathcal{MC}_2 from G_1 with respect to the scheduler \mathfrak{A}_2	61
5.1	pMSC M	67
6.1	Example for a LCFM \mathcal{A}_{pq}	79
6.2	Possible channel contents of LCFM \mathcal{A}_{pq}	80

Chapter 1

Introduction

1.1 Preliminary

Message Sequence Charts (MSCs) have been standardized (*Recommendations*) by the International Telecommunication Union ITU-T as a graphical and textual language used to describe and specify the interaction of components in a system [13]. Interactions between components (which in this thesis are represented as processes) in a MSC are specified as a message exchange. The language of MSCs can be represented in a formal form which describes MSCs purely textually, as well as a visual form providing a graphical representation of MSCs. The intention of a textual form of a MSC is to use it for exchanging between tools and also for automatic formal analysis [14]. The graphical representation of a MSC is very useful since one can depict the order in which communications and events take place [14].

At the beginning, MSCs were principally considered to be in relation with the *Specification and Description Language* SDL. At this point MSCs were used to define requirements for SDL diagrams due to their natural and transparent way of presenting a description of system behavior [11]. After further consideration it has turned out that the standardization of MSCs in relation to SDL has gone beyond SDL guidelines. This led to a look at MSCs as a specification language independent from SDL [12].

MSCs are mainly applied to specify communication behavior of real-time systems (in particular communication switching systems) [13]. Beyond that,

MSCs have gained wide acceptance in software developments. In the early stage of software engineering, MSCs can be used for requirement specifications. Furthermore, MSCs have also been applied for simulation and validation, visualization of test case, documentation of real-time systems, feature interaction detection and much more.

A set of MSCs can be combined into a Message Sequence Graph (MSG). MSGs (also commonly known as High-Level Message Sequence Charts) are nondeterministic finite state machines without input alphabets, where the nodes are represented by MSCs. Where MSCs are used to specify a single episode of message interactions, MSGs are used to specify a set of episodes of message interactions.

There are several approaches to defining a mathematical logic that can be used to describe properties of MSCs. One of them is Propositional Dynamic Logic (PDL) [8]. PDL was first defined to associate a modality to every computer program of a programming language [10].

1.2 Thesis Contribution

We consider MSCs as a suitable language to specify a partial order of sequences of events that occurs in a system. In this thesis, events in a MSC represent the action of sending a message or the action of receiving a message. The process of sending and receiving a message is always associated with a channel. We assume that, for each communication direction of message exchange between processes, one channel is required. Hence, for each process pair there are at most two channels for message exchange (one for each communication direction). MSCs are based on the assumption that the channels used for message exchange are reliable. This means that the channels will never be considered as fault in any possible performance.

In nature, message exchange in every possible system model could always exhibit faulty behavior. Thus, we believe that predefining the channels associated to a MSC as being reliable does not reflect the nature of the characteristics of the components in a system. In this thesis we would like to extend the MSCs by allowing the channels to be faulty. It will be assumed that every message which is in transit in a channel may be lost from the chan-

nel. This will be realized by associating a probability value to every channel that defines the probability of losing a message in the channel. Through this probabilistic extension on MSCs we will also be able to specify quantitative system behaviors.

This extension of MSCs will also be applied to MSGs and PDL. MSGs will be extended in such a way that it will exhibit nondeterministic and also probabilistic behaviors. Each vertex in the extended MSG will be associated to a probabilistic MSC. The probabilistic extension of PDL will allow for the expression of probabilities of a sequence of events. Probabilistic PDL will allow us to fully express properties of probabilistic MSCs.

Last, we will propose a mathematical system that can be used to describe the behavior of probabilistic MSCs. This model is based on *Communicating Finite-State Machines* (CFM) which is a set of local automata communicating by sending and receiving messages to another.

1.2.1 Thesis Outline

First, the thesis will present basic concepts in Chapter 2, which essentially define the basic models that are the backbone of the models which will be presented subsequently. We will encounter the basic models of MSCs and MSGs. Further, the logic PDL will be defined, which can be used to specify properties of MSCs (and also MSGs). In Chapter 3, the basic MSCs will be extended into probabilistic MSCs (pMSCs), where each channel is associated with a probability value. An algorithm whose assignment is to generate all possible scenarios that a pMSC can generate will be presented followed by describing the probabilistic measurement on pMSCs. Combining different pMSCs into a probabilistic MSG (pMSG) will be shown in Chapter 4. This chapter will define the syntax and semantics of pMSGs. The logic PDL, which is used to specify properties of MSCs, will be extended in Chapter 5 with a probabilistic operator. In Chapter 6 we will define an automata model for pMSCs which is based on channel systems. Finally, Chapter 7 will summarize the work and point out some challenges for future works.

Chapter 2

Basic Concepts

This chapter will present the basic concepts for the following chapters. These concepts are standards which are widely used in describing communications models. The aim of this chapter is to give a solid understanding of these standards, so that the transition to the subsequent chapters, which mainly will extend the concepts here, goes easily.

The outline of the chapter is the following: we will start by defining Message Sequence Charts (MSCs). Next, an extended model for MSCs, Message Sequence Graphs (MSGs), will be defined subsequently. This will be followed by defining a mathematical language which will be used to describe MSCs and MSGs properties, which is called Propositional Dynamic Logic (PDL).

2.1 Message Sequence Charts

Abstractly speaking, MSCs are models used to describe the interaction between instances, which in our case are processes. These interactions are mainly described by message exchange between processes. Each process in a MSC can send or receive messages to or from other processes. Messages are sent and received through channels which are assumed to be reliable. The reliability of the channels means that no messages will ever get lost from the channel. Furthermore the channels have a FIFO property, which means that messages will be received in the same order in which they have been sent. The occurrence of send and receive actions in a process are denoted by events. Hence, events in a process are divided into send and receive events.

Beside having a formal definition, MSCs can also be represented visually, which we call here MSCs graphical representation.

Preliminary, before defining MSCs, we have to become familiar with the concept of the *partial order set*, which plays an important role in MSCs (and also in its probabilistic variant pMSCs in the next chapter). A partial order over a set defines a relation between the elements of the set that follows certain rules.

Definition 2.1.1. (*Partial Order*)

A partial order over a set of events E is a binary relation $< \subseteq E \times E$ which is irreflexive, transitive and antisymmetric. $\forall e_1, e_2, e_3 \in E$:

- $\neg(e_1 < e_1)$,
- $(e_1 < e_2) \wedge (e_2 < e_3) \implies (e_1 < e_3)$,
- $(e_1 < e_2) \implies \neg(e_2 < e_1)$.

■

In MSCs, a partial order will be defined over the set of events. From the partial order of a MSC one can read the possible order of its events. For the rest of the paper, a partial order $<$ over a set of events E will be denoted as a tuple $(E, <)$ and will be abbreviated as *poset*.

Definition 2.1.2. (*Isomorphism on Posets*)

Let $(E, <)$ and $(E', <')$ be posets. $(E, <)$ and $(E', <')$ are isomorphic to one another if there exists a bijective function $f : E \rightarrow E'$ such that $\forall e, e' \in E$:

$$e < e' \implies f(e) <' f(e').$$

■

Definition 2.1.3. (*Message Sequence Charts*)

A Message Sequence chart (MSC) is a structure $M = (\mathcal{P}, \Sigma, Act, E, l, m, <)$ such that:

- \mathcal{P} is a finite set of processes and $|\mathcal{P}| \geq 2$,
- Σ is a finite message alphabet,
- Act is a finite set of actions of the form $p!q(a)$ or $q?p(a)$ ($p, q \in \mathcal{P}$, $p \neq q$, $a \in \Sigma$). Sending a message a from process p to process q is denoted by the action $p!q(a)$ and receiving a message a in process q from process p is denoted by the action $q?p(a)$,
- E is a finite set of events which takes place in the processes.

$$E = E_! \uplus E_?$$

where $E_!$ is a set of send events and $E_?$ is a set of receive events,

- $l : E \rightarrow Act$ is a labeling function that assigns to each event an action from the set Act ,
- $m : E_! \rightarrow E_?$ is a matching function that assigns to each send event its corresponding receive event,
- $(E, <)$ is a poset, where

$$< = \left(\left(\bigcup_{p \in \mathcal{P}} <_p \right) \cup <_{comm} \right)^*$$

$<_p$ is a total order over the events on a process p (which is top to bottom) and $<_{comm}$ is a communication order over events (a send event with its corresponding receive event) and $*$ is the transitive closure.

■

The graphical representation of a MSC is depicted by a set of vertical lines arranged one after another, which represent the processes. The vertical lines start with a *process head symbol*, with a process name written above it, and ends with a *process end symbol*. Interactions between the processes are denoted by a horizontal line between the vertical lines. An arrow defines the direction of the interaction. An event is located on each start and end of a horizontal line which denotes the send and receive events in the processes. An example for a MSC M can be depicted from Figure 2.1

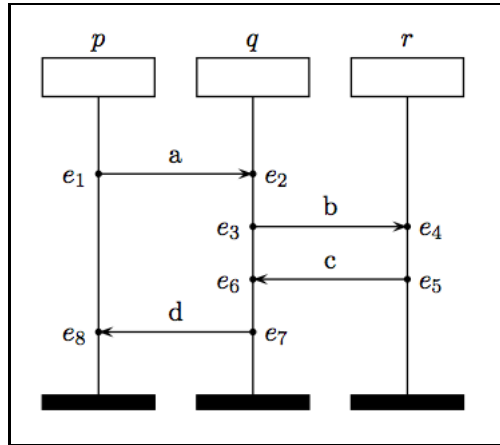


Figure 2.1: Example of MSC M

To have a visualization of posets, next a graphical concept will be defined that will simplify the view of the relations of the elements in a poset.

Definition 2.1.4. (*Hasse Diagram*)

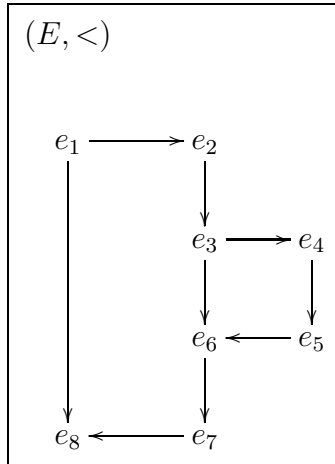
Let $(E, <)$ be a poset of a MSC $M = (\mathcal{P}, \Sigma, Act, E, l, m, <)$. The Hasse diagram of $(E, <)$ is based on a relation $\prec \subseteq E \times E$ and is defined as follows:
 $\forall e, e' \in E$

$$e \prec e' \iff e < e' \wedge \neg(\exists e''. e < e'' < e').$$

■

We use Hasse diagrams as a graphical visualization for the relations in a poset. In a Hasse diagram of a poset $(E, <)$ there is an arrow from an event e to an event e' (where $e, e' \in E$) iff $e \prec e'$.

Example 2.1.1. Consider the MSC M from Figure 2.1. Let $(E, <)$ be the poset of M . The Hasse diagram of $(E, <)$ looks as follows:



For the rest of the thesis it will be assumed that a poset will always be depicted by their Hasse diagram.

2.2 Message Sequence Graphs

A MSC is a structure that defines a single episode of interactions between processes. In many cases one would also like to characterize a set of episodes of interactions between processes which are composed in such a way that the outcome of a composition defines again an episode. Message Sequence Graphs (MSGs) are models that allow us to do so. A MSG is a graph where each node of the graph is associated with a MSC. Every run in the graph characterizes a composition of the MSCs associated with the nodes along the run. A composition of two MSCs produces again a MSC. The composition of MSCs is defined by the *concatenation* operator.

For now, we are only interested in how MSGs are defined and also what are their properties.

Definition 2.2.1. (*Message Sequence Graphs*)

Let \mathcal{M} be a set of MSCs. A Message Sequence Graph (MSG) over \mathcal{M} is a tuple $G = (V, \rightarrow, v_{init}, F, \lambda)$ such that:

- (V, \rightarrow) is a digraph with a finite set of vertices V and $\rightarrow \subseteq V \times V$ a set of edges,

- $v_{init} \in V$ is the initial vertex,
- $F \subseteq V$ is a set of final vertices,
- $\lambda : V \rightarrow \mathcal{M}$ is a function that assigns to every vertex a MSC from \mathcal{M} .

■

A MSG G starts in its initial vertex v_{init} . An arbitrary vertex $v \in V$ can perform a transition to a vertex $v' \in V$ iff $(v, v') \in \rightarrow$. Note that MSGs are nondeterministic systems. If a vertex $v \in V$ has two possible transitions, to the vertex $v' \in V$ or $v'' \in V$ (thus $(v, v'), (v, v'') \in \rightarrow$), then the process of choosing which transition to be performed is made nondeterministically. A MSG is equal to a nondeterministic finite state machine without input alphabet where each node is associated to a MSC.

The behavior of MSGs are defined through their *paths*. A path in a MSG is a finite sequence of vertices where every vertex with its following adjacent must be in relation under \rightarrow .

Definition 2.2.2. (*A MSG path*)

Let $G = (V, \rightarrow, v_{init}, F, \lambda)$ be a MSG. A path in G is a finite sequence of vertices $\pi = v_0 v_1 v_2 \dots v_n$ such that:

- $v_i \in V$ for $0 \leq i \leq n$,
- $(v_i, v_{i+1}) \in \rightarrow$ for $0 \leq i < n$.

The MSG path π is accepting if $v_0 = v_{init}$ and $v_n \in F$.

■

The composition of the MSCs of the vertices along a path produces again a MSC. As mentioned before, this is accomplished by concatenating the MSCs along the path with the concatenation operator. The concatenation of MSCs operates exactly like the concatenation of its probabilistic variant in the next chapter.

Definition 2.2.3. (*MSC of a path*)

Let $\pi = v_0v_1v_2 \dots v_n$ be a MSG path. Through concatenating the MSCs along the path π a MSC $M(\pi)$ will be produced.

$$M(\pi) = \lambda(v_0) \cdot \lambda(v_1) \cdot \lambda(v_2) \cdot \dots \cdot \lambda(v_n).$$

■

Note that the how MSCs are concatenated will not be defined in this chapter. The intention here is to show the properties of a MSG. The concatenation operator will be discussed in detail in Definition 4.2.5 in Section 4.2.

A MSG can be characterized by its MSC language. A MSC language of a MSG is the set of MSCs that can be produced by all accepting paths of the MSG.

Definition 2.2.4. (*The MSC language of a MSG*)

Let $G = (V, \rightarrow, v_{init}, F, \lambda)$ be a MSG. The MSC language of G is defined as follows:

$$L(G) = \{M(\pi) | \pi \text{ is an accepting path in } G\}$$

■

2.3 Propositional Dynamic Logic

This section will present a mathematical language called PDL [8], that can be used to specify the properties of MSCs. The language PDL is composed of over three different categories which are defined inductively. The lowest category is called *path expression*. A PDL path expression is defined over a pair of events in a MSC. A pair of events (e, e') satisfies a PDL path expression if there is a possible sequence of events in the MSC that starts in e and ends in e' that represent a path formulated by the path expression. Thus, PDL path expressions are used to specify sequences of events in a MSC. The second category is called *local formula*. A local formula is defined over a MSC and an event from the MSC. A PDL local formula specifies the property of an

event in a MSC. The last category is called *global formula*. A PDL global formula is defined over a MSC and is used to specify characteristics of the MSC.

In the following, the syntax and semantics of PDL will be stated. Let $M = (\mathcal{P}, \Sigma, Act, E, l, m, <)$ be a MSC used for the following two definitions.

Definition 2.3.1. (*Syntax of PDL*)

The syntax of PDL is defined over three different categories. These categories are:

1. Path expressions, which are ranged over by λ . Let α be a local formula (defined below), then the syntax of a path expression λ is defined as follows:

$$\lambda ::= proc \mid msg \mid \{\alpha\} \mid \lambda; \lambda \mid \lambda + \lambda \mid \lambda^*$$

2. Local formulas, which are ranged over by α . The syntax of a local formula α is defined as follows:

$$\alpha ::= true \mid \sigma \mid \alpha \vee \alpha \mid \neg \alpha \mid \langle \lambda \rangle \alpha \mid \langle \lambda \rangle^{-1} \alpha$$

where $\sigma \in Act$ and λ is a path expression.

3. Global formulas, ranged over by φ . The syntax of a global formula φ is defined as follows:

$$\varphi ::= \exists \alpha \mid \forall \alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

where α is a local formula.

■

In PDL path expressions, *proc* and *msg* determine the direction of the path. The term *proc* refers to the direct predecessor of an event in the same process and *msg* refers to the matching receive event of an event. The local formulas $\langle \lambda \rangle \alpha$ and $\langle \lambda \rangle^{-1} \alpha$ are used to reason about the path λ (where $\langle \lambda \rangle$ means that the path runs forwards and $\langle \lambda \rangle^{-1}$ means that the path runs backwards) and the end of the path is characterized with the local formula α .

Other logical connectives and operators can be deduced from the terms defined above.

Definition 2.3.2. (*Semantics of PDL*)

1. Let $e, e' \in E$, α a PDL local formula and $\lambda, \lambda_1, \lambda_2$ PDL path expressions. The semantics of PDL path expressions is defined over a pair of events as follows:

- $(e, e') \models \text{proc} \iff \text{loc}(e) = \text{loc}(e') \wedge e < e' \wedge \neg(\exists e''. e < e'' < e')$
- $(e, e') \models \text{msg} \iff m(e) = e'$
- $(e, e') \models \{\alpha\} \iff e \models \alpha$
- $(e, e') \models \lambda_1; \lambda_2 \iff \exists e''. (e, e'') \models \lambda_1 \wedge (e'', e') \models \lambda_2$
- $(e, e') \models \lambda_1 + \lambda_2 \iff (e, e') \models \lambda_1 \vee (e, e') \models \lambda_2$
- $(e, e') \models \lambda^* \iff \exists n \geq 0. (e, e') \models \underbrace{\lambda; \lambda; \dots; \lambda}_{n \text{ times}}$

2. Let $\sigma \in \text{Act}$, λ a PDL path expression and $\alpha, \alpha_1, \alpha_2$ PDL local formulas. The semantics of PDL local formulas is defined over M and an event $e \in E$ as follows:

- $M, e \models \text{true} \forall e \in E$
- $M, e \models \sigma \iff l(e) = \sigma$
- $M, e \models \alpha_1 \vee \alpha_2 \iff M, e \models \alpha_1 \text{ or } M, e \models \alpha_2$
- $M, e \models \neg\alpha \iff M, e \not\models \alpha$
- $M, e \models \langle \lambda \rangle \alpha \iff \exists e' \in E. (e, e') \models \lambda \wedge M, e' \models \alpha$
- $M, e \models \langle \lambda \rangle^{-1} \alpha \iff \exists e' \in E. (e', e) \models \lambda \wedge M, e' \models \alpha$

3. Let α be PDL local formula. The semantics of PDL global formulas is defined over M as follows:

- $M \models \exists\alpha \iff \exists e \in E. M, e \models \alpha$
- $M \models \forall\alpha \iff \forall e \in E. M, e \models \alpha$
- $M \models \varphi_1 \vee \varphi_2 \iff M \models \varphi_1 \text{ or } M \models \varphi_2$
- $M \models \varphi_1 \wedge \varphi_2 \iff M \models \varphi_1 \text{ and } M \models \varphi_2$

■

Chapter 3

Message Sequence Charts With Unreliable Channels

3.1 Introduction

To achieve a more natural approach to message exchange between instances, this chapter will define an extension of MSCs which will be called *Probabilistic Message Sequence Charts* (pMSCs). In contrast to MSCs, pMSCs use unreliable, asynchronous, FIFO channels. This means that there is always a possibility that any message can get lost in a channel. Essentially, the syntax of both models, MSCs and pMSCs can not be distinguished. In pMSCs each channel has a certain probability of losing a message. Each message in a channel has the same probability of being lost, independent of the size of the channel in which the message resides and the number of messages in that channel. Further, in the model we define here, channels may have different probability values of losing a message instead of using one uniform probability value for all the channels. Hence, each channel may have a different degree of reliability.

The significant difference between MSCs and their probabilistic variant is in their semantics. A MSC has semantically speaking only one possible scenario in which all messages which are sent will be received - where no channel loses its messages. This is due to the fact that MSCs are based on the assumption of having reliable channels. However, there may be several event orderings in that scenario depending on the partial order relation $<$. On the

contrary, in a pMSC there is more than one possible scenario. Each message loss in the channels (or combination of message losses between different channels) can produce different scenarios. Where again, in regard to $<$ there may be several event orderings in each scenario.

The structure of this chapter is presented in the following way: we will start by defining what a pMSC is, followed by essential definitions for better understanding the characteristics of pMSCs. The next section is focused on designing an algorithm that generates all possible scenarios of a pMSC. Subsequently, the complexity of the algorithm and the number of possible scenarios will be approximated. The last section will handle the method of applying probabilistic measurement to pMSCs.

3.2 Probabilistic Message Sequence Charts

Definition 3.2.1. (*Probabilistic Message Sequence Charts*)

A Probabilistic Message Sequence Chart (pMSC) is a structure $M = (\mathcal{P}, \Sigma, Act, E, l, m, <, C, p_{loss})$ such that:

- $(\mathcal{P}, \Sigma, Act, E, l, m, <)$ is a MSC,
- $C \subseteq \mathcal{P} \times \mathcal{P}$ is a symmetric relation defining the communication channels between two processes,
- $p_{loss} : C \rightarrow [0, 1]$ is a function that assigns to each communication channel a probability. The probability of a channel indicates the probability that a message can get lost in that channel.

■

How MSCs (Definition 2.1.3) and pMSCs (Definition 3.2.1) are defined differ in two aspects. First, the existence of channels between processes is explicitly defined in Definition 3.2.1. Since the relation between processes is symmetrical, it follows that each process pair has two communication channels - one for each direction. Second, a function p_{loss} is introduced in Definition 3.2.1 that assigns a probability value to the communication channels. This value determines the probability that a channel loses a message. Further, each of these channels may have different probability values in losing a message: for

two communication channels (p, q) and (q, p) it is either $p_{loss}(p, q) \neq p_{loss}(q, p)$ or $p_{loss}(p, q) = p_{loss}(q, p)$.

Figure 3.1 illustrate examples for graphical representations of pMSCs. They reflect that visually pMSCs are not distinct from MSCs. Through the graphical representation of a pMSC one can depict the expected behavior of the system. For example, that the event e_1 sends a message a to event e_2 , which has a successor event e_3 that sends a message b to e_4 and so on, can be read directly from pMSC M_1 in Figure 3.1(a). However, if one accounts for the possible loss of messages in a pMSC, one can not deduce all possible behavior that the pMSC poses simply by looking at the graphical representation.

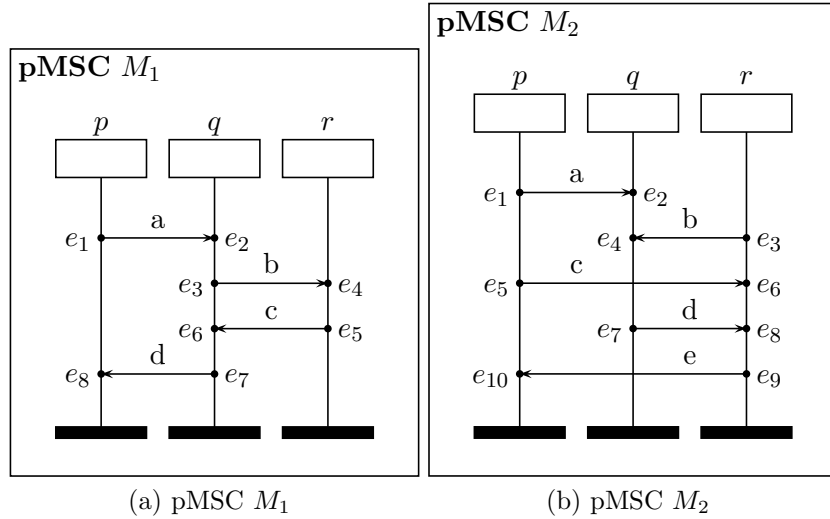


Figure 3.1: Examples of pMSCs

Subsequently, a number of concepts will be presented which are the building blocks of the following sections. Therefore, we define the pMSC $M = (\mathcal{P}, \Sigma, Act, E, l, m, <, C, p_{loss})$ for the following definitions.

Definition 3.2.2. (*The empty pMSC*)

M is called the *empty* pMSC, denoted by M_\emptyset , if there are no events occurring in M .

$$M = M_\emptyset \iff E = \emptyset.$$

■

The empty pMSC M_\emptyset can be understood as a *dummy* pMSC that does not exhibit any behavior. This concept will be helpful in the next chapter where a collection of pMSCs are combined into a graph. Figure 3.2 shows the graphical representation of the empty pMSC M_\emptyset .

Definition 3.2.3. (*Location of an event*)

For an arbitrary event e in pMSC M , $loc(l(e))$ defines the process in which e takes place:

$$loc(p!q(a)) = loc(p?q(a)) = p,$$

for $p, q \in \mathcal{P}, p \neq q$ and $a \in \Sigma$.

■

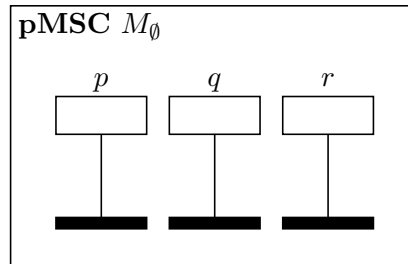


Figure 3.2: The empty pMSC M_\emptyset

Definition 3.2.4. (*Successors of an event*)

Let e be an event in pMSC M . We define $succ_!(e)$ as the set of send events which occur after the event e and $succ_?(e)$ as the set of receive events which occur after the event e .

$$succ_!(e) = \{e' \in E_! | e < e'\},$$

$$succ_?(e) = \{e' \in E_? | e < e'\}.$$

Further we define $succ(e)$ as the union of $succ_!(e)$ and $succ_?(e)$, the set of all events which occur after event e .

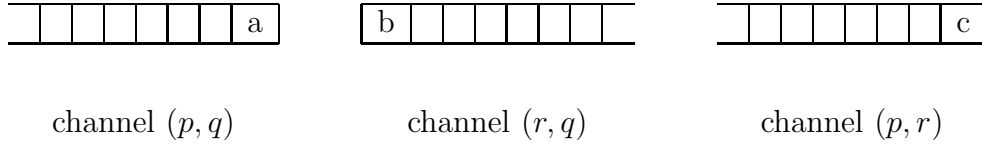
$$succ(e) = succ_!(e) \cup succ_?(e).$$

■

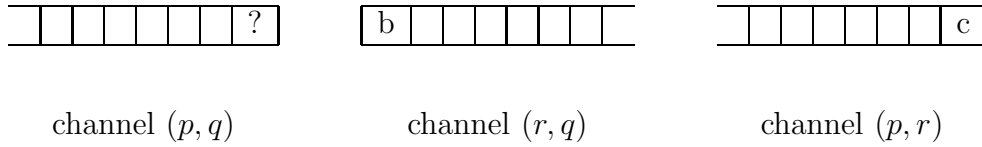
Since channels may lose messages with certain probability, we can never be sure if all events in a pMSC will occur. This feature, that certain events may not occur due to messages lost in the channels, reflects the possible scenarios of the pMSC. Thus, the pMSC behave differently depending on which channels lose a message (or multiple messages) as well as which message is lost. A lost message is to be understood as a message disappearing from the channel. Since the loss of messages takes place in the channels, it implies that the message has to be in the channel before it is lost. This further implies that beforehand the message has to be sent to the channel - by executing a send event. The occurrence of a send event is not directly dependent on the reliability of the channels. It is moreover dependent on whether the send event has a possible receive event that occur before it and furthermore if this receive event occurs or not.

On the other hand, the occurrence of a receive event is directly dependent on the reliability of the channels. For every message lost, there is a receive event that should have occurred. Therefore, a message loss can also be interpreted as a nonoccurrence of a receive event. A nonoccurrence of a receive event e in a pMSC implies that all events that should occur after it will also not occur - this association is based on the partial order relation $<$. The set of these events is denoted by $succ(e)$ as in Definition 3.2.4.

Example 3.2.1. Consider the pMSC M_2 in Figure 3.1(b). Assume that M_2 is in a state in which the events e_1 , e_3 and e_5 have been executed. This means that in the channel (p, q) resides the message a , in the channel (r, q) resides the message b and in the channel (p, r) resides the message c (the rest of the channels are empty).



Notice that the event e_5 can only occur if e_1 has occurred but the event e_3 can occur independently of e_1 and e_5 - since $e_1 < e_5$, $e_1 \not< e_3$ and $e_5 \not< e_3$. Imagine a situation in which the channel (p, q) loses the message a (this is denoted below by substituting the message a with ?).



This implies that the receive event e_2 will never occur - since e_2 should have received the message a . Thereby, all the events in $\text{succ}(e_2)$ will also not occur - the events e_4, e_7, e_8, e_9 and e_{10} . However, the receive event e_6 has no dependency on the receive event e_2 with respect to $<$. Hence, through the nonoccurrence of e_2 no prediction can be made due to the occurrence of e_6 - the receive event e_6 may occur or it may not. Therefore we say that e_6 is *incomparable* to e_2 . This means that by assuming that the receive event e_2 does not occur, M_2 produces two possible scenarios: one in which the receive event e_6 does occur and another one in which it does not.

The number of possible scenarios by assuming that a receive event e does not occur is dependent on the number of receive events that are *incomparable* to e - if no receive event is *incomparable* to e than there is only one possible scenario in which e and all events in $\text{succ}(e)$ do not occur. Hence, to define possible scenarios for the nonoccurrence of a receive event e , a set of *incomparable* receive events to e must be defined.

Definition 3.2.5. (*Set of incomparable receive events*)

Let $\bowtie \subseteq E \times E$ be an irreflexive relation over the set of events in pMSC M such that $\forall e, e' \in E$:

$$e \bowtie e' \iff \neg(e < e') \wedge \neg(e' < e).$$

The set of incomparable receive events $\text{ics}_?(e)$ for an event $e \in E$ is defined as follows:

$$ics_{\gamma}(e) = \{e' \in E_{\gamma} | e \bowtie e'\}.$$

■

A possible scenario of a pMSC is a possible ordering relation of events in that pMSC. In Definition 2.1.1 from the previous chapter we have defined an ordering relation over events that is called poset. For convenience, through the rest of the paper we will say a possible poset instead of a possible scenario.

For all pMSCs there exists a unique poset in which no message is lost and all events in the pMSC occurs. This poset will be called as the *standard* poset.

Definition 3.2.6. (*The standard poset of a pMSC*)

The *standard* poset of pMSC M , denoted by $full(M)$, is the unique poset where no message in the channels is lost.

$$full(M) = (E, <).$$

■

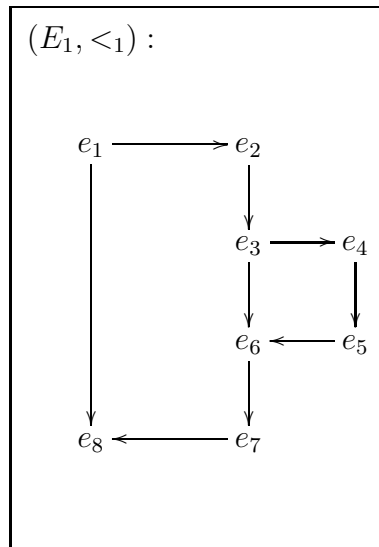


Figure 3.3: $full(M_1)$.

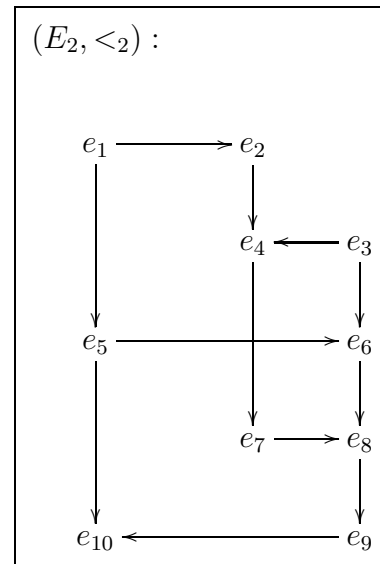


Figure 3.4: $full(M_2)$.

As was the case in the previous chapter, also posets of pMSCs are depicted by their Hasse diagram. Since the standard poset of a pMSC is the poset where none of the channels loses messages, it is as if one considers a MSC. Therefore, the standard poset of a pMSC is the same poset if one considers the pMSC as a MSC. Figures 3.3 and 3.4 show the standard posets $full(M_1)$ and $full(M_2)$ for the pMSC M_1 and M_2 from Figure 3.1.

Definition 3.2.7. (*The lost event*)

Let $e \in E_?$ be a receive event in a poset $(E, <)$, where the message that should have been received by e got lost. Then the event e will be removed from E and replaced by a lost event \odot . This process of event substitution is denoted by $e \rightsquigarrow \odot$ (the index of e will be used as an index for the lost event \odot). Further, the order $<$ will also be adapted for the lost event \odot .

$(E, <)[e \rightsquigarrow \odot] = (E', <')$ such that:

$$E' = E \setminus \{e\} \cup \{\odot\} \text{ and } (E', <') \text{ is isomorphic to } (E, <).$$

$E'_\odot \subset E'$ denotes the set containing all lost events. Hence,

$$E' = E'_1 \uplus E'_? \uplus E'_\odot.$$

■

Example 3.2.2. Consider the pMSC M_1 from Figure 3.1(a) with the standard poset $full(M_1)$. Imagine that the message c got lost from the channel (r, q) . This means that the receive event e_6 will be replaced by the lost event \odot_6 ($e_6 \rightsquigarrow \odot_6$). Hence, the new poset $(E'_1, <'_1)$ is equal to $(E_1, <_1) = full(M_1)$ with $E'_1 = E_1 \setminus \{e_6\} \cup \odot_6$. Further, the set of lost events in M_1 contains only the lost event \odot_6 ($E'_\odot = \{\odot_6\}$).

A graphical representation of M_1 where the receive event e_6 does not occur can be seen in Figure 3.5. Notice that the partial order relation $<'_1$ is a copy of $<_1$ where e_6 is replaced by \odot_6 . Figure 3.5 is only a graphical representation of M_1 where it was previously assumed that the event e_6 will not occur due to the loss of message c . It does not represent a possible poset of M_1 with the assumption that the event e_6 is not occurring.

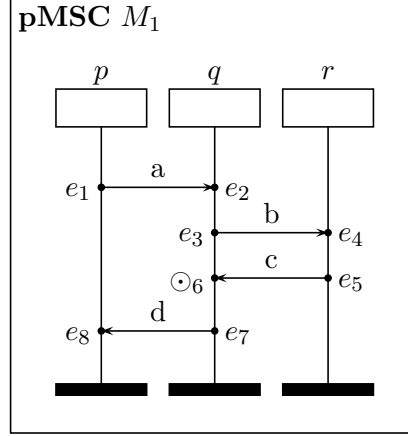


Figure 3.5: pMSC M_1 with a lost event e_6

Definition 3.2.8. (*Posets of a pMSC*)

The set of posets of M , denoted by $\Pi(M)$, is the set containing all possible posets that pMSC M can exhibit. A poset $(E', <')$ is in $\Pi(M)$ if $(E', <')$ fulfills the following conditions (let \bar{e} represent send, receive and loss events, e send and receive events, and \odot loss events):

1. $\forall e \in E': \forall e' \in E. e' < e \implies e' \in E'$,
2. $\forall e \in E'_1: (\exists e' \in E'_2. (m(e) = e' \wedge e' \in E'))$ or $(\exists \odot \in E'_\odot. m(e) = \odot)$,
3. $\forall \odot \in E': \exists e \in E'_1. m^{-1}(\odot) = e$,
4. $\forall \bar{e} \in E': \neg(\exists \odot \in E'_\odot. \odot <' \bar{e})$.

Further, the partial order relation $<'$ is defined as follows:

$$<' = \left((< \cap (E' \times E')) \cup (m^{-1}(E'_\odot), E'_\odot) \right)^*$$

where $*$ is the transitive closure. ■

There are four conditions which define a poset. The first condition states that for any event in a poset, all the events occurring before it in the pMSC

should also be in the poset. This first condition is obvious, since an event can only occur if all preceding events have occurred before. The second condition states that in each poset, a send event should have a matching receive event or a matching lost event. The third condition states that each lost event in a poset should have a corresponding send event included in the poset. The fourth condition assures that there are no events (send, receive nor loss events) occurring after a loss event. Further, it is obvious that it has to be ensured that the send and receive events in E' have to be a subset of the events in E ($E'_1 \cup E'_2 \subseteq E$). This is necessary so that no arbitrary event - a send or receive event not from E - is included in E' .

Example 3.2.3. Consider the pMSC M_1 from Figure 3.1(a), and let's analyze if the following two posets $(E', <')_1$ and $(E', <')_2$ (Figure 3.6) are in $\Pi(M_1)$. Let's start with the first poset $(E', <')_1$. The first condition is not fulfilled by $(E', <')_1$ since one of the predecessor events of e_6 , the event e_5 , is not included in $(E', <')_1$. Since all send events in $(E', <')_1$ have a corresponding receive or lost event, all lost events have a corresponding send event, and there are no events after \odot_4 (with respect to $<'$), the poset $(E', <')_1$ passes the conditions two, three and four. Hence $(E', <')_1 \notin \Pi(M_1)$. On the other hand, the poset $(E', <')_2$ fulfills all of the conditions from Definition 3.2.8. $(E', <')_2$ is the poset from Figure 3.5 where it is assumed that the message c is lost. Hence, $(E', <')_2 \in \Pi(M_1)$.

3.3 Generating posets of a pMSC

This section will be focused on a method with the purpose of generating all possible posets of a pMSC M - which is the set $\Pi(M)$. To generate the set $\Pi(M)$ we will apply an algorithm to the pMSC M that takes the standard poset $full(M)$ as an input and returns the set $\Pi(M)$ after the algorithm is completely executed. Before we show the pseudocode of the algorithm, first we would like to describe the idea behind it.

Since for every receive event in a pMSC there exists at least two possible posets - one where the receive event does occur and another one where it does not occur -, it is natural to assume that a pMSC M has more than one poset (the standard poset) as long as $E \neq \emptyset$ - note that in our definition of pMSCs (Definition 3.2.1) we restrict that every send event must have a corresponding receive event and vice versa; the function m in Definition 3.2.1

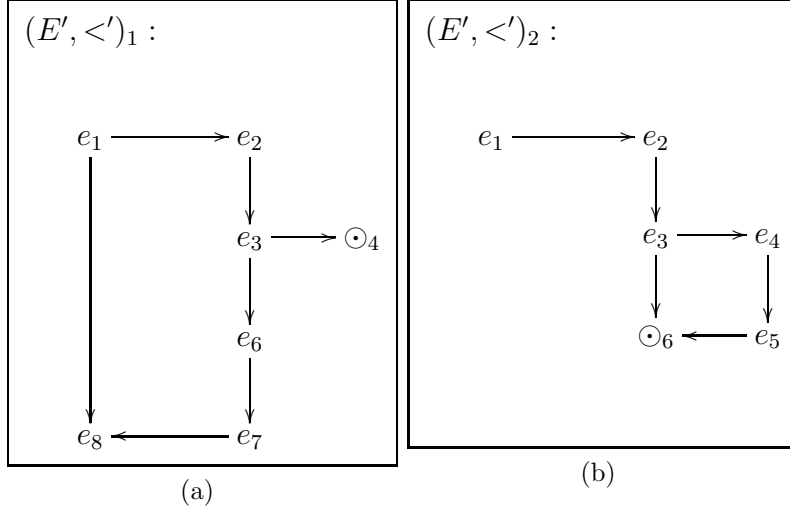


Figure 3.6

is a total, bijective function; thus, we do not allow the existence of a send event without a matching receive event. Each receive event may occur if the message it should receive does not get lost in the channel, or it does not occur if the message it should have received does get lost in the channel. Moreover, if a receive event does occur, then all successor events can also occur as long as there are no messages lost subsequently. On the other hand, if a receive event does not occur then all successor events of it will also not occur. Example 3.2.1 shows that the number of possible posets for the nonoccurrence of a receive event depends on the number of receive events that are incomparable to it (see Definition 3.2.5). Let $e \in E$ be a receive event in M and $ics_{\gamma}(e)$ the set of incomparable receive events of e . If the event e does not occur, then there are a maximum of $2^{|ics_{\gamma}(e)|}$ possible posets based on the nonoccurrence of e . The power set of $ics_{\gamma}(e)$ ($2^{ics_{\gamma}(e)}$) is a set of receive events in which each of these sets can be seen as a combination of receive events that may also not occur if the event e is assumed not to occur. If e is included in each set in $2^{ics_{\gamma}(e)}$, each of these new sets can be mapped to a poset, where the receive events in a set (including e) are asserted not to occur in the associated poset - accordingly they are represented as lost events in the poset. If we collect all of these posets for each receive event in a pMSC, we will obtain all possible posets of the pMSC.

3.3.1 The Algorithm

Based on the idea above, we will introduce a full algorithm which takes the standard poset as an input and returns the set of all possible posets. Further, the algorithm will be applied to two examples so that the working of the algorithm can be analyzed along with what constitutes the number of posets in a pMSC.

Input: the standard poset $full(M)$.

Output: the set of posets $\Pi(M)$.

```

1:  $\Pi(M) := \{full(M)\};$ 
2:  $j := 0;$ 
3: for all  $e_i \in E?$  do
4:   for all  $x \in 2^{ics?(e_i)}$  do
5:      $x := x \cup \{e_i\};$ 
6:      $j ++;$ 
7:      $(E, <)_j := full(M);$ 
8:     for all  $e_k \in x$  do
9:        $e_k \rightsquigarrow \odot_k$ 
10:    end for
11:     $(E, <)_j := (E, <)_j - (\bigcup_{e_k \in x} succ(e_k));$ 
12:     $\Pi(M) := \Pi(M) \cup \{(E, <)_j\};$ 
13:  end for
14: end for

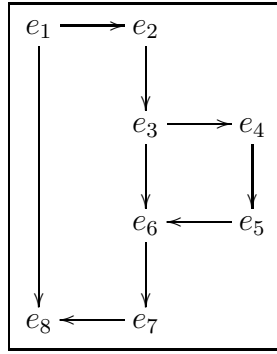
```

(1) Initially, the set $\Pi(M)$ contains only the standard poset $full(M)$. This is the poset where no message is lost. Thus, all events did occur. (2) j is used to index the newly generated posets. (3) The algorithm has to run over all receive events in E to be able to generate all possible posets. (4) To generate all possible posets for a nonoccurrence of a receive event e_i , the algorithm has to run over all the elements in the power set of $ics?(e_i)$. (5) For each element of the power set we include the receive event in consideration. Thus, we can treat all the receive events in x (including the receive event in consideration) as receive events that did not occur and all the other receive events ($E? \setminus x$) as receive events that did occur (in this stage we still have not considered the successor events of the non occurred events). (6) Increase j for the generation of a poset. (7) Define the poset with the index j as the *standard* poset. (8-9) Replace all the receive events in x with the symbol \odot (the index of e will be taken as the index of \odot). (11) Redefine the new poset by removing

all the successor events of all the events in x . This is accomplished by the ‘-’ operator. (12) Include the new generated poset to the set $\Pi(M)$.

The best way to illustrate the individual steps is to apply the algorithm in examples. In the following, the algorithm will be applied to the pMSCs in Figure 3.1. The further intention is to show how the number of possible posets differ in pMSCs with no incomparable receive events and pMSCs with incomparable receive events.

Example 3.3.1. Let $M_1 = (\mathcal{P}_1, \Sigma_1, Act_1, E_1, l_1, m_1, <_1, C_1, p_{loss_1})$ be the pMSC from Figure 3.1a. The standard poset, $full(M_1)$ for M_1 looks as follows:

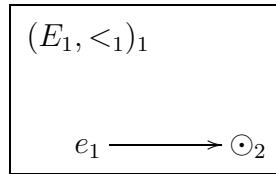


Next we apply the algorithm to the first receive event e_2 in M_1 :

1. $\Pi(M_1) = \{(E_1, <_1)\}$, the set $\Pi(M_1)$ contains only the standard poset $full(M_1)$.
2. $j = 0$, index for the posets is set to 0.
3. The first receive event (e_2) will be analyzed.
4. All elements in $2^{ics_?(e_2)}$ will be taken into consideration.
5. Add the event e_2 into the elements of $2^{ics_?(e_2)}$.
6. Increment the value of the index j .
7. $(E_1, <_1)_1 = full(M_1)$, define the possible poset as the standard poset.

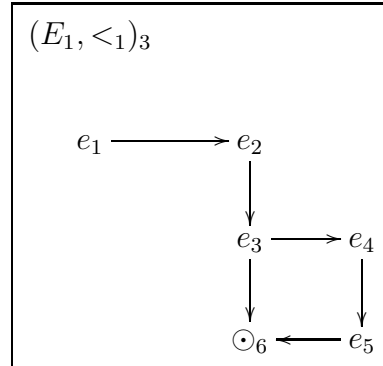
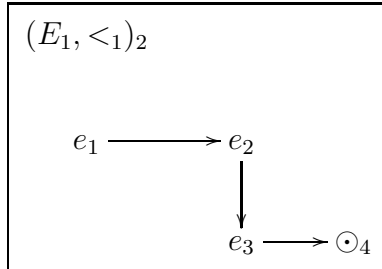
8. (8-9) Since all the receive events in M_1 are comparable to one another ($ics_?(e_2) = ics_?(e_4) = ics_?(e_6) = ics_?(e_8) = \emptyset$), the only element of x is e_2 , thus only $e_2 \rightsquigarrow \odot_2$.
9. Remove all the successor events of e_2 , the set $succ(e_2) = e_3, e_4, e_5, e_6, e_7, e_8$.
10. Include the poset into $\Pi(M_1)$, $\Pi(M) = \{(E_1, <_1)\} \cup \{(E_1, <_1)_1\}$.

The poset generated from e_2 looks as follows:

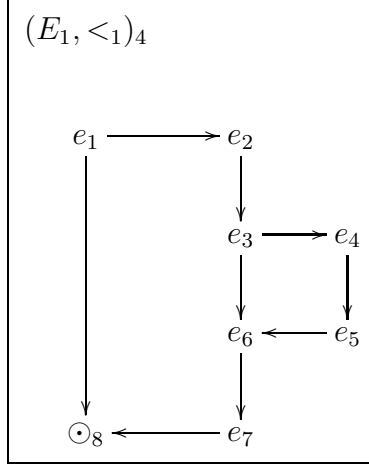


The procedure for the other receive events will not be described in detail, given that they run the same as in e_2 . However, how the posets appear will be shown below.

- The poset $(E_1, <_1)_2$ for the receive event e_4 and the poset $(E_1, <_1)_3$ for the receive event e_6 :



- The poset $(E_1, <_1)_4$ for the receive event e_8 :

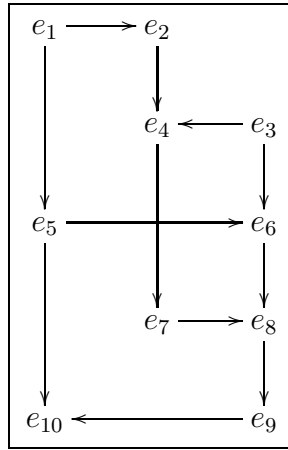


The set of posets $\Pi(M_1)$ for the pMSC M_1 contains all of the posets generated above $\Pi(M_1) = \{(E_1, <_1), (E_1, <_1)_1, (E_1, <_1)_2, (E_1, <_1)_3, (E_1, <_1)_4\}$.

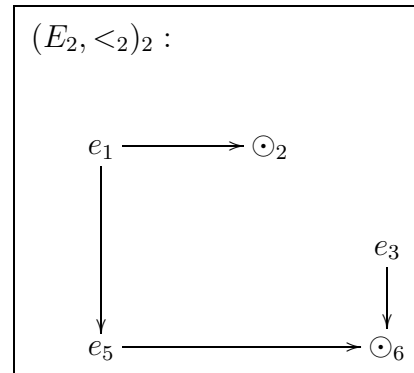
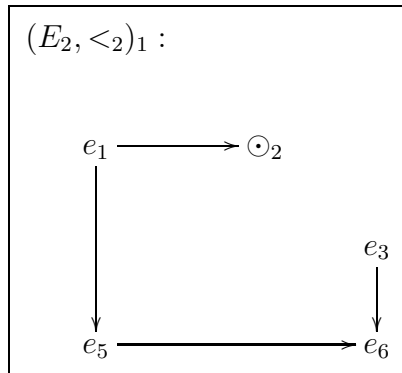
In Example 3.3.1 each receive event generates only one poset. This is due to the fact that all receive events in M_1 are comparable to one another, since for all $e \in E_{1,r}$ is the set $ics_{\gamma}(e) = \emptyset$. Comparing M_2 from Figure 3.1(b) with M_1 (Figure 3.1(a)), M_2 has some pairs of receive events that are incomparable to one another - where all the receive events in M_1 are in total order. This implies that not for all receive events e in M_2 is the set $ics_{\gamma}(e)$ empty. Hence, there is more than one possible poset for these receive events. The next example will show that the number of possible posets can grow with respect to the number of incomparable receive events.

Example 3.3.2. Let pMSC $M_2 = (\mathcal{P}_2, \Sigma_2, Act_2, E_2, l_2, m_2, <_2, C_2, p_{loss_2})$ be the pMSC from Figure 3.1(b). Consider the receive events e_2 and e_6 in M_2 . Even though the order of their corresponding send event is clear - since they take place in the same process -, the order of the receive events e_2 and e_6 cannot be determined. It cannot be said which of the two receive events will occur first. For those receive events e in M_2 where $ics_{\gamma}(e)$ is not empty, there exist at most $2^{ics_{\gamma}(e)}$ possible posets in which the receive event e did not occur. By applying the algorithm to M_2 , it will generate the following posets.

The standard poset $full(M_2)$, in which there are no lost messages looks as follows:

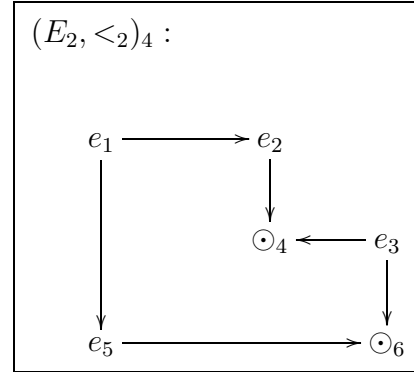
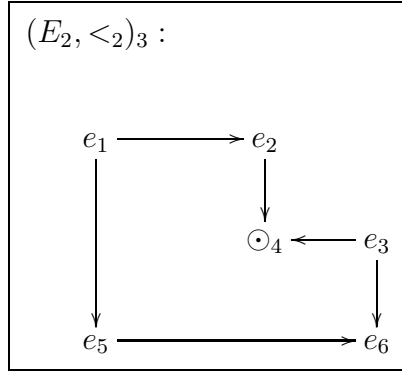


The posets generated from the first receive event e_2 are the following:



The set $ics?(e_2)$ has only one element, the receive event e_6 . Therefore, there are two possible posets for the nonoccurrence of the receive event e_2 . In the first poset $(E_2, <_2)_1$ the receive event e_2 did not occur - e_2 is replaced with \odot_2 - and the receive event e_6 did occur. Further, all the successor events of e_2 are removed. In the second poset $(E_2, <_2)_2$ both the events e_2 and e_6 did not occur. By removing all the successor events of e_2 , the successor events of e_6 have also been removed - since $succ(e_6) \subset succ(e_2)$.

The posets generated from the receive event e_4 are the following:

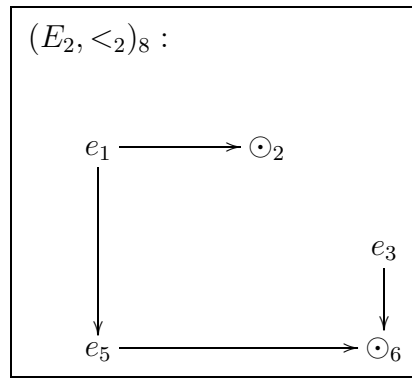
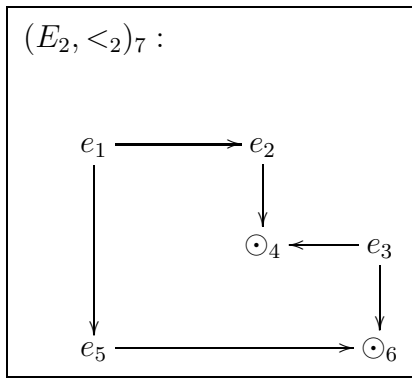
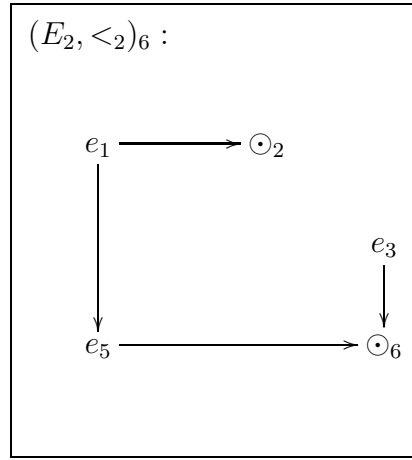
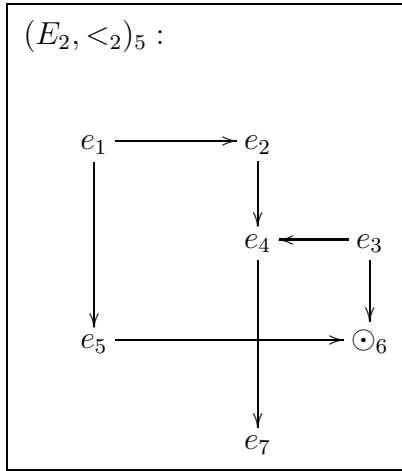


The posets for the receive event e_4 are generated in a similar way as the posets of e_2 . Since the only receive event which is incomparable to e_4 is the event e_6 , there are also two possible posets for e_4 - one in which only the receive event e_4 did not occur and another one in which the receive events e_4 and e_6 did not occur. The events e_7, e_8, e_9 and e_{10} are removed from the posets as these events are successor events of e_4 and e_6 .

The receive event e_6 generates more posets than e_2 and e_4 due to the fact that $ics?(e_6) = \{e_2, e_4\}$. There are four possible posets for the case that e_6 does not occur:

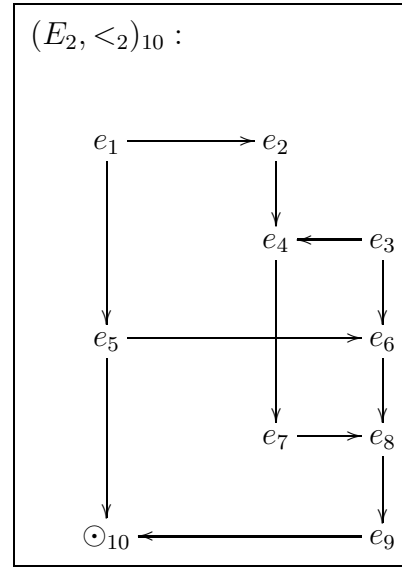
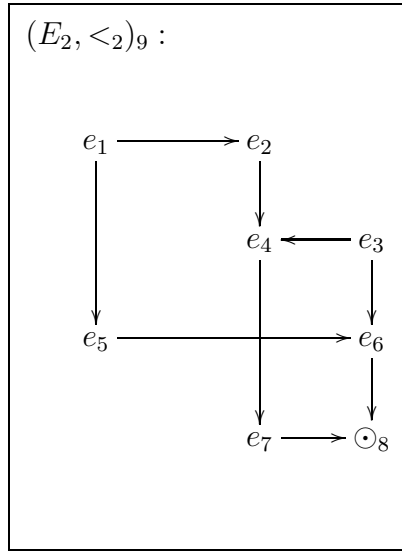
- e_2 and e_4 did occur - the poset $(E_2, <_2)_5$, the events e_8, e_9 and e_{10} are removed (since these are the successor events of e_6).
- e_2 did not occur and e_4 did occur - the poset $(E_2, <_2)_6$, the events e_4, e_7, e_8, e_9 and e_{10} are removed (since these are the successor events of e_2 and e_6).
- e_2 did occur and e_4 did not occur - the poset $(E_2, <_2)_7$, the events e_7, e_8, e_9 and e_{10} are removed (since these are the successor events of e_4 and e_6).
- e_2 and e_4 both did not occur - the poset $(E_2, <_2)_8$, the events e_4, e_7, e_8, e_9 and e_{10} are removed (since these are the successor events of e_2, e_4 and e_6).

The posets generated from the receive event e_6 are the following:



Since the receive events e_8 and e_{10} are successor events of e_2, e_4 and e_6 , and e_{10} is a successor of e_8 , they are comparable to all the receive events in M_2 - $ics_?(e_8) = ics_?(e_{10}) = \emptyset$. Thus, each of these receive events generates one poset. The poset $(E_2, <_2)_9$ is generated from e_8 , where e_8 did not occur and all successor events of e_8 are removed. The receive event e_{10} generates the poset $(E_2, <_2)_{10}$, where the only receive event not occurring is e_{10} (no events are removed).

The posets generated from the receive event e_8 and e_{10} are the following:



$\Pi(M_2) = \{(E_2, <_2), (E_2, <_2)_1, (E_2, <_2)_2, (E_2, <_2)_3, (E_2, <_2)_4, (E_2, <_2)_5, (E_2, <_2)_9, (E_2, <_2)_{10}\}$. The posets $(E_2, <_2)_6$, $(E_2, <_2)_7$ and $(E_2, <_2)_8$ are not being included since $(E_2, <_2)_2 = (E_2, <_2)_6 = (E_2, <_2)_8$ and $(E_2, <_2)_4 = (E_2, <_2)_7$.

3.3.2 Complexity of generating a set of posets and the size of it

The worst case scenario for the algorithm in Section 3.3.1 is that all receive events in a pMSC are incomparable to one another - assuming that there is more than one receive event. Imagine a pMSC M with the following properties (example in Figure 3.7):

1. each receive event is located in a different process and
2. it is never the case that a send event occurs after a receive event in a process.

Using proof by contradiction, it can be shown that the pMSC M with the properties above will have incomparable receive events. First, by contradicting property (1), there must be at least one process in which there is more than one receive event. In this case, the receive events in those processes will

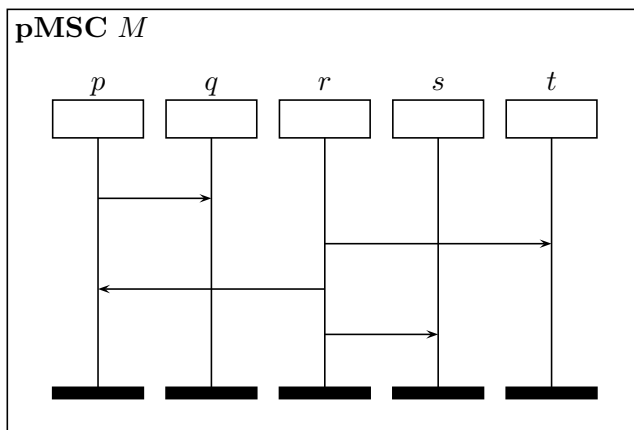


Figure 3.7: An example of a pMSC M with properties 1 and 2.

be comparable to one another (all events in one process are ordered from top to bottom). Hence, it contradicts the assumption that all receive events are incomparable to one another. In Figure 3.8, property (1) is violated by the events e and e' through adding a new message exchange denoted by the dash line. Second, by contradicting property (2), there must be at least one

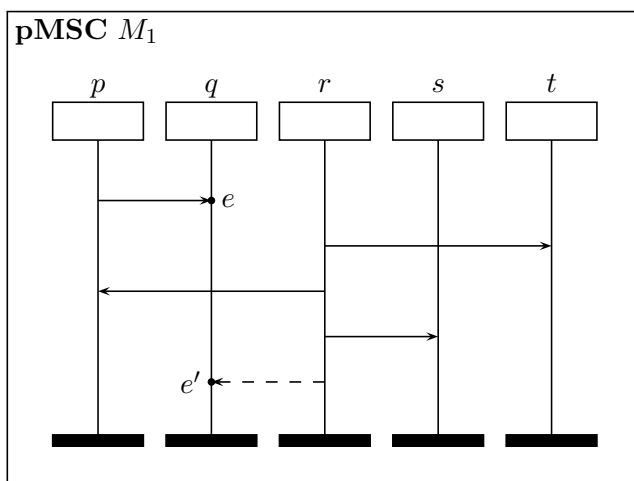


Figure 3.8: An example of a pMSC contradicting property (1).

process in which there is a send event occurring after a receive event. In

this case the receive event will be comparable to the send event that occurs after it. Thus, the receive event will also be comparable to the matching receive event of the send event. Again it contradicts the assumption that all receive events are incomparable to one another. In Figure 3.9 one can depict a violation of property (2) by adding a message exchange denoted by the dash line ($e < e'$ thus $e < e''$). Further, the properties (1) and (2) can

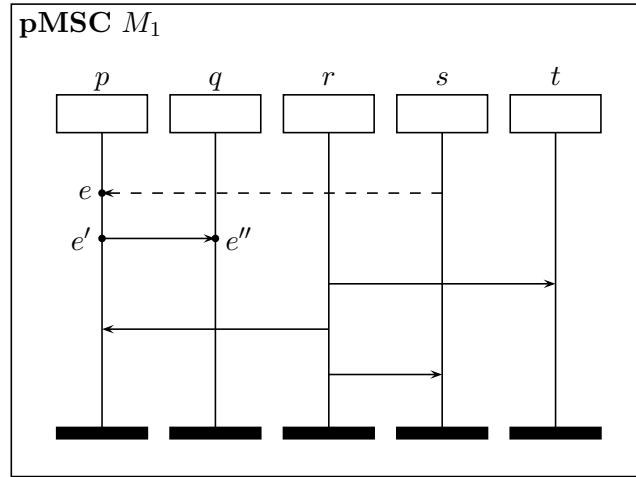


Figure 3.9: An example of a pMSC contradicting property (2).

be combined: if there is a receive event located in a process, then there are no other receive events in that process and no send events occurring after it. By examining the algorithm from Section 3.3.1, in the worst case, when the properties described above hold, the first loop in line (3) will be executed $\frac{n}{2}$ times (if $|E| = n$ then $|E_?| = \frac{n}{2}$). The second loop, in line (4), will be executed $2^{\frac{n}{2}-1}$ times. The last loop in line (8) depends on the size of x - an element from the power set of $ics_?$. The size of x can range from 1 to $\frac{n}{2}$. This could be affirmed since: the size of $ics_?$ is $\frac{n}{2} - 1$ (in the worst case); so the size of an element from the power set of $ics_?$ will range from 0 to $\frac{n}{2} - 1$; since we always add the receive event in consideration into the set x (look at line (5) in the algorithm), the size of x will range from 1 to $\frac{n}{2}$. Thus, it implies that the complexity of the algorithm in Section 3.3.1 is in the worst case exponential in the number of receive events.

Proposition 3.3.1. (Complexity of generating $\Pi(M)$ for a pMSC M)

The worst case complexity of the algorithm for generating all posets for a pMSC M is $\mathcal{O}(2^{|E_\gamma|})$.

Further, by maintaining the properties (1) and (2) above, it follows that the set $\text{succ}(e)$ is empty for each receive event $e \in E_\gamma$ of the pMSC M . A pMSC with such characteristics is only possible if the number of processes is greater than the number of receive events - it must be the case that $|\mathcal{P}| > |E_\gamma|$. This means, in line (11) of the algorithm no events will be removed from the new poset. Every new generated poset is a unique possible poset of nonoccurrence of receive events. These possible posets are deduced from the power set of E_γ (2^{E_γ} consists of all possible combinations of nonoccurrence of receive events). It implies that in a worst case scenario where the constraints above hold ($|\mathcal{P}| > |E_\gamma|$ and $\text{succ}(e) = \emptyset$ for all receive events e) the size of $\Pi(M)$ will be $2^{|E_\gamma|}$. On the other hand, if the number of processes is less than the number of receive events, the size of $\Pi(M)$ is less than $2^{|E_\gamma|}$. This is true because since $|\mathcal{P}| \leq |E_\gamma|$, there must be at least one process where a receive event and one or more send and/or receive events are located in the process. In such case the size of $\Pi(M)$ depends on the size of E_γ and \mathcal{P} and how the receive events are distributed in \mathcal{P} . Since in general there are no restrictions on the correlation between the number of receive events and the number of processes, the size of $\Pi(M)$ is exponential in the relation to the number of receive events and the number of processes.

Proposition 3.3.2. (The number of posets for a pMSC M)

The number of posets ($|\Pi(M)|$) for a pMSC M is $2^{\mathcal{O}(\min(|E_\gamma|, |\mathcal{P}|))}$ in the worst case.

3.4 Probability measure on pMSC

The previous sections of this chapter were centered on defining a pMSC and also describing the possible posets a pMSC could exhibit. The fact that a pMSC could produce different posets is essentially based on the unreliable channels which are equipped with certain probabilities. So far, we have not defined probability measures on pMSCs which will be the focus of this section.

The fundamental idea in this section is to describe how to apply probability measurement to pMSCs. We would like to be able to tell what the probability of a pMSC is. This can be derived from describing how to compute the probability of an event to describing how to compute the probability of a poset and finally to compute the probability of all posets (which defines the probability of the pMSC). Note that by defining the probability of an event we mean the probability of an event in a poset, and not generally in a pMSC - since a receive event in a pMSC cannot be determined in advance if it occurs or not.

Since the occurrence of send events in a poset does not depend on the reliability of the channels, their probability is always equal to one. Whereas the probability of receive events and lost events in a poset can be gained through the probability of the channels. The probability of receive events and lost events depends on the probability of the channel where the message resides (or has resided).

Definition 3.4.1. (*Probability of events in a pMSC poset*)

Let $(E, <) \in \Pi(M)$ be a poset of pMSC M and $e \in E$ an event. Then, the probability of the event e is defined as follows:

$$pr(e) = \begin{cases} 1 & \text{if } e \in E_1, \\ 1 - p_{loss}(c_{pq}) & \text{if } e \neq \odot, e \in E_? \cap E_q, m^{-1}(e) \in E_1 \cap E_p, \\ p_{loss}(c_{pq}) & \text{if } e = \odot, e \in E_? \cap E_q, m^{-1}(e) \in E_1 \cap E_p. \end{cases}$$

for $p, q \in \mathcal{P}$ and $p \neq q$.

■

When the probability of events in a poset is determined, the probability of the poset itself is straightforward. By multiplying the probability of the individual events (send, receive and lost events) in a poset, the probability of the poset is determined.

Definition 3.4.2. (*Probability of a pMSC poset*)

Let $(E, <) \in \Pi(M)$ be a poset of pMSC M . We define the probability of $(E, <)$ as the probability that all the events in E occur. Formally:

$$Pr((E, <)) = \prod_{e \in E} pr(e).$$

■

Finally, the probability of a set of posets can be computed naturally by adding the probability of the individual posets.

Definition 3.4.3. (*Probability of a set of posets*)

Let $\Pi(M)$ be the set of posets of *pMSC* M and $B \subseteq \Pi(M)$ an arbitrary subset of it. To compute the probability of B we sum the probability of all those posets $(E, <)' \in B$.

$$Pr(B) = \sum_{(E, <)' \in B} Pr((E, <)').$$

■

Proposition 3.4.1. (*Probability of a pMSC*)

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m, <, C, p_{loss})$ be a *pMSC*. The probability of M is equal to one, $Pr(M) = Pr(\Pi(M)) = 1$.

Proof. Let $(\Omega, \mathcal{F}, Pr)$ be a probability space (see below) over posets of a set $\Pi(M)$ such that:

- $\Omega = \Pi(M)$, the sample set consists of all possible posets.
- $\mathcal{F} = 2^{\Pi(M)}$.

To simplify the proof, imagine that the elements in $\Pi(M)$ are indexed. Imagine a bijective function $\mathcal{I} : \Pi(M) \rightarrow \{1, 2, \dots, |\Pi(M)|\}$ which has the purpose of indexing the elements in $\Pi(M)$ from 1 to $|\Pi(M)|$.

$$\begin{aligned} Pr(\Pi(M)) &= \sum_{n=1}^{|\Pi(M)|} Pr(\underbrace{(E, <)_n}_{\text{pairwise disjoint events}}) \\ &= Pr\left(\bigcup_{n=1}^{|\Pi(M)|} (E, <)_n\right) \\ &= Pr(\Omega) \\ &= 1 \end{aligned}$$

□

A probability space is a triple $(\Omega, \mathcal{F}, Pr)$ where:

- Ω is an arbitrary non-empty set which comprises all possible outcomes,
- $\mathcal{F} \subseteq 2^\Omega$ is a set of events which is a σ -algebra containing the empty set and is closed under complements and countable unions.
- $Pr : \mathcal{F} \rightarrow [0, 1]$ is a probability measure function on \mathcal{F} with the following properties:
 1. Let $A_1, \dots, A_n \in \mathcal{F}$ be a set of pairwise disjoint sets. Then $Pr(\sqcup A_i) = \sum_1^n Pr(A_i)$,
 2. $Pr(\Omega) = 1$.

Example 3.4.1. Consider the pMSC M_1 from Figure 3.1(a). Lets assume that the probability of message loss for the channels (p, q) and (q, p) is 0.2 and for the channels (q, r) and (r, q) is 0.35 - the probability of message loss for the channels (p, r) and (r, p) is not of interest, since on these channels there are no messages exchanged. The probability of the send events (e_1, e_3, e_5, e_7) are straightforward ($pr(e_1) = pr(e_3) = pr(e_5) = pr(e_7) = 1$). The probability of the receive events e_2 and e_8 are 0.8 and for e_4 and e_6 are 0.65 - the probability of a receive event is $1 - p_{loss}(c_{pq})$ if $e_i \neq \odot_i, e_i \in E_r \cap E_q, m^{-1}(e_i) \in E_1 \cap E_p$. The probability for the events which did not occur are 0.2 for \odot_2 and \odot_8 , and 0.35 for \odot_4 and \odot_6 . The probability of the individual posets in $\Pi(M_1)$ are the following (the posets can be obtained from Example 3.3.1):

$$\begin{aligned}
 Pr((E_1, <_1)) &= pr(e_1) \times pr(e_2) \times pr(e_3) \times pr(e_4) \times pr(e_5) \times pr(e_6) \times pr(e_7) \times pr(e_8) = 0.2704 \\
 Pr((E_1, <_1)_1) &= pr(e_1) \times pr(\odot_2) &= 0.2 \\
 Pr((E_1, <_1)_2) &= pr(e_1) \times pr(e_2) \times pr(e_3) \times pr(\odot_4) &= 0.28 \\
 Pr((E_1, <_1)_3) &= pr(e_1) \times pr(e_2) \times pr(e_3) \times pr(e_4) \times pr(e_5) \times pr(\odot_6) &= 0.182 \\
 Pr((E_1, <_1)_4) &= pr(e_1) \times pr(e_2) \times pr(e_3) \times pr(e_4) \times pr(e_5) \times pr(e_6) \times pr(e_7) \times pr(\odot_8) = 0.0676
 \end{aligned}$$

By adding the probability of the individual posets we obtain the probability

of all the posets $\Pi(M_1)$, which is the probability of the pMSC M_1 .

$$Pr((E_1, <_1)) = 0.2704$$

$$Pr((E_1, <_1)_1) = 0.2$$

$$Pr((E_1, <_1)_2) = 0.28$$

$$Pr((E_1, <_1)_3) = 0.182$$

$$Pr((E_1, <_1)_4) = 0.0676 +$$

$$Pr(\Pi(M_1)) = 1$$

Chapter 4

Composing Probabilistic MSCs

4.1 Introduction

The previous chapter introduced pMSCs, which are a variant of MSCs, where each channel is associated a probability value that defines the probability that a channel can lose messages. This chapter will introduce a concept that combines different pMSCs into one automata-based model, where each node in the automata corresponds to a pMSC. We call these automata-based models *Probabilistic Message Sequence Graphs* (pMSGs).

A pMSG is a variant of MSGs (see Definition 2.2.1), in which each node is labeled with a pMSC instead of a MSC. Therefore, pMSGs are defined over a set of pMSCs. A transition in a pMSG is probabilistic and also non-deterministic. Each node in a pMSG may have several actions to choose to perform a transition. The process of deciding which action to take is done in a nondeterministic way. After an action has been chosen, a probabilistic transition will be performed, with respect to the chosen action, to one of the possible successors.

To begin with, this chapter will start with introducing pMSGs followed by describing the properties of it. Subsequently, we will become familiar with Markov Chains, which is a probabilistic model that will be used as a basic model to specify probabilistic behaviors of pMSGs. At the end of this chapter, the process of transforming a pMSG into a corresponding Markov Chain will be shown.

4.2 Probabilistic Message Sequence Graphs

A pMSC can be used to describe a single entity of probabilistic interaction (in our case message exchange) between instances which could be processes, agents, networks or others. But if one would like to describe compositions over a set of entities of probabilistic interactions (pMSCs), where the transitions from one entity to another also has a notion of probability, pMSCs are very limited in fulfilling such intentions. In Chapter 2 we have defined MSGs (Definition 2.2.1) which is an automata-based model used to describe compositions of MSCs. Therefore we will extend MSGs in a sense of incorporating probabilistic behavior so that it will be compatible to define compositions of pMSCs. Our extended model will also exhibit nondeterminism beside being probabilistic.

Definition 4.2.1. (*Probabilistic Message Sequence Graphs*)

Let \mathcal{M} be a set of pMSCs. A Probabilistic Message Sequence Graph (pMSG) over \mathcal{M} is a tuple $G = (V, Act, P, v_{init}, v_f, \lambda)$ such that:

- V is a finite set of vertices,
- Act is a finite set of actions,
- $P : V \times Act \times V \rightarrow [0, 1]$ is a transition probability function that assigns to each transition a probability. For fixed $v \in V$ and $\alpha \in Act$:

$$\sum_{v' \in V} P(v, \alpha, v') \in \{0, 1\},$$

- $v_{init} \in V$ is the initial vertex,
- $v_f \in V$ is the *final* vertex with the following property:

$$\exists! \alpha \in Act \text{ such that } P(v_f, \alpha, v_f) = 1$$

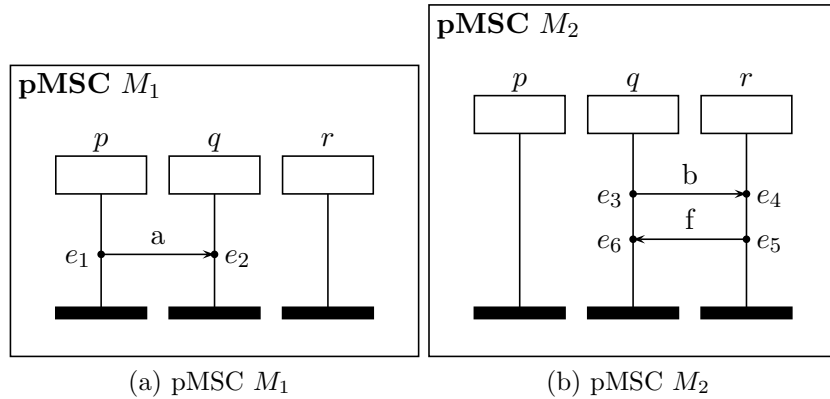
- $\lambda : V \rightarrow \mathcal{M}$ is a function that assigns to each vertex a pMSC, where:

$$\lambda(v) = \begin{cases} M_\emptyset & \text{if } v = v_f, \\ M \in \mathcal{M} \setminus \{M_\emptyset\} & \text{otherwise.} \end{cases}$$

■

A pMSG always starts from its initial vertex v_{init} . Whenever it wants to perform a transition from its current vertex $v \in V$, it will first have to choose nondeterministically from all possible actions which can be executed from the vertex v . After an action $\alpha \in Act$ has been chosen, a transition to a successor vertex v' of v can be performed with the probability $P(v, \alpha, v')$. An accepting run in a pMSG G ends always in its unique final vertex v_f , where each incoming transition to v_f has a transition probability 1. Moreover, v_f also has a self loop with a transition probability 1.

Example 4.2.1. Figure 4.1f shows an example of a pMSG G_1 which is defined over the set of pMSCs $\mathcal{M} = \{M_1, M_2, M_3, M_4, M_\emptyset\}$. The vertices in G_1 are associated to pMSCs in \mathcal{M} in the following way: $\lambda(v_i) = M_i$, for $i \in \{1, 2, 3, 4\}$ and $\lambda(v_5) = M_\emptyset$. The pMSG G_1 starts from its initial vertex v_1 . In v_1 , G_1 has a nondeterministic choice between the actions α_1 and α_2 . By choosing the action α_2 , v_1 has only one possible successor with probability 1 which is the vertex v_4 . By choosing the action α_1 , v_1 has two possible successors, the vertices v_2 and v_3 with a transition probability 0.2 and 0.8. From v_2 there is only one possible action to choose, the action α_3 , in which a transition back to v_2 can be performed with probability 0.35 or a transition to the vertex v_3 with probability 0.65. The final vertex v_5 can be reached from the vertex v_3 by taking the only possible action α_4 and from the vertex v_4 by taking the only possible action α_5 . Both transitions have the probability one. Furthermore, the final vertex v_5 has a self loop transition with probability 1 by taking the action α_6 .



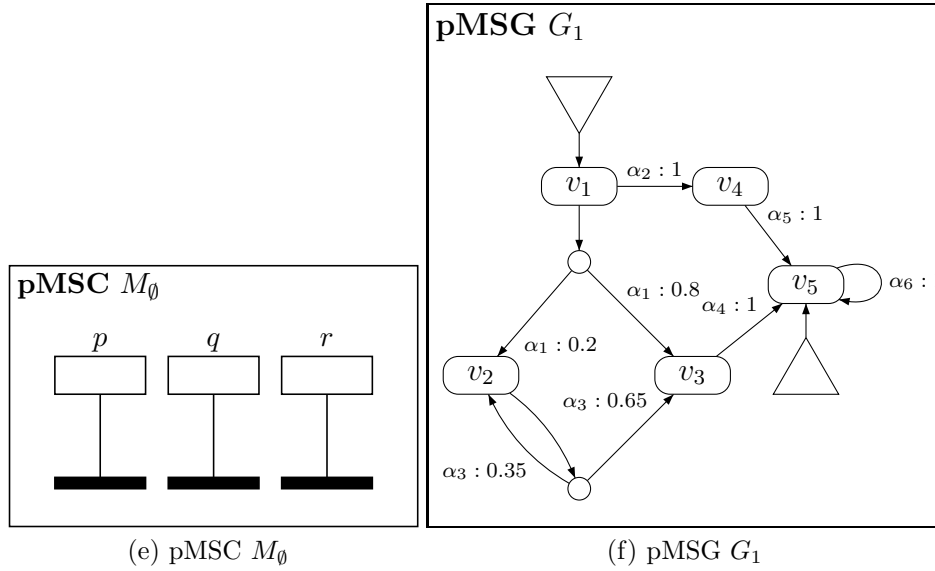
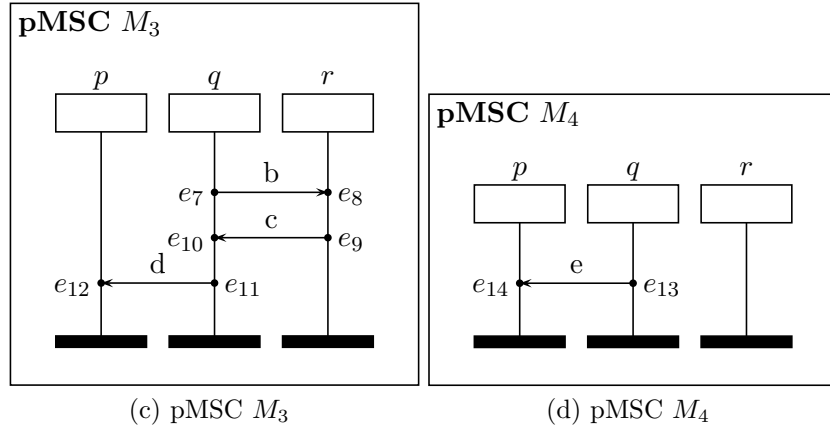


Figure 4.1: pMSG G_1 over $\mathcal{M} = \{M_1, M_2, M_3, M_4, M_0\}$.

Due to the fact that pMSGs have probabilistic and also nondeterministic transitions, we consider pMSGs similar to Markov Decision Processes (since a Markov Decision Process also allows probabilistic and nondeterministic choices in their transitions unlike Markov Chains, which allow only probabilistic ones). The similarity between our definition of pMSGs to Markov

Decision Processes will give an advantage which will be shown in Section 4.4.2

Let $G = (V, Act, P, v_{init}, v_f, \lambda)$ be a pMSG which we will use for the following definitions.

Definition 4.2.2. (*Enabled action in a vertex*)

The set of enabled actions in a vertex $v \in V$ of G , denoted by $Enabled(v)$, is the set of possible actions which can be taken from the vertex v to perform a transition.

$$Enabled(v) = \{\alpha \in Act \mid \sum_{v' \in V} P(v, \alpha, v') = 1\}.$$

■

If we examine the pMSG G_1 from Figure 4.1f, then the enabled actions for the vertices are the following:

- $Enabled(v_1) = \{\alpha_1, \alpha_2\}$
- $Enabled(v_2) = \{\alpha_3\}$
- $Enabled(v_3) = \{\alpha_4\}$
- $Enabled(v_4) = \{\alpha_5\}$
- $Enabled(v_5) = \{\alpha_6\}$

Through a resolution of the probabilistic and nondeterministic choices in a pMSG we can derive a set of possible execution sequences, which are represented via an alternating consecutive sequence of vertices and actions. We define these possible execution sequences in a pMSG as *paths*.

Definition 4.2.3. (*A pMSG path*)

A path π in pMSG G is a finite sequence of vertices and actions which ends in a vertex.

$$\pi = v_0 \xrightarrow{\alpha_1} v_1 \xrightarrow{\alpha_2} v_2 \dots \xrightarrow{\alpha_n} v_n$$

such that:

$$P(v_i, \alpha_{i+1}, v_{i+1}) > 0, \text{ for } 0 \leq i < n.$$

We define $first(\pi)$ as the first vertex in a path π and $last(\pi)$ as its last vertex. The set of all possible paths of pMSG G is denoted by $Path(G)$. To define the set of paths starting from a vertex v we use the notation $Path(v)$. We say that a path π is *accepting* iff $first(\pi) = v_{init}$ and $last(\pi) = v_f$.

■

Example 4.2.2. Consider the pMSG G_1 from Figure 4.1f and the following paths:

$$\pi_1 = v_1 \xrightarrow{\alpha_2} v_4 \xrightarrow{\alpha_5} v_5 \xrightarrow{\alpha_6} v_5 \xrightarrow{\alpha_6} v_5$$

The path π_1 is an accepting path since $first(\pi) = v_1$ which is the initial vertex of G_1 and $last(\pi_1) = v_5$ which is the final vertex of G_1 .

$$\pi_2 = v_1 \xrightarrow{\alpha_1} v_3 \xrightarrow{\alpha_4} v_5$$

π_2 is also an accepting path in G_1 since $first(\pi_2) = v_1$, $last(\pi_2) = v_5$. The path

$$\pi_3 = v_1 \xrightarrow{\alpha_1} v_2 \xrightarrow{\alpha_3} v_2 \xrightarrow{\alpha_2} v_2 \dots \xrightarrow{\alpha_2} v_3$$

is not accepting. The path π_3 ends in a vertex (the vertex v_3) which is not the final vertex of G_1 .

Since a path in a pMSG is an alternating consecutive sequence of vertices and actions which ends in a vertex and a transition from one vertex to another in the path is carried out by an action which has a certain probability, one can also define the probability of paths appearing in a pMSG. The probability of a path is the probability of traversing the sequence of vertices in the corresponding path. This probability can be obtained by multiplying the probability of the individual transitions in the path.

Definition 4.2.4. (*Probability of a pMSG path*)

Let $\pi = v_0 \xrightarrow{\alpha_1} v_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} v_n$ be a path in G . The probability of π is defined as follows:

$$Prob(\pi) = \prod_{i=0}^{n-1} P(v_i, \alpha_{i+1}, u_{i+1}),$$

where P is the transition probability function of the pMSG G , $\alpha_{i+1} \in Enabled(v_i)$ and $P(v_i, \alpha_{i+1}, v_{i+1}) > 0$ for $0 \leq i < n$. ■

Example 4.2.3. Consider the path $\pi = v_1 \xrightarrow{\alpha_1} v_2 \xrightarrow{\alpha_3} v_2 \xrightarrow{\alpha_3} v_2 \xrightarrow{\alpha_3} v_3 \xrightarrow{\alpha_4} v_5 \xrightarrow{\alpha_6} v_5$ in G_1 . The probability of this path is the following:

$$\begin{aligned} Prob(\pi) &= P(v_1, \alpha_2, \alpha_2) \cdot P(v_2, \alpha_3, v_2) \cdot P(v_2, \alpha_3, v_2) \cdot P(v_2, \alpha_3, v_3) \cdot P(v_3, \alpha_4, v_5) \cdot P(v_5, \alpha_6, v_5) \\ &= 0.2 \cdot 0.35 \cdot 0.35 \cdot 0.65 \cdot 1 \cdot 1 \\ &= 0.015925 \end{aligned}$$

Definition 4.2.3 states that a path in a pMSG is a sequence of vertices and actions that follows certain conditions. Furthermore, a pMSG is defined over a set of pMSCs, such that each vertex in a pMSG is associated to one pMSC. Thus, a path in a pMSG can be illustrated as a concatenation of pMSCs that are associated to the vertices, which defines the path. The concatenation of two pMSCs emerge from the action connecting the two pMSCs. The following definition defines how two pMSCs can be concatenated, which results again in a pMSC.

Definition 4.2.5. (*Concatenation of pMSCs*)

Let $M_1 = (\mathcal{P}_1, \Sigma_1, Act_1, E_1, l_1, m_1, <_1, C_1, p_{loss_1})$ and $M_2 = (\mathcal{P}_2, \Sigma_2, Act_2, E_2, l_2, m_2, <_2, C_2, p_{loss_2})$ be pMSCs. The concatenation of M_1 and M_2 , denoted by $M_1 \cdot M_2$, is the pMSC $M = (\mathcal{P}, \Sigma, Act, E, l, m, <, C, p_{loss})$ which is defined in the following way:

- $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $Act = Act_1 \cup Act_2$,
- $E = E_1 \uplus E_2$,
- $l(e) = \begin{cases} l_1(e) & \text{if } e \in E_1, \\ l_2(e) & \text{if } e \in E_2, \end{cases}$
- $m(e) = \begin{cases} m_1(e) & \text{if } e \in E_1, \\ m_2(e) & \text{if } e \in E_2, \end{cases}$

- $\leq_1 \cup \leq_2 \cup \{(e, e') \mid e \in E_1, e' \in E_2, \text{loc}(e) = \text{loc}(e')\}$,
- $C = C_1 \cup C_2$,
- $p_{\text{loss}}(c_{pq}) = \begin{cases} p_{\text{loss}_1}(c_{pq}) = p_{\text{loss}_2}(c_{pq}) & \text{if } p, q \in \mathcal{P}_1 \cap \mathcal{P}_2, \\ p_{\text{loss}_1}(c_{pq}) & \text{if } p, q \in \mathcal{P}_1 \setminus \mathcal{P}_1 \cap \mathcal{P}_2, \\ p_{\text{loss}_2}(c_{pq}) & \text{if } p, q \in \mathcal{P}_2 \setminus \mathcal{P}_1 \cap \mathcal{P}_2. \end{cases}$

■

Definition 4.2.5 above is defined based on three assumptions. First, if there is a letter used to denote a process in \mathcal{P}_1 which is also used to denote a process in \mathcal{P}_2 , then these are identical processes - i.e., there is a letter q used to denote a process in \mathcal{P}_1 and also in \mathcal{P}_2 . This implies that $q \in \mathcal{P}_1 \cap \mathcal{P}_2$. Further if there are processes p and q used in \mathcal{P}_1 and also in \mathcal{P}_2 , then $p_{\text{loss}_1}(c_{pq}) = p_{\text{loss}_2}(c_{pq})$ and $p_{\text{loss}_1}(c_{qp}) = p_{\text{loss}_2}(c_{qp})$. If we wish to allow pairs of processes (p, q) which are used in \mathcal{P}_1 and \mathcal{P}_2 to have different channel probabilities in M_1 and M_2 , then for the concatenation of M_1 and M_2 we could take the minimum (or the maximum) of the channel probabilities. For example, if $p, q \in \mathcal{P}_1$ with $p_{\text{loss}_1}(c_{pq}) = 0.5$ and $p, q \in \mathcal{P}_2$ with $p_{\text{loss}_2}(c_{pq}) = 0.25$ then the probability of the channel after the concatenation would be $p_{\text{loss}}(c_{pq}) = 0.25$ if the minimum is chosen ($p_{\text{loss}}(c_{pq}) = 0.5$ respectively if we choose to take the maximum). Second, E_1 and E_2 are assumed to be disjoint - $E_1 \cap E_2 = \emptyset$. Third, the definition above is based on an asynchronous concatenation. This means that through the concatenation, the events are ordered by process. This still means that for all $p \in \mathcal{P}$, all the events from M_1 will be executed first in process p and then the events from M_2 . It does not mean that all events from M_2 have to wait for all events from M_1 to be executed first, as in a synchronous concatenation.

Example 4.2.4. Consider the pMSCs M_2 and M_3 from Figure 4.1b and 4.1c. Figure 4.2 shows the concatenation $M = M_2 \cdot M_3$.

We find that it is not necessary to define concatenation over the set of posets. The concatenation of two sets of posets $\Pi(M_1)$ and $\Pi(M_2)$ from two different pMSCs M_1 and M_2 can be obtained by:

1. apply the concatenation of M_1 and M_2 to obtain a new pMSC M .

$$M = M_1 \cdot M_2$$

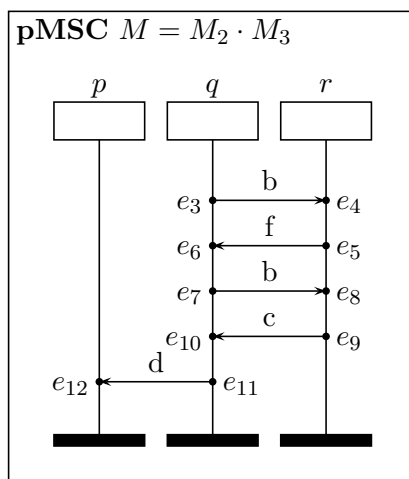


Figure 4.2: Concatenation of M_2 and M_3 .

2. apply the algorithm from Section 3.3 to M to obtain the set of posets $\Pi(M)$.

Definition 4.2.6. (*pMSC of a path*)

Let $\pi = v_0 \xrightarrow{\alpha_1} v_1 \xrightarrow{\alpha_2} v_2 \dots \xrightarrow{\alpha_n} v_n$ be a path in G . The pMSC produced from the path π is that which evolves by concatenating the corresponding pMSCs of the vertices in π .

$$M(\pi) = \lambda(u_0) \cdot \lambda(u_1) \cdot \dots \cdot \lambda(u_n).$$

■

In Definition 4.2.6 we have stated that by concatenating the pMSCs of the vertices along a path of a pMSG will induce a pMSC. Furthermore, from Definition 4.2.3 we can obtain the probability of any path a pMSG can produce. Using the Definitions 4.2.6 and 4.2.3 we can now determine the probability of a poset $(E, <)_{\pi} \in \Pi(M(\pi))$ of a pMSC $M(\pi)$ produced by a path π in a pMSG G .

Definition 4.2.7. (*Probability of a poset in a pMSC path*)

Let π be a path in pMSG G and $M(\pi)$ the corresponding pMSC of the

path π . The probability of a poset $(E, <)_{\pi} \in \Pi(M(\pi))$ is defined in the following way:

$$Prob((E, <)_{\pi}) = pr((E, <)_{\pi}) \times Prob(\pi).$$

■

The probability of the poset, $pr((E, <)_{\pi})$, is computed as in the previous chapter (Definition 3.4.2). The probability of the pMSC $M(\pi)$, from the path π , in G is equal to the probability of the path π in G , i.e., $Prob(M(\pi)) = Pr(\Pi(M(\pi))) \times Prob(\pi) = Prob(\pi)$. Notice that $Prob(M(\pi))$ denotes the probability of $M(\pi)$ in G and $Pr(\Pi(M(\pi)))$ denoted the probability of all the posets in $M(\pi)$.

To define the language of a pMSG we adapt a similar method used for probabilistic automata with infinite words [5].

Definition 4.2.8. (*The language of a pMSG*)

Let $p \in [0, 1]$ be a threshold. The Language $L_p(G)$ of pMSG G is the set of pMSCs which are produced from all accepting paths π and π has a probability of at least p .

$$L_p(G) = \{M(\pi) | \pi \text{ is an accepting path in } G \text{ and } Prob(\pi) \geq p\}.$$

■

4.3 Markov Chain

To reason about the characteristics of a pMSG, in the matter of its probabilistic behavior, we would like to introduce in this section a well known probabilistic system that is used to model probabilistic choices, namely Markov Chains. The Markov Chain we would like to define is also known as a discrete-time Markov Chain, which is a time-abstract model where decisions between possible outcomes are solely probabilistic in nature. Formally a Markov Chain consists of a nonempty countable set of states in which a transition from one state to another is simply probabilistic. There is a transition probability function that determines the probability of transitions between states. A Markov Chain behaves like a transition system except that the successor

states are chosen probabilistically instead of nondeterministically.

The intention in this section is at first to define the probabilistic system Markov Chain. This will be followed by presenting the properties of a Markov Chain and also how to associate a probability space to a Markov Chain for reasoning about quantitative and qualitative properties. Accordingly we will describe the transformation from a pMSG to a Markov Chain which is carried out by associating a scheduler (also referred to as an adversary) to the pMSG.

A Markov Chain is a probabilistic system in the sense that transitions between states are based on a transition probability function. Imagine a state s with two possible successor states s_1 and s_2 . This means that being in state s the choice of moving to state s_1 or s_2 is determined by a probability function which assigns to each transition a probability value. Moreover, the probability to move from one state to another depends only on the current state of the the system. Thus, the predecessor states do not have any effect on the transition probability of the current state. The complete independence of the transition probability function to the predecessor states is known as the *memoryless property*.

Definition 4.3.1. (*Markov Chain*)

A Markov chain is a system $\mathcal{MC} = (S, P, i_{init}, AP, L)$ such that:

- S is a nonempty countable set of states,
- $P : S \times S \rightarrow [0, 1]$ is a transition probability function that assigns to each transition a probability value such that for all s :

$$\sum_{s' \in S} P(s, s') = 1,$$

- $i_{init} : S \rightarrow [0, 1]$ is an initial distribution that assigns to the states a probability for being the initial state of the system such that:

$$\sum_{s \in S} i_{init}(s) = 1,$$

- AP is a set of atomic propositions,

- $L : S \rightarrow 2^{AP}$ is a labeling function that assigns a set of atomic propositions to each state.

■

A Markov Chain consists of a set of states S , where each state $s \in S$ has a certain probability of being an initial state. The probability of being an initial state will be determined by the initial distribution function i_{init} such that all states $s \in S$ with $i_{init}(s) > 0$ are possible initial states. A transition in a Markov Chain is solely probabilistic. The probability of a transition will be determined by the transition probability function P . The probability that the system will move from s to an arbitrary state s' is $P(s, s')$ where all states $s' \in S$ with $P(s, s') > 0$ are possible successors for the state s . Further, each state in a Markov Chain is labeled with a set of atomic propositions. This is mainly helpful to interpret logical formulas over Markov Chains. A Markov Chain is depicted by its corresponding digraph where the vertices in the digraph represents the states of the Markov Chain and furthermore there is an edge from the vertex s to s' iff $P(s, s') > 0$.

Example 4.3.1. Consider a simple weather model in which there are only two possible states of weather which are *sunny* or *rainy*. Each state of weather is assigned with an initial distribution of 0.5. The weather outcome for the next day depends only on the current state of weather. If the current state of weather is sunny, then there is a 70% possibility that the next day will also be sunny. When the current state of weather is rainy, then the following day would also be rainy with the possibility of 45%. The Markov Chain for the mentioned weather model can be depicted in Figure 4.3

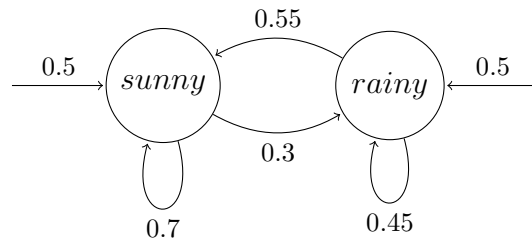


Figure 4.3: A Markov Chain for a simple weather model.

A path π in a Markov Chain is an infinite sequence of states $s_0s_1s_2\dots \in S^\omega$ such that $P(s_i, s_{i+1}) > 0$ for $i \geq 0$. Every prefix of an infinite path π is a finite path if it ends in a state. For a Markov Chain \mathcal{MC} the set of all infinite paths is denoted by $Path(\mathcal{MC})$ and the set of all finite paths is denoted by $Path_{fin}(\mathcal{MC})$.

The reason that we use a Markov Chain is the simplicity to reason about qualitative and quantitative properties through incorporating a σ -algebra and a probability measure to the set of paths. Hence, we can associate probabilities to these sets of paths. For example we would like to question if a Markov Chain runs through a certain path (which can be a bad behavior or a good behavior) with a certain desired probability. Or we would like to know the probability of all good (or bad) paths of a Markov Chain.

The probability space of a Markov Chain \mathcal{MC} is defined as follows:

- $\Omega = Path(\mathcal{MC})$,
- $\mathcal{F} = Cyl(Path_{fin}(\mathcal{MC}))$ is the smallest σ -algebra containing all the cylinder sets over all finite paths of \mathcal{MP} . Where

$$Cyl(\bar{\pi}) = \{\pi \in Path(\mathcal{MC}) \mid \bar{\pi} \text{ is a prefix of } \pi\}$$

for a finite path $\bar{\pi} \in Path_{fin}(\mathcal{MC})$,

- The probability measure on \mathcal{F} (in this case the cylinder set spanned by the finite paths) is defined as follows:

$$Pr(Cyl(\bar{\pi})) = i_{init}(s_0) \cdot P(\bar{\pi})$$

where $\bar{\pi} = s_0s_1s_2\dots s_n \in Path_{fin}(\mathcal{MC})$ is a finite path in \mathcal{MC} , $0 \leq i \leq n$ and $P(\bar{\pi}) = \prod_0^{n-1} P(s_i, s_{i+1})$ (P is the transition probability function of \mathcal{MC}).

4.4 From pMSG to Markov Chain

In the previous section we have defined a Markov Chain which will be used as our probabilistic model. The benefit of using a Markov Chain is the straightforwardness in associating a probability space to reason about qualitative and

quantitative properties of the system. Further, using a Markov Chain we can imitate the probability behavior of a pMSG as long as we can resolve the nondeterminism. This section will be concentrated in transforming a pMSG to an equivalent Markov Chain. This means that we would like to generate a Markov Chain from a pMSG without losing the characteristic of the pMSG.

Since a transition in a pMSG is also nondeterministic beside being probabilistic, we have to find a way to resolve this nondeterminism before associating it to a Markov Chain. The nature of pMSGs being probabilistic and also nondeterministic shows its similarity to a Markov Decision Process. In a Markov Decision Process there is a common procedure used to resolve the nondeterminism by using a scheduler. A scheduler decides in every nondeterministic branch which action to take. This motivated us to apply schedulers to resolve nondeterminism in a pMSG. A scheduler in a pMSG is a function that assigns an action to each vertex in a way that the pMSG will never have to choose between actions when being situated so.

Definition 4.4.1. (*Scheduler for a pMSG*)

Let $G = (V, Act, P, v_{init}, v_f, \lambda)$ be a pMSG. A scheduler \mathfrak{A} for G is a function $\mathfrak{A} : V \rightarrow Act$ such that $\mathfrak{A}(v) \in Enabled(v)$ for all $v \in V$.

■

Associating a scheduler to a pMSG will transform the pMSG in a probabilistic system where all nondeterministic choices will be resolved with respect to the chosen scheduler. This allows us to assign a Markov Chain to every pMSG governed by a scheduler. Since each vertex in a pMSG may have several actions to choose nondeterministically, it implies that there are different schedulers that can be associated to a pMSG. Note that the scheduler defined here is memoryless. This means that the scheduler chooses the next action independent from the previous chosen actions.

In the following we will define a transformation of a pMSG governed by a scheduler to a corresponding Markov Chain. The resulting Markov Chain will behave exactly like the pMSG under the influence of the scheduler. This means that there is a one-to-one correspondence between paths in the pMSG governed by the scheduler and paths in the corresponding Markov Chain.

Definition 4.4.2. (*From pMSG to Markov Chain*)

Let $G = (V, Act, P_G, v_{init}, v_f, \lambda)$ be a pMSG and \mathfrak{A} a scheduler for G . The corresponding Markov Chain for G with respect to \mathfrak{A} , $\mathcal{MC}_{G_{\mathfrak{A}}} = (S, P_{\mathcal{MC}}, i_{init}, AP, L)$, is defined as follows:

- $S := V$,
- $P_{\mathcal{MC}_{G_{\mathfrak{A}}}} : S \times S \rightarrow [0, 1]$ such that:

$$P_{\mathcal{MC}_{G_{\mathfrak{A}}}}(v, v') = P_G(v, \mathfrak{A}(v), v'),$$

- $i_{init}(v) = \begin{cases} 1 & \text{if } v = v_{init} \\ 0 & \text{otherwise.} \end{cases}$

■

The Markov Chain induced from a pMSG governed by a scheduler has the set of vertices of the pMSG as its set of states. The transition probability function of the Markov Chain is obtained by applying the scheduler on the actions in the transition probability function of its pMSG. Furthermore we set the initial distribution of the Markov Chain to one iff it is applied to the corresponding state of the initial vertex in the pMSG. The use of a set of atomic propositions has been omitted here since there is no interest of interpreting logical formulas over the Markov Chain.

By examining the pMSG G_1 from Figure 4.1f, a nondeterministic choice takes place only in the vertex v_1 - the nondeterministic choice between the actions α_1 and α_2 . Thus, there are two possible schedulers for G_1 . The first possible scheduler \mathfrak{A}_1 chooses always the action α_1 in vertex v_1 and the second possible scheduler \mathfrak{A}_2 chooses always the action α_2 when in v_1 . Thus, G_1 can be transformed to a Markov Chain by associating the scheduler \mathfrak{A}_1 or \mathfrak{A}_2 .

Example 4.4.1. Let \mathcal{MC}_i be the Markov Chain obtained by associating G_1 with the scheduler \mathfrak{A}_i (for $i \in \{1, 2\}$). Then the Markov Chains \mathcal{MC}_1 and \mathcal{MC}_2 looks as follows:

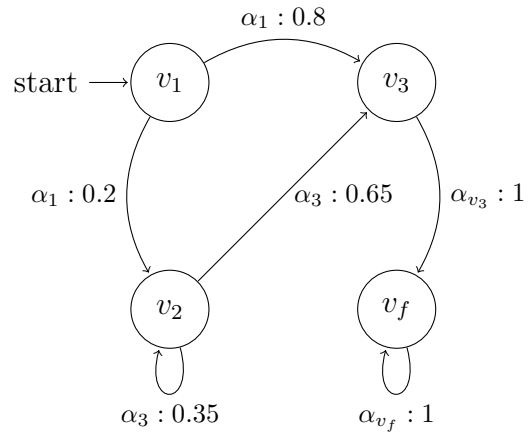


Figure 4.4: A Markov Chain \mathcal{MC}_1 from G_1 with respect to the scheduler \mathfrak{A}_1 .

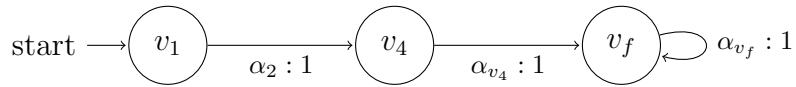


Figure 4.5: A Markov Chain \mathcal{MC}_2 from G_1 with respect to the scheduler \mathfrak{A}_2 .

Chapter 5

Associating Logic to pMSCs

5.1 Introduction

This chapter introduces a specification language which will be used to express the properties of pMSCs (it also can be used for pMSGs). A specification language for pMSCs should be able to express the properties of an event and furthermore it should be able to express sequences of events equipped with a probability value. Using such a specification language, we would be able to express the execution order of events with probabilities, i.e, we could express the probability that the events $e_1e_2e_3e_4e_5$ are executed in this order with probability 0.342 and e_5 is the maximal event.

Since pMSCs are basically extended models of MSCs (that the channels in pMSCs can lose messages with a predefined probability), we adopted a language used to specify MSCs and extend it with a probabilistic notion. There are several studies which have been focused on defining or extending a specification language for the purpose of MSCs [8]. In this chapter we present an extended version of PDL, called Probabilistic Propositional Dynamic Logic (PPDL). PPDL extends PDL in such a way that path expressions are equipped with a probability operator that defines the probability of the path. Moreover, in PPDL we omit the use of quantification in global formulas as in PDL. Beside these two differences, both specification languages have syntactically similar definitions.

The structure of this chapter is the following: first, the syntax and semantics

of PPDL will be defined. This will be followed by comprehensive examples to show how to check if a pMSC satisfies a PPDL formula. In the end of this chapter we will summarize a method for model checking MSCs over PDL, and the problem that we encounter if we apply this method for model checking pMSCs over PPDL.

5.2 Probabilistic Propositional Dynamic Logic

PPDL is an extension of PDL where paths are equipped with a probabilistic operator. This probability operator is applied to the forward and backward-path formulas. In PDL the formula $\langle \lambda \rangle \alpha$ is used to express the occurrence of a forward-path $\langle \lambda \rangle$, where the last event in the path λ has to fulfill the local formula α (analog for the backward-path formulas $\langle \lambda \rangle^{-1} \alpha$). In our PPDL language we enrich the path expressions with a probability operator such that the path is linked to a probability bound, i.e., the formula $\langle \lambda \rangle_{\sim p} \alpha$ means that the forward-path $\langle \lambda \rangle_{\sim p}$ is bound by the probability value p and the last event has to fulfill the local formula α (analog for the backward-path formulas $\langle \lambda \rangle_{\sim p}^{-1} \alpha$). Thus, the probability of a path in a pMSC (the occurrence of sequences of events in a pMSC) can be formulated mathematically using PPDL.

5.2.1 Syntax of PPDL

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m, <, C, p_{loss})$ be a pMSC and $\Pi(M)$ the set of posets of M . The two following definitions will be defined over the pMSC M .

Definition 5.2.1. (*Syntax of PPDL*)

The syntax of PPDL is defined over three different categories. These categories are:

1. Path expressions, which are ranged over by λ . Let α be a local formula (defined below), then the syntax of a path expression λ is defined as follows:

$$\lambda ::= proc \mid msg \mid \{\alpha\} \mid \lambda; \lambda \mid \lambda + \lambda \mid \lambda^*$$

- Local formulas, which are ranged over by α . The syntax of a local formula α is defined as follows:

$$\alpha ::= true \mid \sigma \mid \alpha \vee \alpha \mid \neg \alpha \mid \langle \lambda \rangle_{\sim p} \alpha \mid \langle \lambda \rangle_{\sim p}^{-1} \alpha$$

where $\sigma \in Act$, λ is a path expression, $\sim \in \{\leq, \geq\}$ and $p \in [0, 1]$.

- Global formulas, ranged over by φ . The syntax of a global formula φ is defined as follows:

$$\varphi ::= \alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

where α is a local formula.

■

Upon closer inspection, Definition 5.2.1 does not differ so much from Definition 2.3.1. Definition 5.2.1 extends PDL in associating the probability operator \sim_p to paths, where $\langle \lambda \rangle_{\leq p}$ means that the path λ has a probability less than or equal to p and $\langle \lambda \rangle_{\geq p}$ means that the path λ has a probability greater than or equal to p (analog for the backward-path $\langle \lambda \rangle_{\sim p}^{-1}$).

5.2.2 Semantics of PPDL

The semantics of PPDL path expressions are defined exactly as in PDL over a pair of events. On the other hand, the semantics of PPDL local and global formulas are not defined over a pMSC M , but over the set of posets $\Pi(M)$.

Definition 5.2.2. (*Semantics of PPDL*)

- The semantics of PPDL path expressions are defined as in Definition 2.3.2 for PDL path expressions (see Section 2.3).
- Let $\sigma \in Act$, λ a PPDL path expression and $\alpha, \alpha_1, \alpha_2$ PPDL local formulas. The semantics of PPDL local formulas is defined over $\Pi(M)$ and an event $e \in E$ as follows:

- $\Pi(M), e \models true \forall e \in E \setminus E_{\odot}$
- $\Pi(M), e \models \sigma \iff l(e) = \sigma$
- $\Pi(M), e \models \alpha_1 \vee \alpha_2 \iff \Pi(M), e \models \alpha_1 \text{ or } \Pi(M), e \models \alpha_2$

- $\Pi(M), e \models \neg\alpha \iff \Pi(M), e \not\models \alpha$
- $\Pi(M), e \models \langle \lambda \rangle_{\sim p} \alpha \iff \exists e' \in E. (e, e') \models \lambda \wedge \Pi(M), e' \models \alpha \wedge \wp(\langle \lambda \rangle_{\sim p} \alpha)$
where $\wp(\langle \lambda \rangle_{\sim p} \alpha) =$

$$\begin{cases} \Pr(\{(E, <' \in \Pi(M) | e \in (E, <' \wedge (E, <'', e \models \langle \lambda \rangle \alpha)\} \leq p & \text{if } \sim = \leq \\ \Pr(\{(E, <' \in \Pi(M) | e \in (E, <' \wedge (E, <'', e \models \langle \lambda \rangle \alpha)\} \geq p & \text{if } \sim = \geq \end{cases}$$

- $\Pi(M), e \models \langle \lambda \rangle_{\sim p}^{-1} \alpha \iff \exists e' \in E. (e', e) \models \lambda \wedge \Pi(M), e' \models \alpha \wedge \wp(\langle \lambda \rangle_{\sim p}^{-1} \alpha)$
where $\wp(\langle \lambda \rangle_{\sim p}^{-1} \alpha) =$

$$\begin{cases} \Pr(\{(E, <' \in \Pi(M) | e \in (E, <' \wedge (E, <'', e \models \langle \lambda \rangle^{-1} \alpha)\} \leq p & \text{if } \sim = \leq \\ \Pr(\{(E, <' \in \Pi(M) | e \in (E, <' \wedge (E, <'', e \models \langle \lambda \rangle^{-1} \alpha)\} \geq p & \text{if } \sim = \geq \end{cases}$$

3. Let α be a PDDL local formula. The semantics of PDDL global formulas is defined over $\Pi(M)$ as follows:

- $\Pi(M) \models \alpha \iff \exists e \in E_{min} \wedge \Pi(M), e \models \alpha$
- $\Pi(M) \models \varphi_1 \vee \varphi_2 \iff \Pi(M) \models \varphi_1 \text{ or } \Pi(M) \models \varphi_2$
- $\Pi(M) \models \varphi_1 \wedge \varphi_2 \iff \Pi(M) \models \varphi_1 \text{ and } \Pi(M) \models \varphi_2$

where $E_{min} = \{e \in E | \neg \exists e'. e' < e\}$.

■

PPDL semantics for local formulas are essentially similar to PDL. The semantics of their forward- and backward-path formulas are extended with a term describing the condition for the probability of paths. Consider the semantics of the forward-path formula in PPDL:

$$\begin{aligned} \Pi(M), e \models \langle \lambda \rangle_{\sim p} \alpha \\ \iff \\ \exists e' \in E. \underbrace{(e, e') \models \lambda}_1, \underbrace{\Pi(M), e' \models \alpha}_2 \wedge \underbrace{\wp(\langle \lambda \rangle_{\sim p} \alpha)}_3 \end{aligned}$$

1. The first term $(e, e') \models \lambda$ defines that the path λ should start from the event e and ends in e' .

2. The second term $\Pi(M), e' \models \alpha$ defines that given the poset $\Pi(M)$ and the last event from the path λ (in our case here the event e') the local formula α must be satisfied.
3. The third terms determine the probability aspect. We know that $\wp(\langle \lambda \rangle_{\sim p} \alpha) =$

$$\begin{cases} Pr(\{(E, <)' \in \Pi(M) | e \in (E, <)' \wedge (E, <)', e \models \langle \lambda \rangle \alpha\}) \leq p & \text{if } \sim = \leq \\ Pr(\{(E, <)' \in \Pi(M) | e \in (E, <)' \wedge (E, <)', e \models \langle \lambda \rangle \alpha\}) \geq p & \text{if } \sim = \geq \end{cases}$$

The probability boundary for the posets satisfying the formula $\langle \lambda \rangle \alpha$ depends on which symbol is associated to \sim , greater than or equal, or less than or equal ($\sim \in \{\geq, \leq\}$). I.e., if $\sim = \geq$ then the sum of the probabilities of all posets satisfying the local formula $\langle \lambda \rangle \alpha$ must be greater than or equal to the probability value p .

Example 5.2.1. Imagine a pMSC M with three processes *User*, *Interface* and *Resource*. The pMSC M starts by the *User* sending a request message to the *Interface* for accessing the *Resource*. After receiving the request message, the *Interface* forwards the request message to the *Resource*. The *Resource* grants the request by sending a granting message to the *Interface* which will be forwarded to the *User* by sending a message denoted by *yes*. The graphical representation is depicted in Figure 5.1.

Imagine some probabilistic behavior of pMSC M which is expected to hold or not. Let's first consider a desired behavior, where the path starting from the *User* sends the request message until the *Resource* grants the request, by sending the message *grant* to the *Interface*, will be given a probability of at least 0.5. Given the pMSC M , this behavior can be expressed by the following PDDL global formula:

$$\varphi_1 = \langle msg; proc; msg; proc; msg \rangle_{\geq 0.5} Interface?Resource(grant)$$

The path expression in φ_1 - $msg; proc; msg; proc; msg$ - specifies the path of events $e_1 e_2 e_3 e_4 e_5 e_6$. This path of events is desired to have a probability of at least 0.5 and the last event in this path - e_6 - has to satisfy the local formula $Interface?Resource(grant)$.

To examine whether pMSC M satisfies the global formula φ_1 , first we have to know the set of all possible posets $\Pi(M)$ for the pMSC M . This set can

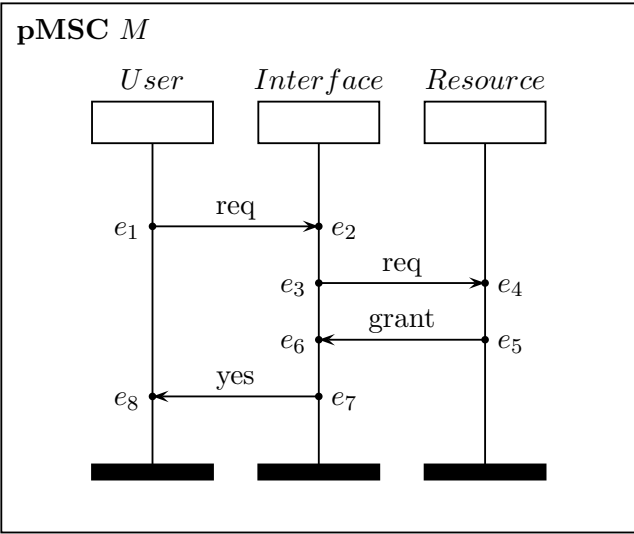
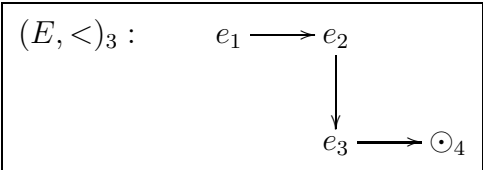
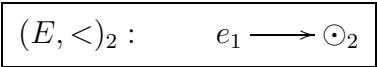
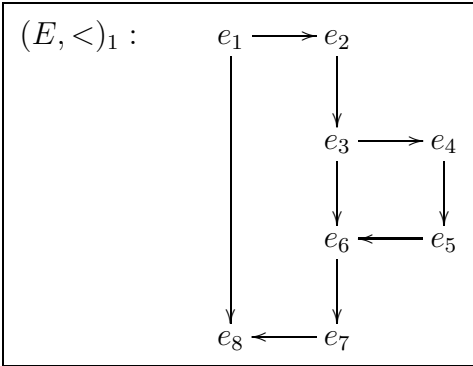
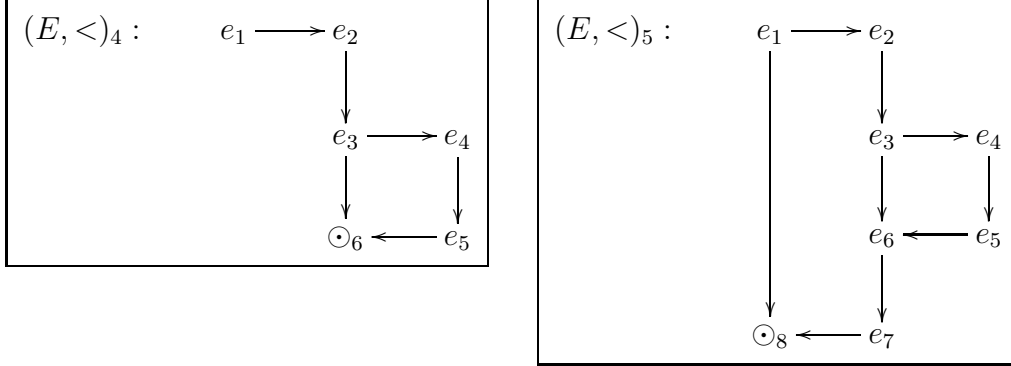


Figure 5.1: pMSC M .

be generated by applying the algorithm in Section 3.3.1. By executing the algorithm we will obtain the following posets:





$\Pi(M) = \{(E, <)_1, (E, <)_2, (E, <)_3, (E, <)_4, (E, <)_5\}$ is the set of posets for the pMSC M . The first poset $(E, <)_1$ is the standard poset in which all the events in M occur. In the second poset $(E, <)_2$ the message *req* which is sent from the *User* to the *Interface* is lost in the channel. Thus, all successor events of e_2 (which are the events e_3, e_4, e_5, e_6, e_7 and e_8) will also not occur in the poset. The third poset $(E, <)_3$ describes the poset where the *Interface* has received the message *req* from the *User*, but while forwarding the message to the *Resource* it is lost in the channel. This implies that the receive event e_4 will not occur, which implies to the nonoccurrence of the events e_5, e_6, e_7 and e_8 (which are the successor events of e_4). $(E, <)_4$ is the poset where the message granting access to the *Resource* is not received by the *Interface*, which is represented by the lost event \odot_6 . Accordingly the successor events e_7 and e_8 of the event e_6 will not occur in the poset. The last poset $(E, <)_5$ describes the loss of the message *yes* from the *Interface* to the *User*. This results in the lost event e_8 . Since the receive event \odot_8 does not have any successors, all events occur in the poset except e_8 .

To see which posets in $\Pi(M)$ satisfy the global formula φ_1 , one needs to examine in which posets in $\Pi(M)$ the path of events $e_1e_2e_3e_4e_5e_6$ do appear. These are the posets $(E, <)_1$ and $(E, <)_5$. Let's assume that the probability of losing a message in the channels are the following:

- $p_{loss}(User, Interface) = p_{loss}(Interface, User) = 0.121$, and
- $p_{loss}(Interface, Resource) = p_{loss}(Resource, Interface) = 0.074$.

By multiplying the probability of the events in the posets (see Definition 3.4.2) we obtain the probability of $(E, <)_1 \approx 0.6625$ and the probability of

$$(E, <)_5 \approx 0.0912.$$

Next we will analyze if the $\Pi(M)$ satisfies the global formula φ_1 .

$$\begin{aligned} \Pi(M) &\models \langle msg; proc; msg; proc; msg \rangle_{\geq 0.5} \underbrace{Interface?Resource(grant)}_{=\alpha} \\ &\iff \\ \exists e \in E_{min} \wedge \Pi(M), e &\models \underbrace{\langle msg; proc; msg; proc; msg \rangle_{\geq 0.5}}_{=\lambda} \underbrace{Interface?Resource(grant)}_{=\alpha} \\ &\iff \\ \underbrace{\exists e \in E_{min}}_{\text{satisfied by } e_1} \wedge &\underbrace{\exists e' \in E.(e, e') \models \lambda}_{\text{satisfied by the path } e_1 e_2 e_3 e_4 e_5 e_6} \wedge \underbrace{\Pi(M), e' \models \alpha}_{\text{satisfied by } e_6} \wedge \underbrace{\wp(\langle \lambda \rangle_{\geq 0.5} \alpha)}_{?} \end{aligned}$$

What is left to prove is the last inequation $\wp(\langle \lambda \rangle_{\geq 0.5} \alpha)$. Based on the set $\Pi(M)$ only the posets $(E, <)_1$ and $(E, <)_5$ satisfy the local formula $\langle \lambda \rangle \alpha$ (note: $\lambda = msg; proc; msg; proc; msg$ and $\alpha = Interface?Resource(grant)$). This implies that:

$$\begin{aligned} Pr(\{(E, <)' \in \Pi(M) | e \in (E, <)' \wedge (E, <)' \models \langle \lambda \rangle \alpha\}) &= \\ Pr((E, <)_1) + Pr((E, <)_5) &\approx \\ 0.6625 + 0.0912 &= \\ 0.7537 & \end{aligned}$$

Therefore,

$$\Pi(M) \models \langle msg; proc; msg; proc; msg \rangle_{\geq 0.5} Interface?Resource(grant).$$

Example 5.2.2. Let's consider now an undesired behavior of the pMSC M such that the message *yes* from the *Interface* never arrives to the *User*, even when it has been granted from the *Resource*. This scenario should occur with a very low probability - for example a probability of 0.2 at most. This undesired behavior can occur in two different scenarios:

1. The first scenario is that the *Resource* has sent the message *grant* to the *Interface* but the *Interface* did not receive it. Thus, all the successor events of e_6 (*Interface* receiving the message *grant* from *Resource*) will not occur.

2. The second possible scenario is that the message *grant* has been received by the *Interface* and therefore, the *Interface* can send the message *yes* to the *User*, but this will not be received by the *User*.

The two scenario above can be written in a PDDL global formula φ_2 as follows:

$$\langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false$$

with

- $\lambda_1 = ((msg; proc)^2; \{Resource!Interface(grant)\}; msg)$ and
- $\lambda_2 = ((msg; proc)^2; \{Resource!Interface(grant)\}; (msg; proc); \{Interface!User(yes)\}; msg)$.

Through closer examination on the set $\Pi(M)$ it turns out that the poset $(E, <)_4$ reflects the behavior of the first scenario and the poset $(E, <)_5$ reflects the behavior of the second scenario. Thus, $(E, <)_4$ and $(E, <)_5$ are the only two posets in $\Pi(M)$ that could satisfy φ_2 .

The satisfiability of the global formula φ_2 is solved in the following way:

$$\begin{aligned}
& \Pi(M) \models \varphi_2 \\
& \iff \\
& \Pi(M) \models \langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false \\
& \iff \\
& \exists e \in E_{min} \wedge \Pi(M_1), e \models \langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false \\
& \iff \\
& \underbrace{\exists e \in E_{min}}_{\text{satisfied by } e_1} \\
& \wedge \\
& \left(\left(\underbrace{\exists e' \in E.(e, e') \models \lambda_1}_{\text{satisfied by the path } e_1 e_2 e_3 e_4 e_5 \odot_6} \right) \vee \left(\underbrace{\exists e'' \in E.(e, e'') \models \lambda_2}_{\text{satisfied by the path } e_1 e_2 e_3 e_4 e_5 e_6 e_7 \odot_8} \right) \right)
\end{aligned}$$

$$\begin{array}{c}
\wedge \\
\underbrace{\Pi(M_1), e' \models false}_{\text{satisfied by } \odot_6 \text{ and } \odot_8} \\
\wedge \\
\wp(\langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false)
\end{array}$$

Further:

$$\begin{aligned}
Pr(\{(E, <)' \in \Pi(M) | e \in (E, <)' \wedge (E, <)' \models \langle \lambda_1 + \lambda_2 \rangle false\}) = \\
Pr((E, <)_4) + Pr((E, <)_5) \approx \\
0.0602 + 0.0912 = \\
0.1514
\end{aligned}$$

This implies the validity of $\wp(\langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false)$, and therefore

$$\Pi(M_1) \models \langle \lambda_1 + \lambda_2 \rangle_{\leq 0.2} false.$$

5.3 Model Checking

Model checking is a fully automated process which analyzes if a model satisfies a certain specification. In cases of models with messages exchanged, the model is generally represented by an automaton-based model and the specification is represented by a formal mathematical system. If the automaton-based model and the formal mathematical system are appointed, then model checking centers on the question: if the automata-based model satisfies a specification formula from the formal mathematical system. If \mathcal{A} is an automaton-based model describing a scenario of message exchanges and φ is a specification formula from a formal mathematical system, then the model checking verifies if $L(\mathcal{A}) \subseteq L(\varphi)$ (where $L(\mathcal{A})$ and $L(\varphi)$ define the language of \mathcal{A} and φ respectively).

5.3.1 Model Checking of PDL over MSCs

This section will discuss an approach from Bollig, Kuske and Meinecke [8] for model checking MSCs over PDL formulas. The intention here is to show the idea of their method and to point out the difficulty in adopting this method to our model.

The Model and The Specification Language

The model checking approach in [8] uses slightly modified Communication Finite-State Machines (CFMs) as an automaton-based model for describing the behavior of a MSC. Furthermore, to represent the specification to be examined they use PDL. A CFM is a model with a finite set of local automata communicating over a finite set of channels. Each local automaton can execute send and receive actions, that specify the sending and receiving of messages. The behavior of each local automaton is associated to a process, which means that a local automaton represents a process. The channels are asynchronous FIFO channels. Based on the characteristics of CFMs, it is widely used to represent MSCs. The PDL syntax and semantics used in [8] is comparable to Definition 2.3.1 and Definition 2.3.2.

In general a CFM is of the form $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{init}, F)$, which is defined over a finite set of processes \mathcal{P} and a finite set of message contents Σ . For each process $p \in \mathcal{P}$ there is a local automaton (S_p, Δ_p) with S_p as the set of local states and Δ_p the set of local transitions. \mathbb{D} is a set of control messages or synchronization messages. The global initial state is denoted by $s_{init} \in S_{\mathcal{A}}$, where $(S_{\mathcal{A}} = \prod_{p \in \mathcal{P}} S_p)$ and $F \subseteq S_{\mathcal{A}}$ is a set of global final states. The set of local transitions in the local automata are defined as follows:

$$\Delta_p \subseteq S_p \times Act \times S_p$$

where in our case Act is a set of actions of the form $p!q(a)$ or $p?q(a)$ (for all $p, q \in \mathcal{P}$, $p \neq q$ and $a \in \Sigma$). In [8] the local transitions on the local automata have been modified. They have extended the transitions with a sequence of bits such that the input of a transition is not solely the set of actions Act , but also a sequence of bits $\{0, 1\}^n$. Thus, they get the following local transition relation:

$$\Delta_p \subseteq S_p \times Act \times \{0, 1\}^n \times S_p$$

The Model Checking Method

The approach in [8] is based on an idea of using a specification in the early stage of developing a system. If we would like to gain a system based on a predefined specification, it is more reasonable to use the specification directly to synthesize the desired system. What they did is that they translated a PDL specification that is to be examined to a corresponding CFM. The

translation from a PDL specification φ (which is a global formula) to a CFM \mathcal{A}_φ is carried out without imposing some restrictions on both sides. Through the translation, [8] obtained a CFM whose size is exponential in the size of the PDL specification and the number of processes.

We will not discuss the translation from a PDL formula φ to its corresponding CFM \mathcal{A}_φ in detail (see [8]). The translation from the formula φ to \mathcal{A}_φ is not straightforward. A PDL formula φ is defined over local formulas α_i that are quantified (by using the quantifier \exists and \forall), which is called *basic global formulas*, which further can be combined using two-place positive boolean operators (such as \wedge and \vee). Furthermore, a PDL local formula α is composed of the truth value *true* or an action of the set *Act* or a path expression (which can go forward and backwards) followed by a local formula or combination of the previous terms. This all follows that a PDL formula φ is constructed inductively. This also provokes the model checking method in [8]. The idea is also to build a CFM \mathcal{A}_φ inductively from the parts of the PDL formula φ .

A formula φ will be broken down into its subformulas. For each subformula $\beta \in \text{sub}(\varphi)$ a CFM \mathcal{A}_β will be modeled. The task of these CFMs $\{\mathcal{A}_\beta\}$ is to confirm if the subformulas are satisfied by the MSC being examined. Therefore, [8] have associated to every MSC a function $c : E \rightarrow \{0, 1\}^n$ that maps a sequence of bits to every event in the MSCs. By doing so, each subformula β has a reserved bit $c_\beta(e)$ in the bit sequence of every event $e \in E$. Hence, the CFM \mathcal{A}_β can confirm if the bit associated to the subformula β in every bit sequence is set to 1 for exactly those events that satisfy the subformula β . By running all the CFMs of the subformulas synchronously we will obtain the overall CFM for the formula φ .

The model checking in [8] turns out to be PSPACE-complete for existentially B -bounded MSCs.

Model Checking on pMSCs

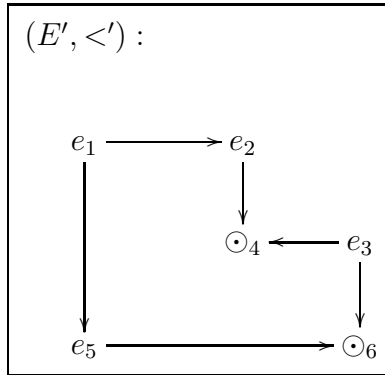
We think that the model checking approach in [8] can not be applied directly to our pMSCs defined in this paper. This is mainly founded based on two points: first, the PDL global formulas in [8] are defined over MSCs where on the other hand, our PDDL global formulas are defined over the set of posets;

second, the forward and backward-path formulas in our PPDL are equipped with a probability operator. Thereby, the CFM for the PPDL global formula must also be defined with a probability operator.

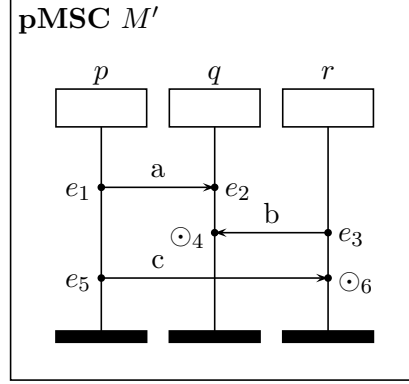
We believe that through some modification in the approach in [8] we could apply the model checking of pMSCs over PPDL formulas. The idea is basically that the model checking of a pMSC over a PPDL global formula can be determined through model checking a set of MSCs over a PDL global formula.

The general model checking question is: given a pMSC M and a PPDL global formula φ , do the set of posets $\Pi(M)$ of M satisfy the formula φ ($\Pi(M) \models \varphi$)? The model checking approach idea that will be presented subsequently considers two cases. The first case considers PPDL global formulas that do not have forward and/or backward-path formulas as subformulas. The second case considers PPDL global formulas with forward- and/or backward-path formulas as subformulas.

Consider the following poset $(E', <')$ of a pMSC:



We can consider the above poset as a MSC M' . The idea to transform a pMSC poset to a MSC is straightforward. The MSC M' looks as follows:



Thus, from a set of posets $\Pi(M)$ of a pMSC M we can obtain a set of MSCs \mathcal{M} with $|\Pi(M)| = |\mathcal{M}|$. For the first case, where the PDDL global formula does not have any forward- and/or backward path formulas as subformulas, we can consider the PDDL global formula as a PDL global formula - since there is no probability operator associated. In this case, we could just apply the model checking from [8] to every MSC obtained from the posets in $\Pi(M)$. For the latter case, where the PDDL global formula has forward and/or backward-path formulas as subformulas, the model checking is not as straightforward as in the first case. In the second case, the PDDL global formula is associated with probability operators on the forward and backward-path subformulas. Each PDDL global formula with probability association can be transformed into a PDL global formula by ignoring the probability operators on the forward and backward-path subformulas. Further, we apply the model checking to all the MSCs obtained from the posets over the PDL global formula (which is obtained by ignoring the probability operators) as in [8]. For each MSC that satisfies the PDL global formula we compute the probability of the corresponding poset. By adding all these probabilities and comparing them with the probability bounds of the forward and backward-path subformulas we will obtain the satisfiability of the pMSC over the PDDL global formula.

Note that the idea described above is an idea of a possible approach for model checking pMSCs over PDDL formulas using the method in [8]. The idea still lacks precision in the individual steps. The detailed approach will not be discussed in this work and will be left as a motivation for further research.

Chapter 6

An Automaton Model for pMSCs

The best way to analyze a model is to transform it into an equivalent mathematical system. By doing so, the behavior of the model to be examined will be reflected in an equivalent mathematical system. Thus, each possible execution of the examined model can be mimicked by the corresponding mathematical system. This implies that the language defined by the model is exactly the same language defined by the mathematical system.

A well known mathematical system used to describe communication protocols such as MSCs is a *Communicating Finite-State Machine* (CFM) (see Section 5.3.1). A CFM consists of several local automata, where each local automaton describes the behavior of a process, such that there is a one-to-one correspondence between the behavior of a process in a MSC with its local automaton. A local automaton communicates with other local automata by sending and/or receiving messages through a channel. For every pair of local automata there are two channels for communication, one for each direction. The channels have an unbounded asynchronous FIFO property and are assumed to be reliable.

In this chapter we would like to extend the basic CFM with probabilistic and nondeterministic properties in such a way that it can simulate the behavior of a pMSC. The structure of this chapter will be the following: first the syntax and semantics of *Lossy Communication Finite-State Machines* (LCFM) will be defined. A LCFM is basically a CFM with the presump-

tion that channels can lose messages due to the fact that the channels are unreliable. This kind of mathematical system has been presented before in [16, 2, 9] where it is called *Lossy Channel Systems*. By using a LCFM as the basic model, we will further expand the syntax and semantics of it to obtain a model that can imitate the behavior of a pMSC. We will call this model *Probabilistic Lossy Communicating Finite-State Machine* (PLCFM).

6.1 Lossy Communication Finite-State Machines

A CFM is basically a set of finite state automata that are communicating to one another through channels. The actions that can be taken in the automaton are basically send and receive actions. Each of these actions is associated with a message. Through executing a send action, the message associated with the action will be attached to the end of the sequence of messages residing in the channel. On the other hand, the first message in the sequence will be removed from the channel by executing a receive action - the message at the beginning of the sequence must be identical to the message associated to the action and certainly the channel cannot be empty.

CFMs are based on the assumption that the channels used are reliable. This means that it is assured that no message in the channels will ever get lost. These kinds of mathematical systems are ideal to model MSCs since MSCs are also based on reliable channels. For modeling pMSCs, we need a system similar to CFMs that is based on unreliable channels.

LCFMs are CFMs where messages in the channels may get lost. This is so because the channels in LCFMs are assumed to be unreliable. This implies that every message has a possibility of getting lost while being in transit in a channel. The syntax of a LCFM does not differ from the one of a CFM. The possibility of losing messages is expressed in the semantics of LCFM.

This section starts by defining the syntax of LCFMs followed by how configurations of LCFMs are defined. A configuration in a LCFM defines a global state - which is a cross product of local states of the local automata - associated with its channel contents. Next, the transition between LCFM

configurations will be discussed, which is summarized in the semantics of LCFM. In the subsequent subsection, LCFMs will be extended to PLCFMs, where the syntax of PLCFMs is equipped with a probability function which assigns a probability value to the channels. The semantics of PLCFMs will be defined as Markov Decision Processes.

6.1.1 Syntax of LCFM

Let \mathcal{P} be a finite set of processes (at least two), $C \subseteq \mathcal{P} \times \mathcal{P}$ a relation between a pair of processes, defining their communication channel and Σ a finite set of message contents. Further, besides having asynchronous and FIFO properties, the channels C are also unreliable in the sense that messages in the channel may get lost.

Definition 6.1.1. (*Syntax of a Lossy Communicating Finite-State Machine*)

A Lossy Communicating Finite-State Machine (LCFM) over \mathcal{P} and Σ is a structure $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F)$ where:

- $((S_p, \Delta_p))_{p \in \mathcal{P}}$ are local automata,
 - S_p is a nonempty set of local states,
 - $\Delta_p \subseteq S_p \times Act \times S_p$ is a set of local transitions, where Act is a set of actions of the form $p!q(a)$ or $p?q(a)$ ($p, q \in \mathcal{P}, p \neq q$ and $a \in \Sigma$).
- $s_{init} \in S_{\mathcal{A}}$ is the global initial state ($S_{\mathcal{A}} = \prod_{p \in \mathcal{P}} S_p$),
- $F \subseteq S_{\mathcal{A}}$ is a set of global final states.

■

An example of a LCFM is depicted in Figure 6.1. The LCFM \mathcal{A}_{pq} consist of two local automata (S_p, Δ_p) and (S_q, Δ_q) , where the former can be considered to represent the behavior of a process p and the latter the behavior of a process q . The global initial state of \mathcal{A}_{pq} is $s_{init_{pq}} = (p_1, q_1)$ and the unique global final state is $F = \{(p_2, q_2)\}$.

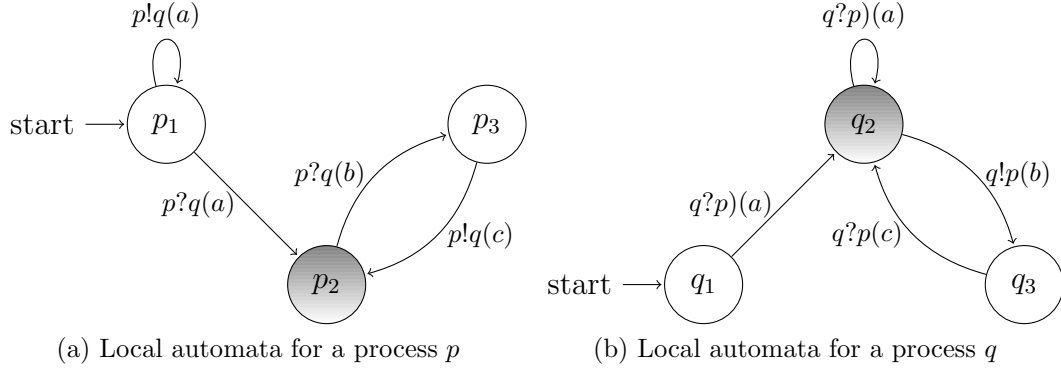


Figure 6.1: Example for a LCFM \mathcal{A}_{pq} .

There have been several approaches to how a lost message should be defined [16, 2, 9]. In this paper we will take the approach that was first introduced in [2] and later also used in [16, 6, 3, 1]. In this approach the loss of a message will be defined on the semantics of the system, instead of extending the local transition of the system as in [9]. First, a type of relation between strings (which later will be applied to the configurations) will be specified. This is essentially important in defining the meaning of message lost in the semantics of LCFM.

Let $x, y \in \Sigma^*$ be strings over Σ . We say that the string x is a substring of y , denoted by $x \sqsubseteq y$, if by deleting arbitrary numbers (probably zero) of symbols in y at arbitrary places, we obtain the string x . Further, $\#(x, y)$ denotes the number of possibilities there is to delete symbols in the string y to obtain the string x .

Example 6.1.1. Consider the two strings $x = baa$ and $y = baaba$. The string x is a substring of the string y , since by deleting arbitrary symbols in y we obtain x . Thus $x \sqsubseteq y$. Further, there are three possibilities to delete symbols in y to obtain x : $\underline{b}aaba$, $ba\underline{a}ba$ and $baab\underline{a}$. Thus $\#(x, y) = 3$.

Definition 6.1.2. (*Configuration of a Lossy Communication Finite-State Machine*)

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F)$ be a LCFM. $\sigma = \langle \bar{s}, \eta \rangle$ is a configuration of \mathcal{A} with:

- $\bar{s} \in S_{\mathcal{A}}$ is a global state of \mathcal{A} ,
- $\eta : C \rightarrow \Sigma^*$ is a function that maps a channel on its content.

■

Consider the LCFM \mathcal{A}_{pq} from Figure 6.1. Imagine that the local automata (S_p, Δ_p) is in the local state p_2 and (S_q, Δ_q) in the local state q_2 . Further the content of the channels (p, q) and (q, p) are as in Figure 6.2. This situation can be captured by the configuration $\sigma = \langle \bar{s}, \eta \rangle$, where $\bar{s} = (p_2, q_2)$, $\eta((p, q)) = aa$ and $\eta((q, p)) = \epsilon$.

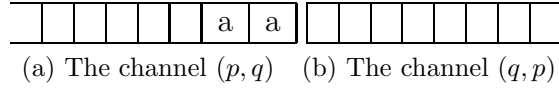


Figure 6.2: Possible channel contents of LCFM \mathcal{A}_{pq}

To specify the meaning of a channel losing messages, it is necessary to define substring relations on configurations. To be more precise, the substring relation will be defined based on the content of the channels in a configuration.

Definition 6.1.3. (*Substring relation on configurations*)

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F)$ be a LCFM over \mathcal{P} and Σ and σ, σ' configurations of \mathcal{A} . We say that σ is a substring of σ' , denoted with $\sigma \sqsubseteq \sigma'$, if the following condition holds (let $\sigma = \langle \bar{s}, \eta \rangle$ and $\sigma' = \langle \bar{s}', \eta' \rangle$):

- $\bar{s} = \bar{s}'$,
- $\eta(p, q) \sqsubseteq \eta'(p, q)$ for all $(p, q) \in C$.

■

6.1.2 Semantics of LCFM

The operational behavior of a LCFM is defined as a transition system. The states of the corresponding transition system are the set of configurations of the LCFM. Due to the fact that the channels in a CFM are unbounded,

the number of configurations of the LCFM is infinite. This implies that the corresponding transition system also has infinite number of states. There are two kinds of transitions in the corresponding transition system. The first kind is referred to as *perfect* transitions. In a perfect transition no messages will be lost from the channels. These transitions are governed by the transition rules of the local automata of the LCFM. A perfect transition between two configurations is possible if there is a local state in the former configuration with its possible successor in the latter configuration. Further, all other local states remain unchanged in the two configurations. The content in the channels of the two configurations are identical, except the channel associated to the action in the LCFM which is responsible for the transition (in this channel either a message will be added to the channel or removed from the channel). The second kind of transitions is called a *lossy* transitions. In a lossy transition the global state of the predecessor and successor remains the same. The only thing that can change is the content of the channels in the successor configuration.

Definition 6.1.4. (*Semantics of a Lossy Communicating Finite-State Machine*)

For a LCFM $\mathcal{A} = ((S_p, \Delta_p)_{p \in \mathcal{P}}, s_{init}, F)$ over \mathcal{P} and Σ , the semantics of LCFM is defined as a transition system $TS_{\mathcal{A}} = (\mathcal{S}, \longrightarrow)$ with:

- \mathcal{S} is the set of configurations of LCFM \mathcal{A} ,
- $\longrightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is a set of transitions. There are two types of transitions: a *perfect* transition $\longrightarrow_{perfect}$, which is determined by $(\Delta_p)_{p \in \mathcal{P}}$ and a *lossy* transition \longrightarrow_{lossy} which is determined by the substring relation on configurations. The two transitions are defined as follows:
 - For $\sigma, \sigma' \in \mathcal{S}$, there is a transition $\sigma \longrightarrow_{perfect} \sigma'$ if one of the following applies:
 - * **sending a message:** there is a transition $\delta \in \Delta_p$ with $\delta = (s_p, p!q(a), s'_p)$ such that
 - $\sigma = \langle \bar{s}, \eta \rangle$ with $\bar{s}[p] = s_p$,
 - $\sigma' = \langle \bar{s}', \eta' \rangle$ with $\bar{s}'[p] = s'_p$ and $\eta'(p, q) = a \cdot \eta(p, q)$,
 - $\bar{s}[r] = \bar{s}'[r]$ for all $r \in \mathcal{P} \setminus \{p\}$,
 - $\eta(q, r) = \eta'(q, r)$ for all $(q, r) \in C \setminus \{(p, q)\}$.

- * **receiving a message:** there is a transition $\delta \in \Delta_p$ with $\delta = (s_p, p?q(a), s'_p)$ such that
 - $\sigma = \langle \bar{s}, \eta \rangle$ with $\bar{s}[p] = s_p$ and $\eta(p, q) = w \cdot a$ ($w \in \Sigma^*$),
 - $\sigma' = \langle \bar{s}', \eta' \rangle$ with $\bar{s}'[p] = s'_p$ and $\eta'(p, q) \cdot a = \eta(p, q)$,
 - $\bar{s}[r] = \bar{s}'[r]$ for all $r \in \mathcal{P} \setminus \{p\}$,
 - $\eta(q, r) = \eta'(q, r)$ for all $(q, r) \in C \setminus \{(p, q)\}$.
- For $\sigma, \sigma' \in \mathcal{S}$, there is a transition $\sigma \xrightarrow{\text{lossy}} \sigma'$ if there exists a perfect transition $\sigma \xrightarrow{\text{perfect}} \sigma''$ and $\sigma' \sqsubseteq \sigma''$.

■

Definition 6.1.5. (*A run of a Lossy Communication Finite-State Machine*)

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F)$ be a LCFM over \mathcal{P} and Σ . Further let $TS_{\mathcal{A}} = (\mathcal{S}, \longrightarrow)$ be its corresponding transition system. A run of LCFM \mathcal{A} in $TS_{\mathcal{A}}$ is a sequence of configurations $\rho = \sigma_0 \sigma_1 \dots \sigma_n$ such that:

- $\sigma_0 = \langle s_{init}, \eta_\epsilon \rangle$, where η_ϵ is the function which assigns to each channel the empty string ϵ ,
- for every $i \in \{1, \dots, n\}$ there exists a transition \longrightarrow_t in $TS_{\mathcal{A}}$ with $t \in \{\text{perfect}, \text{lossy}\}$ such that $\sigma_{i-1} \longrightarrow_t \sigma_i$.

A run $\rho = \sigma_0 \sigma_1 \dots \sigma_n$ is an accepting run of LCFM \mathcal{A} if $\sigma_n \in F \times \{\eta_\epsilon\}$.

■

6.2 Probabilistic Lossy Communication Finite-State Machine

In the previous section we have introduced a mathematical system that can be used to describe communication between processes that is based on unreliable channels. This means that without any notification a message can be lost from any arbitrary channel in the system. We have called this mathematical system a Lossy Communication Finite-State Machine (LCFM) which is based on the work in [16, 2, 9].

Instead of just assuming that a channel can lose messages at any time without any reason, we would like to assume that these losses are due to a probabilistic behavior. We believe this is a more natural way of specifying why messages get lost from the channels. Thus, it is reasonable to adopt probabilistic behavior to LCFMs defined in the previous section.

One of the first approaches in adopting probabilistic behavior into a system with finite state automaton communicating through unreliable channels can be traced back to [15]. The probabilistic variant which will be defined here will be called Probabilistic Lossy Communicating-Finite State Machines (PLCFM).

6.2.1 Syntax of PLCFM

A PLCFM is basically a LCFM where the loss of a message is associated with a predefined probability. The probability of losing a message depends on where the message is in transit. A PLCFM extends a LCFM by defining a probability function that determines the probability of the individual channels losing a message.

Definition 6.2.1. (*Syntax of a Probabilistic Lossy Communicating Finite-State Machine*)

A Probabilistic Lossy Communicating Finite-State Machine (PLCFM) over \mathcal{P} and Σ is a structure $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F, p_{loss})$ where:

- $((S_p, \Delta_p))_{p \in \mathcal{P}}, i_{init}, F$ is a LCFM,
- $p_{loss} : C \rightarrow [0, 1]$ is a function which assigns to each channel the probability of message loss in that channel.

■

While the work in [16, 15, 3, 1] uses a solely probabilistic system as their semantic model, in this paper we would like to pursue the approach in [6] where the semantic model has both probabilistic and also nondeterministic behavior. As stated in [6], we also think that a system with both probabilistic and nondeterministic behavior is more suitable for our model. If the semantic model is defined as a Markov Chain, as [16, 15, 3, 1], the nondeterministic choices between the possible transitions in the local automaton

must be regarded as being probabilistic. And this is not the case.

We will use a Markov Decision Process (a MDP) as the semantic model of a PLCFM. This will ensure that the possible transitions without message loss (the perfect transitions) in a configuration will be governed nondeterministically and the transitions with loss messages (the loss transitions) will be determined probabilistically (see Definition 6.2.4)

Definition 6.2.2. (*Markov Decision Process*)

A Markov Decision Process is a system $\mathcal{MP} = (S, Act, P, i_{init})$ such that:

- S is a finite set of states,
- Act is a finite set of actions,
- $P : S \times Act \times S \rightarrow [0, 1]$ is a transition probability that assigns for a state and an action the probability to reach another state.

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\}$$

- $i_{init} : S \rightarrow [0, 1]$ is an initial distribution that assigns to the states a probability for being the initial state of the system.

$$\sum_{s \in S} i_{init}(s) = 1,$$

■

6.2.2 Semantics of PLCFM

Our semantic for PLCFM is based on the *local-fault* model from [6, 3, 1]. In a local-fault model [3, 1], a perfect transition (where no messages are lost) is always followed by a lossy transition (where messages can get lost). Furthermore, the probability of losing a message p_{loss} in a channel is applied to every message in the channel regardless of the number of messages in the channel. This implies that a message in a channel, i.e. (p, q) , is lost with a probability $p_{loss}(p, q)$ and kept in the channel with a probability $1 - p_{loss}(p, q)$.

Definition 6.2.3. (*Semantics of a Probabilistic Lossy Communicating Finite-State Machine*)

For a PLCFM $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F, p_{loss})$ over \mathcal{P} and Σ , the operational semantics of PLCFM is defined as a Markov Decision Process $\mathcal{MP}_{\mathcal{A}} = (S_{\mathcal{MP}}, Act_{\mathcal{MP}}, P_{\mathcal{MP}}, i_{init_{\mathcal{MP}}})$ with:

- $S_{\mathcal{MP}} = S_{\mathcal{A}} \times \{\eta\}$ is the set of all configurations of PLCFM \mathcal{A} ,
- $Act_{\mathcal{MP}} = Act \cup \{\tau\}$ is the set of actions in $\mathcal{MP}_{\mathcal{A}}$ (Act is the same set of actions used in Definition 6.1.1),
- $P_{\mathcal{MP}} : S_{\mathcal{MP}} \times Act_{\mathcal{MP}} \times S_{\mathcal{MP}} \rightarrow [0, 1]$ is the transition probability function on the configurations based on an action. For $\alpha \in Act_{\mathcal{MP}}$ and $\sigma, \sigma' \in S_{\mathcal{MP}}$ (with $\sigma = \langle \bar{s}, \eta \rangle$, $\sigma' = \langle \bar{s}', \eta \rangle$), $P_{\mathcal{MP}}(\sigma, \alpha, \sigma')$ is defined as follows:

- if $\alpha \in Act_{\mathcal{MP}} \setminus \{\tau\}$ and α makes $\sigma \xrightarrow{perfect} \sigma'$ possible, then $P_{\mathcal{MP}}(\sigma, \alpha, \sigma') = 1$
- if $\alpha = \tau$, $\bar{s} = \bar{s}'$ and $\sigma \xrightarrow{lossy} \sigma'$, then $P_{\mathcal{MP}}(\sigma, \alpha, \sigma') = P(\eta, \eta')$, with $P(\eta, \eta') =$

$$\prod_{(p,q) \in C} (p_{loss}(p, q)^{|\eta(p,q)| - |\eta'(p,q)|} \cdot (1 - p_{loss}(p, q))^{|\eta'(p,q)|} \cdot \#(\eta'(p, q), \eta(p, q)))$$

- otherwise $P(\sigma, \alpha, \sigma') = 0$

- $i_{init_{\mathcal{MP}}}(\sigma) = \begin{cases} 1 & \text{if } \sigma = \langle s_{init}, \eta_{\epsilon} \rangle \\ 0 & \text{otherwise} \end{cases}$

■

$P(\eta, \eta')$ computes the probability that by losing some messages the channel content $\eta(C)$ results in the channel content $\eta'(C)$ ($\eta(C)$ can be seen as a concatenation of contents of all channels in C). The probability of one message loss is independent of the size of the channel and also independent of other messages being lost. The first term $(p_{loss}(p, q)^{|\eta(p,q)| - |\eta'(p,q)|})$ in $P(\eta, \eta')$ computes the probability of the lost messages. $|\eta(p, q)| - |\eta'(p, q)|$ returns the number of messages being lost, which is used as the exponent for the probability that the channel loses messages. This is obvious, since: if the

probability of losing a message in a channel is 0.123 then the probability of losing four messages in the channel is 0.123^4 . Since the probability of message loss is accounted by the first term, the second term $(1 - p_{loss}(p, q))^{|η'(p, q)|}$ deals with messages that are kept in the channel. The probability that a message is kept in a channel (p, q) is $1 - p_{loss}(p, q)$ and $|η'(p, q)|$ is the number of messages left in the channel. Then it is straightforward that the probability of $|η'(p, q)|$ messages kept in the channel is $(1 - p_{loss}(p, q))^{|η'(p, q)|}$. The last term denotes the number of possibilities in deleting symbols, to obtain the channel content in which messages have been lost.

Example 6.2.1. Consider two channel contents $η(p, q) = baaba$ and $η'(p, q) = baa$ with 0.123 as the probability of losing a message in the channel. Then we have:

$$\begin{aligned} P(η, η') &= p_{loss}(p, q)^{|η(p, q)| - |η'(p, q)|} \cdot (1 - p_{loss}(p, q))^{|η'(p, q)|} \cdot \#(η'(p, q), η(p, q)) \\ &= 0.123^2 \cdot 0.877^3 \cdot 3 = 0.0306147176 \end{aligned}$$

The property of the local-fault model, where a perfect transition is always followed by a lossy transition, is defined in the runs of a PLCFM. Notice that a lossy transition does not necessarily means that messages certainly will get lost from the channels. A lossy transition also allows that no message is lost.

Definition 6.2.4. (*A run of a Probabilistic Lossy Communication Finite-State Machine*)

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, s_{init}, F)$ be a PLCFM over \mathcal{P} and Σ . Further let $\mathcal{MP}_{\mathcal{A}} = (S_{\mathcal{MP}}, Act_{\mathcal{MP}}, P_{\mathcal{MP}}, i_{init_{\mathcal{MP}}})$ be its corresponding MDP. A run of PLCFM \mathcal{A} in $\mathcal{MP}_{\mathcal{A}}$ is a sequence of transitions $\rho = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \sigma_n$ such that:

- $\sigma_0 = \langle s_{init}, \eta_\epsilon \rangle$, where η_ϵ is the function which assigns to each channel the empty string ϵ ,
- $\alpha_1, \alpha_{1+2i} \in Act \setminus \{\tau\}$ and $\alpha_{2i} = \tau$ (for $i \in \mathbb{N}$).

A run $\rho = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \sigma_n$ is an accepting run of PLCFM \mathcal{A} if $\sigma_n \in F \times \{\eta_\epsilon\}$.

■

6.3 An Approach in Realizing pMSCs

In the previous sections we have defined how a system should look (syntactically and semantically) such that they can describe pMSCs. This system, PLCFM, is an extension of CFMs with probability associated to the channels, while the transitions are probabilistic as also nondeterministic. The matter which is discussed here is how to synthesize a PLCFM from a set of pMSCs. We give an approach to this problem and leave the details for further works. The problem of synthesizing a PLCFM \mathcal{A}_{PLCFM} from a set of pMSCs \mathbb{M}_{pMSC} is simplified into synthesizing a CFM \mathcal{A}_{CFM} from a set of MSCs \mathbb{M}_{MSC} since that syntactically $\mathbb{M}_{pMSC} = \mathbb{M}_{MSC}$. Notice that a pMSC is syntactically speaking a MSC. The differences between MSCs and their probabilistic versions is described in their behaviors. We think that if there is a method which can synthesize a CFM from a set of MSCs, where the synthesized CFM describes exactly those behaviors possessed by the MSCs (not more and not less), than this method can be used to implement a PLCFM from a set of pMSCs.

The idea is first to consider the pMSCs \mathbb{M}_{pMSC} as MSCs \mathbb{M}_{MSC} and use the method to synthesize a CFM \mathcal{A}_{CFM} , which precisely describes the behaviors of the MSCs in \mathbb{M}_{MSC} . Next, to maintain the PLCFM \mathcal{A}_{PLCFM} for \mathbb{M}_{pMSC} we consider the \mathcal{A}_{CFM} as a LCFM \mathcal{A}_{LCFM} (see Section 6.1). By associating the probabilities of the channels from \mathbb{M}_{pMSC} to the channels in \mathcal{A}_{LCFM} we will get the PLCFM \mathcal{A}_{PLCFM} .

There have been several methods presented for the synthesis of message-passing systems from a set of MSCs [7]. Since the method in [7] synthesizes a CFM that precisely exhibits the behaviors of the MSCs, we think that this method can also be used to gain a PLCFM from a set of pMSCs using the idea explained above. [7] uses a learning algorithm, which is an extension of Angluin's learning algorithm [4]. The algorithm consists of a *Learner*, which has initially no knowledge about the machine to be taught, and by asking questions to a *Teacher/Oracle*, the *Learner* successively learns about the machine.

Chapter 7

Conclusion

7.1 Summary

In this Thesis we have extended MSCs into pMSCs in a way that the channels are associated with probabilities that define the probability of losing a message. The semantics of a pMSC is defined by its set of posets. Each poset in a pMSC describes one possible execution of the pMSC. The different posets are an implication of different message losses in the channels. To generate all possible posets of a pMSC we have presented an algorithm whose complexity is $\mathcal{O}(2^{|E_{\mathcal{P}}|})$ and where the number of posets generated is $2^{\mathcal{O}(\min(|E_{\mathcal{P}}|, |\mathcal{P}|))}$ in the worst case. Probability measurements can be applied to pMSCs in a simple manner, where the probability of a send event in a poset is always equal to 1 and the probability of receive events and loss events depend on the probability of the channel associated to them. The product of the probabilities of the events in a poset defines the probability of the poset itself.

Through extending MSGs to pMSGs, where each vertex is represented as a pMSC, we are able to describe the composition of pMSCs. Transitions in a pMSG exhibit the notion of probabilistic and nondeterministic behavior. Every vertex chooses nondeterministically an action from all the possible actions and performs a probabilistic transition to a successor state with respect to the previous chosen action. A transition from a vertex v to a vertex v' is always associated to the concatenation of the pMSC associated to the vertex v with the pMSC associated to the vertex v' . Concatenation in pMSCs is handled in a similar manner as with MSCs. For reasoning about qualitative

and quantitative properties of pMSGs we have used Markov Chains based on its simplicity. A transformation from a pMSG to a Markov Chain can be realized through associating a scheduler to the pMSG which resolves the nondeterminism.

To express the properties of pMSCs we have defined an extension of PDL specification language which we called PDDL. Using PDDL we are now able to express the properties of events in pMSCs with probabilities and furthermore also the probability of a sequence of events occurring in a pMSC. The syntax of PDDL is basically similar to the syntax of PDL with the addition that the forward- and backward-path expressions are equipped with a probability operator. We defined the semantics of PDDL local and global formulas over the set of posets, where on the other hand PDL local and global formulas are defined over the MSC itself. The model checking problem for PDDL over pMSCs has to be examined in a slightly different way than the case with PDL and MSCc. But we still conclude that model checking of PDDL over pMSCs can be reduced to model checking of PDL over MSCs.

Furthermore we have presented PLCFM which is a CFM where the channels are unreliable, such that messages can be lost without any notification. Moreover, the losses are due to probabilistic behavior. To realize such mathematical system we use the Markov Decision Process as its semantical model. Thus, transitions with no message loss are expressed through nondeterministic transitions and transitions with message loss are expressed through probabilistic transitions. The semantic we defined for PCLFM is based on the *local-fault* model from [6, 3, 1]. In addition, we present an idea for the realization of a PCLFM from a set of pMSCs. The idea is that to realize a PLCFM from a set of pMSCs we could revert to methods for realizing CFM from a set of MSCs. We think that this is possible only if the resulted CFM describes precisely those behaviors featured by the MSCs.

7.2 Future Work

In this Thesis, we believe there are two topics which are subject to further research. The first one is model checking of PDDL over pMSCs. Most of the works found are subject to model checking of PDL over MSCs [8]. We have presented a possible approach in the probabilistic case. It would be interest-

ing to analyze if this approach is correct and implementable. Further, this approach falls back on model checking of PDL over MSCs. It would also be interesting to work on an approach which does not rely on model checking of PDL over MSCs.

The second topic is the feasibility of PLCFMs. Also in this case we have found difficulties in presenting a method based on a direct translation of pMSCs to a PLCFM. As in the model checking case, the idea of our approach is also based upon using the methods used in realizing CFM from MSCs.

Bibliography

- [1] P. A. Abdulla, N. Bertrand, A. Rabinovich, and Ph. Schnoebelen. Verification of probabilistic systems with faulty communication. *Inf. Comput.*, 202(2):141–165, November 2005.
- [2] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91 – 101, 1996.
- [3] Parosh Aziz Abdulla and Alexander Rabinovich. Verification of probabilistic systems with faulty communication. In *Proceedings of the 6th International conference on Foundations of Software Science and Computation Structures and joint European conference on Theory and practice of software*, FOSSACS’03/ETAPS’03, pages 39–53, Berlin, Heidelberg, 2003. Springer-Verlag.
- [4] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, November 1987.
- [5] Christel Baier, Nathalie Bertrand, and Marcus Größer. Probabilistic automata over infinite words: Expressiveness, efficiency, and decidability. In Jürgen Dassow, Giovanni Pighizzini, and Bianca Truthe, editors, *DCFS*, volume 3 of *EPTCS*, pages 3–16, 2009.
- [6] Nathalie Bertrand and Philippe Schnoebelen. Model checking lossy channels systems is probably decidable. In *Proceedings of the 6th International conference on Foundations of Software Science and Computation Structures and joint European conference on Theory and practice of software*, FOSSACS’03/ETAPS’03, pages 120–135, Berlin, Heidelberg, 2003. Springer-Verlag.

- [7] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, and Martin Leucker. Learning communicating automata from mscs. *IEEE Transactions on Software Engineering*, 36:390–408, 2010.
- [8] Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), September 2010.
- [9] Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distrib. Comput.*, 7(3):129–135, March 1994.
- [10] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
- [11] Jens Grabowski and Ekkart Rudolph. Putting Extended Sequence Charts to Practice. In *In: SDL'89 - The Language at Work (Editors: O. Faergemand, R. Reed), North-Holland, 1989*, January 1989.
- [12] Jens Grabowski and Ekkart Rudolph. Message Sequence Chart (MSC) - A Survey of the new CCITT Language for the Description of Traces within Communication Systems. *CCITT SDL Newsletter, No 16*, May 1993.
- [13] ITU-TS. ITU-TS recommendation Z.120anb: Formal semantics of message sequence charts. Technical report, ITU-TS, Geneva, 1998.
- [14] ITU-TS. ITU-TS recommendation Z.120: Message sequence chart 2011 (MSC11). Technical report, ITU-TS, Geneva, 2011.
- [15] Purush Iyer and Murali Narasimha. Probabilistic lossy channel systems. Technical report, Raleigh, NC, USA, 1996.
- [16] Ph. Schnoebelen. The verification of probabilistic lossy channel systems. In *In Validation of Stochastic Systems A Guide to Current Research, LNCS 2925*, pages 445–465. Springer, 2004.