

UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali



Corso di Laurea Specialistica in Informatica
within European Master in Informatics

Final Thesis

Probabilistic Message Sequence Charts

Relatori / Readers

Prof. Dr. Ir. Joost-Pieter Katoen
RWTH Aachen University
Prof. Paola Quaglia
University of Trento

Laureanda / Graduant

Chitra Hapsari Ayuningtyas

Anno accademico 2008/2009

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, 9 September 2009

Chitra Hapsari Ayuningtyas

Abstract

Message Sequence Charts (MSCs) are heavily used in various phases of system development, ranging from requirements specification to test-case scenarios of system implementations. An MSC specifies example runs of the system, by illustrating the communications between several processes using message exchanges. Formally speaking, an MSC specifies a partial order of events, and thus a set of possible runs. One of the deficiencies of MSCs is that we cannot use it to specify quantitative system behaviors such as, e.g., to specify that a message can be lost with probability 0.001. Similarly, in Message Sequence Graphs (MSGs), which are digraphs where vertices are labeled with MSCs, alternative composition can be expressed but cannot be quantified. This thesis deals with probabilistic extensions of MSCs and MSGs, aiming to overcome the limitations described above.

Probabilistic MSCs (pMSCs) are defined using probabilistic lossy channel systems where a probability of message loss is specified for every channel such that a message sent using this channel can be lost with the specified probability. We provide semantics of the model in terms of partial ordered sets (posets). We also show how to work with the semantics to compute probabilities of possible scenarios defined by the posets.

We defined probabilistic MSGs (pMSGs) as a means to combine a set of pMSCs by enabling both probabilistic and nondeterministic system behavior to be specified using the model. We first provide a concatenation operator to combine a set of pMSCs then discuss about the syntax and semantics of pMSG. We present Asynchronous Leader Election Protocol as a case study and show how this protocol can be specified using our models.

We also provide a logic to reason about properties of pMSCs. Our logic is an extension of Propositional Dynamic Logic (PDL) for message-passing systems. We extend PDL by adding probabilistic operators and modify its semantics in order to make it suitable with the partial order semantics of pMSCs.

Acknowledgements

This thesis concludes my study in Embedded System Informatics within the European Master of Informatics program at the University of Trento and RWTH Aachen, as a part of Erasmus Mundus project funded by the European Commission. I started this work on April 2009 at the Chair of Software Modeling and Verification in the Department of Computer Science of RWTH Aachen.

I would like to express my sincere appreciation and gratitude to my direct supervisor, Joost-Pieter Katoen, for his guidance, patience, and care through all this work for the last six months. I would also like to thank my other supervisor, Paola Quaglia at the University of Trento, for her cooperation and understanding. Further thanks also to the colleagues at the Chair for Software Modeling and Verification of RWTH Aachen for their support and feedback.

I would also like to thank my family and close friends. My parents, Agus Hadisoeryo and Amrit Saptari Widiatmi, for their support, love, and understanding. My best friends, Arya, Ibai, Tresna, and Malou, for the night chats during hard days, and for always being there though we are miles apart. Afshan, Divya, Ola, and Kiran for the laughters and the great time together. And for all other friends and relatives who give me reasons to live my days with joys.

Contents

Abstract	iv
Acknowledgements	iv
1 Introduction	1
1.1 Message Sequence Charts	1
1.1.1 History and variants	1
1.1.2 Uses and implementations of Message Sequence Charts	2
1.1.3 Properties of MSCs and model checking problems	3
1.2 Channel systems	4
1.3 Probability and nondeterminism	4
1.4 Thesis contribution	5
1.5 Organization of this thesis	5
2 Probabilistic Extension of Message Sequence Charts	6
2.1 Introduction and motivation	6
2.2 Basic Message Sequence Charts	7
2.2.1 Syntax	8
2.2.2 Drawing conventions	9
2.2.3 Semantics	10
2.2.4 Coregions	12
2.3 Probabilistic Message Sequence Charts	12
2.3.1 Syntax	13
2.3.2 Semantics	16
2.3.3 Linearization	22
2.3.4 Probabilistic semantics	23
2.3.4.1 Probability of events	23
2.3.4.2 Probability of posets	24
2.3.4.3 Probability of linearizations	26

3	Probabilistic Extension of Message Sequence Graphs	28
3.1	Introduction and motivation	28
3.2	Probabilistic models	28
3.2.1	Markov chains	29
3.2.2	Markov decision processes	31
3.3	Concatenation operator	32
3.3.1	Concatenation of posets	33
3.3.2	Concatenation of pMSCs	37
3.4	Probabilistic Message Sequence Graphs	40
3.4.1	Syntax	40
3.4.2	Language of a pMSG	43
3.4.3	Adversary	45
3.5	Case Study : Asynchronous Leader Election Protocol	50
4	Properties of Message Sequence Charts	54
4.1	Introduction	54
4.2	Propositional Dynamic Logic for MSCs	55
4.3	Probabilistic Propositional Dynamic Logic for pMSCs	58
5	Conclusions	64
5.1	Summary	64
5.2	Related work	65
5.3	Future work	66
	Bibliography	69
A	Domain Theory	73
A.1	Partially Ordered Set	73
A.2	Probability Theory	74

List of Figures

1.1	MSC of PIN validation on ATM	2
2.1	Example of a BMSC	10
2.2	FIFO property	11
2.3	Environment representation	12
2.4	BMSC examples with and without coregion	13
2.5	A pMSC and its posets	16
2.6	A pMSC with incomparable events	17
2.7	A pMSC with complete and incomplete posets	18
2.8	Example of a pMSC	19
2.9	Inconsistent events elimination	20
2.10	The final set of posets	21
2.11	Subsumption of posets	21
2.12	Incomplete poset and the posets it subsumes	25
3.1	Markov chain for the weather problem	30
3.2	Life cycle of a machine as MDP	33
3.3	Different interpretations of concatenation	34
3.4	pMSCs to be concatenated	35
3.5	The resulting poset from concatenation	36
3.6	Posets to be concatenated	37
3.7	Concatenation of posets is not associative	38
3.8	Concatenation of M_1 and M_2	39
3.9	pMSG example	41
3.10	The underlying MDP of pMSG in Figure 3.9	42
3.11	Example of pMSCs included in $\mathcal{L}(G)$	45
3.12	Markov chain \mathcal{MC}_{A_2}	47

3.13	Markov chain \mathcal{MC}_{A_1}	48
3.14	pMSC of π	49
3.15	pMSG of Asynchronous Leader Election Protocol	51
3.16	The next part from vertex v_2 of Figure 3.15	52
3.17	pMSC of a vertex and its labeled poset	53
4.1	Simple client-server program	57
4.2	Simple client-server program as pMSC	61
5.1	Client-server program as pMSG	66
5.2	Posets of $M(\pi)$	67
5.3	Modification on v_2 and the resulting posets	68

List of Tables

2.1	Linearization of pMSC with incomparable events	23
-----	--	----

Chapter 1

Introduction

1.1 Message Sequence Charts

Message Sequence Chart (MSC) is a graphical specification language standardized by the International Telecommunication Union (ITU) [24] for describing communication behavior of a number of distributed entities by means of message exchanges. MSCs have gained wide acceptance to be used as a mechanism to specify scenario-based specifications of concurrent systems. MSCs are used in various phases of system development, ranging from requirements specification to test-case scenarios of system implementations. They are employed to formalize the design by describing patterns of interactions between processes or objects within the system in the form of ordered message exchanges. They also provide visual notations which give an intuitive understanding of the intended system behavior.

Figure 1.1 depicts a simplified example of the verification procedure of an automatic teller machine (ATM). The client first needs to enter his PIN to the machine which then asks the bank to verify it. In this case, the bank sends an acknowledgement to the machine that the PIN is valid, then the transaction options will be displayed by the machine to the client. Each entity in the system, in this example: the client, the ATM and the bank, is represented by a vertical line which is started with a blank rectangle and ended in a filled rectangle. Horizontal arrows represent message exchanges between entities, with the message content written above the arrows. The arrows are ordered from top to bottom, indicating the order of the execution.

In its simplest form, an MSC depicts message exchanges between processes within the system and describes one single execution of the system. More features have later been developed to enable system designers to specify a collection of scenarios. *Message Sequence Graphs* (MSGs), also called *high-level MSCs* (HMSCs) in ITU-T standard [28], combine MSCs using choice, repetition and concatenation operators.

1.1.1 History and variants

MSCs have long been used by ITU Study Groups in their recommendations, initially used to support SDL methodology for describing possible scenarios of systems. The

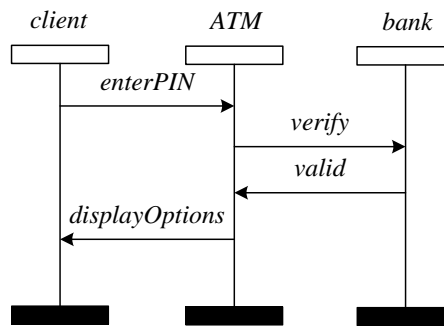


Figure 1.1: MSC of PIN validation on ATM

ITU-T SDL community suggested the standardization of MSCs which eventually led to the first ITU-T MSC recommendation Z.120 [24]. Later there have been revisions of this standard in 1996 [25], 1999 [27], and 2004 [28]. ITU-T has also provided formal semantics to accompany the syntax with an unambiguous meaning [26].

There have been many MSC dialects developed over the past few years, triggered by the increasing significance of interaction based specification. Compositional MSCs (CMSCs) [19] are extensions of MSCs which allow unmatched send-events because the matching receive events can be defined after an arbitrary delay. MSCs have also been incorporated into the UML notational framework [12] where they are called *sequence diagrams*. Sequence diagrams are derived from use cases and combined with object oriented concept. Compared to MSCs, sequence diagrams do not have the same advanced and well defined syntax and semantics. They also focus more on flow control based on synchronous communication mechanisms instead of asynchronous ones applied by MSCs. *Live Sequence Charts* (LSCs) [14] are extensions of MSCs where charts, messages, locations and conditions are specified as either universal or existential. It provides the possibility of specifying the behavior of the system as either mandatory or possible, which cannot be done in MSCs. STAIRS [42] is another variant based on UML 2.0 [18]. It describes system specification as object interactions which characterized three kinds of traces: desirable, undesirable, and irrelevant to the system. In MSC semantics, desirable and undesirable behaviors are implemented using *guards*.

We will use the syntax of basic MSC from ITU standard in this thesis, simplified to cover only send, receive and local actions. We remove the notions of condition and timer to avoid complication in the definition.

1.1.2 Uses and implementations of Message Sequence Charts

MSCs in software engineering. Since the concept of MSC has been adapted in UML as sequence diagrams, it is well-known to software engineers. In the object-oriented system development methodologies, the main role of MSCs is to capture system requirements in the form of scenarios that the system should exhibit, but MSCs have also been implemented in other phases of software engineering process. In the requirement engineering phase, MSCs are derived from use cases by explicitly

visualizing causal relationship between entities to describe system behaviors [35]. To bring gaps between the requirement and design phases, MSCs' behavioral scenarios are used to generate SDL (Specification and Description Language) design specification [31] and statecharts [32]. MSCs are also used to specify test purposes [17] and to generate test cases [7] in the testing phase of the system.

MSC Tools. A few tools have been developed to support system development based on MSCs. Bell Labs developed some tools to construct and edit MSCs in graphical form, detect logical inconsistencies, and do fragments searching on a set of MSCs [22]. **MESA** [8] is an MSC based tool which supports editing, syntactic analysis, model-based analysis and code synthesis for MSCs and HMSCs. **MSCAN** [10] is another tool which offers features for editing, displaying, debugging and model checking HMSCs. As **MESA**, it does specification checking on non-local choice property, and offers some more properties to be checked such as local and global cooperativity, regularity and other related requirements.

1.1.3 Properties of MSCs and model checking problems

One of the important tasks in system modeling is to check whether the model satisfies certain properties. There are many different approaches proposed to specify properties of MSCs and to do model checking for deciding whether an MSC satisfies the specified properties.

The first approach is done by checking properties on the linearizations of MSCs, which are the result of completing the partial orders on events implied by the MSCs into total orders. The language of an MSC is defined as the set of all possible linearizations of the MSC. The properties are specified as an automaton and the model checking is done by building an automaton which accepts all the undesirable executions and another automaton which accepts the language of the MSC, then check whether the intersection of the two automata is empty. Given a set of MSCs as an MSG, it was shown that model checking problem on the linearizations of the MSG is undecidable. This problem is tackled by limiting the class of the MSG by imposing a bound on the number of messages in the communication channels. [5]

Another approach is done by specifying the properties of HMSCs as HMSCs. A technique well known by checking emptiness of intersecting a system automaton and a property automaton is adopted by intersecting two HMSCs. Unfortunately, checking emptiness of HMSCs intersection is also undecidable [36].

There also exist some attempts to define logics to specify MSC properties directly on the partial order implied by the MSC rather than of their linearizations. Hence these logics do not distinguish between different linearizations of the partial order execution. It was shown that monadic second order (MSO) logic can be used to specify properties of a set of MSCs represented by an MSG [33] and that the model checking problem is decidable. A temporal logic TLC^- [37], which is a fragment of the logic TLC [4], has also been used to assert properties of HSMCs and interpreted over the partial order structure. Another temporal logic was developed for Lamport diagrams [34] which can be seen as the partial orders generated by the MSCs,

and therefore is similar to the other logics defined over the structure of MSCs. A propositional dynamic logic (PDL) developed for MSCs [11] combines elements from TLC⁻ and [34] and is inspired from dynamic LTL [21]. This logic is basically the original PDL [16] which was developed for computer programs, applied to the MSC framework. Beside the usual operators for future event properties, this version of PDL also extends the logic by adding past operators to express properties of events that have already been executed.

In this work, we will extend PDL logic with probabilistic notions to specify properties of probabilistic MSCs.

1.2 Channel systems

Channel system is a common model for asynchronous communication protocols where some components are connected to each other by communication channels [9]. In most of the definitions, the components are finite state automata which evolve asynchronously and the channels behave as unbounded FIFO buffers. Component A may move from one state to another by sending a message m to component B via channel c_{AB} where the message will be enqueued. On the other side, component B may move from one state to another by consuming the message m from channel c_{AB} , which is only possible if the channel is not empty and the first message in the queue is m . Message m is then removed from the queue.

In a *lossy channel system* [2], messages can be lost nondeterministically at any time after being sent. *Probabilistic lossy channel systems* [29] allows message losses to happen with a certain probability, making the systems behave probabilistically. These models are intended to model fault-tolerant protocols where the communication channels are not reliable.

We are inspired by probabilistic lossy channel systems and incorporate this concept in our model such that any message sent from one process to another in an MSC can be lost with a certain probability.

1.3 Probability and nondeterminism

Probability expresses the measure of certainty or likelihood of potential events to occur. There are various applications of probabilistic notion in computer science. Probability can be used to describe the likelihood of a certain phenomena (noise, malfunction, intrusion) which affect the performance of the system, to specify the likelihood of some specific system behaviors to be chosen, or, as mentioned in the previous chapter, to describe unreliability of the system. A model is *probabilistic* when different alternatives are represented and a probability is assigned for each alternative.

A *nondeterministic* model also represents different alternatives, but the way one alternative is chosen over the others is not specified. Nondeterminism is mostly

needed to abstract away from details. For example, it is possible to write a high level system specifications where certain aspects of the behavior are left open for the implementation phase. Another use of nondeterminism is in concurrent systems where the order in which independent components perform their computations is not specified. Nondeterminism is also useful to resolve competition for communication, as used in some well-known randomized protocols. In security domain, nondeterminism is used for example, in a password generator which should select password from a pool and make the selection nondeterministically from the perspective of the users and the attackers.

Our model described in this thesis will allow both probabilistic and nondeterministic behavior to be represented.

1.4 Thesis contribution

One of the deficiencies of MSCs is that quantitative system behaviors cannot be specified. We develop a probabilistic extension of MSC, defined over probabilistic lossy channel system. We define its syntax and semantics in terms of partial ordered set (poset).

Our second contribution is a model to combine a set of probabilistic MSCs, probabilistic Message Sequence Graphs (pMSGs). We define the concatenation operator for pMSCs, then define the syntax and semantics of pMSGs. We discuss about the language accepted by the model and provide a case study.

Our last contribution is a logic to specify properties of pMSCs, Probabilistic Propositional Dynamic Logic (PPDL), which is interpreted directly over the structure of pMSC. We discuss about the syntax and semantics of the logic and show how to work with the semantics with some examples.

1.5 Organization of this thesis

This thesis is further organized as follows. Chapter 2 describes the syntax and semantics of probabilistic Message Sequence Chart (pMSC), the notion of linearization and how to compute its probability. Chapter 3 describes the syntax and semantics of probabilistic Message Sequence Graph (pMSG) as a model to define combination of pMSCs. It also describes pMSG semantics as Markov Decision Processes (MDP). Chapter 4 describes the probabilistic variant of Propositional Dynamic Logic (PPDL) as a way to specify properties of pMSCs. Chapter 5 summarizes the main results of this work, discusses related work, and provides some thoughts about future work.

Chapter 2

Probabilistic Extension of Message Sequence Charts

2.1 Introduction and motivation

Message Sequence Chart (MSC) is a well-known formalism to provide a scenario-based language for the specification and description of the communication behavior of system components by means of message exchanges.

The latest ITU-T standard [28] defines some main characteristics of the MSC language as follows:

- MSC is a scenario language because it describes the order of events and communications,
- MSC can describe incomplete behaviors as well as complete ones,
- MSC is widely applicable and not designed for one single application domain,
- MSC has a formal expressive power because of its formal notation and an intuitive appearance because of its graphical diagrams,
- MSC enables structured design by supporting the combination of simple scenarios to form more complete ones,
- MSC can be interpreted as the behavior the system should exhibit as well as disallowed scenarios.

Although it has now been generalized to be employed in many application domains, when MSC was first standardized by ITU-T [24], it was originally intended to specify communication behavior for real-time systems, in particular telecommunication switching systems as this domain was the main focus of the ITU. In telecommunication systems, participating nodes are communicating with each other by transmitting signals using communication channels. When modeled using MSCs, it is normally assumed that these channels are free of errors such that signals transmitted from

one node to another will always be perfectly received. In reality however, this is not the case. Communication channels are often unreliable; messages can be lost due to hardware or software failures, reordered because of delay, modified or duplicated.

Communication protocols have formally been modeled using channel systems, which are finite state machines that communicate over unbounded channels [9]. Channel failures have also been incorporated into this model in various manners. A model of errors, called *completely specified protocols* [15], defines that messages in front of a queue can be lost. This notion is generalized in another model by allowing messages from anywhere in the queue to be lost [2]. Other types of errors such as insertion and deletion have also been considered to be modeled [13], but in this thesis we are particularly interested in message losses.

It is believed that providing a quantitative measure on faults is a more realistic model to describe unreliable channels. Some approaches on probabilistic lossy channel system have been developed to model communication systems' behaviors. Schnoebelen [40] points out that there are two main approaches to define probabilistic lossy channel systems. Both approaches define a fixed probability of message loss for any channel, p_{loss} , but interpret it in different ways. The *global-fault model* [40] assumes that p_{loss} is distributed over all messages currently in the buffer. Thus, when there are more messages in the channel, the probability of each message to get lost is smaller. The *local-fault model* [1, 41] on the other hand, applies p_{loss} to any single message independently from the other messages such that any message can be lost with probability p_{loss} .

We adopt the interpretation of probability of message loss from local-fault model to be included in our model. We assume that every process included in the MSC is connected with each other by unreliable communication channels with probabilistic behavior. We define the probability of message loss for every channel such that every single message sent using this channel can be lost with the specified probability.

In this chapter, we first introduce the Basic Message Sequence Chart (BMSC) notation. We then discuss about the probabilistic extension of BMSC which will cover the syntax, semantics, and probability measure over linearizations.

2.2 Basic Message Sequence Charts

According to the ITU-T formal semantics of MSC [26], the core language of MSC is called *Basic Message Sequence Chart* (BMSC). It concentrates only on communications and local actions. The ITU-T standard defines MSC in terms of graphical and textual representations. In this section we will introduce the formal semantics of BMSC which captures the essence of the model from the ITU-T semantics. Our definition of BMSC emphasizes that its structure is based on *partial order* of the events. Partial order model allows events to be unordered if they can concurrently occur, which can happen in concurrent systems specified using the MSC. Some basic theory about partial order is included in Appendix A.1.

2.2.1 Syntax

Definition 2.2.1 Basic Message Sequence Chart

A Basic Message Sequence Chart M is a structure $(\mathcal{P}, \Sigma, Act, E, l, m, C, <)$ where:

- $\mathcal{P} = \{p, q, r, \dots\}$ is a finite set of processes,
- $\Sigma = \{a, b, c, \dots\}$ is a message alphabet,
- Act is a finite set of actions. The processes in \mathcal{P} are able to communicate with each other by sending and receiving messages taken from Σ . There are three allowed actions as the followings, with $p, q \in \mathcal{P}$ and $a \in \Sigma$:
 - *send* action, $p!q(a)$ stands for process p sending message a to q ,
 - *receive* action, $q?p(a)$ stands for process q receiving message a from p ,
 - *internal* action, $p(a)$ stands for process p performing internal action involving message a ,
- E is a finite set of events. E can be partitioned based on the process where the events are located, or based on the types of the events.

$$E = \bigsqcup_{p \in \mathcal{P}} E_p = E_! \bigsqcup E_? \bigsqcup E_{int}$$

E_p denotes the set of events which are located in process p . There are three types of events allowed in BMSM: send events $E_!$, receive events $E_?$, and internal events E_{int} . $E_{p!}$ denotes the set of send events which are located in process p . Similar notations can be defined for the set of receive and internal events,

- $l: E \mapsto Act$ is a labeling function given by:

$$l(e) = \begin{cases} l(e) = p!q(a) & \text{if } e \in E_{p!}, p, q \in \mathcal{P}, p \neq q, a \in \Sigma, \\ l(e) = p?q(a) & \text{if } e \in E_{p?}, p, q \in \mathcal{P}, p \neq q, a \in \Sigma, \\ l(e) = p(a) & \text{if } e \in E_{p_{int}}, p \in \mathcal{P}, a \in \Sigma, \end{cases}$$

- $m: E_! \mapsto E_?$ is a matching function between send events and their corresponding receive events satisfying:

$$(m(e) = e' \wedge l(e) = p!q(a)) \Rightarrow l(e') = q?p(a), p, q \in \mathcal{P}, p \neq q, a \in \Sigma,$$

- C is a finite set of channels used by processes in \mathcal{P} for communication. For each pair of processes $p, q \in \mathcal{P}$, two channels are attached, c_{pq} for sending messages from p to q and c_{qp} for the other communication direction,
- $< \subseteq E \times E$ is a partial order over E defined by:

$$< = \bigcup_{p \in \mathcal{P}} <_p \cup \{(e, m(e)) \mid e \in E_!\}$$

The first part of the definition denotes the union of $<_p$, which is the total order (or 'top-to-bottom' order) of events in process p . The second part denotes the communication order $<_c$, which tells that a send event must happen before its corresponding receive event,

□

The partial order $<$ describes the visual ordering of events and is obtained from the representation of the graphical chart. For two events $e, e' \in E$, $e < e'$ holds if and only if one of the following holds:

- e immediately precedes e' in a process $p \in \mathcal{P}$ such that e and e' belong to process p and there is no event between e and e' in p ,
- $e <_c e'$, i.e., e is a send event and e' is the matching receive event for e ,
- there exists an event $e'' \in E$ such that $e < e''$ and $e'' < e'$ (transitivity property).

Definition 2.2.2 *Consistent and complete poset*

The set of events E and the partial order $<$ form a partially ordered set (poset) $(E, <)$.

A poset is *consistent* if every receive event is preceded by its corresponding send event. Formally, $\forall e \in E_{\gamma}. \exists e' \in E_{\iota}. m(e') = e \wedge e' < e$.

A poset is called *complete* if every send event is succeeded by its corresponding receive event. Formally, $\forall e \in E_{\iota}. \exists e' \in E_{\gamma}. m^{-1}(e') = e \wedge e < e'$.

□

2.2.2 Drawing conventions

We adopt the following drawing conventions. A process is drawn as a vertical line which is started with an empty rectangle as the *head symbol* and ended with a filled rectangle as the *end symbol*. The symbols do not represent creation and termination of the process, but the start and the end of the description. The name of the process is written above the head symbol. A message sent from one process to another is drawn as a directed arrow, starting from the sending process and ending at the receiving process. The message content is written on top of the arrow. Internal action is drawn as rectangle with the message written inside. It is attached to a process where the action is located. When necessary, the names of the events can be written at their locations in the process line.

Partial orders are drawn as directed graphs. Two events are connected with a directed arrow, such that $e \longrightarrow e'$ denotes $e < e'$.

Example 2.2.1 *Basic MSC*

Figure 2.1 illustrates an example of a BMSC M with three processes, $\mathcal{P} = \{p, q, r\}$. We have $\Sigma = \{a, b, c\}$ and $E = \{e_1, e_2, e_3, e_4, e_5\}$. The events are labeled as the followings: $l(e_1) = p!q(a)$, $l(e_2) = q?p(a)$, $l(e_3) = q(b)$, $l(e_4) = q?r(c)$, and $l(e_5) = r!q(c)$. e_2 is the matching receive event for e_1 as e_4 is for e_5 , i.e., $m(e_1) = e_2$ and $m(e_5) = e_4$. We can also write the matching pairs as $m^{-1}(e_2) = e_1$ and $m^{-1}(e_4) = e_5$.

The right part of the figure depicts the partial order of the BMSC. We have for example, $e_1 < e_2$, which means that e_2 can only occur after e_1 occurs. e_4 has directed arrows from e_3 and e_5 . This means that e_4 can occur after both e_3 and e_5 occur. We can see however, that e_1 and e_5 are not connected by the partial order, therefore it is not specified which event may occur first. We say that e_3 and e_5 are incomparable under the partial order.

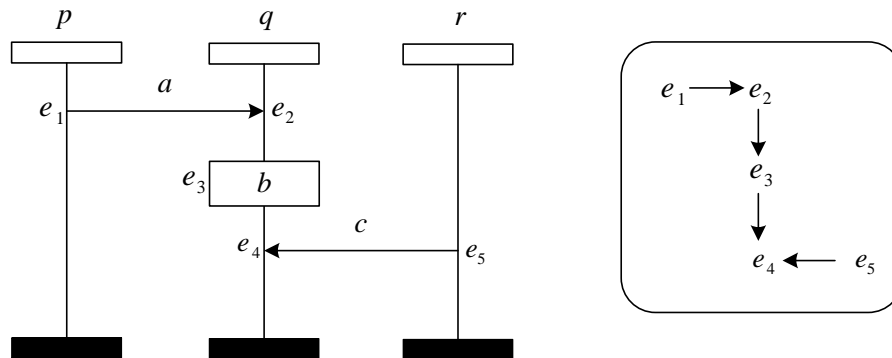


Figure 2.1: Example of a BMSC

□

2.2.3 Semantics

A BMSC describes the events to be executed by the system, along with the information on the order in which they can be executed. A BMSC consists of a number of processes on which events are specified. Each process executes the events in the order given on the vertical line from top to bottom. Therefore, the events specified on a process are totally ordered in time. However, the elapse of time between two consecutive events is not specified.

The processes basically run independently from each other. The only dependency of the processes come from the rule that a message should be sent before it is received. In Figure 2.1 it is possible to first send and receive message a , execute local action b , then send and finally receive c . Because of the asynchronous communication, event e_5 can be executed before or after e_1, e_2 and e_3 but restrictedly before its matching receive event e_4 .

We also define some assumptions on the semantics of BMSC, some are different from the original semantics in ITU-T standard [26].

Instantaneous execution. It is assumed that the execution of an event consumes no time.

Linear execution. No two events can be executed at the same time.

FIFO and non-degeneracy property. We assume FIFO (first in first out) behavior such that message arrows on the same channel do not cross each other. Formally, for all $e, e' \in E$, $a, b \in \Sigma$ and $p, q \in \mathcal{P}$,

$$e < e' \wedge l(e) = p!q(a) \wedge l(e') = p!q(b) \text{ implies } m(e) < m(e')$$

Figure 2.2(a) gives an example of a situation where FIFO property holds. Message a is sent before b and is also received before b . Figure 2.2(b) however, is an example of non-FIFO property. Message a is sent before b but is received after.

Figure 2.2(c) is an illustration of degeneracy condition. A BMSC degenerates if it reverses the order in which two identical messages sent by any process $p \in \mathcal{P}$ are received by another process $q \in \mathcal{P}$. We also disallow this behavior and require that BMSC satisfies the non-degeneracy property.

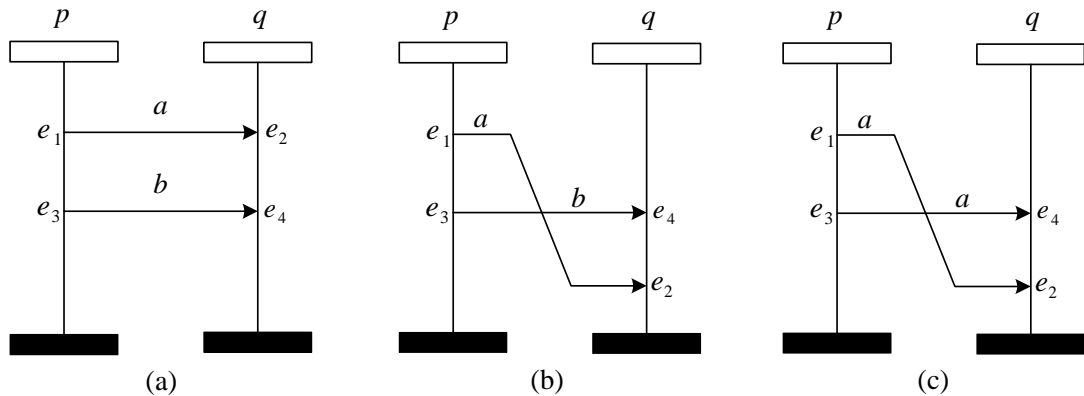


Figure 2.2: FIFO property

Closed-system assumption. The original semantics of BMSC allows processes to send and receive messages from the environment, which are represented by message arrows originated from the exterior part of the BMSC. The messages sent by the environment are not ordered in time. In this thesis however, we require that the environment should be represented as another process in the system. This requirement is illustrated in Figure 2.3, with the left part representing the environment as external entity and the right part representing the environment as a part of the system, drawn as a process.

Completeness and consistency. We require the poset $(E, <)$ of BMSC to be complete and consistent as defined before.

The ITU-T standard extends BMSC with other basic concepts such as process creation and termination, timer handling, incomplete messages and conditions. We exclude those additional concepts from BMSC specification in our work.

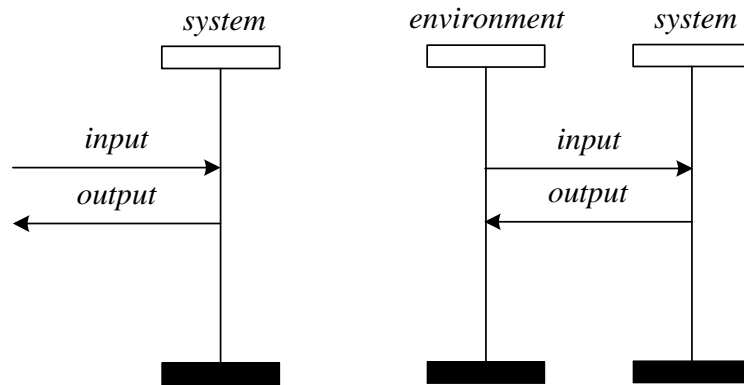


Figure 2.3: Environment representation

2.2.4 Coregions

Coregions are introduced in the ITU-T standard to enable the specification of unordered events on a process. Coregion is a part of process line where the events specified within that part are unordered in time. Thus, the events belong in a coregion are incomparable under the partial order. Coregions are graphically indicated by vertical dashed lines instead of solid lines.

Example 2.2.2 Coregions

Figure 2.4 below depicts examples of BMSCs with their corresponding partial orders. The left part of Figure 2.4 presents an example of BMSC without coregions. Here event e_3 (q sending message b to r) must happen after e_2 (q receiving message a from p). In the right part of Figure 2.4 on the other hand, e_2 and e_3 are located in a coregion. Thus, q can send message b to r without waiting for message a to arrive from p .

□

2.3 Probabilistic Message Sequence Charts

We extend the basic MSC by introducing the possibility for the messages to get lost with a certain probability when being sent from one process to another. In order to do that, we attach a probability of message loss to each channel in the MSC structure and introduce message loss events to the definition of MSC. Our model is called the *probabilistic Message Sequence Charts* (pMSC).

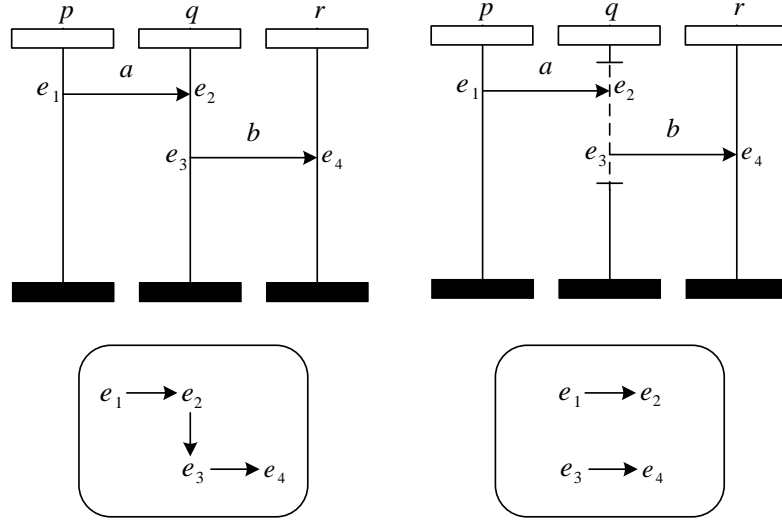


Figure 2.4: BMSC examples with and without coregion

2.3.1 Syntax

Definition 2.3.1 Probabilistic Message Sequence Charts

A probabilistic MSC (pMSC) is a structure $M = (\mathcal{P}, \Sigma, Act, E, l, m_\gamma, m_\zeta, Posets, C, p_{loss})$ such that \mathcal{P}, Σ and C are as before for BMSC and:

- Act is a finite set of actions. As for BMSC, send, receive and internal actions are allowed. Additionally, we allow actions $p \not\zeta q(a)$ denotes that the message $a \in \Sigma$ sent by process p to q is lost,
- E is a set of events with $E = \uplus_{p \in \mathcal{P}} E_p = E_l \uplus E_r \uplus E_{int} \uplus E_\zeta$, where E_ζ is the set of loss events which is ranged over \bar{e}, \bar{e}' etc.,
- l is the labeling function as for BMSC, added with labeling for loss events, $l(\bar{e}) = p \not\zeta q(a)$ if $\bar{e} \in E_\zeta, p \neq q, a \in \Sigma$,
- $m_\gamma : E_l \mapsto E_r$ is a matching function between send events and their corresponding receive events as the function m in BMSC,
- $m_\zeta : E_l \mapsto E_\zeta$ is a matching function between send events and their corresponding loss events satisfying

$$(m_\zeta(e) = e' \wedge l(e) = p!q(a)) \Rightarrow l(e') = p \not\zeta q(a), p \neq q, a \in \Sigma,$$

- $Posets$ is a set of posets,
- $p_{loss} : C \mapsto [0, 1]$ is the probability of message loss defined for the communication channels.

□

The above definition extends our former definition of BMSC with loss actions, loss events, the matching function between send and loss events and the probability of message losses. We use a probabilistic lossy channel system as the communication channels between the processes instead of a reliable one.

Drawing conventions. We keep the drawing conventions from the Basic MSC. We do not draw loss events explicitly so that the graphical representation of a pMSC only describes the scenario where all messages are received.

Loss events. We introduce a notion for a message loss event, \bar{e} , such that for each send event $e \in E_l$ with the labeling $l(e) = p!q(a), p \neq q, a \in \Sigma$, we have \bar{e} as the corresponding loss event when message a gets lost after being sent from p and intended to be delivered at q .

Events location. We define the location of the events with a function $loc : E \mapsto \mathcal{P}$ such that for every $e \in E, p, q \in \mathcal{P}$, and $a \in \Sigma$ we have the following rules:

- if $e \in E_l$ and $l(e) = p!q(a)$ then $loc(e) = p$,
- if $e \in E_r$ and $l(e) = q?p(a)$ then $loc(e) = q$,
- if $e \in E_{int}$ and $l(e) = p(a)$ then $loc(e) = p$,
- if $e \in E_{\bar{q}}$ and $l(e) = p_{\bar{q}}q(a)$ then $loc(e) = q$.

The set of posets. We first need to define the top-bottom order for every process \langle_p . In BMSC, \langle_p simply depicts the order of the events as visualized in the graphical representation of the BMSC. We now denote the union of these visual orders for all processes as \langle_{vis} . Since we now have loss events, although they do not appear in the graphical representation of the pMSC, we need to specify how they are comparable to other events at the process where they are located at. Because a loss event replaces the occurrence of the intended receive event and the locations of both events are the same (i.e., the process where the message is intended to be delivered at), we redefine \langle_p such that whenever a receive event e is comparable to another event e' under \langle_{vis} , the corresponding loss event is also comparable to e' under \langle_p . Formally, $\forall e, e' \in E$ and $\forall p \in \mathcal{P}$, $e \langle_p e'$ holds if one of the following holds:

- $e \langle_{vis} e'$, e precedes e' in the graphical representation,
- $e \in E_{\bar{q}} \wedge \exists e'' \in E_r. m_{\bar{q}}^{-1}(e'') = m_{\bar{q}}^{-1}(e) \wedge e'' \langle_{vis} e'$. Intuitively, if e is a loss event which happens instead of the corresponding receive event e'' and e'' is supposed to occur before e' in process p , then e should also occur before e' ,
- $e' \in E_{\bar{q}} \wedge \exists e'' \in E_r. m_{\bar{q}}^{-1}(e'') = m_{\bar{q}}^{-1}(e) \wedge e \langle_{vis} e''$. Intuitively, if e is an event which is supposed to occur before a receive event e'' but the loss event e' occurs instead of e'' , then e should occur before e' .

We also redefine the communication order $<_c$ to include the relation between send events and their corresponding loss events as follows:

$$<_c = \{(e, m_?(e)) \mid e \in E_1\} \cup \{(e, m_?(e)) \mid e \in E_1\}$$

A poset $(E', <')$ belongs to $Posets$ if it satisfies the following conditions:

- $E' \subseteq E$ and for all $(E'', <'') \in Posets$, if $E'' = E'$ then $<'' = <'$,
- any pair of events in E' which are comparable under $<_p$ for any process $p \in \mathcal{P}$ also comparable under $<'$, i.e., $<' = <_p \cap (E' \times E')$.

Definition 2.3.2 *Complete poset*

We redefine the notion of completeness such that a poset $(E', <') \in Posets$ is *complete* if it contains all send events and every send event has either a matching receive or loss event. Formally, $(E', <')$ is complete if it satisfies the following conditions:

- $E_1 \subseteq E'$,
- $\forall e \in E_1. \exists e' \in E'. e' = m_?(e) \oplus e' = m_?(e)$.

□

Definition 2.3.3 *Consistent poset*

We also redefine the notion of consistency such that a poset $(E', <')$ is *consistent* if it satisfies all conditions below:

- every receive event is preceded by its corresponding send event, i.e., $\forall e \in E'_?. \exists e' \in E'_!. m_?(e') = e \wedge e' <' e$,
- every loss event is preceded by its corresponding send event, i.e., $\forall e \in E'_\$. \exists e' \in E'_!. m_?(e') = e \wedge e' <' e$,
- if the message sent by a send event is received, then it is not lost, vice versa, i.e., $\forall e \in E'_!. (m_?(e) \in E' \Rightarrow m_?(e) \notin E'_\$) \wedge (m_?(e) \in E'_\$ \Rightarrow m_?(e) \notin E')$,
- loss events are *maximal*, i.e., $\forall e \in E'_\$. \nexists e' \in E'. e <' e'$.

□

Definition 2.3.4 *Maximal poset*

A poset $(E', <') \in Posets$ is the *maximal poset* of the pMSC if it satisfies the following conditions:

- $E' = E \setminus E'_\$$,
- $<' = <_{vis} \cup \{(e, m_?(e)) \mid e \in E_1\}$.

A maximal poset describes the scenario where no message is lost, which is exactly described by the graphical representation of the pMSC.

□

2.3.2 Semantics

Probabilistic lossy channel system. As mentioned before, instead of using perfect channels for communication, pMSC is using a probabilistic lossy channel system where a certain probability of message loss is defined for every channel. We adopt the *local-fault model* [1, 41] where p_{loss} is defined as the probability of message loss for each channel such that every single message sent using the channel has probability p_{loss} to get lost. It is possible to define the channel as a perfect channel by specifying $p_{loss} = 0$ which means that the every sent message is received with probability 1 (or lost with probability 0). Thus, it is also possible to define a channel where all messages sent get lost after being sent by specifying $p_{loss} = 1$.

The set of posets. Since every message sent can be lost, there are several scenarios that may happen during the execution of a pMSC. Each poset basically represents one unique scenario where certain messages get lost. Lost messages may affect the execution because they disable events which are supposed to be executed when the messages are delivered properly. We require *all posets to be consistent* but they *do not have to be complete*.

Example 2.3.1 *The set of posets*

We take a look once more to the example in Figure 2.4. We redraw the BMSC on the left (without coregion) and redefine it as a pMSC by specifying $p_{loss}(c_{pq}) = 0.1$ for all $p, q \in \mathcal{P}$ which means that any single message sent with any channel can be lost with probability 0.1 after being sent. The pMSC is depicted in Figure 2.5 along with its posets.

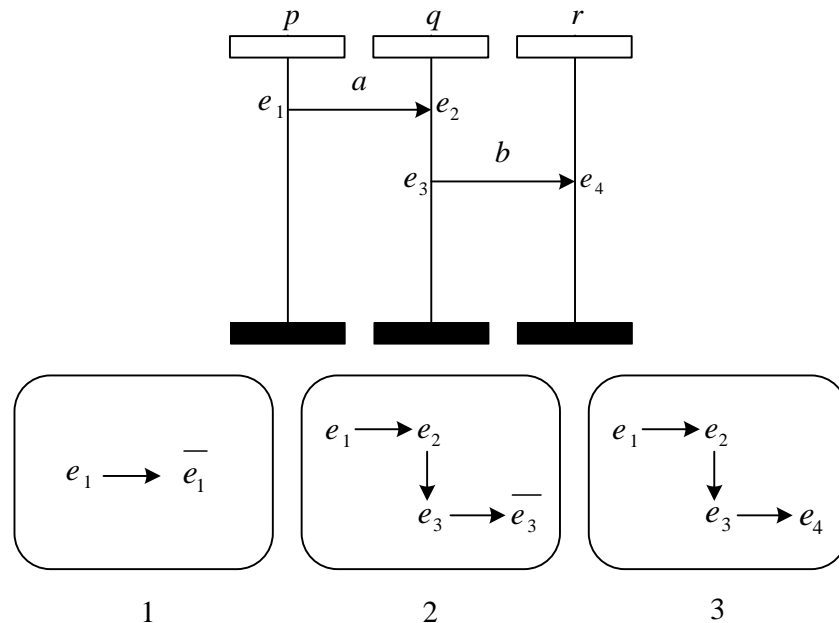


Figure 2.5: A pMSC and its posets

The first poset describes the scenario when message a gets lost. Since we have $\bar{e}_1 <_p e_3$, e_3 cannot occur. As the consequence, e_4 also cannot occur because a

receive event can occur only if the corresponding send event occurs before. These rulings are determined using the conditions for consistency property.

In the second poset, message a is delivered but message b gets lost after being sent. The third poset is the maximal poset because it does not contain any loss event. It describes the scenario depicted by the graphical representation of the pMSC and is exactly the same with the poset we have for the BMSC depicted in the left side of Figure 2.4.

In this example, all posets are consistent but the first poset is not complete while the second and the third are.

□

Example 2.3.2 *The set of posets with incomparable events*

The BMSC with a coregion from the right side of Figure 2.4 shows an example of incomparable events. We again redefine this BMSC as a pMSC by declaring that the channels are probabilistic lossy channels. e_2 and e_3 are incomparable under $<_q$ so unlike the previous example, e_3 can occur without waiting for e_2 to occur first. This results in four posets as seen in the picture. The last poset is the maximal poset which is the same as the poset for BMSC. The first and the third poset show us that when message a gets lost, it does not affect the sending and receiving process of message b because their events are incomparable. Message b can be delivered, as depicted in the first poset, or lost, as seen in the third poset. In this example, all posets are consistent and complete.

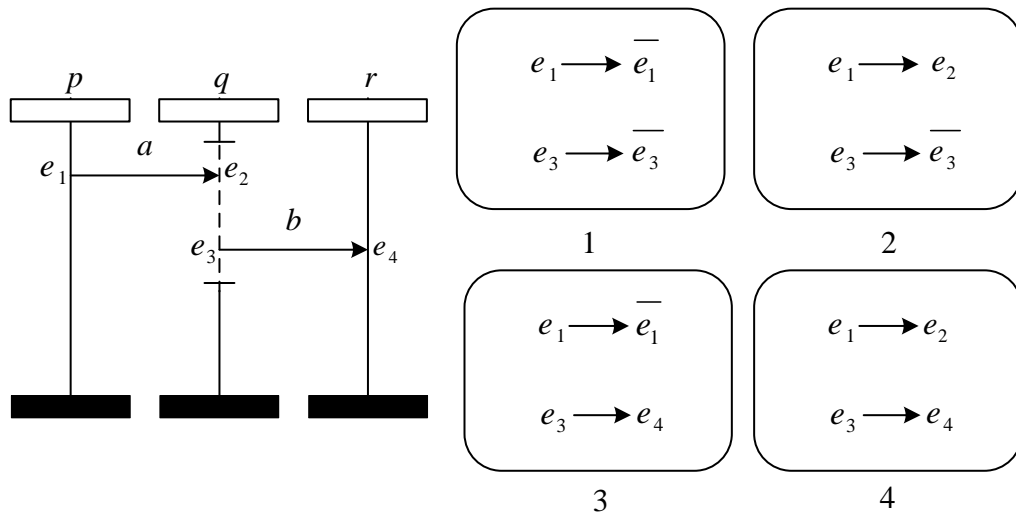


Figure 2.6: A pMSC with incomparable events

□

Example 2.3.3 *Incomplete poset*

The first poset in Example 2.3.1 is incomplete because it does not contain all send events in E . Here we show another example where a poset is incomplete because there exists a send event without its matching receive or loss event.

The first poset in Figure 2.7 shows an example of this situation. Since the loss event \bar{e}_1 occurs instead of e_2 , no event which is supposed to be executed after e_2 can occur, so we cannot include either e_4 or \bar{e}_3 in the poset. This results in the depicted poset, where e_3 does not have either matching receive or loss event. In this case, the consistency property is still satisfied but the poset is not complete. We simply do not know what happens to message b , whether it is received successfully by process q or gets lost after being sent from r .

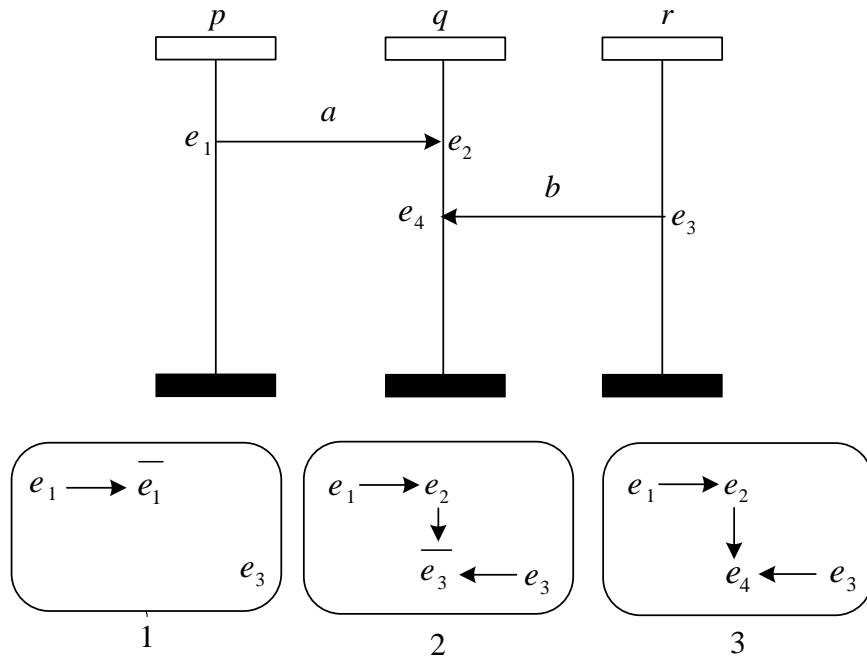


Figure 2.7: A pMSC with complete and incomplete posets

□

Remark. *On generating the set of posets*

We describe a procedure to generate the set of posets of a pMSC. We start by thinking that every poset represents a scenario where only a subset of the send events are successful, i.e., the messages they send are delivered. The following steps are then performed:

1. From the set of send events E_l , we first compute its powerset, $2^{E_l} = \{\emptyset, \dots, E_l\}$, where each element represents a set of successful send events.
2. For each set of send events $E'_l \in 2^{E_l}$, we build a poset $(E', <')$ by including $m_r(e)$ for every $e \in E_l$ if $e \in E'_l$ and $m_s(e)$ if $e \notin E'_l$ and $< = (<_{vis} \cup <_c) \cap (E' \times E')$.

At this point we will get $|2^{E_1}|$ complete posets, but we need to check for its consistency. We will need this set of complete posets later in the chapter, so we denote the set as $Complete(Posets)$.

3. For every poset $(E', <')$, we check its elements for consistency. Based on the conditions for consistency property, an element e is inconsistent if it satisfies one of the following conditions:
 - e is a receive event but it is not preceded by its matching send event,
 - e is a loss event but it is not preceded by its matching send event,
 - e is located at process p but there is a loss event e' occurring before e in process p .

The formal definition for the conditions above can be derived from the conditions for consistency property.

We check every element for consistency by checking all chains which start from the minimal elements of the poset, $min(E', <')$. For every chain, we start with the first element. If the element is consistent, we keep it, otherwise we eliminate it from the poset. We then move to its successor in the chain and apply the same checking procedure. These steps are repeated until we reach the maximal element of every chain, $max(E', <')$. The resulting poset after inconsistent elements elimination is $(E', <')$. It is possible that two different posets result in a same final poset after the elimination procedure.

4. The set of posets, $Posets$, is the union of all $(E', <')$ from the previous steps.

This procedure will result into a set of consistent posets which are not necessarily complete because of the inconsistent events elimination procedure.

Example 2.3.4 *Generating the set of posets*

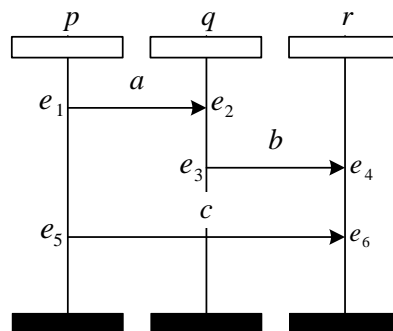


Figure 2.8: Example of a pMSC

From the pMSC depicted in Figure 2.8 we have the set of send events $E_1 = \{e_1, e_3, e_5\}$. We first compute the powerset of E_1 ,

$$2^{E_1} = \{\emptyset, \{e_1\}, \{e_3\}, \{e_5\}, \{e_1, e_3\}, \{e_1, e_5\}, \{e_3, e_5\}, \{e_1, e_3, e_5\}\}$$

Next, we build poset $(E', <')$ for each $E'_i \in 2^{E_i}$, then eliminate its inconsistent elements. This step is illustrated in Figure 2.9. The set of successful events E' is written in the first column. The second column is the built poset with its inconsistent elements being marked, and the last column depicts the resulting poset.

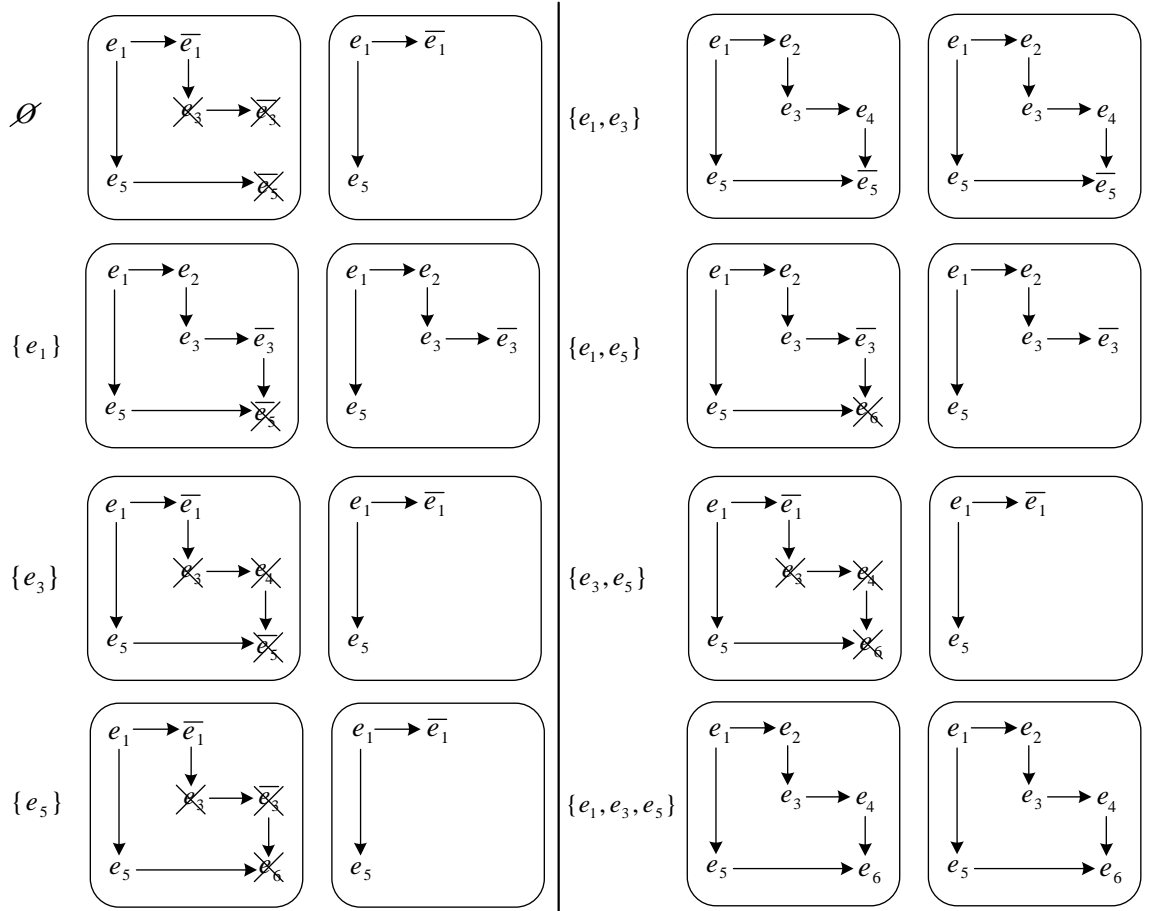


Figure 2.9: Inconsistent events elimination

We can see from Figure 2.9 that some of the resulting posets are equal. The final *Posets* is the union of the resulting posets, depicted in Figure 2.10.

□

Subsumption of scenarios. The scenario represented by poset $(E', <') \in Posets$ subsumes the scenario represented by poset $(E'', <'') \in Posets$ iff $(E', <')$ is the superset of $(E'', <'')$, i.e., $(E'', <'') \subseteq (E', <')$.

Let $(E', <') \in Posets$ be a poset of a pMSC and $Subs(E', <') = \{(E'', <'') | (E'', <'') \subseteq (E', <'), (E'', <'') \in Posets\}$ be the set of all posets whose scenarios are subsumed by

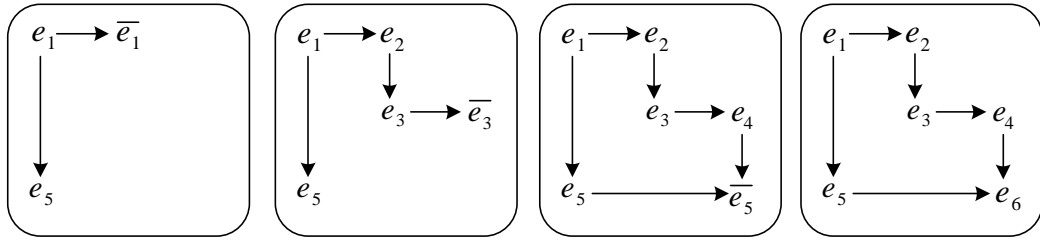


Figure 2.10: The final set of posets

$(E', <')$. Given $(E', <')$, it generalizes the posets in $Subs(E', <')$, and it is not known which $(E'', <'') \in Subs(E', <')$ is actually executed. Therefore, we can consider $(E', <')$ as the union of all scenarios represented by the posets in $Subs(E', <')$, i.e.:

$$(E', <') = \bigcup_{(E'', <'') \in Subs(E', <')} (E'', <'')$$

During generation procedure of the set of posets, every inconsistent poset is modified by removing inconsistent events, resulting into a poset which is the subposet of the original one. Let us take a look in the first poset in Figure 2.5. This poset is consistent but not complete. In this poset, because message a gets lost, we do not know what will actually happen if message b is sent. Thus, this poset covers two scenarios described by two posets where message b is sent, one where message b is delivered and the other where message b gets lost, which are illustrated in Figure 2.11. These two posets are inconsistent. During the generation procedure of the posets, e_3 and e_4 are eliminated from the first poset while e_3 and \bar{e}_3 are eliminated from the second poset, both resulting in the same poset which is the subposet of the two.

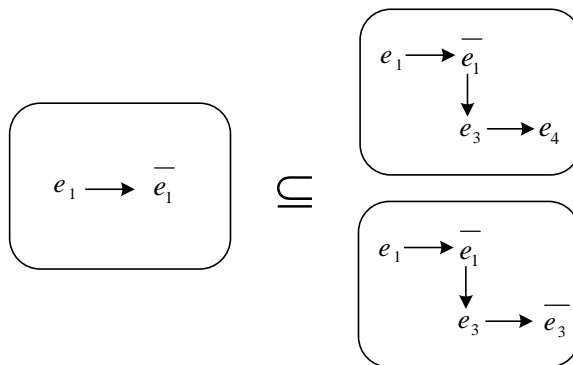


Figure 2.11: Subsumption of posets

As another example, every poset in Figure 2.9 which has its inconsistent events eliminated is modified into another poset which is the subposet the original one.

2.3.3 Linearization

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m?, m\ddagger, Posets, C, p_{loss})$ be a pMSC.

Every poset $(E', <') \in Posets$ specifies the temporal order of how the events are supposed to be executed in the pMSC. Two events can be incomparable under $<'$, which means that we do not specify which of the two should be executed first. During the execution itself however, since we do not allow two events to occur at the same time, one event will always be executed before the other. The order of the events during the real execution forms a total order in which every event is comparable with each other. This total order is formalized as linearization, with the formal definition given below.

Definition 2.3.5 *Linearization of pMSC*

A *linearization* of pMSC M is a finite word over E' with $(E', <') \in Posets$, obtained by considering a total ordering of the events in E' which is consistent with the partial order $<'$. Formally, a word $w = e_1e_2\dots e_n \in (E')^*$ is a linearization of $(E', <') \in Posets$ such that whenever $e_i <' e_j$, we have $i < j$ (note that $<$ in $i < j$ is the ordering in natural numbers). For $1 \leq i \leq n$, let $w_i = e_i$ denotes the i -th element (event) in w .

□

From every poset $(E', <') \in Posets$ of a pMSC M , we get a set of linearizations in which the difference between one linearization with another is the ordering of the events. We denote this set as $Lin_M(E', <')$, and when the pMSC context is known, we shorten the notation as $Lin(E', <')$ such that

$$Lin(E', <') = \{w \in (E')^* \mid w \text{ is a linearization of } (E', <')\}$$

All linearizations belong to $Lin(E', <')$ are composed from the same set of events, E' , and are consistent with the partial order $<'$.

We can now define $Lin(M)$ as the total set of linearizations of a pMSC M such that

$$Lin(M) = \bigcup_{(E', <') \in Posets} Lin(E', <').$$

We will also use the notation $Lin(Posets)$ as the set of linearizations generated from $Posets$ when the pMSC context is known, i.e., $Lin(Posets) = Lin(M)$ given $Posets$ the set of posets of M .

Example 2.3.5 *Linearization of pMSCs*

In Figure 2.5 we get exactly one linearization from each poset because all events are comparable to every other event in the poset such that there is only one possible ordering for each poset. We have $Lin(M) = \{e_1\bar{e}_1, e_1e_2e_3\bar{e}_3, e_1e_2e_3e_4\}$.

From Figure 2.6 on the other hand, we get more than one linearization from each poset because some events are not comparable to each other under the partial order.

This results into the possibility to have more than one execution orders from each poset. The linearizations of the pMSC for every poset $(E_i, <_i) \in Posets, i \in [1, 4]$ are given in Table 2.1.

i	$Lin(E_i, <_i)$
1	$\{e_1\bar{e}_1e_3\bar{e}_3, e_1e_3\bar{e}_1\bar{e}_3, e_1e_3\bar{e}_3\bar{e}_1, e_3\bar{e}_3e_1\bar{e}_1, e_3e_1\bar{e}_3\bar{e}_1, e_3e_1\bar{e}_1\bar{e}_3\}$
2	$\{e_1e_2e_3\bar{e}_3, e_1e_3e_2\bar{e}_3, e_1e_3\bar{e}_3e_2, e_3\bar{e}_3e_1e_2, e_3e_1\bar{e}_3e_2, e_3e_1e_2\bar{e}_3\}$
3	$\{e_1\bar{e}_1e_3e_4, e_1e_3\bar{e}_1e_4, e_1e_3e_4\bar{e}_1, e_3e_4e_1\bar{e}_1, e_3e_1e_4\bar{e}_1, e_3e_1\bar{e}_1e_4\}$
4	$\{e_1e_2e_3e_4, e_1e_3e_2e_4, e_1e_3e_4e_2, e_3e_4e_1e_2, e_3e_1e_4e_2, e_3e_1e_2e_4\}$

Table 2.1: Linearization of pMSC with incomparable events

As we see in the table, incomparable events can occur in any order. In the fourth poset for example, the only order we require is that e_2 should occur after e_1 and e_4 after e_3 . We can see that based on the partial order, e_3 is not comparable to e_1 and e_2 so it can be executed before, after, or between the two events.

□

2.3.4 Probabilistic semantics

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m_?, m_?, Posets, C, p_{loss})$ be a pMSC. In this section we will define some probabilistic semantics for events, posets and linearizations.

2.3.4.1 Probability of events

The probability of message loss allows us to define the probability of each event $e \in E$ of pMSC M as follows:

$$Pr(e) = \begin{cases} 1 & \text{if } e \in E_! \cup E_{int}, \\ 1 - p_{loss}(c_{pq}) & \text{if } e \in E_? \text{ and } l(e) = q?p(a), \\ p_{loss}(c_{pq}) & \text{if } e \in E_? \text{ and } l(e) = p \& q(a) \end{cases}$$

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

The probability of a set of events $E = \{e_1, e_2, \dots, e_n\}$ is the product of the probability of all of its events,

$$Pr(E) = \prod_{i=1}^n Pr(e_i).$$

This result comes from the interpretation that the probability of E is the probability that all of its events being executed. Because e_1, e_2, \dots, e_n are independent from each other, we can consider E as $\{e_1 \cap e_2 \cap \dots \cap e_n\}$ and just apply the probability theory on the intersection of independent events.

2.3.4.2 Probability of posets

We define the probability of a poset $(E', <') \in Posets$ as the probability of its set of events,

$$Pr(E', <') = Pr(E').$$

Example 2.3.6 Probability of posets

Using again Figure 2.7, with $p_{loss}(c_{pq}) = 0.2$ and $p_{loss}(c_{rq}) = 0.3$ we get the following result:

$$\begin{aligned} Pr(E_1, <_1) &= Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_3) &= 1 \times 0.2 \times 1 &= 0.20 \\ Pr(E_2, <_2) &= Pr(e_1) \times Pr(e_2) \times Pr(e_3) \times Pr(\bar{e}_3) &= 1 \times 0.8 \times 1 \times 0.3 &= 0.24 \\ Pr(E_3, <_3) &= Pr(e_1) \times Pr(e_2) \times Pr(e_3) \times Pr(e_4) &= 1 \times 0.8 \times 1 \times 0.7 &= 0.56 \end{aligned}$$

□

Probability of union of set of posets. From all posets of a pMSC, during execution only one scenario represented by one particular poset will actually be executed. When we join two posets, $(E_1, <_1) \cup (E_2, <_2)$, with $(E_1, <_1), (E_2, <_2) \in Posets$, the interpretation is that either one of the two posets will actually be executed. Since every poset represent a distinct scenario, we can apply the probability theory on the union of independent events such that

$$\begin{aligned} Pr((E_1, <_1) \cup (E_2, <_2)) &= Pr(E_1, <_1) + Pr(E_2, <_2) \\ &= Pr(E_1) + Pr(E_2) \end{aligned}$$

This result also holds when $E_1 \cap E_2 \neq \emptyset$. Let us consider the case where $E_1 \cap E_2 = E$. As mentioned before, we consider a set of events as the intersection of all its independent events. Thus, we can consider E_i as $(E \cap (E_i \setminus E))$ so we have $Pr(E_i) = Pr(E) \times Pr(E_i \setminus E)$ for $i = \{1, 2\}$. In this case,

$$\begin{aligned} Pr((E_1, <_1) \cup (E_2, <_2)) &= Pr(E) \times Pr(E_1 \setminus E) + Pr(E) \times Pr(E_2 \setminus E) \\ &= Pr(E_1) + Pr(E_2) \end{aligned}$$

Probability of incomplete posets. When a poset $(E', <') \in Posets$ is incomplete, its scenario subsumes the scenarios of other posets which are complete but inconsistent. We will show that the probability of $(E', <')$ equals to the sum of probability of the posets it subsumes.

Lemma 2.3.6 Probability of incomplete poset

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m_?, m_!, Posets, C, p_{loss})$ be a pMSC and $(E', <') \in Posets$ an incomplete poset of M . Let $CompSubs(E', <')$ be the set of complete posets of

Posets which are subsumed by $(E', <')$, i.e., $CompSubs(E', <') = Complete(Posets) \cap Subs(E', <')$. The probability of $(E', <')$ satisfies the following property:

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

Proof:

From the posets generation procedure we are sure that $CompSubs(E', <')$ is a maximal set, i.e., it contains all possible posets whose scenarios are subsumed by $(E', <')$. Therefore, given $(E', <')$ we know that one of the members of $CompSubs(E', <')$ actually occurs so we can consider $(E', <')$ as the union of all posets that it subsumes,

$$(E', <') = \bigcup_{(E'', <'') \in CompSubs(E', <')} (E'', <'')$$

and therefore,

$$\begin{aligned} Pr(E', <') &= Pr\left(\bigcup_{(E'', <'') \in CompSubs(E', <')} (E'', <'')\right) \\ &= \sum_{(E'', <'') \in CompSubs(E', <')} Pr(E'', <'') \end{aligned}$$

□

Example 2.3.7 *Probability of incomplete posets*

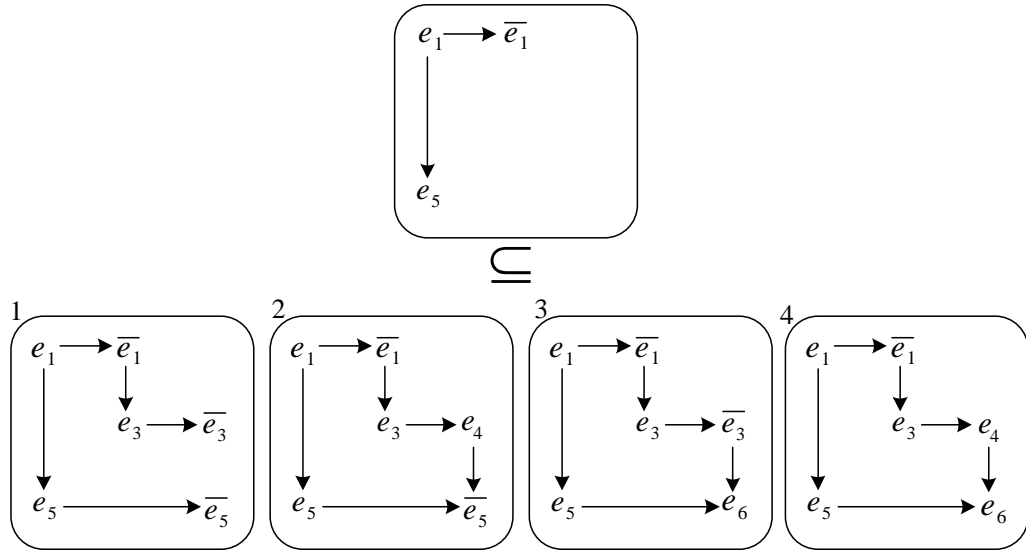


Figure 2.12: Incomplete poset and the posets it subsumes

Let us take a look at the posets in Figure 2.9. We will compute the probability of one of its incomplete final poset. Figure 2.12 depicts the poset and all the complete posets it subsumes. Suppose the probability of message loss is 0.2 for all channels. We first compute the probability of the complete posets:

$$\begin{aligned}
Pr(E_1, <_1) &= Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_3) \times Pr(\bar{e}_3) \times Pr(e_5) \times Pr(\bar{e}_5) &= 1 \times 0.2 \times 1 \times 0.2 \times 1 \times 0.2 &= 0.008 \\
Pr(E_2, <_2) &= Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_3) \times Pr(e_4) \times Pr(e_5) \times Pr(\bar{e}_5) &= 1 \times 0.2 \times 1 \times 0.8 \times 1 \times 0.2 &= 0.032 \\
Pr(E_3, <_3) &= Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_3) \times Pr(\bar{e}_3) \times Pr(e_5) \times Pr(e_6) &= 1 \times 0.2 \times 1 \times 0.2 \times 1 \times 0.8 &= 0.032 \\
Pr(E_4, <_4) &= Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_3) \times Pr(e_4) \times Pr(e_5) \times Pr(e_6) &= 1 \times 0.2 \times 1 \times 0.8 \times 1 \times 0.8 &= 0.128
\end{aligned}$$

Next, we compute the probability of the incomplete poset:

$$Pr(E, <) = Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_5) = 1 \times 0.2 \times 1 = 0.2$$

and then show that the probability equals to the sum of the probability of the posets it subsumes:

$$\begin{aligned}
Pr(E, <) &= Pr(E_1, <_1) + Pr(E_2, <_2) + Pr(E_3, <_3) + Pr(E_4, <_4) \\
&= 0.008 + 0.032 + 0.032 + 0.128 \\
&= 0.2
\end{aligned}$$

□

2.3.4.3 Probability of linearizations

From a poset $(E', <') \in Posets$, we can get a set of linearizations $Lin(E', <')$ which are composed from the same set of events E' and consistent to the partial order $<'$. We define the probability of $Lin(E', <')$ as the probability of the poset itself, i.e.,

$$Pr(Lin(E', <')) = Pr(E', <').$$

Nondeterminism on linearizations. We define the probability semantics on the set of linearizations $Lin(E', <')$ instead of on each single linearization $w \in Lin(E', <')$ because the difference between each linearization is only the ordering of the events. Given a poset $(E', <')$ and two events $e, e' \in E'$, if e and e' are not comparable under $<'$, we do not know which event will occur first. The actual order is chosen nondeterministically during execution. Based on this reasoning, we do not want to specify that a linearization $w \in Lin(E', <')$ in which e occurs before e' has a larger, less, or equal probability than another linearization $w' \in Lin(E', <')$ in which e' occurs first, because of the nondeterminism.

As an example, we have in Table 2.1 the set of linearizations for the complete poset:

$$Lin(E_4, <_4) = \{e_1e_2e_3e_4, e_1e_3e_2e_4, e_1e_3e_4e_2, e_3e_4e_1e_2, e_3e_1e_4e_2, e_3e_1e_2e_4\}$$

We cannot say, for example, that $Pr(e_1e_2e_3e_4) = Pr(e_1e_3e_2e_4)$ because we do not know the probability of e_2 happens before e_3 . This applies also to the other linearizations in the set. We simply define that the probability of having e_1, e_2, e_3, e_4 executed with any ordering consistent to the partial order as $Pr(Lin(E_4, <_4)) = Pr(E_4, <_4) = Pr(\{e_1, e_2, e_3, e_4\})$.

Probability of the total set of linearizations. The total set of linearizations is the union of the set of linearizations of all posets belong to $Posets$. Thus, the

probability of the total set of linearizations of M is defined as follows:

$$\begin{aligned} Pr(Lin(M)) &= Pr(Lin(Posets)) \\ &= Pr(\bigcup_{(E', \langle' \rangle) \in Posets} Lin(E', \langle' \rangle)) \\ &= \sum_{(E', \langle' \rangle) \in Posets} Pr(Lin(E', \langle' \rangle)) \end{aligned}$$

Proposition 2.3.1 *The probability of the total set of linearizations of a pMSC*

Given a pMSC $M = (\mathcal{P}, \Sigma, Act, E, l, m?, m\$, Posets, C, p_{loss})$, the probability of the total set of its linearizations, $Pr(Lin(M)) = Pr(Lin(Posets))$ equals 1.

Proof:

We give the proof in terms of probability space (See Appendix A.2 for notations and basic theory). Let (Ω, ξ, Pr) a probability space over the linearizations of M such that:

- the sample space Ω is the total set of linearizations of M , i.e., $\Omega = Lin(M) = Lin(Posets) = \bigcup_{(E', \langle' \rangle) \in Posets} Lin(E', \langle' \rangle)$,
- the events $\xi = \bigcup_{n \geq 1} Ev_n$ is mapped to $Lin(Posets) = \bigcup_{1 \leq n \leq |Posets|} (E', \langle' \rangle)_n$ such that $Ev_n = Lin(E', \langle' \rangle)_n$ for $1 \leq n \leq |Posets|$,
- since $Lin(E', \langle' \rangle)_n, 1 \leq n \leq |Posets|$ are disjoint sets,

$$\begin{aligned} \sum_{1 \leq n \leq |Posets|} Pr(Lin(E', \langle' \rangle)_n) &= Pr(\bigcup_{1 \leq n \leq |Posets|} Lin(E', \langle' \rangle)_n) \\ &= Pr(Lin(Posets)) \\ &= Pr(\Omega) \\ &= 1 \end{aligned}$$

We are sure that Ω has covered every possible scenarios because of the generation procedure we define before, that is, $Complete(Posets)$ are generated from the powerset of the send events. When a member of $Complete(Posets)$ is not in $Posets$ then it is not consistent and is subsumed by an incomplete but consistent poset. Because the probability of an incomplete poset is the total probability of the set of complete posets it subsumes (Lemma 2.3.6), the equation defined above holds. □

Example 2.3.8 Consider the posets of pMSC depicted in Figure 2.10. Suppose the probability of message loss is 0.2 for all channels. We compute the probability of the linearizations as follows:

$$\begin{array}{lll} Pr(Lin(E_1, \langle_1 \rangle)) = Pr(e_1) \times Pr(\bar{e}_1) \times Pr(e_5) & = 1 \times 0.2 \times 1 & = 0.2 \\ Pr(Lin(E_2, \langle_2 \rangle)) = Pr(e_1) \times Pr(e_2) \times Pr(e_3) \times Pr(\bar{e}_3) \times Pr(e_5) & = 1 \times 0.8 \times 1 \times 0.2 \times 1 & = 0.16 \\ Pr(Lin(E_3, \langle_3 \rangle)) = Pr(e_1) \times Pr(e_2) \times Pr(e_3) \times Pr(e_4) \times Pr(e_5) \times Pr(\bar{e}_5) & = 1 \times 0.8 \times 1 \times 0.8 \times 1 \times 0.2 & = 0.128 \\ Pr(Lin(E_4, \langle_4 \rangle)) = Pr(e_1) \times Pr(e_2) \times Pr(e_3) \times Pr(e_4) \times Pr(e_5) \times Pr(e_6) & = 1 \times 0.8 \times 1 \times 0.8 \times 1 \times 0.8 & = 0.512 \end{array}$$

We now compute the total and show that it equals 1,

$$\begin{aligned} Pr(Lin(Posets)) &= Pr(Lin(E_1, \langle_1 \rangle)) + Pr(Lin(E_2, \langle_2 \rangle)) + Pr(Lin(E_3, \langle_3 \rangle)) + Pr(Lin(E_4, \langle_4 \rangle)) \\ &= 0.2 + 0.16 + 0.128 + 0.512 \\ &= 1 \end{aligned}$$

□

Chapter 3

Probabilistic Extension of Message Sequence Graphs

3.1 Introduction and motivation

Message Sequence Graph (MSG) is a formalism to specify a collection of MSCs. MSG is a directed graph with an initial vertex and a set of final vertices where each vertex is labeled with an MSC, and it defines some ways to combine them using concatenation, choice, and repetition.

We would like to incorporate both probabilistic and nondeterministic behavior in our model of MSG. Probabilistic aspects are useful and sometimes necessary in system modeling. It can be used to quantify system performance, for example to define the likelihood of the system to fail in a given period of time, or to quantify possible outcomes of randomized actions of the system such as tossing a coin. Nondeterminism can be used to model interface between the system and an unpredictable environment. It is also employed in randomized distributed algorithms where the processes involved are behaving nondeterministically in nature. We choose Markov decision process [38] as the base of our model because it employs both probabilistic and nondeterministic choices.

In this chapter we first describe Markov chains and Markov decision processes as the probabilistic models that we are going to use in our extension of MSG. Next, we define the concatenation operator as a way to combine pMSCs, followed by the discussion about the probabilistic extension of MSG which covers the syntax and semantics of the model. A case study is presented at the end of the chapter to give an illustration on how the probabilistic extension of MSG can be applied to model real life situations.

3.2 Probabilistic models

Systems in real world are subject to various stochastic phenomena. In order to model these random phenomena, *probabilities* and *nondeterminism* are incorporated into

system models. *Discrete time Markov chain* (MC) is a common model to represent probabilistic systems. Markov chains are transition systems with probability distribution for the successors of each state such that the next state from the current one is chosen probabilistically. Some systems such as randomized distributed systems also exhibit nondeterminism beside probabilistic behavior. Since Markov chain only allows probabilistic behaviors, *Markov decision process* (MDP) is used to allow both nondeterministic and probabilistic behaviors of the system to be specified.

3.2.1 Markov chains

Markov chains employ probabilistic choices among successor states of transition systems. When the system is in state s , the next state is chosen according to a probability distribution which only depends on the current state. Markov chains exhibit *memoryless property* such that the future states depend only on the current state and are independent of past states.

At each step the system may change its state from the current state to another state according to a probability distribution. The changes of state are called *transitions* and the probabilities associated with the transitions are called *transition probabilities*.

Definition 3.2.1 (*Discrete Time*) *Markov Chain*

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

- S is a nonempty set of states,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a *transition probability function* such that for all states s ,

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

- $i_{init} : S \rightarrow [0, 1]$ is a *initial distribution* 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 which assigns to each state a set of atomic propositions.

\mathcal{MC} is a finite Markov chain if S and AP are finite. If we are not dealing with logic, the atomic propositions and the labeling function can be removed from the definition.

□

The transition probability function \mathbf{P} is a probability distribution. For each state $s \in S$, the probability which performs the transition, i.e., moving from s to another state $s' \in S$, is denoted as $\mathbf{P}(s, s')$. The initial distribution i_{init} specifies the probability

of each state to be the initial state, which is the state where the system starts to evolve. *Initial states* are all state $s \in S$ with $i_{init}(s) > 0$.

The *underlying graph* induced by a Markov chain is a set of vertices and directed edges where each vertex corresponds to a state in the Markov chain and the edge $s \rightarrow s'$ exists if and only if $\mathbf{P}(s, s') > 0$. The edges are labeled with the corresponding transition probability. The label can be omitted if the transition probability equals 1, that is, when the current state only has a single successor state.

$Post(s)$ denotes the set of successor states of s , i.e., $Post(s) = \{s' \in S | \mathbf{P}(s, s') > 0\}$. $Pre(s)$ is the set of predecessor states of s such that $Pre(s) = \{s' \in S | \mathbf{P}(s', s) > 0\}$.

Example 3.2.1 *The weather in the Land of Oz*

According to Kemeny, Snell and Thompson [30], the Land of Oz is not blessed with good weather. They never have two consecutive days with nice weather. When the day is nice, the people can be sure that in the following day it will be rainy or snowy. If they have snow or rain, there is a half chance that the following day will have the same weather. If the weather changes, only with half chance that it will change into a nice weather in the next day.

This problem is represented as Markov chains in Figure 3.1. Every state represents one type of weather. The transition probability from one state to another is associated with every edge.

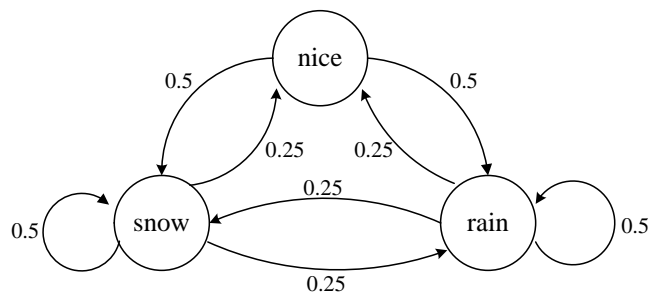


Figure 3.1: Markov chain for the weather problem

□

A *path* of a Markov chain is an infinite sequence $\pi = s_0 s_1 s_2 \dots \in S^\omega$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. Any finite prefix of π which ends in a state is a *finite path*. $Paths(\mathcal{MC})$ denotes the set of (infinite) paths in \mathcal{MC} and $Paths_{fin}(\mathcal{MC})$ the set of finite paths. $Paths(s, \mathcal{MC})$ denotes the set of all paths in \mathcal{MC} which start from state s and $Paths_{fin}(s, \mathcal{MC})$ the set of finite paths which start from s .

The notion of probabilities in the Markov chain \mathcal{MC} is formalized by associating a probability space with \mathcal{MC} (see Appendix A.2 for notations). The outcomes are defined as the infinite paths of \mathcal{MC} , that is, $\Omega^{\mathcal{MC}} = Paths(\mathcal{MC})$. The σ -algebra associated with \mathcal{MC} is generated by the *cylinder sets* spanned by the finite paths

in \mathcal{MC} . Given a finite path $\hat{\pi} = s_0s_1\dots s_n \in Paths_{fin}(\mathcal{MC})$, the cylinder set of $\hat{\pi}$ is given as

$$Cyl(\hat{\pi}) = \{\pi \in Paths(\mathcal{MC}) | \hat{\pi} \text{ is a prefix of } \pi\}.$$

The probability of a finite path $\hat{\pi} = s_0s_1s_2\dots s_n$ is defined as

$$\mathbf{P}(\hat{\pi}) = \prod_{0 \leq i < n} \mathbf{P}(s_i, s_{i+1}).$$

Let $\xi^{\mathcal{MC}}$ be the smallest σ -algebra on $Paths(\mathcal{MC})$ which contains all cylinder sets $Cyl(\hat{\pi})$ for all $\hat{\pi} \in Paths_{fin}(\mathcal{MC})$. There exists a probability measure Pr on the σ -algebra $\xi^{\mathcal{MC}}$ where the probabilities for the events (i.e., the cylinder sets) are given by

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

where $\hat{\pi} = s_0s_1s_2\dots s_n$.

When we consider paths which start in a certain state s , we apply the same construction with $\Omega^{\mathcal{MC}} = Paths(s, \mathcal{MC})$. s is not necessarily the initial state of the Markov chain \mathcal{MC} , so we apply initial distribution $i_s(s') = 1$ if $s = s'$ and $i_s(s') = 0$ otherwise, for all $s' \in S$.

3.2.2 Markov decision processes

Markov decision process (MDP) is a variant of Markov chain which permits both probabilistic and nondeterministic choices. This means that it is possible in a state to nondeterministically choose between more than one probability distributions on the successor states. The chosen distribution then determines the probability of which successor state will be taken.

Definition 3.2.2 *Markov decision process (MDP)*

A Markov decision process is a structure $\mathcal{M} = (S, Act, \mathbf{P}, i_{init}, AP, L)$ with:

- S is a finite set of states,
- Act is a set of actions,
- $\mathbf{P}: S \times Act \times S \rightarrow [0, 1]$ is the transition probability function such that for all states $s \in S$ and actions $\alpha \in Act$:

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

- $i_{init}: S \rightarrow [0, 1]$ is a *initial distribution* 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 which assigns to each state a set of atomic propositions.

An action α is *enabled* in state s if and only if $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. $Act(s)$ denotes the set of enabled actions in s . For any state $s \in S$, it is required that $Act(s) \neq \emptyset$. $Post(s, \alpha)$ denotes α -successors of s , which are the set of states s' such that $\mathbf{P}(s, \alpha, s') > 0$. When α is not enabled in s then $Post(s, \alpha)$ is the empty set.

As in Markov chains, if we are not dealing with logic, the atomic propositions and the labeling function can be removed from the definition. □

The system evolves by starting from one of the initial states which is chosen probabilistically from the initial distribution. On entering every state s , the system makes a nondeterministic choice between enabled actions in s . When an action $\alpha \in Act(s)$ is chosen, the system then perform a probabilistic choice to select the next state among its successors in $Post(s, \alpha)$. That is, given α the chosen action, the probability of making transition from s to s' is $\mathbf{P}(s, \alpha, s')$.

A *path* of \mathcal{M} is an infinite sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ with $s_i \in S$, $\alpha_i \in Act(s_i)$, such that $\mathbf{P}(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 0$. Any finite prefix of π which ends in a state is a *finite path*.

Example 3.2.2 *Life cycle of a machine*

The MDP in Figure 3.2 depicts the life cycle of a machine. The machine can be in one of the three states at any time: good, deteriorating or broken. In every state, the choice whether to perform maintenance or to ignore the condition of the machine is a nondeterministic choice. Maintenance brings the machine into good state. When the machine is in good or deteriorating state and maintenance is not performed, there is 0.1 probability that the machine will go into a worse state. Otherwise, it will stay in the current state. A broken machine stays broken without maintenance and revives into its good state otherwise. □

3.3 Concatenation operator

We recognize that there are two different interpretations of concatenation of MSC. Let us consider the concatenation of two BMSCs M_1 and M_2 illustrated in Figure 3.3. Under the *synchronous concatenation*, all events in M_1 finish before any event in M_2 occurs as shown in Figure 3.3(a). When *asynchronous concatenation* (also called weak concatenation) is applied, the concatenation is done process by process, resulting in the partial order depicted by 3.3(b).

In this work we always use the asynchronous interpretation for concatenation.

Since a pMSC has a set of posets instead of a single one, we need a mechanism to concatenate sets of posets. Figure 3.4 depicts two pMSCs, M_1 and M_2 , with their set of posets. We will use this example throughout the section to illustrate how the two pMSCs can be concatenated.

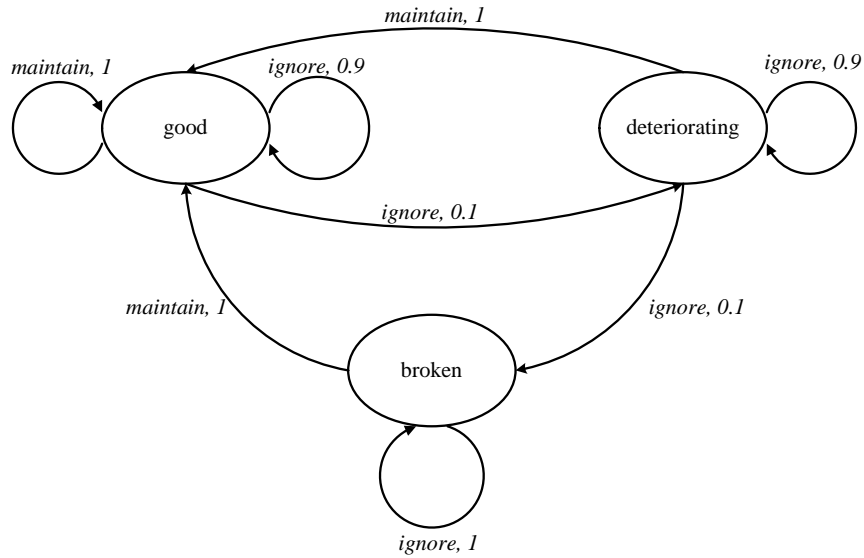


Figure 3.2: Life cycle of a machine as MDP

We will start by defining a mechanism to concatenate two single posets then continue to describe the mechanism for concatenating two sets of posets.

3.3.1 Concatenation of posets

In this subsection we define a concatenation operator to combine single posets. Since concatenation operator is used to combine two pMSCs and we require all posets of pMSC to be consistent, we want to make sure that when we combine two posets from two different pMSCs, we will get a poset which is also consistent.

Example 3.3.1 Concatenation of posets

Let us take a look of what may resulted from the concatenation of poset 1b of M_1 and poset 2a of M_2 from Figure 3.4. Since in poset 1b event \bar{e}_1 occurs instead of e_2 , no other events in process q can occur afterwards. Because of this, event e_3 of poset 2a cannot occur. Thus, when we concatenate the two posets, we have to check

- if any loss event occurs in the first poset and make sure that no events belong to the same process in the second poset occurs after.

Next, since e_3 does not occur, e_4 cannot occur afterwards. Therefore,

- for any receive event in the second poset, we need to check whether the matching send event occurs before. This can also be generalized for loss events.

The next event we care about is e_6 which is located at process r . There is no loss event occurs before in the process, but since its predecessor in the second poset, e_4 , does not occur, then e_6 does not occur either. It means that we also need to check

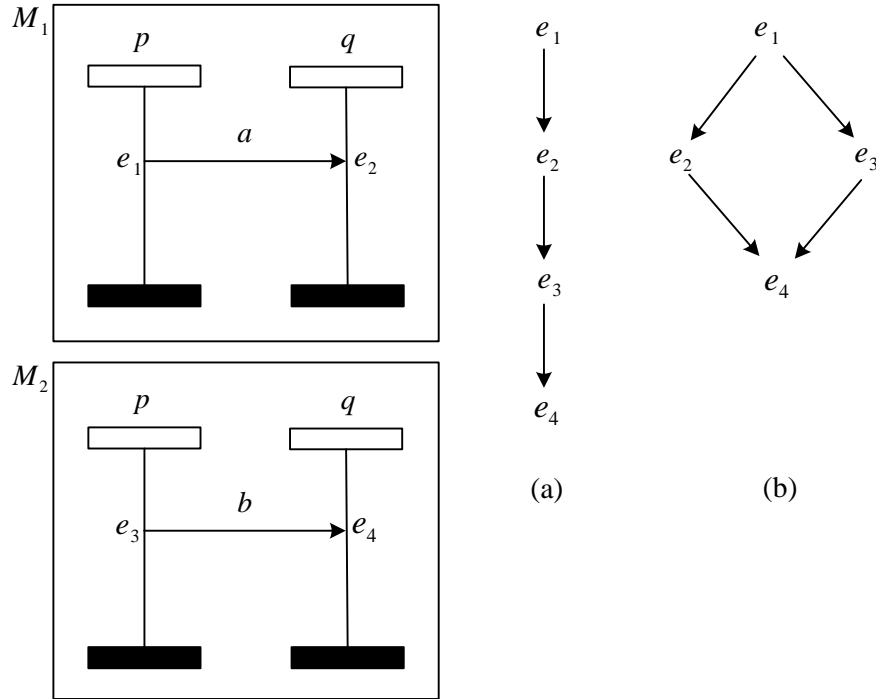


Figure 3.3: Different interpretations of concatenation

- for every event in the second poset, whether its predecessors in the second poset occurs before.

□

The example above highlights some conditions needed to define the concatenation operator to guarantee that the resulting poset is consistent. We give the formal definition of the concatenation operator as follows.

Definition 3.3.1 *Concatenation of posets*

Let $(E_1, <_1)$ and $(E_2, <_2)$ be two posets with $E_1 \cap E_2 = \emptyset$. The concatenation of two posets $(E_1, <_1)$ and $(E_2, <_2)$ is the poset $(E_1, <_1) \cdot (E_2, <_2) = (E, <)$ where:

- $E = E_1 \cup E'_2$ and $E'_2 \subseteq E_2$ is the smallest set such that for all $e \in E'_2$,
 - $\neg \exists e' \in E_1 \setminus \{e\}. loc(e') = loc(e)$,
 - $\forall e' \in E_2. e' <_2 e \wedge loc(e') = loc(e) \Rightarrow e' \in E'_2$,
 - if $e \in E_1$ then $m_1^{-1}(e) \in E'_2$, and
 - if $e \in E_2$ then $m_2^{-1}(e) \in E'_2$.

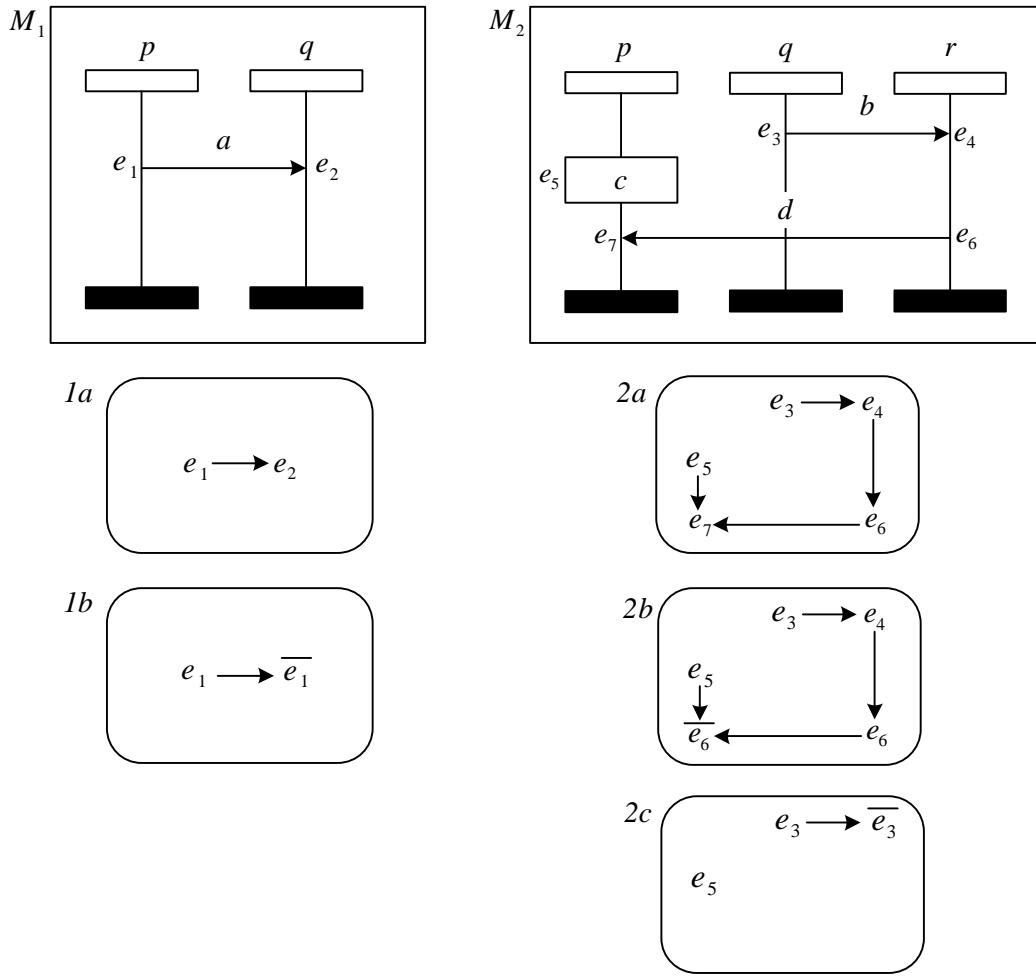


Figure 3.4: pMSCs to be concatenated

- $\leq = \leq_1 \cup \leq'_2 \cup \{(e, e') \mid e \in E_1 \wedge e' \in E'_2 \wedge loc(e) = loc(e') \text{ for all } p \in \mathcal{P}_1 \cup \mathcal{P}_2\}$ with $\leq'_2 = \leq_2 \cap (E'_2 \times E'_2)$. \leq'_2 denotes the partial order \leq_2 imposed on the set of events $E'_2 \subseteq E_2$. The last part of the union is needed to glue \leq_1 and \leq'_2 process by process. It denotes the 'top-to-bottom' order of events such that events of E_1 which are located in process p precede events of E_2 in p for all $p \in \mathcal{P}_1 \cup \mathcal{P}_2$.

□

Example 3.3.2 *Concatenation of posets, continued*

In Example 3.3.1, we discuss about concatenating poset 1b of M_1 with poset 2a of M_2 . Based on the concatenation operator defined above the resulting poset $1b \cdot 2a$ is given in Figure 3.5. In fact, the same poset will be resulted from the concatenation between 1b with any other poset of M_2 , that is, $1b \cdot 2a = 1b \cdot 2b = 1b \cdot 2c$.

□

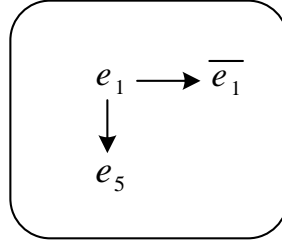


Figure 3.5: The resulting poset from concatenation

Proposition 3.3.1 *Consistency of concatenation of posets*

Let $(E_1, <_1)$ and $(E_2, <_2)$ be two posets with $E_1 \cap E_2 = \emptyset$. The concatenation of $(E_1, <_1)$ and $(E_2, <_2)$, $(E, <) = (E_1, <_1) \cdot (E_2, <_2)$, is consistent if $(E_1, <_1)$ and $(E_2, <_2)$ are consistent.

Proof:

Assume that $(E_1, <_1)$ and $(E_2, <_2)$ are consistent. By definition, $E = E_1 \cup E'_2$ and $E'_2 \subseteq E_2$ is the smallest set such that for all $e \in E'_2$,

- $\neg \exists e' \in E_{1\ddagger}. loc(e') = loc(e)$. Since loss events in E_1 and E_2 are maximal, this guarantees that loss events in E are also maximal by making sure that if a loss event occurs the first poset, no other event in the second poset which has the same location with the loss event will occur. (i)
- $\forall e' \in E_2. e' <_2 e \wedge loc(e') = loc(e) \Rightarrow e' \in E'_2$. This make sure that the partial order in the second poset is preserved such that if one event cannot occur (possibly because of a loss event in the first poset), its successors cannot occur either.
- if $e \in E_{\ddagger}$ then $m_{\ddagger}^{-1}(e) \in E'_2$. It makes sure that every receive event in E'_2 is preceded by its corresponding send event. Since E_1 already satisfies this condition, $E = E_1 \cup E'_2$ also satisfies the condition. (ii)
- if $e \in E_{\ddagger}$ then $m_{\ddagger}^{-1}(e) \in E'_2$. Thus, every loss event in E'_2 is preceded by its corresponding send event. Since E_1 already satisfies this condition, $E = E_1 \cup E'_2$ also satisfies the condition. (iii)

Since $(E, <)$ satisfies (i),(ii),and (iii), by definition it is consistent if it also satisfies the condition that whenever a message sent by a send event is received, then it is not lost and vice versa. Since $(E_1, <_1)$ and $(E_2, <_2)$ are consistent and $E_1 \cap E_2 = \emptyset$, if we have a receive event e (loss event, resp.) in E_1 , it is not possible to have the corresponding loss event (receive event, resp.) e' , $m_{\ddagger}^{-1}(e) = m_{1\ddagger}^{-1}(e')$, in E_2 , and therefore in E'_2 . Therefore, $(E, <) = (E_1, <_1) \cdot (E_2, <_2)$ is consistent.

□

Remark. *Associative Property*

Concatenation operator of posets is not associative. Let $(E_1, <_1), (E_2, <_2), (E_3, <_3)$ be posets. We have that $((E_1, <_1) \cdot (E_2, <_2)) \cdot (E_3, <_3) \neq (E_1, <_1) \cdot ((E_2, <_2) \cdot (E_3, <_3))$. We will show this property by example.

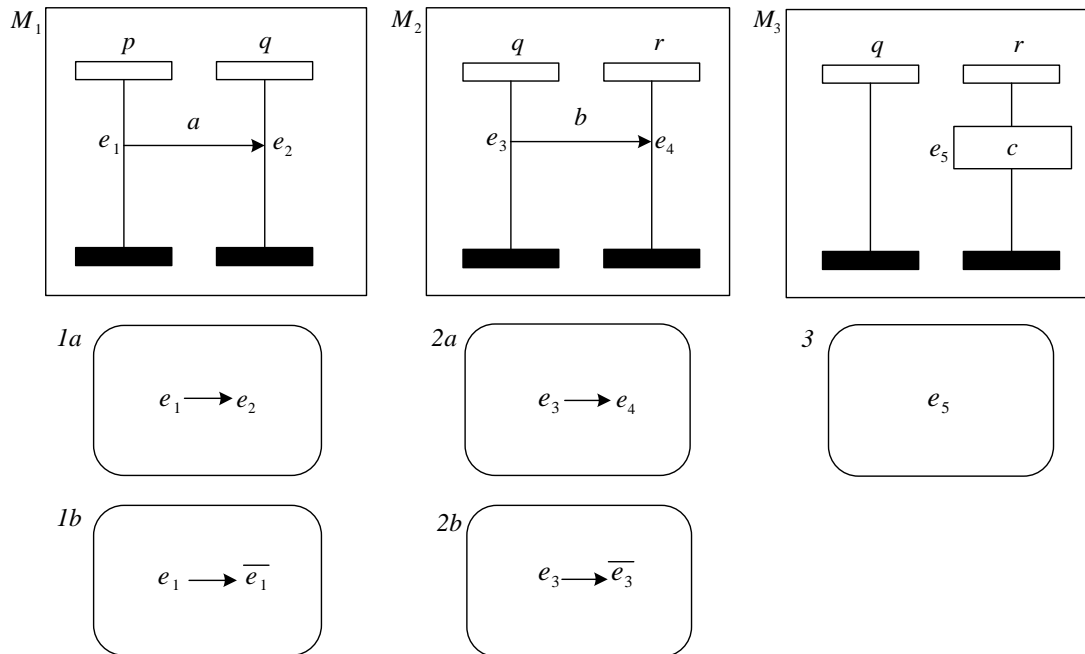


Figure 3.6: Posets to be concatenated

Figure 3.6 shows an example of three pMSCs and their posets. We concatenate poset $1b, 2a$ and 3 and shows that $(1b \cdot 2a) \cdot 3 \neq 1b \cdot (2a \cdot 3)$, as shown in Figure 3.7. As we see in the figure, if we concatenate poset $1b$ and $2a$ first, e_5 will be in the final result because $(1b \cdot 2a)$ eliminates e_3 and e_4 , and since \bar{e}_1 and e_5 do not belong to the same process, e_5 is not eliminated. However, if we concatenate poset $2a$ and 3 first, and then concatenate $1a$ with $(2a \cdot 3)$, e_3 will be eliminated first because of its mutual location with \bar{e}_1 , e_4 will be eliminated because a receive event cannot occur without its corresponding send event, and finally, e_5 will be eliminated because it cannot occur without e_4 occurs first according to $(2a \cdot 3)$.

3.3.2 Concatenation of pMSCs

Every edge in an MSG represent the natural operation of MSC concatenation. Here we describe how we can concatenate pMSCs to be later used to describe the language of our probabilistic extension of MSG.

Let $M_i = (\mathcal{P}_i, \Sigma_i, Act_i, E_i, l_i, m_i, m_{\bar{i}}, Posets_i, C_i, p_{loss_i}), i \in \{1, 2\}$ be pMSCs with $E_1 \cap E_2 = \emptyset$. When we concatenate M_1 with M_2 , it means that we need to concatenate two sets of posets, $Posets_1$ with $Posets_2$. We also need to take care other components of the resulting pMSC beside the set of posets.

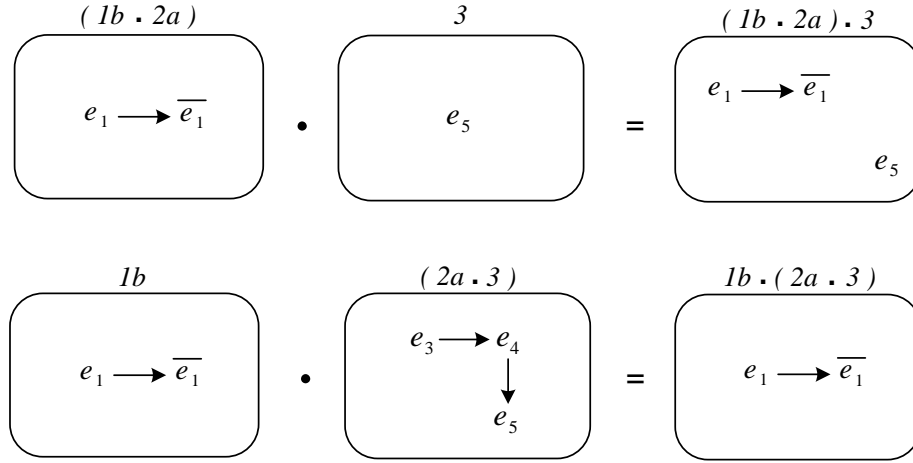


Figure 3.7: Concatenation of posets is not associative

Definition 3.3.2 *Concatenation of pMSCs*

The concatenation of two pMSCs M_1 and M_2 is the pMSC $M_1 \cdot M_2 = (\mathcal{P}, \Sigma, Act, E, l, m_?, m_{\ddagger}, Posets, C, p_{loss})$ with:

- $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $E = E_1 \cup E_2$,
- $l(e) = \begin{cases} l_1(e) & \text{if } e \in E_1, \\ l_2(e) & \text{if } e \in E_2, \end{cases}$
- $m_?(e) = \begin{cases} m_{?_1}(e) & \text{if } e \in E_1, \\ m_{?_2}(e) & \text{if } e \in E_2, \end{cases}$
- $m_{\ddagger}(e) = \begin{cases} m_{\ddagger_1}(e) & \text{if } e \in E_1, \\ m_{\ddagger_2}(e) & \text{if } e \in E_2, \end{cases}$
- $Posets = Posets_1 \times Posets_2 = \{Poset \cdot Poset' \mid Poset \in Posets_1 \text{ and } Poset' \in Posets_2\}$, which is the union of all concatenations of each $Poset \in Posets_1$ with each $Poset' \in Posets_2$,
- $C = C_1 \cup C_2 \cup \{c_{pq}, c_{qp} \mid p \in P_1 \setminus P_2, q \in P_2 \setminus P_1\}$,
- p_{loss} is the probability attached to each channel such that $p_{loss}(c_{pq}) = p_{loss_i}(c_{pq})$ if $p, q \in P_i, i \in \{1, 2\}$.

□

One important assumption when we concatenate two pMSCs M_1 and M_2 is we assume that both pMSCs are describing the same system such that when identical variable names are used in both pMSCs, they are representing the same object in the system. For example, if a process p belongs to both \mathcal{P}_1 and \mathcal{P}_2 then $p \in \mathcal{P}_1$ and $p \in \mathcal{P}_2$ are representing the same process/component of the system. When it is not the case then the process needs to be renamed. The implication of this assumption is when we declare a loss probability for a channel in C_1 , $p_{loss_1}(c_{pq})$, if $c_{pq} \in C_2$ then we should have $p_{loss_1}(c_{pq}) = p_{loss_2}(c_{pq})$ for any $p, q \in \mathcal{P}_1 \cup \mathcal{P}_2$ because c_{pq} in C_1 should represent the same channel as c_{pq} in C_2 .

Example 3.3.3 Concatenation of pMSCs

Figure 3.8 depicts $M_1 \cdot M_2$ from the pMSCs in Figure 3.4 with its set of posets.

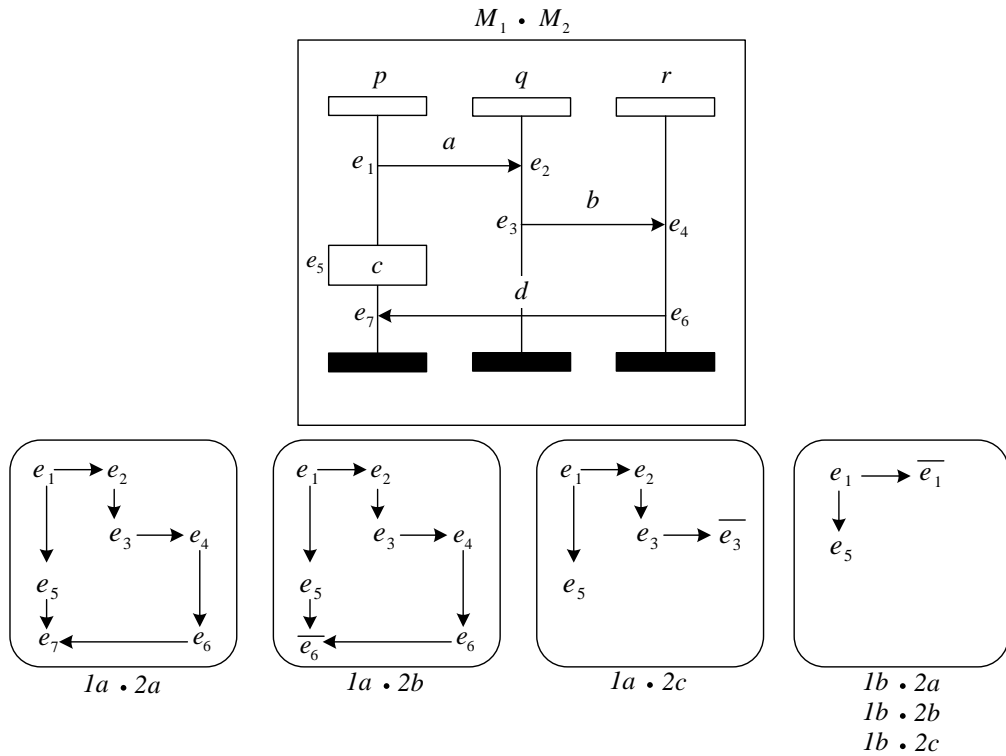


Figure 3.8: Concatenation of M_1 and M_2

□

3.4 Probabilistic Message Sequence Graphs

In this section we introduce probabilistic Message Sequence Graphs as a mechanism to combine pMSCs.

3.4.1 Syntax

Probabilistic Message Sequence Graph (pMSG) is a special structure of Markov decision process where each state represents a probabilistic Message Sequence Charts (pMSC). We define the formal syntax of the model below.

Definition 3.4.1 *Probabilistic Message Sequence Graph (pMSG)*

A probabilistic Message Sequence Graph (pMSG) is a structure $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ where:

- V is a finite set of vertices,
- $Distr$ is a finite set of distributions on V ,
- $\mathbf{P}: V \times Distr \times V \rightarrow [0, 1]$ is the transition probability function such that for all vertices $v \in V$ and distribution $\mu \in Distr$:

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

- $v_{init} \in V$ is initial vertex,
- $F \subseteq V$ is a set of final vertices, i.e., $v \in V$ with $Distr(v) = \emptyset$,
- \mathfrak{M} is a finite set of pMSCs,
- $L: V \rightarrow \mathfrak{M}$ is a mapping between each vertex $v \in V$ to a pMSC $M \in \mathfrak{M}$.

A distribution μ is *enabled* in vertex v if and only if $\sum_{v' \in V} \mathbf{P}(v, \mu, v') = 1$. $Distr(v)$ denotes the set of enabled distributions in v . Let $\mu_v(v')$ denotes $\mathbf{P}(v, \mu, v')$, which is the probability of going from v to v' under distribution μ . $Post(v, \mu)$ denotes the set of successor vertices of v under μ , that is, $Post(v, \mu) = \{v' \in V \mid \mu_v(v') > 0\}$. If v' is the only successor of v under μ , i.e., $\mu_v(v') = 1$, then μ is a unique distribution and we write the distribution as $\mu^{v':1}$.

□

Drawing conventions. We adopt the following drawing conventions. A vertex v can be drawn with one of this following ways:

- as a circle with the name of vertex written inside the circle when the pMSC $L(v)$ is known or drawn somewhere else,

- as a box with the pMSC $L(v)$ drawn inside the box.

We indicate initial vertex with ∇ and final vertices with \triangle . We use directed solid lines to indicate outgoing edges from a vertex v to distributions $\mu \in Distr(v) \setminus \{\mu^{v':1}, v' \in V\}$. The solid line ends in a filled circle which represents the probabilistic choice. If we have a unique distribution $\mu^{v':1} \in Distr(v)$, we use the solid line to connect v and v' directly. We use directed dotted lines to indicate the probabilistic choice from the filled circle to all states supporting the distribution, i.e., $v' \in V$ with $\mu_v(v') > 0$.

Example 3.4.1 *pMSG*

Figure 3.9 depicts a pMSG $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, \lambda)$ with $V = \{v_1, v_2, v_3, v_4\}$. We have $Distr(v_1) = \{\mu, \mu^{v_2:1}\}$ with $\mu_{v_1}(v_3) = 0.2$ and $\mu_{v_1}(v_4) = 0.8$, $Distr(v_3) = \{\mu^{v_1:1}\}$, and $Distr(v_i) = \emptyset$ for $i = 2, 4$. v_1 is initial vertex and $F = \{v_2, v_4\}$ is the set of final vertices. For a set of pMSCs $M = \{M_1, M_2, M_3, M_4\}$, we have $L(v_i) = M_i$.

When system is in v_1 , first it will make a nondeterministic choice between μ and $\mu^{v_2:1}$. If μ is chosen then the next vertex will be v_3 with probability 0.2 or v_4 with probability 0.8. Otherwise, the next vertex will be v_2 with probability 1.

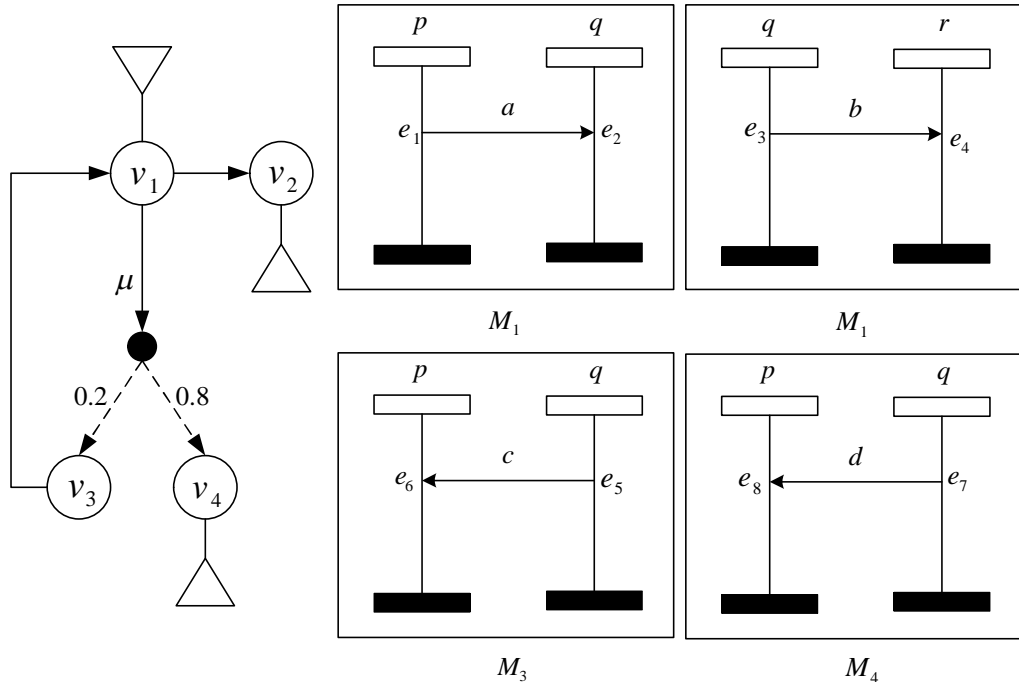


Figure 3.9: pMSG example

□

The Underlying MDP of a pMSG . A pMSG is basically an MDP with a special treatment for the initial distribution and the final vertices. We will relate the definition of pMSG with the original definition of MDP.

Let $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ be a pMSG and $\mathcal{M}_G = (S, Act, \mathbf{P}, i_{init})$ the underlying MDP of G . $V, Distr, \mathbf{P}$ of G correspond to S, Act, \mathbf{P} of \mathcal{M}_G respectively, with additional states and distributions which are going to be discussed later. We require that G has only one initial state denoted by v_{init} . In the underlying MDP we define the initial distribution such that for every vertex v , $i_{init}(v) = 1$ iff $v = v_{init}$ and $i_{init}(v) = 0$ otherwise. Since we are not talking about logic, the MDP does not have any atomic proposition and labeling function.

The last difference between a pMSG and an MDP is that a pMSG has a set of final vertices with no outgoing transition while MDP requires every state to have probability distribution and therefore, outgoing transitions. In the definition of MDP the system may have infinitely many transitions because it has no final states, but for pMSG we want the system to have finite behaviors such that every accepting path ends in one of the final vertices of the system. When we need to represent the pMSG as its underlying MDP, we need a translation mechanism to make sure that every state in the MDP has outgoing transitions without changing the finite behavior of the system. We solve this problem by adding a new state in \mathcal{M}_G for each final vertex of G . The set of states S of \mathcal{M}_G is defined as $V \cup S_F$ where $S_F = \{v' | v \in F\}$, i.e., we add a new state v' for each final vertex v . Next, we add a transition from v to v' with probability 1 and a self transition to each v' , also with the probability of 1. The states in S_F have an empty pMSC as the label, that is, pMSC with an empty set of events. Intuitively, after the system finishes an execution of the corresponding pMSG which ends at one of its final vertices, it will go to a new state where nothing will be executed. The system will stay in this state forever because it has a self transition as the only outgoing transition, thus making this state as the “final state” of the system after one execution of the pMSG.

Example 3.4.2 *Underlying MDP of a pMSG*

Figure 3.10 below depicts the underlying MDP of pMSG G in Figure 3.9.

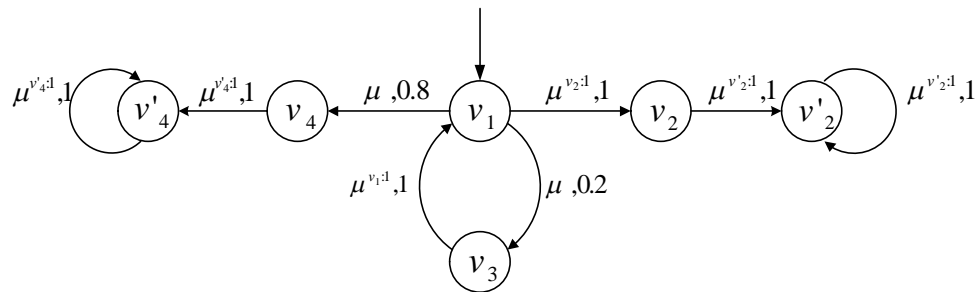


Figure 3.10: The underlying MDP of pMSG in Figure 3.9

□

System behavior. The system specified as a pMSG evolves as its underlying MDP. When the system is in a vertex v , it will first make a nondeterministic choice of $\mu \in Distr(v)$, then a probabilistic choice to go to the next vertex v' such that $\mu_v(v') > 0$.

3.4.2 Language of a pMSG

Let $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ be a pMSG.

A path of G describes one execution sequence of made by resolving nondeterministic and probabilistic choices.

Definition 3.4.2 Path of pMSG

A path of G is a sequence $\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} \dots$ with $v_i \in V$, $\mu_i \in Distr(v_i)$, $\mu_{i(v_i)}(v_{i+1}) > 0$ for $i = 0, 1, 2, \dots$

When a path ends in a vertex then it is a *finite path*. The first element of a finite path π is denoted by $first(\pi)$ and the last element is denoted by $last(\pi)$. A path π is *accepting* iff it is finite, $first(\pi) = v_{init}$ and $last(\pi) \in F$.

□

Example 3.4.3 Paths of a pMSG

For pMSG depicted in Figure 3.9, we have $\pi_1 = v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \xrightarrow{\mu} v_4$ and $\pi_2 = v_1 \xrightarrow{\mu^{v_2:1}} v_2$ as both accepting paths. $\pi_3 = v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} \dots \xrightarrow{\mu} v_3$ which describes a finite loop between v_1 and v_3 , is a finite but not accepting path because $v_3 \notin F$. $\pi_4 = v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} \dots \in (v_1 \xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}})^\omega$ is an infinite path and is not accepting.

□

There is a one-to-one correspondence between paths of G and the paths in the underlying MDP \mathcal{M}_G . If π is an accepting path of G ,

$$\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_{n-1}} v_n$$

such that $v_n = last(\pi)$ is an accepting vertex, the corresponding path of the MDP \mathcal{M}_G is given by

$$\pi_{\mathcal{M}_G} = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_{n-1}} v_n (\xrightarrow{\mu^{v'_n:1}} v'_n)^\omega$$

Otherwise, the corresponding MDP path is exactly the same as π .

Vice versa, if a path $\pi_{\mathcal{M}_G}$ is a path of \mathcal{M}_G and it has the following structure:

$$\pi_{\mathcal{M}_G} = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_{n-1}} v_n \left(\xrightarrow{\mu^{v'_n:1}} v'_n \right)^\omega$$

where v_n is an accepting vertex in the corresponding pMSG G , then the corresponding path in G is given by

$$\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_{n-1}} v_n$$

Otherwise, the corresponding pMSG path is exactly the same as $\pi_{\mathcal{M}_G}$.

Since every vertex in G represents a pMSC, we can associate a pMSC with each path by concatenating pMSCs corresponding to individual vertices in the path. We call this pMSC the *pMSC of a path*.

Definition 3.4.3 *pMSC of a path*

Let $\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} \dots \xrightarrow{\mu_{n-1}} v_n$ be a path of pMSG G , the pMSC of π is given as:

$$M(\pi) = L(v_0) \cdot L(v_1) \cdot \dots \cdot L(v_n) = \prod_{i=0}^n L(v_i)$$

□

Definition 3.4.4 *pMSC language of a pMSG*

The *pMSC language* accepted by pMSG is defined as the set of pMSCs associated with all accepting paths of the pMSG,

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

For any pMSC M , we say that G generates M if $M \in \mathcal{L}(G)$.

□

Definition 3.4.5 *Word language of a pMSG*

The *word language* of G is the union of the linearizations of its pMSC language,

$$\begin{aligned} \text{Lin}(\mathcal{L}(G)) &= \text{Lin}(\{M(\pi) | \pi \text{ is an accepting path of } G\}) \\ &= \bigcup_{\pi \in \text{the set of accepting paths of } G} \text{Lin}(M(\pi)) \end{aligned}$$

□

Example 3.4.4 *pMSC language of pMSG*

In Example 3.4.3, we have two accepting paths, $\pi_1 = v_1 \left(\xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \right)^* \xrightarrow{\mu} v_4$ and $\pi_2 = v_1 \left(\xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \right)^* \xrightarrow{\mu^{v_2:1}} v_2$ where $\left(\xrightarrow{\mu} v_3 \xrightarrow{\mu^{v_1:1}} v_1 \right)^*$ describes finite loops (possibly none) between v_3 and v_1 . We have the pMSCs of the paths as follow:

$$\begin{aligned} M(\pi_1) &= M_1 \cdot (M_3 \cdot M_1)^* \cdot M_4 \\ M(\pi_2) &= M_1 \cdot (M_3 \cdot M_1)^* \cdot M_2 \end{aligned}$$

The pMSC language accepted by G is

$$\mathcal{L}(G) = \{M_1 \cdot (M_3 \cdot M_1)^* \cdot M_4, M_1 \cdot (M_3 \cdot M_1)^* \cdot M_2\}$$

The word language of G is defined as

$$\text{Lin}(\mathcal{L}(G)) = \text{Lin}(M_1 \cdot (M_3 \cdot M_1)^* \cdot M_4) \cup \text{Lin}(M_1 \cdot (M_3 \cdot M_1)^* \cdot M_2)$$

Figure 3.11 depicts two examples of pMSCs which are included in the language of G . Figure 3.11(a) depicts pMSC of the path $\pi = v_1 \cdot (v_3 \cdot v_1)^2 \cdot v_4$ and Figure 3.11(b) depicts pMSC of the path $\pi = v_1 \cdot v_2$.

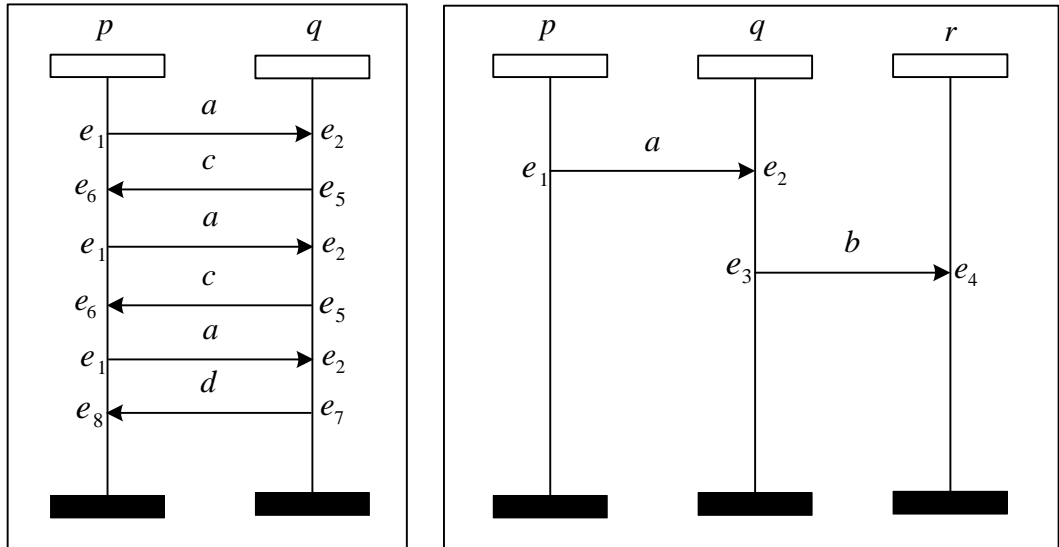


Figure 3.11: Example of pMSCs included in $\mathcal{L}(G)$

□

3.4.3 Adversary

During the execution of a pMSG, a mechanism is needed to resolve the nondeterminism, that is, to choose a distribution for every current vertex to go to the next one. This mechanism is important when we want to reason about probabilities of sets of paths of a pMSG because pMSGs are not augmented with a single probability measure. As in MDPs, the nondeterminism in pMSGs is resolved using an *adversary*. An adversary, also known as a scheduler, strategy or policy, chooses in any vertex v one of the enabled distributions $\mu \in \text{Distr}(v)$. After a distribution is

chosen, the probabilistic choice is resolved by the system itself. The adversary can also choose the distribution probabilistically, but in this work we are focusing on *nonprobabilistic adversary*.

Definition 3.4.6 *Adversary of pMSGs*

Let $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ be a pMSG. An *adversary* for G is a function $A : S \rightarrow Distr$ such that $A(v) \in Distr(v)$ for all $v \in V$. A path $\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} \dots$ is an A -path if $\mu_i = A(s_i)$ for all $i \geq 0$.

□

In the literature, an adversary sometimes takes the path as its input and chooses for the last vertex of the path, $last(\pi)$, a distribution $\mu \in Distr(last(\pi))$, thus making it possible to choose a different distribution depends on the past history. For example, consider an adversary A for the MDP depicted in Figure 3.2 which chooses to perform maintenance in every state if the machine has broken at least once, otherwise it chooses to do nothing. In this work, for the sake of simplicity, our adversary has no memory of the past history. To be precise, we consider a memoryless adversary which is sufficient because the main interest would only be in maximal and minimal reachability probabilities.

Definition 3.4.7 *Memoryless adversary of pMSGs*

Let $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ be a pMSG. A *memoryless adversary* for G is a function $A : S \rightarrow Distr$ such that $A(v) = \mu \in Distr(v)$ for every $v \in V$. In other words, a memoryless adversary is an adversary which always chooses the same distribution $\mu \in Distr(v)$ whenever state v is reached, independent from the past history.

□

For the MDP in Figure 3.2, an adversary which chooses to ignore the condition whenever the machine is in good or deteriorating state and chooses to perform maintenance whenever the machine is broken, is a memoryless adversary.

The behavior of a pMSG G under an adversary A induces a computation tree which can be described as a Markov chain \mathcal{MC} whose states are the states of the underlying MDP of G , \mathcal{M}_G , and the transition probability is given by the distributions selected by A .

Definition 3.4.8 *Markov chain of a pMSG*

Let $G = (V, Distr, \mathbf{P}, v_{init}, F, \mathfrak{M}, L)$ be a pMSG, $\mathcal{M}_G = (S, Act, \mathbf{P}, i_{init})$ the underlying MDP of G and A an adversary of G . The Markov chain induced by A is given by

$$\mathcal{MC}_A = (S, \mathbf{P}_A, i_{init})$$

where for $s, s' \in S$,

$$\mathbf{P}_A(s, s') = \mathbf{P}(s, A(s), s'),$$

and $i_{init}(s) = 1$ iff $s = v_{init}$ and $i_{init}(s) = 0$ otherwise.

□

There is a one-to-one correspondence between A -paths of G and the paths in the Markov chain \mathcal{MC}_A . For an A -path

$$\pi = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots$$

and the corresponding MDP path of $\pi_{\mathcal{M}_G}$

$$\pi_{\mathcal{M}_G} = v_0 \xrightarrow{\mu_0} v_1 \xrightarrow{\mu_1} v_2 \xrightarrow{\mu_2} \dots$$

the corresponding path in the Markov chain \mathcal{MC}_A is given as the sequence of vertices of the corresponding MDP path of π ,

$$\pi_A = v_0 v_1 v_2 \dots$$

Note that if π is an accepting path with $last(\pi) = v_n$ then the corresponding path in the Markov chain will have the following form:

$$\pi_A = v_0 v_1 v_2 \dots v_n (v'_n)^\omega$$

Vice versa, if $\pi_A = v_0 v_1 v_2 \dots$ is a path of \mathcal{MC}_A then the corresponding path of G is given by

$$\pi = v_0 \xrightarrow{A(v_0)} v_1 \xrightarrow{A(v_1)} v_2 \xrightarrow{A(v_2)} \dots$$

such that $\mathbf{P}(v_i, A(v_i), v_{i+1}) > 0$ for $i \geq 0$. If π_A has the form of $\pi_A = v_0 v_1 v_2 \dots v_n (v'_n)^\omega$ with v_n a final vertex of G then the corresponding path of G is given by

$$\pi = v_0 \xrightarrow{A(v_0)} v_1 \xrightarrow{A(v_1)} v_2 \xrightarrow{A(v_2)} \dots \xrightarrow{A(v_{n-1})} v_n$$

Example 3.4.5 *Markov chain induced by an adversary*

Consider the pMSG from Example 3.4.1. Adversary A_1 always selects distribution μ in state v_1 and A_2 always selects distribution $\mu^{v_2:1}$ in state v_1 . Note that the other states have only one distribution to choose from.

The Markov chain \mathcal{MC}_{A_2} is the finite chain given by Figure 3.12.

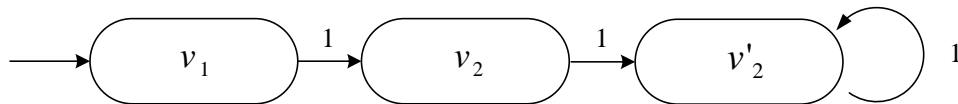
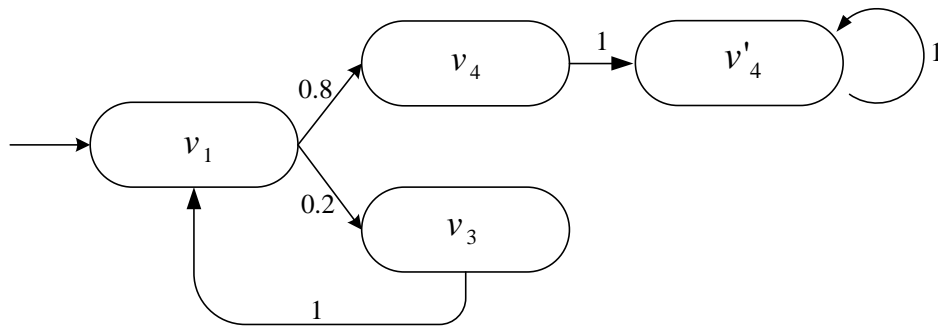


Figure 3.12: Markov chain \mathcal{MC}_{A_2}

And the Markov chain \mathcal{MC}_{A_1} is given by Figure 3.13.

Figure 3.13: Markov chain \mathcal{MC}_{A_1}

□

Since an adversary A induces a Markov chain \mathcal{MC}_A from a pMSG G , we can now reason about the probabilities of sets of A -paths. Let $Pr_{\mathcal{MC}_A}$, or simply Pr_A , denote the probability measure associated with \mathcal{MC}_A .

Let $\pi = v_0 \xrightarrow{A(v_0)} v_1 \xrightarrow{A(v_1)} \dots \xrightarrow{A(v_{n-1})} v_n$ be an A -path of pMSG G and $\pi_A = v_0 v_1 v_2 \dots v_m, m \geq n$ be its corresponding path of \mathcal{MC}_A . The probability of path π given the adversary A is defined as:

$$Pr_A(\pi) = \prod_{i=0}^{n-1} A(v_i).$$

Let $M(\pi)$ be the pMSC of path and $Lin(M(\pi))$ the set of linearizations of $M(\pi)$. The probability of $Lin(M(\pi))$ given the path π and the scheduler A is given as the probability of the set of linearizations times the probability of the path,

$$Pr_A(Lin(M(\pi))) = Pr(Lin(M(\pi))) \times Pr_A(\pi).$$

Remember that the probability of a set of linearizations is defined by the probability of its posets.

Example 3.4.6 *Probability of linearization of a path*

We see again the Markov chain \mathcal{MC}_{A_1} induced by adversary A_1 in Figure 3.13. The corresponding pMSG with the set of pMSCs is given in Figure 3.9. We consider a finite path of \mathcal{MC}_{A_1} , $\pi_{A_1} = v_1 v_3 v_1 v_4 \in Paths_{fin}(\mathcal{MC}_{A_1})$. The corresponding pMSG path is

$$\pi = v_1 \xrightarrow{0.2} v_3 \xrightarrow{1} v_1 \xrightarrow{0.8} v_4$$

and the pMSC of π along with its posets is given in Figure 3.14.

The probability of π given scheduler A_1 is

$$Pr_{A_1}(\pi) = 0.2 \times 1 \times 0.8 = 0.16.$$

We specify the probability of message loss for the channels $p_{loss}(c_{pq}) = p_{loss}(c_{qp}) = 0.2$. Let us first compute the probability that all messages are delivered. This scenario is depicted by the 5-th poset, which is the maximal poset for the pMSC $M(\pi)$. The probability of the set of linearizations of the poset is given as

$$\begin{aligned} Pr(Lin_{M(\pi)}(E_5, <_5)) &= Pr(E_5) \\ &= Pr(e_1)^2 \times Pr(e_2)^2 \times Pr(e_5) \times Pr(e_6) \times Pr(e_7) \times Pr(e_8) \\ &= 1^2 \times 0.8^2 \times 1 \times 0.8 \times 1 \times 0.8 \\ &= 0.512 \end{aligned}$$

Finally, we can compute the probability that all messages are delivered when the path π is executed given scheduler A_1 as follows:

$$\begin{aligned} Pr_{A_1}(Lin_{M(\pi)}(E_5, <_5)) &= Pr(Lin_{M(\pi)}(E_5, <_5)) \times Pr_{A_1}(\pi) \\ &= 0.512 \times 0.16 \\ &= 0.08192 \end{aligned}$$

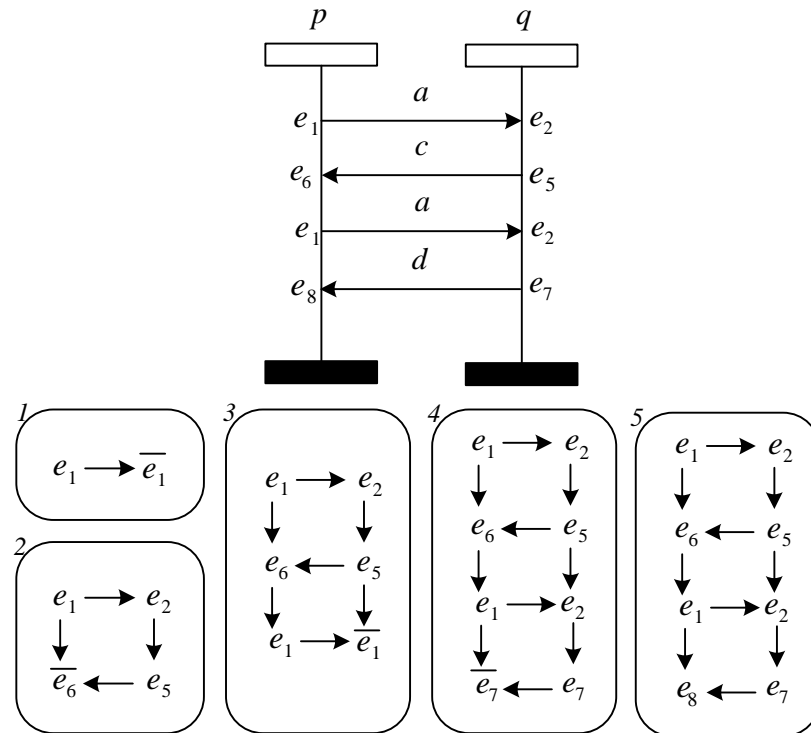


Figure 3.14: pMSC of π

3.5 Case Study : Asynchronous Leader Election Protocol

We consider the asynchronous leader election protocol of Itai and Rodeh [23] as an example of how pMSG can be applied to model protocols for distributed system. The protocol considers N processes in a network with unidirectional ring topology and the goal is to elect a leader (a uniquely designated process) by sending messages around the ring. Process i is connected to its neighbors by FIFO channels, c_i to send messages to process $i + 1$ and c_{i-1} to receive message from process $i - 1$, with addition considered as modula of the ring size N .

The protocol begins with all processes being *active*. In the active mode process i chooses a bit, 0 or 1 each with probability 0.5, and sends this choice to its successor. Process i receives the bit chosen by process $i - 1$. If the bit of process i is 1 and the bit it receives is zero then process i switches to the *passive* mode. Otherwise it stays active and participates in the next round, which is started by every active process choosing a new bit. In the passive mode, process i just acts as a relay node, passing the bit from process $i - 1$ to $i + 1$. When there is only one active process left, this process becomes the leader. This protocol assumes that all channels are reliable and all messages sent will be received.

Figure 3.15 represents the the initial part of the protocol for a network with $N = 3$ as a pMSG. We have 2^3 possible bit combinations chosen by 3 processes, each with the same probability 0.125. Internal action labeled $p(x)$ denotes random bit choice by process $p \in \{p, q, r\}$ with $x \in \{0, 1\}$. We use coregions after the internal events because we do not specify which process sends its bit first or which bit is received first. This will be chosen nondeterministically during the run time, depends on which linearization will be executed. We can see that if all three processes choose the same bit then no process becomes passive while in the other cases one process which receives bit 0 and chooses bit 1 becomes passive.

Figure 3.16 depicts the next part of vertex v_2 , which is the next round of the case where process q becomes passive because it receives bit 0 and chooses bit 1 in the previous round. Process p and r stay active. The active processes then start the second round by making new bit choices. We can see that q as the passive process does not have coregion because it can only relay the message from p to r so it has to wait for the message from p to arrive before passing it to r . The protocol terminates when one of the processes becomes the leader while the others are passive.

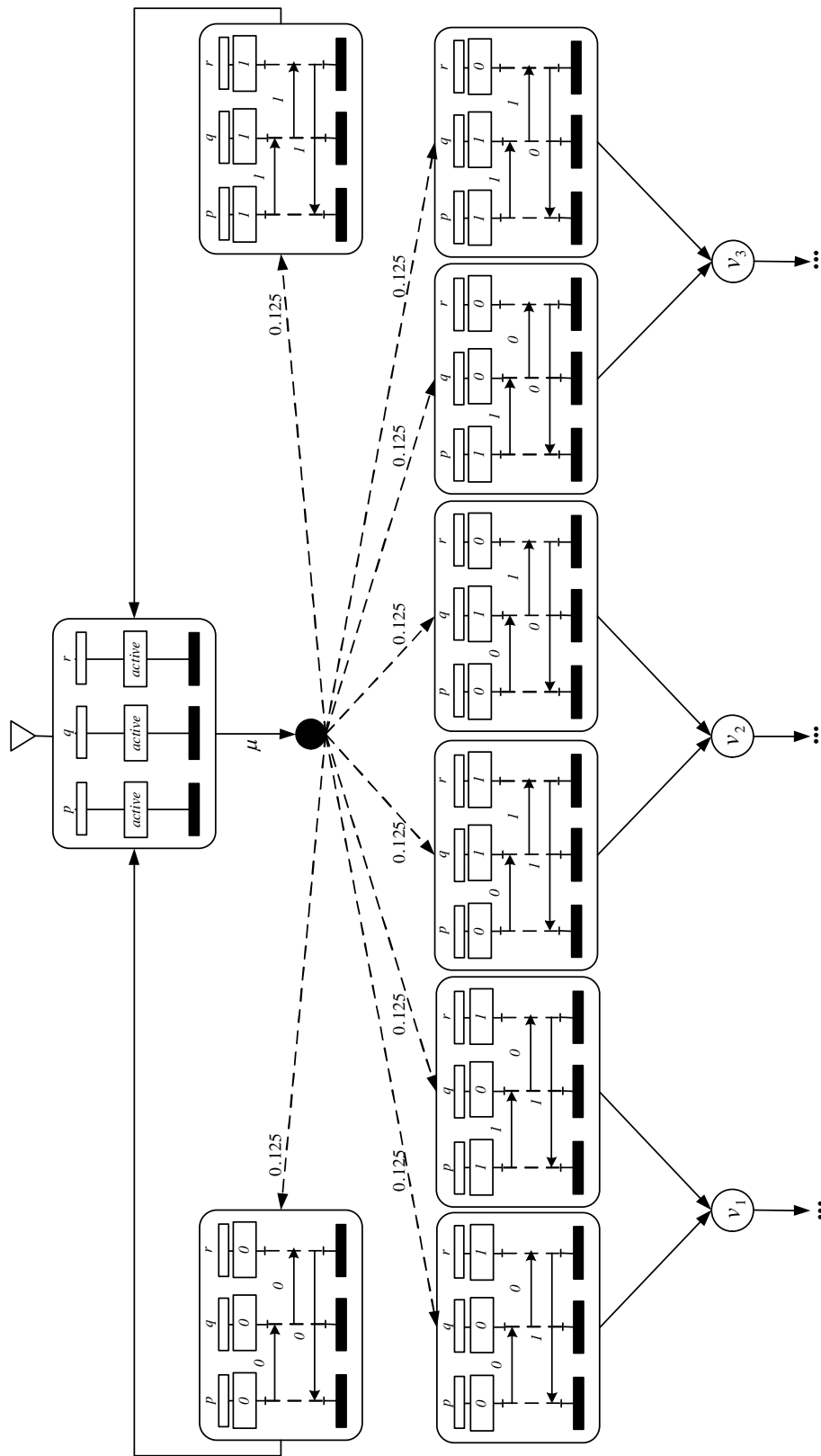
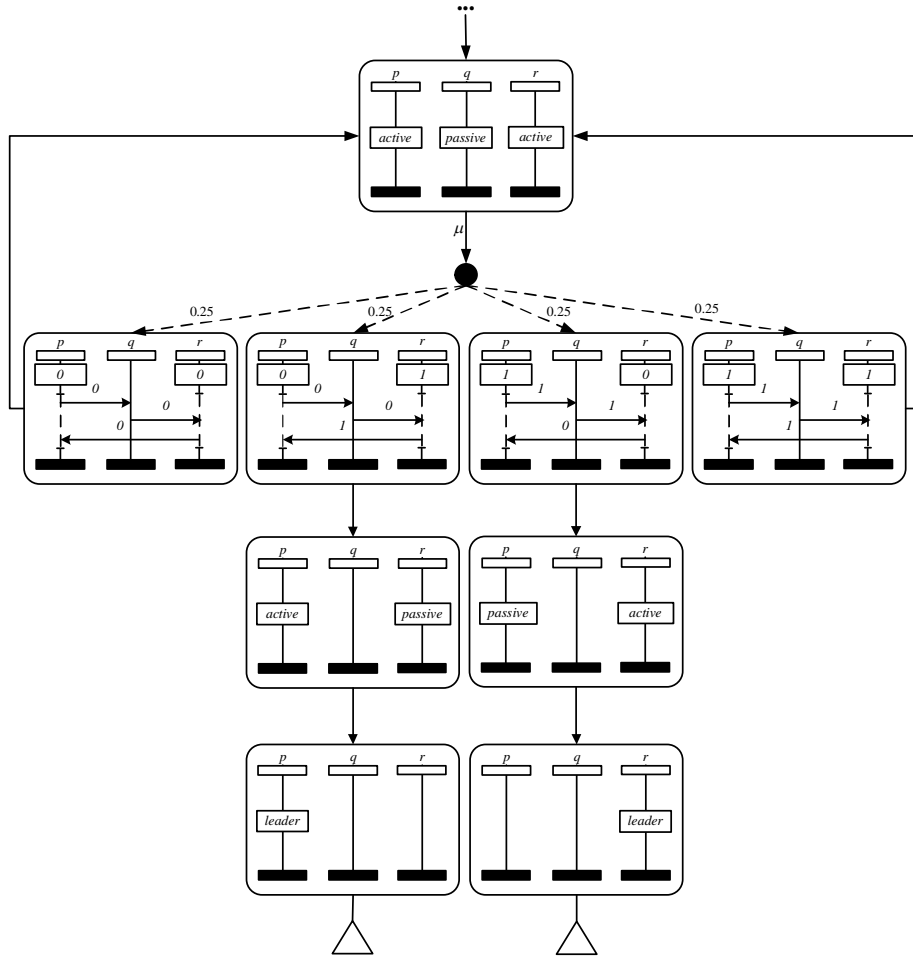


Figure 3.15: pMSG of Asynchronous Leader Election Protocol


 Figure 3.16: The next part from vertex v_2 of Figure 3.15

In this example, every vertex of the pMSG only has one probability distribution so it can be represented as a Markov chain. The nondeterminism is specified, as mentioned before, in the linearization level of the pMSCs where the system has to choose which process should send its message first. Let us take a closer look to one of the vertex of the pMSG and its *labeled poset* given in Figure 3.17. A labeled poset is a poset where the events are represented as its action label. Some of its possible linearizations are given below:

$$\begin{aligned}
 w_1 &= p(0).q(0).r(1).p!q(0).q?p(0).q!r(0).r?q(0).r!p(1).p?r(1) \\
 w_2 &= p(0).q(0).r(1).q!r(0).r?q(0).p!q(0).q?p(0).r!p(1).p?r(1) \\
 w_3 &= q(0).q!r(0).r(1).r?q(0).r!p(1).p(0).p?r(1).p!q(0).q?p(0)
 \end{aligned}$$

In w_1 , the processes first choose the bits then send and receive the bits with the order: $p - q - r$. In w_2 process q starts sending its bit first and in w_3 we can see that the internal actions do not have to be all executed in the beginning and can be executed only when needed.

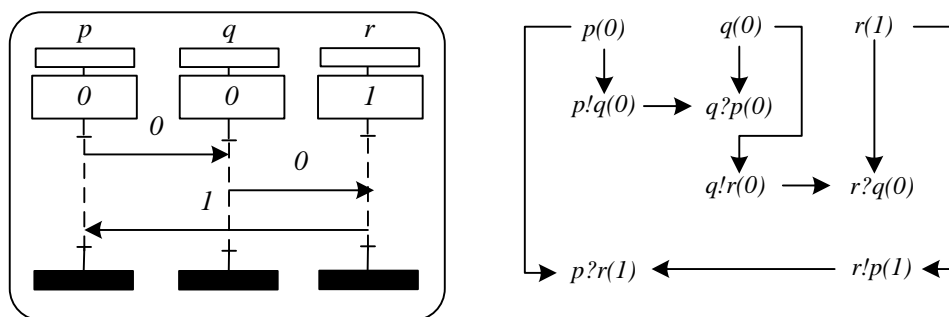


Figure 3.17: pMSC of a vertex and its labeled poset

Chapter 4

Properties of Message Sequence Charts

4.1 Introduction

We would like to be able to specify properties of pMSCs and pMSGs with suitable logics. As mentioned in the first chapters, various logics have been developed to express properties of MSCs. Some logics are defined over linearizations [5], while some others are defined over the structure of the MSCs [11, 33, 37]. We prefer the latter type of logic because it offers a more direct approach to specify the properties based on how the models are defined.

The logic defined for pMSCs should be able to express properties of events as well as paths imposed by the partial orders. It should also incorporate probabilistic notions to reason about the probability of events and posets. We would like to be able for example, to express the following properties:

- the probability that a message sent by a send event will be received in the next event is more than 0.95,
- the system will eventually terminate (or a termination event will eventually occur),
- the number of requests send by a client to the server does not exceed 3.

The logic for pMSC presented in this chapter is an extension of Propositional Dynamic Logic (PDL) for message-passing systems [11]. In this work we will only discuss the logic defined to specify properties of pMSCs, which we call the probabilistic PDL (PPDL). We do not provide procedures to model check pMSCs against PPDL formulas and leave it for future work.

We start by giving the definitions of the original PDL for MSCs and then present our logic, PPDL, to specify properties of pMSCs.

4.2 Propositional Dynamic Logic for MSCs

Propositional Dynamic Logic for MSCs [11] is interpreted directly over the structure of the MSCs and not over linearizations. It combines elements from different logics defined for MSCs. The past operators which are provided to reason about events which have already been executed, are inspired by the ones defined for the logic on Lamport diagrams [34], and the existential interpretation of the until operator from the logic TLC^- [37]. The logic is also inspired by dynamic LTL [21], an extension of LTL for traces which extends the standard LTL by indexing the until operator with a regular expression to make it more flexible. Different from dynamic LTL, the path expressions in PDL are not bound to the events of a single process, but can also move from one process to another.

PDL defines the properties of MSCs in two stages: at the level of events and at global level. Local formulae are defined by using path expressions. Local formulae express properties of single events in MSCs and path expressions express properties of paths. Global formulae express global properties of the MSC by expressing whether or not there exists an event which satisfies certain event formulae. The formulae are defined over the action labels of the events in the MSCs.

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m, <)$ be a basic MSC.

Definition 4.2.1 Syntax of PDL

The PDL *local formulae* are formed according to the following grammar:

$$\Phi ::= true \mid a \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \langle\varphi\rangle\Phi \mid \langle\varphi\rangle^{-1}\Phi$$

where $a \in Act$ and φ is a path expression.

The *path expressions* are formed according to the following grammar:

$$\varphi ::= proc \mid msg \mid \{\Phi\} \mid \varphi_1; \varphi_2 \mid \varphi_1 + \varphi_2 \mid \varphi^*$$

where Φ is a local formula and φ_1, φ_2 are path expressions. Intuitively, the *proc* operator refers to the immediate successor belonging to the same process as the current event, *msg* refers to the matching receive event if the current event is a send event and $\{\Phi\}$ provides a means to reason only about the current event in the path.

PDL *global formulae* have the following syntax:

$$\gamma ::= \exists\Phi \mid \forall\Phi \mid \gamma_1 \vee \gamma_2 \mid \gamma_1 \wedge \gamma_2$$

where Φ is local formula and γ_1, γ_2 are global formulae.

□

For local formulae, other Boolean connectives such as conjunction \wedge , implication \rightarrow , equivalence \leftrightarrow , and exclusive or \oplus can be derived using the Boolean connectives \vee and \neg . The until operator U can be derived as follows:

$$\Phi_1 \text{U} \Phi_2 = \langle (\{\Phi_1\}; (proc + msg))^* \rangle \Phi_2$$

The local formula $\Phi_1 \mathbf{U} \Phi_2$ holds if there is a future event for which Φ_2 holds and all events until that future event satisfy Φ_1 . The operator is derived by using the combination of several expressions. The expression $\{\Phi_1\}; (proc + msg)$ checks if the current event satisfies Φ_1 then moves to the next event, which can be either the immediate successor of the current event at the same process or its matching receive event. We use the iterative operator $(\{\Phi_1\}; (proc + msg))^*$ to do this iteratively with unbounded number of times until we reach an event in which Φ_2 holds, $\langle (\{\Phi_1\}; (proc + msg))^* \rangle \Phi_2$.

The eventually operator \diamond can be derived similarly as follows:

$$\diamond \Phi = true \mathbf{U} \Phi = \langle (\{true\}; (proc + msg))^* \rangle \Phi$$

and the always operator \square as

$$\square \Phi = \neg \diamond \neg \Phi$$

Definition 4.2.2 *Satisfaction relation for PDL*

Let $e \in E$ be an event of MSC M . The satisfaction relation \models is defined for local formulae by

$$\begin{aligned} e \models a & \quad \text{iff } a = l(e) \\ e \models \Phi_1 \vee \Phi_2 & \quad \text{iff } e \models \Phi_1 \text{ or } e \models \Phi_2 \\ e \models \neg \Phi & \quad \text{iff } \text{not } e \models \Phi \end{aligned}$$

The semantics of *forward*-path expressions $\langle \varphi \rangle \Phi$ of the event e is given by

$$\begin{aligned} e \models \langle proc \rangle \Phi & \quad \text{iff } \exists e' \in E'. e \leq_p e' \text{ for some } p \in \mathcal{P} \text{ and } e' \models \Phi \\ e \models \langle msg \rangle \Phi & \quad \text{iff } \exists e' \in E'. e' = m_{\tau}(e) \text{ and } e' \models \Phi \\ e \models \langle \{\Phi\} \rangle \Phi' & \quad \text{iff } e \models \Phi \text{ and } e \models \Phi' \\ e \models \langle \varphi_1; \varphi_2 \rangle \Phi & \quad \text{iff } e \models \langle \varphi_1 \rangle \langle \varphi_2 \rangle \Phi \\ e \models \langle \varphi_1 + \varphi_2 \rangle \Phi & \quad \text{iff } e \models \langle \varphi_1 \rangle \Phi \text{ or } e \models \langle \varphi_2 \rangle \Phi \\ e \models \langle \varphi^* \rangle \Phi & \quad \text{iff } \exists j \geq 0. e \models (\langle \varphi \rangle)^j \Phi \end{aligned}$$

where \leq is a binary relation such that $e \leq e'$ iff $loc(e) = loc(e')$, $e \leq_{loc(e)} e'$ and $\forall e'' \in E. loc(e'') = loc(e) \wedge e \leq_{loc(e)} e'' \leq_{loc(e)} e'$ we have $e = e''$. Intuitively, $e \leq e'$ means that e' is the immediate successor of e in process $loc(e)$. The formula $(\langle \varphi \rangle)^j \Phi$ stands for repeating j times the expression $\langle \varphi \rangle$ before Φ , $\langle \varphi \rangle_1 \langle \varphi \rangle_2 \dots \langle \varphi \rangle_j \Phi$.

The semantics of *backward*-path expressions $\langle \varphi \rangle^{-1} \Phi$ of the event e is given by:

$$\begin{aligned} e \models \langle proc \rangle^{-1} \Phi & \quad \text{iff } \exists e' \in E. e' \succ_p e \text{ for some } p \in \mathcal{P} \text{ and } e' \models \Phi \\ e \models \langle msg \rangle^{-1} \Phi & \quad \text{iff } \exists e' \in E. e' = m_{\tau}^{-1}(e) \text{ and } e' \models \Phi \\ e \models \langle \{\Phi\} \rangle^{-1} \Phi' & \quad \text{iff } e \models \Phi \text{ and } e \models \Phi' \\ e \models \langle \varphi_1; \varphi_2 \rangle^{-1} \Phi & \quad \text{iff } e \models \langle \varphi_1 \rangle^{-1} \langle \varphi_2 \rangle^{-1} \Phi \\ e \models \langle \varphi_1 + \varphi_2 \rangle^{-1} \Phi & \quad \text{iff } e \models \langle \varphi \rangle^{-1} \Phi \text{ or } e \models \langle \varphi_2 \rangle^{-1} \Phi \\ e \models \langle \varphi^* \rangle^{-1} \Phi & \quad \text{iff } \exists j \geq 0. e \models (\langle \varphi \rangle^{-1})^j \Phi \end{aligned}$$

The semantics of global formulae is as follows:

PDL model checking. The model checking problem considered in [11] is to decide, given an MSC M and a PDL global formula γ , whether M satisfies γ . The model checking is done by constructing a communicating finite-state machine (CFM) \mathcal{A}_γ for each PDL global formula γ . The decision procedure is done with an inductive method. The events of the MSC are colored with an additional bit for each subformula of γ . A CFM is constructed for each of these subformulas whose task is to check whether the the bit corresponding to the subformula is set at those events where the subformula holds. The overall CFM \mathcal{A}_γ is obtained by running synchronously all the CFMs built from the subformulas. The model checking problem for CFMs against PDL formulae is shown to be in PSPACE for existentially bounded MSCs, that is, MSCs in which the number of messages which can be contained in the channels are bounded with a finite number.

4.3 Probabilistic Propositional Dynamic Logic for pMSCs

Probabilistic PDL (PPDL) is an extension of PDL tailored to pMSCs. We extend PDL by incorporating probabilistic operators $\mathbb{P}_J\langle\varphi\rangle\Phi$ and $\mathbb{P}_J\langle\varphi\rangle^{-1}\Phi$ where φ is a path expression, Φ is a local formula and J is an interval of $[0,1]$ which indicates a lower and/or upper bound on the probability. The meaning of the formula $\mathbb{P}_J\langle\varphi\rangle\Phi$ in an event e is that the probability of e to satisfy the forward-path expression $\langle\varphi\rangle\Phi$ meets the bounds given by J .

Since a pMSC has a set of posets instead of a single one (as MSCs) and that each poset might impose a different path from the others, we want to reason about the likelihood of the events to satisfy the path expressions.

The path expressions φ are defined as for PDL, except that a bounded iterative operator is additionally incorporated. We also remove the existential and universal quantification from the global formulae and reinterpret the meaning of the formulae.

Let $M = (\mathcal{P}, \Sigma, Act, E, l, m_?, m_!, Posets, C, p_{loss})$ be a pMSC.

Definition 4.3.1 *Syntax of PPDL*

PPDL *local formulae* are formed according to the following grammar:

$$\Phi ::= true \mid a \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \mathbb{P}_J\langle\varphi\rangle\Phi \mid \mathbb{P}_J\langle\varphi\rangle^{-1}\Phi$$

where $a \in Act$, φ is a path expression, and $J \subseteq [0,1]$ is an interval with rational bounds.

Rather than writing the intervals explicitly, often abbreviations are used; e.g., $\mathbb{P}_{\leq 0.5}\langle\varphi\rangle\Phi$ denotes $\mathbb{P}_{[0,0.5]}\langle\varphi\rangle\Phi$, $\mathbb{P}_{=0.5}\langle\varphi\rangle\Phi$ denotes $\mathbb{P}_{[0.5,0.5]}\langle\varphi\rangle\Phi$, and $\mathbb{P}_{>0.5}\langle\varphi\rangle\Phi$ stands for $\mathbb{P}_{]0.5,1]}\langle\varphi\rangle\Phi$.

The *path expressions* are formed according to the following grammar:

$$\varphi ::= proc \mid msg \mid \{\Phi\} \mid \varphi_1; \varphi_2 \mid \varphi_1 + \varphi_2 \mid \varphi^* \mid \varphi^{[m,n]}$$

where Φ is an event formula, φ_1, φ_2 are path expressions, and $[m, n]$ with $m, n \in \mathbb{N}$ is a closed interval.

The expression $\varphi^{[m,n]}$ is a bounded version of the unbounded iterative operator φ^* , where the number of the iteration shall meet the interval given by $[m, n]$. Abbreviations are also used for $[m, n]$ interval; e.g., $\varphi^{\leq 5}$ denotes $\varphi^{[5,5]}$ and $\varphi^{\leq 0}$ denotes $\varphi^{[0,5]}$.

PPDL *global formulae* have the following syntax:

$$\gamma ::= \Phi \mid \gamma_1 \vee \gamma_2 \mid \gamma_1 \wedge \gamma_2$$

where Φ is an event formula and γ_1, γ_2 are global formulae.

□

As in PDL, other Boolean connectives can be derived using the Boolean connectives \vee and \neg . The until operator \mathbf{U} and the eventually operator \diamond can be derived as follows:

$$\begin{aligned} \mathbb{P}_J \Phi_1 \mathbf{U} \Phi_2 &= \mathbb{P}_J(\{\{\Phi_1\}; (proc + msg)^*\}) \Phi_2 \\ \mathbb{P}_J \diamond \Phi &= \mathbb{P}_J(\{\{true\}; (proc + msg)^*\}) \Phi \end{aligned}$$

The always operator \square can be derived using the duality of eventually and always and the duality of lower and upper bounds. An event holds with probability at most p if and only if the dual event holds with probability at least $1-p$. Thus, we can define the always operator as follows:

$$\mathbb{P}_{\leq p} \square \Phi = \mathbb{P}_{\geq 1-p} \diamond \neg \Phi$$

Definition 4.3.2 *Satisfaction relation for PPDL*

Let $e \in E$ be an event of pMSC M . The satisfaction relation \models is defined for local formulae as for PDL, except for the probabilistic operators:

$$\begin{aligned} e \models \mathbb{P}_J \langle \varphi \rangle \Phi &\text{ iff } Prob(e \models \langle \varphi \rangle \Phi) \in J \\ e \models \mathbb{P}_J \langle \varphi^{-1} \rangle \Phi &\text{ iff } Prob(e \models \langle \varphi \rangle^{-1} \Phi) \in J \end{aligned}$$

We define the semantics of the probability expressions later on.

The semantics of *forward-path* expressions $\langle \varphi \rangle \Phi$ of an event e given a poset $(E', <)$ such that $e \in E'$ is given by

$$\begin{aligned} e, (E', <) \models \langle proc \rangle \Phi &\text{ iff } \exists e' \in E'. e \prec_p e' \text{ for some } p \in \mathcal{P} \text{ and } e', (E', <) \models \Phi \\ e, (E', <) \models \langle msg \rangle \Phi &\text{ iff } \exists e' \in E'. e' = m_?(e) \text{ and } e' \models \Phi \text{ and } e', (E', <) \models \Phi \\ e, (E', <) \models \langle \{\Phi\} \rangle \Phi' &\text{ iff } e, (E', <) \models \Phi \text{ and } e, (E', <) \models \Phi' \\ e, (E', <) \models \langle \varphi_1; \varphi_2 \rangle \Phi &\text{ iff } e, (E', <) \models \langle \varphi_1 \rangle \langle \varphi_2 \rangle \Phi \\ e, (E', <) \models \langle \varphi_1 + \varphi_2 \rangle \Phi &\text{ iff } e, (E', <) \models \langle \varphi_1 \rangle \Phi \text{ or } e, (E', <) \models \langle \varphi_2 \rangle \Phi \\ e, (E', <) \models \langle \varphi^* \rangle \Phi &\text{ iff } \exists j \geq 0. e, (E', <) \models (\langle \varphi \rangle)^j \Phi \\ e, (E', <) \models \langle \varphi^{[m,n]} \rangle \Phi &\text{ iff } \exists m \leq j \leq n. e, (E', <) \models (\langle \varphi \rangle)^j \Phi \end{aligned}$$

The base cases for the semantics of *backward*-path expressions $\langle\varphi\rangle^{-1}\Phi$ of an event e given a poset $(E', <)$ such that $e \in E'$ is given by

$$\begin{aligned} e, (E', <) \models \langle proc \rangle^{-1}\Phi & \text{ iff } \exists e' \in E. e' \succ_p e \text{ for some } p \in \mathcal{P} \text{ and } e', (E', <) \models \Phi \\ e, (E', <) \models \langle msg \rangle^{-1}\Phi & \text{ iff } \exists e' \in E'. e' = m_{\bar{\gamma}}^{-1}(e) \text{ and } e' \models \Phi \text{ and } e', (E', <) \models \Phi \end{aligned}$$

The other clauses are defined similarly by replacing $\langle \cdot \rangle$ with $\langle \cdot \rangle^{-1}$.

Now we define the semantics of the probability expression as follows:

$$\begin{aligned} Prob(e \models \langle\varphi\rangle\Phi) & = Pr(\{(E', <)|e, (E', <) \models \langle\varphi\rangle\Phi\}) \\ Prob(e \models \langle\varphi\rangle^{-1}\Phi) & = Pr(\{(E', <)|e, (E', <) \models \langle\varphi\rangle^{-1}\Phi\}) \end{aligned}$$

where Pr is a probability measure over posets as defined in Chapter 2.

The semantics of global formulae is given by:

$$\begin{aligned} M \models \Phi & \text{ iff } \forall e \in E_{min}. M, e \models \Phi \\ M \models \gamma_1 \vee \gamma_2 & \text{ iff } M \models \gamma_1 \text{ or } M \models \gamma_2 \\ M \models \gamma_1 \wedge \gamma_2 & \text{ iff } M \models \gamma_1 \text{ and } M \models \gamma_2 \end{aligned}$$

where E_{min} denotes the set of minimal events in E .

□

Since a pMSC has more than one poset, the forward- and the backward-path expressions $\langle\varphi\rangle\Phi$ and $\langle\varphi\rangle^{-1}\Phi$ have different satisfiability result for each poset. Therefore, we define the satisfiability of these formulae by incorporating the poset in which the event e belongs to. The semantics of the expressions are equal to those of PDL, except that the satisfiability is now defined over an event in a particular poset, that is, we have $e, (E', <) \models \langle\varphi\rangle\Phi$ and $e, (E', <) \models \langle\varphi\rangle^{-1}\Phi$. Since every poset is equipped with a probability of being executed, we can now reason about the likelihood that an event e satisfies a forward-path expression, $Pr(e \models \langle\varphi\rangle\Phi)$, by computing the probability of the set of posets in which e belongs to, that satisfy $\langle\varphi\rangle\Phi$, $Pr(\{(E', <)|e, (E', <) \models \langle\varphi\rangle\Phi\})$. Thus, when we have more than one poset satisfying $\langle\varphi\rangle\Phi$, we simply sum up the probability of each of those poset. Similar reasoning is defined for the backward-path expression.

We also incorporate a bounded iterative operator $\varphi^{[m,n]}$ to complement the unbounded operator φ^* . The intuitive meaning is that while the unbounded operator allows the path expression φ to be repeated in a finite but nondeterministic number of times, the bounded form requires the expression to be repeated in a finite number inside the interval $[m, n]$.

In the original PDL, the existential quantification $\exists\Phi$ is used to express the existence of some events satisfying Φ while the universal quantification $\forall\Phi$ expresses that Φ holds at all events. We modify the global formulae by removing the existential and universal quantification and define the semantics on the minimal events of the pMSC

such that $M \models \Phi$ holds if there is exist a minimal event which satisfies Φ . The reason behind this modification is that when pMSC is used to model a system, the emphasis of the modeling is on the scenario of the system, which means that we really care about how the events are ordered to be executed. Therefore, we think that it would be more suitable to use global formulae for reasoning about the whole scenario which is started from one of the minimal events than to reason about individual events as imposed by the quantifications.

Example 4.3.1 *PPDL example*

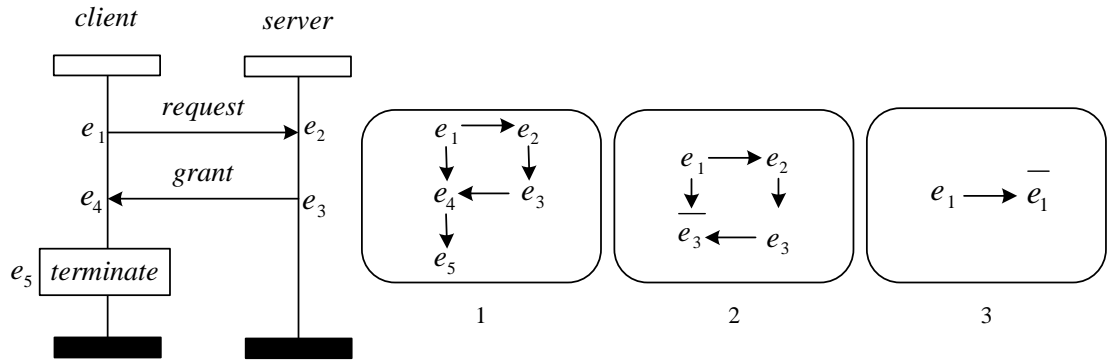


Figure 4.2: Simple client-server program as pMSC

Figure 4.2 depicts a simple client-server program where the client sends a request and then terminates after the server grants the request. The PPDL global formula

$$\gamma_1 = \mathbb{P}_{\geq 0.95} \langle msg \rangle server ? client (request)$$

expresses that the request sent by the client will be received by the server with probability at least 0.95.

The PPDL formula

$$\gamma_2 = \mathbb{P}_{\geq 0.90} \langle \diamond \rangle client (terminate)$$

asserts that the client finally terminates with probability at least 0.90.

Suppose the probability of message loss is 0.02 for the channel in both directions. We first compute the probability of each poset with the procedure given in Chapter 2 and then show whether formula γ_1 holds in the pMSC M . We get the following probabilities for the posets:

$$Pr(E_1, <_1) = 0.9604 \quad Pr(E_2, <_2) = 0.0196 \quad Pr(E_3, <_3) = 0.02$$

We have

$$\begin{aligned} M \models \mathbb{P}_{\geq 0.95} \langle msg \rangle server ? client (request) \\ \text{iff} \\ \forall e \in E_{min}. M, e \models \mathbb{P}_{\geq 0.95} \langle msg \rangle server ? client (request) \end{aligned}$$

We only have one minimal event, e_1 , so we need to check whether e_1 satisfies the local formulae $\mathbb{P}_{\geq 0.95}\langle msg \rangle server?client(request)$. We have

$$\begin{aligned} e_1 \models \mathbb{P}_{\geq 0.95}\langle msg \rangle server?client(request) \\ \text{iff} \\ Prob(e_1 \models \langle msg \rangle server?client(request)) \geq 0.95 \end{aligned}$$

and

$$\begin{aligned} Prob(e_1 \models \langle msg \rangle server?client(request)) \\ = \\ Pr(\{(E', <) | e_1, (E', <) \models \langle msg \rangle server?client(request)\}) \end{aligned}$$

For $i \in \{1, 2, 3\}$, we need to check whether $e_1, (E_i, <_i) \models \langle msg \rangle server?client(request)$. Event e_1 in posets $(E_1, <_1)$ and $(E_2, <_2)$ satisfies this formula because in these two posets e_1 has the matching receive event e_2 and $e_2 \models server?client(request)$. Thus,

$$\begin{aligned} Prob(e_1 \models \langle msg \rangle server?client(request)) &= Pr(\{(E_1, <_1), (E_2, <_2)\}) \\ &= 0.9604 + 0.0196 \\ &= 0.98 \end{aligned}$$

and therefore, $M \models \gamma_1$.

□

Example 4.3.2 PDDL example

Now consider again Figure 4.1 where the client has to make three requests and receive the grant from the server for each request before it terminates. We can specify this requirement for the client as PDDL formula

$$\begin{aligned} &client(start) \\ &\quad \wedge \\ &\mathbb{P}_{\geq 0.95}((proc; client!server(request); (proc; client?server(grant)))^3; proc)client(terminates) \end{aligned}$$

Let γ be the global formula above, $\gamma_1 = client(start)$, $\gamma_2 = \mathbb{P}_{\geq 0.95}((proc; client!server(request); (proc; client?server(grant)))^3; proc)client(terminates)$, $\gamma = \gamma_1 \wedge \gamma_2$ and M be the pMSC. The only minimal event in the pMSC is e_1 , so we need to check whether e_1 satisfies both γ_1 and γ_2 . We can easily see that $e_1 \models client(start)$, now we continue by checking whether $e_1 \models \gamma_2$.

Let φ_1 be the path expression $\langle (proc; client!server(request); (proc; client?server(grant)))^3 \rangle$, φ_2 be the path expression $\langle proc \rangle$ and φ be the path expression $\langle \varphi_1; \varphi_2 \rangle$. Next, let Φ be the local formula $client(terminates)$ and Φ' be the local formula $\mathbb{P}_{\geq 0.95}\langle \varphi \rangle \Phi$. We have

$$e_1 \models \mathbb{P}_{\geq 0.95}\langle \varphi \rangle \Phi \text{ iff } Prob(e_1 \models \langle \varphi \rangle \Phi) \geq 0.95$$

For a poset $(E', <)$, we have

$$\begin{aligned} e_1, (E', <) &\models \langle \varphi \rangle \Phi \\ &\models \langle \varphi_1; \varphi_2 \rangle \Phi \\ &\models \langle \varphi_1 \rangle \langle \varphi_2 \rangle \Phi \end{aligned}$$

Let φ'_1 be the path expression $\langle proc; client!server(request) \rangle$ and φ''_1 be the path expression $\langle proc; client?server(grant) \rangle$ so that $\varphi_1 = \langle (\varphi'_1; \varphi''_1)^{=3} \rangle$. We have

$$\begin{aligned} e_1, (E', <) &\models \langle \varphi_1 \rangle \langle \varphi_2 \rangle \Phi \\ &\models \langle (\varphi'_1; \varphi''_1)^{=3} \rangle \langle \varphi_2 \rangle \Phi \\ &\models \langle \varphi'_1; \varphi''_1 \rangle \langle \varphi'_1; \varphi''_1 \rangle \langle \varphi'_1; \varphi''_1 \rangle \langle \varphi_2 \rangle \Phi \end{aligned}$$

The only event which satisfies Φ is the last event e_{14} and from the visual representation of the pMSC, we can easily see that each pair of send and receive events depends on the previous pair. If the previous one gets lost then the next one cannot be executed, so the only poset in which e_{14} will be executed is the maximal poset $(E_{max}, <_{max})$, where no message gets lost. In this maximal poset, when the current event is e_1 , φ'_1 reasons about the next event, e_2 , and from e_2 , φ''_1 reasons about e_4 such that now we need to check whether

$$e_4, (E_{max}, <_{max}) \models \langle \varphi'_1; \varphi''_1 \rangle \langle \varphi'_1; \varphi''_1 \rangle \langle \varphi_2 \rangle \Phi$$

Since we repeat $\langle \varphi'_1; \varphi''_1 \rangle$ two more times, the current event will be e_{12} . We have

$$e_{12}, (E_{max}, <_{max}) \models \langle \varphi_2 \rangle \Phi$$

because $e_{14}, (E_{max}, <_{max}) \models \Phi$ and $e_{12} <_{max} e_{14}$.

Suppose the probability of message loss is 0.02 for the channel in both directions. The probability of the maximal poset can be computed, and we get $Pr(E_{max}, <_{max}) = 0.886$, which is less than 0.95. Therefore, we have $e_1 \not\models \mathbb{P}_{\geq 0.95} \langle \varphi \rangle \Phi$ and $M \not\models \gamma$.

Another way to define the same requirement is by using the backward-path formula

$$\begin{aligned} &\mathbb{P}_{\geq 0.95} \diamond (client(terminates) \\ &\quad \wedge \\ &\mathbb{P}_{=1} \langle \langle \langle (proc; client?server(grant)); (proc; client!server(request)) \rangle^{=3}; proc \rangle^{-1} client(start) \rangle \end{aligned}$$

□

Chapter 5

Conclusions

This chapter summarizes this thesis by briefly discussing the conclusions from the previous chapters, related work, and the some directions for future work.

5.1 Summary

Probabilistic Message Sequence Charts (pMSCs). We extended Basic Message Sequence Charts (BMSC) [26] by incorporating probabilistic lossy channel systems into the model to provide a more realistic approach for modelling unreliable systems. We adopt the *local-fault model* [1, 41] in which any single message can be lost with a certain probability, independently from the other messages. The probability is attached to the channels such that in a channel with probability of message loss p_{loss} , any message can be lost with probability p_{loss} . We define the syntax of pMSC by extending the definition of BMSC with the notion of message loss events and probability of message loss attached to every channel.

Probability of message loss may introduce several different scenarios that may happen during execution of the pMSC because message loss events can cause other events to be disabled. We model every possible scenario as a partial ordered set (poset) such that a poset represents a case where some particular messages get lost while the others do not. Thus, unlike BMSC, a pMSC has a set of posets instead of a single one. We provide formal definition that maps a pMSC onto a set of posets.

We also provide probabilistic semantics for events, posets and linearizations so that we can reason about how likely a scenario represented by a sequence of events to be executed given the probability of message loss for the channels.

Probabilistic Message Sequence Graphs (pMSGs). We define pMSG, which is a digraph with pMSCs as vertices, as a mechanism to combine pMSCs by incorporating both probabilistic and nondeterministic behaviors into our model. We start by providing the definition of concatenation operator for posets and pMSCs and show how this operator preserves the consistency of the posets in the pMSC. pMSG is defined as a special structure of Markov decision process (MDP) where each vertex represents a pMSC. The system evolves by choosing a distribution at each vertex

nondeterministically, then chooses the next vertex probabilistically based on the chosen distribution. We also provide definitions of the language accepted by pMSGs. Since every vertex in a pMSG represents a pMSC, we associate a pMSC with each path of pMSG by concatenating pMSCs corresponding to individual vertices in the path.

As in MDPs, nondeterministic choices in pMSGs are resolved by an adversary. We use memoryless adversaries in our work which is sufficient as long as we are only interested in maximal and minimal reachability probabilities. We show how an adversary induces Markov chains from a pMSG and given an adversary, how to compute the probability of linearizations from the induced paths.

We provide a pMSG model of a well-known Asynchronous Leader Election Protocol as a case study.

Probabilistic Propositional Dynamic Logic (PPDL). PPDL is introduced in this thesis in order to specify properties of pMSCs. The logic is an extension of Propositional Dynamic Logic (PDL) for message-passing systems [11]. We introduce probabilistic operators because pMSCs may have more than one possible poset with different probabilities so that the likelihood of a formula to hold is defined with a certain probability instead of boolean yes or no. We modify some syntax and semantics of PDL operators and provide the reasoning. We also provide some examples on how we can specify properties of pMSCs and how the semantics work.

5.2 Related work

STAIRS [39,42] is a sequence diagrams-based formalism which specifies interactions between components in terms of traces. Unlike our poset-based semantics, the semantics of STAIRS is given as a set of interaction obligations where each obligation is a pair consisting of a set of positive (allowed) traces and a set of negative (illegal) traces. Another difference between STAIRS and our approach is that STAIRS distinguishes nondeterminism in specifications as either underspecification or inherent nondeterminism. Underspecification means that the specification leaves some freedom of choice for the implementation or further refinement. For example, in an automatic teller machine specification, it is specified that the money should be delivered and the card should be returned back to the client, but whether the card is returned before or after the money is left for the implementation. On the other hand, inherent nondeterminism means that all alternatives must be reflected in the final implementation. For example, the flip of a coin should guarantee that both head and tail are possible outcomes. STAIRS uses different operators for nondeterminism, one for underspecification and another one for inherent nondeterminism. In our work, underspecification and inherent nondeterminism are not distinguished.

Probabilistic STAIRS [39] allows probabilities to be specified by introducing probabilistic operator to reason about probabilistic choices in the system. Combining probabilistic STAIR's underspecification and/or nondeterminism operator with the probabilistic operator will result in the same sense of nondeterminism and probability notions of our pMSGs. Unlike pMSCs, probabilistic STAIRS is not based on

(probabilistic lossy) channel systems. Message losses can only be specified one by one for each sending using the probability operator.

5.3 Future work

As discussed in Chapter 3, we define a pMSG as an MDP with special treatment for the final vertices. We hope that this definition would enable us to specify the properties of pMSGs using well-known logics for MDPs. We consider using Probabilistic Computation Tree Logic (PCTL) to specify properties of pMSGs. The intuitive procedure is to use PCTL and replace atomic propositions with global formulae such that every vertex is labeled with a global formula satisfied by the pMSC represented by the vertex. It turns out however, that we cannot simply apply PCTL to pMSG without having to go deep into all pMSCs which are represented by the vertices of the pMSG.

We illustrate this problem by a simple example. Consider a client-server program represented as a pMSG in Figure 5.1. We define an adversary which chooses μ when the system is in vertex v_1 . Suppose we would like to check whether in one step the client will terminate with probability more than 0.7. This property can be represented as a PCTL formula

$$\mathbb{P}_{>0.7}(\bigcirc_{client}(terminates))$$

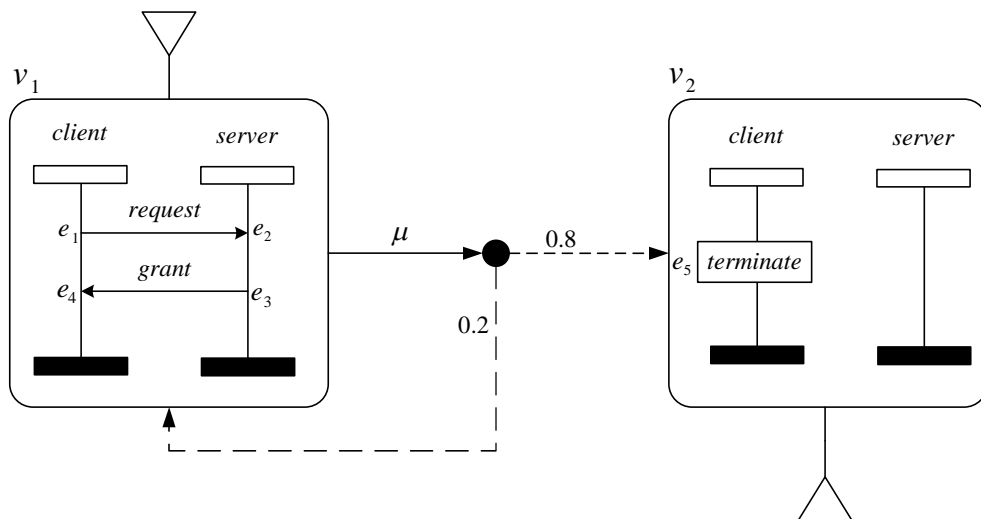


Figure 5.1: Client-server program as pMSG

If we only consider the pMSG representation, we can see that v_2 satisfies the formula $client(terminates)$ and thus $\bigcirc_{client}(terminates)$ is satisfied by the path $\pi = v_1v_2$. If the channels are reliable, then this is exactly the case. However, if the channels

are unreliable, there would be more than one poset generated by the pMSC of the path $M(\pi)$ and it is possible that not all of these posets satisfy the formula. Figure 5.2 depicts three posets of $M(\pi)$. We can see that only in the first poset does the client terminate (represented as event e_5), so to compute the probability that the formula $\bigcirc_{client}(terminates)$ is satisfied, we also need to compute the probability that the first poset is executed.

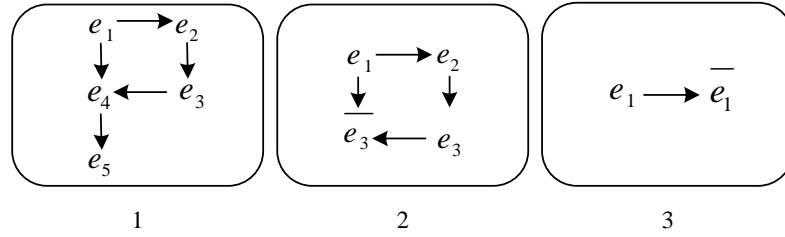


Figure 5.2: Posets of $M(\pi)$

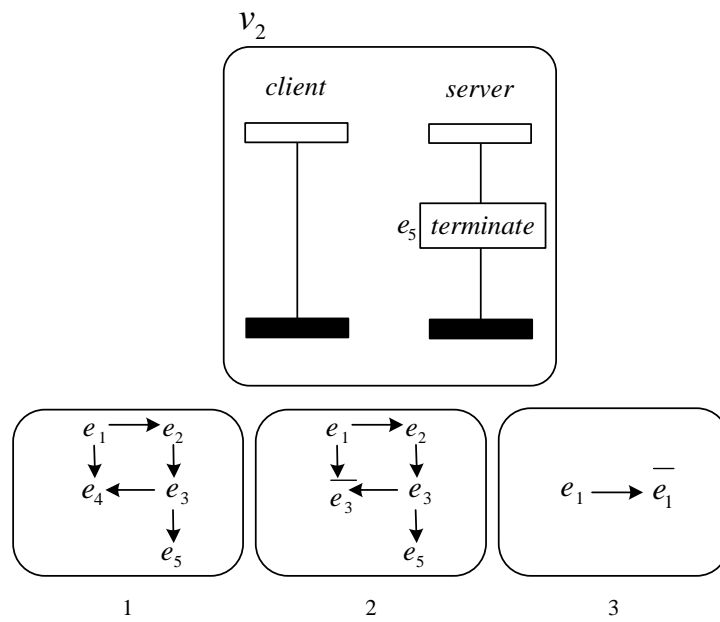
Now consider a slight modification of the pMSC represented by vertex v_2 given in Figure 5.3. We now consider that the server will terminate after giving the grant to the client by the following PCTL formula

$$\mathbb{P}_{>0.7}(\bigcirc_{server}(terminates))$$

The formula $\bigcirc_{server}(terminates)$ is now satisfied if either the first or the second poset is executed while in the previous case only one poset satisfies the formula.

From the illustration above, we can see that it is possible that a vertex satisfies a global PPDL formula, but when concatenated to another vertex, the formula is no longer satisfied with the same probability. In order to apply PCTL on pMSGs, we have to concatenate the pMSCs of the paths and check every possible poset generated by those pMSCs to check the satisfaction relations. We would like to stop at this point and leave the problem of specifying properties of pMSG for further studies.

As mentioned earlier in chapter 4, we also leave the model checking problem of pMSCs against PPDL formulas for future work.

Figure 5.3: Modification on v_2 and the resulting posets

Bibliography

- [1] P. A. Abdulla and A. Rabinovich. Verification of Probabilistic Systems with Faulty Communication. In *Proc. 6th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'03)*, volume 2620 of LNCS, pages 39–53. Springer, 2003.
- [2] Parosh Abdulla and Bengt Jonsson. Verifying Programs with Unreliable Channels. In *LICS '93., Proceedings of 8th Annual IEEE Symposium on Logic in Computer Science*, 1993.
- [3] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and Verification of MSC Graphs. In *In Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808, London, UK, 2001. Springer-Verlag.
- [4] Rajeev Alur, Doron Peled, and Wojciech Penczek. Model-Checking of causality properties. In *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, page 90. IEEE Computer Society, 1995.
- [5] Rajeev Alur and Mihalis Yannakakis. Model Checking of Message Sequence Charts. *Concurrency Theory*, LNCS 1664:780, 1999.
- [6] Christel Baier and Marta Kwiatkowska. Model Checking for a Probabilistic Branching Time Logic with Fairness. *Distributed Computing*, 11:125–155, 1998.
- [7] Paul Baker, Paul Bristow, Clive Jervis, David King, and Bill Mitchell. Automatic Generation of Conformance Tests from Message Sequence Charts . *Telecommunications and Beyond: The Broader Applicability of MSC and SDL*, LNCS 2599:170–198, 2003.
- [8] Hanène Ben-Abdallah and Stefan Leue. MESA: Support for Scenario-Based Design of Concurrent Systems. *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1384, 1998.
- [9] G.V. Bochmann. Finite State Descriptions of Communication Protocols. *Computer Networks*, 3, 5:361–371, 1978.
- [10] Benedikt Bollig, Carsten Kern, Markus Schlütter, and Volker Stolz. MSCan - A Tool for Analyzing MSC Specifications . *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 3920:455–458, 2006.

- [11] Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional Dynamic Logic for Message-Passing Systems. *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, LNCS 4855:303–315, 2007.
- [12] Grady Booch, Ivar Jacobson, and James Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.
- [13] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels . *Information and Computation*, Volume 124, Issue 1:20–31, 1996.
- [14] Werner Damm and David Harel. LSC's: Breathing Life into Message Sequence Charts. In *In Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, 1999.
- [15] Alain Finkel. Decidability of the Termination Problem for Completely Specified Protocols. *Distributed Computing*, Volume 7, Number 3:129–135, 1994.
- [16] Michael J. Fischer and Richard E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.
- [17] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. Test Case Generation with Test Purpose Specification by MSCs. In *SDL '93, Using Objects: Proceedings of the Sixth SDL Forum*, 1993.
- [18] Object Management Group. UML Superstructure Specification. Document: formal/2005-07-04, 2005.
- [19] Elsa L. Gunter, Anca Muscholl, and Doron A. Peled. Compositional Message Sequence Charts. In *In Proc. 7th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 01), volume 2031 of Lect. Notes in Comp. Sci*, pages 496–511. Springer, 2001.
- [20] Indranil Gupta, Robbert Van Renesse, and Kenneth P. Birman. A Probabilistically Correct Leader Election Protocol for Large Groups. In *In Proceedings of the 14th International Symposium on Distributed Computing*, pages 89–103, 2000.
- [21] Jesper G. Henriksen and P. S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, 96:1–3, 1997.
- [22] Gerard J. Holzmann, Doron A. Peled, and Margaret H. Redberg. Design Tools for Requirements Engineering. *Bell Labs Technical Journal*, 2:86–95, 1997.
- [23] Alon Itai and Michael Rodeh. Symmetry Breaking In Distributed Networks. *Information and Computation*, 88:150–158, 1981.
- [24] ITU-T. Recommendation Z.120. Message Sequence Charts, 1993.
- [25] ITU-T. Recommendation Z.120. Message Sequence Charts, 1996.

- [26] ITU-T. Formal semantics of Message Sequence Charts, 1998.
- [27] ITU-T. Recommendation Z.120. Message Sequence Charts, 1999.
- [28] ITU-T. Recommendation Z.120. Message Sequence Charts, 2004.
- [29] Purush Iyer and Murali Narasimha. Probabilistic Lossy Channel Systems. In *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, LNCS vol. 1214*, 1997.
- [30] John G. Kemeny, J. Laurie Snell, and Gerald L. Thompson. *Introduction to Finite Mathematics*. Prentice Hall, Englewood Cliffs, New Jersey, 3rd edition, 1974.
- [31] Ferhat Khendek and Xiao Jun Zhang. From MSC to SDL: Overview and an Application to the Autonomous Shuttle Transport System. *Scenarios: Models, Transformations and Tools.*, LNCS 3466:228–254, 2005.
- [32] Ingolf Krüger, Radu Grosu, Peter Scholz, and Manfred Broy. From MSCs to statecharts. In *Proceedings of the IFIP WG10.3/WG10.5 international workshop on Distributed and parallel embedded systems*, 1998.
- [33] Parthasarathy Madhusudan. Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs. In *Proc. 27th International Colloquium on Automata, Languages and Programming*, volume LNCS 2076, pages 396–407. Springer, 2001.
- [34] B. Meenakshi and R. Ramanujam. Reasoning about Layered Message Passing Systems. In *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 268–282, 2003.
- [35] Andrew Miga, Daniel Amyot, Francis Bordeleau, Donald Cameron, and Murray Woodside. Deriving Message Sequence Charts from Use Case Maps Scenario Specifications. In *Maps Scenario Specifications. 10th SDL Forum.*, pages 268–287. Springer, 2001.
- [36] Anca Muscholl, Doron Peled, and Zhendong Su. Deciding Properties for Message Sequence Charts. In *FoSSaCS '98: Proceedings of the First International Conference on Foundations of Software Science and Computation Structure*, 1998.
- [37] Doron Peled. Specification and Verification of Message Sequence Charts. In *FORTE/PSTV 2000: Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, pages 139–154. Kluwer, B.V., 2000.
- [38] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

-
- [39] Atle Refsdal, Knut Eilif Husa, and Ketil Stølen. Specification and Refinement of Soft Real-Time Requirements Using Sequence Diagrams. *Formal Modeling and Analysis of Timed Systems*, LNCS 3829:32–48, 2005.
- [40] Ph. Schnoebelen. The Verification of Probabilistic Lossy Channel System. In *In Validation of Stochastic Systems, A Guide to Current Research*, LNCS 2925, 2004.
- [41] N. Vertrand and Ph. Schnoebelen. Model Checking Lossy Channels Systems is Probably Decidable. In *Proc. 6th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'03)*, volume 2620 of LNCS, pages 120–135. Springer, 2003.
- [42] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams . *Software and Systems Modeling*, 4:355–357, 2005.

Appendix A

Domain Theory

A.1 Partially Ordered Set

In order theory, a partially ordered set (poset) formalizes the concept of an ordering of the elements of a set. A poset consists of a set and a binary relation which describes the ordering of the elements of the set. Below we give the formal definitions of the notions.

Definition A.1.1 *Partial order of a set*

Let X be a set of elements. A *partial order* over X is a relation $< \subseteq X \times X$ such that:

- $<$ is irreflexive, i.e., $\neg(x < x)$,
- $<$ is transitive, i.e., $x < x' \wedge x' < x''$ implies $x < x''$,
- $<$ is antisymmetric, i.e., $\neg(x < x' \wedge x' < x)$,

for all $x, x', x'' \in X$.

□

The partial order defined above is often called a *strict partial order* because it requires every pair to be irreflexive. In a *non-strict partial order*, normally denoted by “ \leq ”, irreflexivity is replaced by reflexivity ($x \leq x$).

A partial order is different from a *total order* in that some pairs of elements may not be related to each other. In a total order we have an additional condition which is *totality*. A relation having the property of totality means that any pair of elements in the set of the relation are mutually comparable under the relation. Formally, in a total order we have the condition: $x < x'$ or $x' < x$ for all $x, x', x'' \in X$.

Definition A.1.2 *Partially ordered set (Poset)*

A *partially ordered set* (poset) is a set taken together with a partial order on it. Formally, a poset is a tuple $(X, <)$ where X is a set of elements and $<$ is a partial order over X .

□

Let $P = (X, <)$ be a poset. If $x, x' \in X$ and either $x < x'$ or $x' < x$, we say x and x' are *comparable* in $<$; else we say x and x' are *incomparable* in $<$. An element $x \in X$ is called a *minimal* element (respectively, *maximal* element) if there is no element $x' \in X$ such that $x' < x$ (respectively, $x < x'$) in $<$. $\min(X, <)$ denotes the set of all minimal elements of $(X, <)$ and $\max(X, <)$ the set of all maximal elements.

When $P = (X, <)$ is a poset and Y is a nonempty subset of X , the binary relation $<' = < \cap (Y \times Y)$ is a partial order on Y and we call the poset $(Y, <')$ a *subset* of $(X, <)$, denoted by $(Y, <') \subseteq (X, <)$.

Given a poset $P = (X, <)$ and a subset $C \subseteq X$, C is a *chain* if every distinct pair of elements of C is comparable in $<$. When $<$ is a total order then X is a chain.

A.2 Probability Theory

Definition A.2.1 Probability space

We define the probability space as a triple (Ω, ξ, Pr) . (Ω, ξ) is a σ -algebra such that:

- the *sample space* Ω is the set of all possible outcomes,
- the *events* $\xi \subseteq 2^\Omega$ is a set consisting of subsets of Ω which contains the empty set and is closed under complementation and countable unions, i.e.,
 - $\emptyset \in \xi$,
 - if $Ev \in \xi$, then $\overline{Ev} = \Omega \setminus Ev \in \xi$,
 - if $Ev_1, Ev_2, \dots \in \xi$, then $\bigcup_{n \geq 1} Ev_n \in \xi$.

A *probability measure* on (Ω, ξ) is a function $Pr : \xi \rightarrow [0, 1]$ such that $Pr(\emptyset) = 0$ and $Pr(\Omega) = 1$, and if $(Ev_n)_{n \geq 1}$ is a family of pairwise disjoint events $Ev_n \in \xi$, it holds that

$$Pr\left(\biguplus_{n \geq 1} Ev_n\right) = \sum_{n \geq 1} Pr(Ev_n)$$

□