REASONING ABOUT DISTRIBUTED MESSAGE PASSING SYSTEMS

Thesis submitted in
Partial Fulfilment of the
Degree of Doctor of Philosophy (Ph.D.)

by

B. Meenakshi
Theoretical Computer Science Group
The Institute of Mathematical Sciences
Chennai 600 113.

UNIVERSITY OF MADRAS
Chennai 600 005

July 2004

DECLARATION

I declare that the thesis entitled **Reasoning about distributed message passing systems** submitted by me for the Degree of Doctor of Philosophy is the record of work carried out by me during the period from August 1999 to June 2004 under the guidance of Dr. R. Ramanujam and has not formed the basis for the award of any degree, diploma, associateship, fellowship, titles in this or any other University or other similar institution of higher learning.

July 2004 B.Meenakshi

The Institute of Mathematical Sciences
C.I.T. Campus, Taramani
Chennai 600 113, India.

CERTIFICATE

This is to certify that the Ph.D. thesis submitted by B. Meenakshi to the University of Madras, entitled **Reasoning about distributed message passing systems**, is a record of bonafide research work done during the period 1999-2004 under my guidance and supervision. The research work presented in this thesis has not formed the basis for the award of any degree, diploma, associateship, fellowship, titles in this University or any other University or Institution of Higher Learning.

It is further certified that the thesis represents independent work by the candidate and colloboration when existed was necessitated by the nature and scope of problems dealt with.

R.Ramanujam

July 2004 Thesis Supervisor

The Institute of Mathematical Sciences
C.I.T. Campus, Taramani
Chennai 600 113, India.

Abstract

This thesis is concerned with the use of formal methods towards automatic verification of distributed message passing systems with a fixed finite number of agents. We concentrate on developing logics to reason about behaviours of these systems. Behaviours of such systems are modeled using Lamport diagrams which are partial orders over a set of events which is partitioned into those of individual agents such that the events of each agent form a total order. The partially ordered relation in a Lamport diagram is intended to model the underlying causal ordering between events of the system. They can also be used to model message passing in the system in which case we designate certain events as being send and/or receive events.

Many protocols are systems which consist of repetitions of a fixed number of finite communication patterns. We model the behaviour of such systems using Layered Lamport Diagrams (LLDs). The layers of these diagrams are finite Lamport diagrams which consist of events that belong to a particular finite communication pattern. The layers are then composed to obtain the infinite layered Lamport diagram.

We first introduce a natural modal logic with *global* $\mathbf{X}$ (next), $\mathbf{Y}$ (previous), $\mathbf{F}$ (future) and $\mathbf{P}$ (past) modalities over Lamport diagrams. The satisfiability problem of this logic turns out to be undecidable, even if we retain only the global $\mathbf{X}$ ($\mathbf{Y}$) modality and restrict the other one to special receive (special send) propositions which talk about a message being sent (received) without being able to specify anything about the content of the message etc.

Given this, we consider how we might obtain decidable logics. In the first approach, we restrict the syntax and consider a temporal logic over Lamport diagrams. The logic has *local* temporal modalities ($\mathbf{X}$ and $\mathbf{U}$) for each agent and in addition, it has a *weakly global* $\mathbf{Y}$ modality. This new modality, when asserted at an agent i talks about a local formula of agent j being true at the last j-local state visible to i. We show that the satisfiability problem of this logic is decidable using the automata-theoretic approach. We use a distributed automaton model called System of Communicating Automata (SCA) and show that the emptiness problem for these automata is decidable. Given a formula of the logic, we associate an SCA with it such that the language of the SCA is non-empty iff the formula is satisfiable. The decidability of the logic follows from the decidability of the emptiness problem for SCAs. A suitable model checking problem is also shown to be decidable.

In the second approach, we consider ways of restricting the models to obtain

decidable and expressive logics to reason about behaviours of distributed systems. We consider LLDs as models and extend the modal logic above to a temporal logic interpreted over them. The temporal logic reflects the layered structure of LLDs—the basic modal logic introduced above talks about the properties of layers and the temporal modalities are used to talk about sequence of layers that make up the LLD. It turns out that the satisfiability problem of this logic is also undecidable, even if we restrict the size of the layers to be uniformly bounded. However, we obtain decidability when we consider models based on communication closed and bounded LLDs, or when we consider models based on LLDs whose layers have bounded channel capacity. We again prove decidability using the automata-theoretic approach, by introducing automata models like diagram automata and fragment automata whose emptiness problem is also shown to be decidable.

We investigate the use of a suitable monadic second order logic over Lamport diagrams as a specification language. Not surprisingly, the satisfiability problem of this logic is again undecidable. We then show that it is decidable over the class of models based on communication closed and bounded LLDs and also over models based on channel bounded LLDs as above.

We wind up by considering Message Sequence Charts (MSCs) as an alternate model for representing behaviours of message passing systems. MSCs are graphs over an underlying set of events (which is partitioned into those of send, receive and local events) such that the events of each agent are totally ordered. MSCs also have an explicit matching relation which maps each send event with its corresponding receive event and the ordering defined by the local total orders and the matching function is required to be a partial order over the set of events. We show that Lamport diagrams possess more modelling features than that of MSCs and constitute the underlying partial order in an MSC. We also consider other known models to represent collections of MSCs, namely that of Message Sequence Graphs (MSGs) and Compositional Message Sequence Graphs (CMSGs) and compare relevant classes of LLDs with those generated by MSGs and CMSGs. It turns out that the classes of LLDs are strictly more expressive than the partial orders generated by MSGs and CMSGs. We also define an MSO over MSCs and show that model checking CMSGs against these MSO specifications is decidable.

Acknowledgments

I would like to thank my advisor Prof. R. Ramanujam for his guidance and support all these years. He has been extremely patient with me through the course of this work and this thesis would not have taken shape without his support and teaching. I have learnt most of what i know in logic and concurrency theory from him. Apart from being my thesis advisor, Jam has been of tremendous support to me in my personal life and his suggestions and support has kept me going for all these years. I am very grateful to Jam for all the help and patience.

Madhavan Mukund and Kamal Lodaya have been my doctoral committee members through these years. Apart from guiding me during the meetings and many other times, they have also taught me fundamental courses in logic and computability through which I have learnt a lot. I would like to thank them for their support and teaching.

The Theoretical Computer Science group at IMSc and CMI welcomed me into their group and guided us through an excellent course work through which I have learnt many beautiful things in this subject. I am grateful to all of them for their excellent teaching—V. Arvind, Kamal Lodaya, Madhavan Mukund, Meena Mahajan, K. Narayan Kumar, R. Ramanujam, Anil Seth, K. V. Subrahmanyam, P. S. Thiagarajan and Venkatesh Raman.

I have enjoyed discussing and working with P. Madhusudan on message sequence charts and his questions have always encouraged me to think more on a particular topic. I would like to thank him for his valuable friendship too.

Many people have helped me with financial support for my visits abroad to attend various schools and conferences from which I have learnt much. I would like to thank all of them—Prof. Anca Muscholl, Prof. Pascal Weil, Prof. Wolfgang Thomas, organisers of the Marktoberdorf Summer School 2002, Prof. Teodor Knapik, Prof. Lenore Zuck, Prof. Rajeev Alur and Prof. Shankar Natarajan.

I would like to thank my husband Deepak D'Souza for all his love. He has taught me all the basics of practical Computer Science, including how to use emacs etc. Academic interactions with him have been very useful and I have learnt to work hard from him. Thanks Deepak, for all the love, support and companionship.

S. P. Suresh has been a constant and good companion from my early days in the institute. Discussions with him both during course work and after have always been enjoyable.

My brother Karthik has been one among my best friends all these years and I would not have made it this far without his love and support. I would also like to thank my parents for their love. Thank you appa, for all the patience and amma, for opening up the doors of higher education to me.

I thank many friends at IMSc who have made my years here enjoyable—Suneeta, Gyan, Balaji, Sumithra, Vinu, Naveen and Paramita. I owe thanks also to Annie, my good friend from college days, my cousins Shuba and Svetha for their friendship and love and my father-in-law for encouraging me to finish the thesis.

Contents

Chapter 1

Introduction

Computers are ubiquitous these days—they control and manage systems in fields of avionics, industrial processes, space ships and many more applications which are safety critical. The number and complexity of computer controlled systems being developed and used is increasing day-by-day. The safety critical nature of these systems has motivated the development of rigorous methods to show that they are reliable. *Testing* and *formal verification* are two such reliability methods [Pel01]. While testing is done directly on code, formal verification is usually done at the design stage in a software development process. Various successful industrial case studies have illustrated the usefulness of formal verification in minimizing the number of defects in the product being developed [CW96, Rus96].

1.1 Formal verification

Formal verification [CW96, Rus96] is the technique of proving in a formal, mathematical way that a program satisfies its requirement. The program and its requirement or specification are modelled using a mathematical language. Two basic techniques of formal verification are *theorem proving* and *model checking*. In the semi-automatic technique of theorem proving, both the program and the specification are modelled as a set of formulas in some logical language. The program then satisfies a specification iff the formula corresponding to the specification can be proved to be a logical consequence of the set of formulas representing the program.

Model checking is a fully automatic technique wherein the program is usually modelled as a finite transition system and the specification is written as a formula in a logical language. Unfortunately, model checking becomes undecidable even for some simple specifications which require checking for the reachability of a specific state as the transition systems corresponding to arbitrary programs usually have infinitely many states. Hence, research in this area of formal methods has concentrated on identifying classes of programs/systems which can be formally verified, in coming up with suitable specification formalisms and also in reducing the resource requirements to real-life limits through theoretically better algorithms or even heuristic means.

Finite state systems An important class of problems which have been identified to be tractable are those involving checking correctness of programs/systems which can be realized as finite state systems. It turns out that this is not a big restriction as many systems where verification is required like hardware, client-server applications, design of multi-threaded applications etc. are typically finite state systems. Even for systems which are not finite state, it is possible to consider an abstraction of the system which is finite state in order to verify certain specifications like deadlock etc.

Logics as specification languages Specification of a system/program is a property which describes how it ought to or ought not to behave. Various formalisms are available to state specifications of systems. Temporal logics [MP91, HR04] are the most commonly used specification languages—they are basically modal logics which are geared towards reasoning about behaviours of systems that evolve with time. Temporal logics come in two fundamental varieties— linear-time temporal logic (LTL) and branching-time temporal logics (CTL, CTL*), depending on the assumption made about how time evolves with respect to the system considered. Model checking algorithms are available for both the varieties.

Monadic second order logic (MSO) [Tho90, Tho97] is another specification formalism that is more expressive than temporal logics i.e., it is capable of specifying a wider range of properties of systems. Apart from being used as a specification language, the rich theory of MSO logics are also useful in identifying classes of finite state systems. Systems whose behaviours can be ‘identified’ by an MSO formula (i.e., systems whose set of behaviours is exactly the set of those which satisfy the MSO formula) are usually finite state systems (the set of behaviours of finite state systems is usually called a regular language).

Automata theoretic approach to model checking As mentioned before, model checking is a fully automatic way of doing formal verification where the program (or its finite state abstraction) is modelled as a finite transition system and the specification is given as a formula in some logic, usually temporal logic [CGP00]. Models of the formula identify the set of good behaviours of the system and the problem of model checking is to check if the behaviours of the program to be verified are good. One well-established way of doing model checking is by using the *automata-theoretic approach* [VW86]. The main observation in this approach is that one can construct a finite state automaton that accepts the set of all sequences (models) that satisfy the given formula. Then, the problem of model checking is to check if the language accepted by the transition system (representing the program) is a subset of the language accepted by the automaton constructed for the specification. Since this problem is decidable for finite state systems, so is the problem of model checking.

The automata-theoretic approach to solve the model checking problem is also useful to show that the satisfiability problem for temporal logics is decidable. The satisfiability problem for logics is the problem of checking if a given formula is satisfiable, i.e., whether it has a model or not. By using the automata-theoretic approach, we construct an automaton accepting precisely the set of models of a given formula. Consequently, the formula is satisfiable iff the language accepted by the constructed automaton is non-empty. Since this problem is decidable for finite state automata, we also have an algorithm for solving the satisfiability problem for the logic.

1.2 Distributed systems

So far, we have assumed that the system and programs that we are trying to verify are those representing a single entity without any notion of a component. In many practical situations, the program or system is usually a *distributed system* that has many components which are usually called *agents* or *processes.* Agents constitute spatially separated entities that the distributed system is composed of. Typical examples of distributed systems are telecommunication systems, client-server applications, multi-threaded code etc. Sometimes, distributed systems also come with a notion of an environment which is an abstraction of the system that it is built into. The agents perform local computations, interact with the environment and also communicate with other agents. Agents of the system can also dynamically create and/or destroy new agents as the computation of the system evolves.

Distributed systems are usually categorized depending on the way in which the agents of the system communicate with each other.

- In a distributed system with *synchronous* communication, the agents have a set of common actions to be performed jointly and they execute such actions at the same instance of time (measured with respect to a global common clock).
- Agents can also communicate in an *asynchronous* way where the system has a set of dedicated channels or buffers through which the agents send messages to other agents and receive messages from them.
- Finally, there are distributed systems where the agents communicate by reading from and writing onto certain *shared variables* which are stored in a common memory.

Verification of distributed systems Formal methods for verification of distributed systems has been an interesting area of research due to the complex nature and the wide variety of the systems available. Research in this area not only involves developing models for distributed systems but also notations to talk about their behaviours and logics to specify the properties involving their behaviours. Traditional models of concurrent systems and their behaviours like Petri nets, event structures, partial orders etc. [WN95, Win87, Pra86] are generic models capturing notions of concurrency, dependency, conflict etc. between events. They are not well suited as specialized models of distributed systems as they do not model notions like agents, local computations etc. explicitly. Also, these models are, in general, not finite state and this rules out the possibility of developing model checking techniques directly based on these models. This calls for the development of alternate models to talk about special classes of distributed systems and their behaviours in the context of verification.

One standard approach towards verifying distributed systems is to define a global transition system corresponding to the distributed system and to capture system behaviour as a sequence of actions of the global transition system. The states of such a global transition system talk about the local states of the agents, values of all the variables (both global variables and those local to agents), contents of buffers etc. In distributed systems, agents can have independent or concurrent actions which are those that can occur without the influence of any other agent. The global behaviour of the system in terms of sequences of actions involves all possible inter leavings of these independent actions. The properties to be verified are also translated into equivalent properties of the global system involving standard temporal logics. We can then use traditional techniques for solving the verification problem.

However, such an approach is not feasible as the global system is typically not finite state and the verification problem becomes undecidable. Even in cases where the global system is finite state (or can be abstracted into one), the number of possible global states is large as we have to take all possible inter leavings of actions of various agents into consideration. This leads to state explosion problem and traditional formal verification algorithms typically do not perform well on such huge systems. Also, logics which talk about a global sequence of actions of the systems usually fail to capture interesting properties involving events local to agents of the system. Also, the satisfiability problem of such global logics again becomes undecidable [LPRT95, AMP98].

To overcome these drawbacks, various alternate techniques have been proposed to formally verify distributed systems. One common way is to come up with models of distributed systems such that their behaviour is captured component-wise or locally and also to specify properties of systems using logics that talk about the local components. System models based on local computations usually talk about behaviour of the system as a partial order on events of individual agents describing the way they evolve locally unlike the global state-based approach mentioned above. Such approaches have resulted in various useful techniques being developed for modelling, specifying and verifying distributed systems.

Study of distributed systems using partially ordered models also has other advantages. The partial order based models to represent behaviours do not differentiate between computations that are equivalent upto possible inter leavings of independent actions. This natural assumption respects the concurrency in the distributed system. Also, restricting attention to behaviour evolving in terms of local events helps to overcome the state space explosion problem mentioned above as we can reason about the system using its locally specified behaviour without considering the global system.

For example, in the context of distributed systems where the processes communicate synchronously, various distributed automata models have been developed to talk about computations in an agent-based fashion [Zie87, Thi94, MR02]. The behaviour of these automata are also captured in a distributed fashion using *traces* [Maz87] which are basically partial orders whose structure is tuned towards describing these systems. These partial orders talk about sequences of local actions of agents and also model the synchronization of agents using common actions. Various expressive temporal logics have also been developed to describe local properties of these systems in terms of traces [Thi94, APP95, Ram96, TW97]. The fact that these logics are expressive and that their satisfiability problem is decidable makes them ideal specification languages for such systems unlike global logics which are undecidable [AMP98].

1.3 Distributed message passing systems

We now consider systems whose agents communicate asynchronously by exchanging messages across channels. Unlike synchronous distributed systems discussed above, the agents of these systems typically do not share common actions and proceed with their computations in an autonomous fashion [LL90]. The agents also exchange messages with each other through dedicated channels when buffers are not full, the sender can put its message into the buffer meant for the receiver and proceed with its computation without having to wait for the receiver. However, computations can get blocked while waiting for a particular message or an agent might get stuck while trying to send a message through a buffer whose capacity is already full.

In the context of formal methods for distributed message systems, the traditional approach of working with a global system model does not work here due to the presence of buffers whose capacity can be potentially unbounded. States of the global system include the contents of the buffers apart from information regarding variables etc. Consequently, the global system need not be finite state. On the other hand, restricting the capacity of buffers to be uniformly bounded would mean restricting the possible behaviours of the system being modeled. On the specifications side too, it would be interesting to come up with logics which can be used to reason about the behaviour without having to talk about buffers explicitly.

The main aim of this thesis is to develop suitable models to describe behaviours of distributed message passing systems and to come up with suitable logics to reason about them. We also propose certain models for distributed systems and use these system models to solve the satisfiability problem of the logics we consider. The goal is to come up with local event-based models describing behaviours and suitable logics which talk about the way in which local computations evolve. This, we believe, would overcome the problems related to global representations and logics and aid towards formally verifying such systems.

Models and logics for distributed message passing systems

Few models of message passing distributed systems and their behaviours have been proposed in literature. We survey them below.

Asynchronously communicating sequential agents Asynchronously Communicating Sequential Agents (ACSAs) were introduced in [LRT92] as models describing behaviours of distributed message passing systems. The systems can have an arbitrary number of agents and ACSAs are specialized partial orders describing behaviours of such systems. The events are mapped to particular agents and the partial order depicts causal dependency of events with the restriction that the past view of every event restricted to its agent should be linearly ordered. The authors also propose a suitable tense logic to reason about ACSAs and provide a sound and complete axiomatization of this logic. This logic has agent-based past and future modalities which refer to the local past and local future respectively. The formulas are interpreted at local states. However, since ACSAs describe the general causal dependence of events and do not make any assumptions regarding the way in which the events evolve, it is not clear if there is a finite state automata-theoretic model whose behaviour can be described using ACSAs.

A model checking algorithm for a variant of the logic introduced in [LRT92] has been described in [HNW99]. The authors consider a slight variant of the logic introduced in [LRT92] and they show that the problem of checking whether an asynchronous distributed net system (where buffer capacity is assumed to be bounded) whose behaviour is described by ACSAs, satisfies a given formula is decidable. However, the complexity of the algorithm is non-elementary in the size of the formula and the authors do not address the decidability of the satisfiability problem for the logic.

Layered distributed systems Other models for describing computations of distributed message passing systems have been proposed in [EF82, PZ92] where the authors consider asynchronous distributed systems with an arbitrary number of agents. However, these papers focus more on synthesizing distributed systems by decomposing their behaviours into layers which can be composed sequentially and parallely and not on formally modelling and specifying the system.

Lamport diagrams Lamport in [Lam78] introduced certain partial orders to represent computations of distributed message passing systems with a fixed number of agents. Lamport diagrams are partial orders over the set of event occurrences of a distributed system. The event occurrences are partitioned into those of individual agents in such a way that the events of each agent form a linear order. The underlying partially ordered relation captures the causal dependence of event occurrences of the distributed system. We work with Lamport diagrams as as models representing behaviours of distributed message passing systems in this thesis and will present more details in further chapters.

Lamport diagrams can be thought of as a restricted sub-class of ACSAs, which in turn are a sub-class of general representations like event structures. Event structures are very general models of computations of systems where the underlying notion of time is assumed to be branching and they also model conflict in between event occurrences in addition to causality and concurrency. ACSAs can be thought of as restricted event structures which again concentrate only on modeling causality between event occurrences of the distributed system. However, they assume that the past of every event occurrence is totally ordered while the future need not be. Lamport diagrams pose further restriction on ACSAs and insist that all the event occurrences of each agent be totally ordered.

Message sequence charts and related models A formalism closely related to Lamport diagrams is that of Message Sequence Charts (MSCs) [RGG96]. The language of MSCs is a standard language of the International Telecommunication Union [ITU97]. While Lamport diagrams describe communication patterns of a system with main focus on the causal order between the events, the main focus of MSCs is on the description of messages exchanged between the agents of the system and the underlying causal order is required to be a partial order. More precisely, an MSC is a graph over an at most countable set of events (which are partitioned into a set of send, receive and local events) such that there is a bijection between the set of send and receive events. Again, like in Lamport diagrams, each event belongs to a particular agent and the events of each agent are totally ordered. The ordering defined by the total orders of the various agents and the bijection is required to be a partial order over the set of events.

Lamport diagrams constitute an abstract generalization of MSCs and possess additional features to describe simultaneous send and receive events. In fact, Lamport diagrams are partial orders generated by MSCs [AHP96]. We will present a detailed comparison between Lamport diagrams and MSCs later in the thesis.

Various models for distributed message passing systems based on MSCs have been proposed in the literature. Message Sequence Graphs (MSGs) and hierarchical MSCs are models prescribed in the ITU standard to represent collections of MSCs using operations of choice, concatenation and repetition. The model checking problem for MSCs and MSGs against specifications given as finite state automata is studied in [AY99]. The authors show that the problem is undecidable for systems whose models are MSGs but is decidable for systems whose models are bounded MSGs (i.e., MSGs where the capacity of the channel between each pair of agents is assumed to be bounded).

An automata-theoretic model called Message Passing Automata (MPA) for accepting linearizations of MSCs is proposed in [HMKT00b] which accepts regular collections of MSCs. An MPA is a collection of finite state automata, one per agent and the global behaviour of an MPA is obtained by combining the behaviour of each of the local automata along with the contents of buffers, which are bounded. A characterization of regular MSC languages in terms of a monadic second order logic over bounded MSCs is also presented in [HMKT00b].

In [HMKT00a], it is shown that the class of languages of MSCs generated by bounded MSGs is the same as that of finitely generated regular MSC languages, i.e., regular MSC languages which can be obtained from a finite set of MSCs by using operations of union, concatenation and iteration. The collections of MSCs generated by MSGs is incomparable with regular collections of MSCs in general.

A model checking algorithm for MSGs against MSO specifications is presented in [Mad01]. However, the problem of checking satisfiability of MSO over arbitrary MSCs is undecidable [Thi01].

Compositional Message Sequence Graphs (CMSGs) were introduced in [GMP01] as a generalization of MSGs. They are finite graphs whose nodes are labelled by Compositional Message Sequence Charts (CMSCs). CMSCs are like MSCs but, possess unmatched send and receive events in addition to the normal send and receive events of MSCs. These are send (receive) events whose corresponding receive (send) events are not present within the same CMSC. They have to be matched up with appropriate send/receive events while combining CMSCs using the operation of concatenation in an CMSG. We show that CMSGs subsume MSGs and regular MSC languages and also present a model checking algorithm for CMSGs against MSO specifications in the thesis.

1.4 Contributions of the thesis

We work with distributed message passing systems with a fixed finite number of agents in this thesis. We consider n-agent systems and denote the set of agents by $[n] = \{1, 2, \ldots, n\}$. The main focus of the thesis is to investigate notions to represent behaviours of such systems and to develop suitable logics to reason about the behaviours. Towards working on the decidability of various logics, we also propose various automata-theoretic models of these systems and investigate some of their closure properties. These lead to natural model checking problems where the distributed message passing system is modelled using appropriate automata.

Lamport diagrams and layered Lamport diagrams We embark on this study by starting with the model of Lamport diagrams to represent behaviours of distributed message passing systems in Chapter 2. As mentioned above, Lamport diagrams are partial orders representing the causal dependency between event occurrences of a distributed system. We set up notations to talk about send and receive events and also about various properties of Lamport diagrams like those of states, sequentializations etc. and prove a few propositions to be used later.

Lamport diagrams are ideal as models for systems which occur as a parallel composition of sequential behaviour—the event occurrences of each agent are linearly ordered in a Lamport diagram which itself is a partial order on the entire set of event occurrences. However, there are many distributed systems which consist of repeated patterns of finite communication protocols. These patterns are then composed to obtain the full behaviour of the system. To capture the behaviour of such systems, we introduce the notion of a Layered Lamport Diagram (LLD). There can be various types of layering depending on the underlying system behaviour—layerings with only bounded number of events (bounded LLDs), layerings where every send event and its corresponding receive event occur within the same layer (communication closed LLD), layerings where there are only a bounded number of send events whose corresponding receive events occur in later layers (channel bounded LLDs) etc. We introduce layered Lamport diagrams and the various layerings in Chapter 2 and discuss their properties.

Automata models for distributed message passing systems In Chapter 3, we introduce various automata-theoretic models that we work with in the thesis. These models will be used in later chapters to solve the satisfiability problem of appropriate logics using the automata-theoretic approach. We introduce two kinds of automata—the first kind called System of Communicating Automata (SCA) run on Lamport diagrams and accept them and the second kind of automata (diagram automata and fragment automata) accept various layered Lamport diagrams. With the goal of using these automata mainly for solving the satisfiability problem of certain logics, we prove that the emptiness problem for these automata are decidable and also prove certain relevant closure properties.

Modal logics over Lamport diagrams We then consider in Chapter 4, the question of defining appropriate linear time temporal logics to reason about systems whose computations are Lamport diagrams. As mentioned above, the traditional option is to consider the set of all sequentializations of all the Lamport diagrams representing the behaviour of a system and use standard LTL to reason about their properties. To reiterate, there are many drawbacks in such an approach. Firstly, even simple Lamport diagrams have sequentializations which do not form a regular language. For example, consider the Lamport diagram corresponding to the producer-consumer problem given in Figure 1.4. The producer (agent 1) repeatedly sends messages labelled a to the consumer (agent 2) who receives them as b. The set of all finite sequentializations of this Lamport diagram yields the language L over

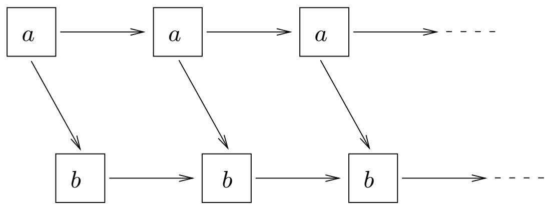


Figure 1.1: Lamport diagram representing a behaviour of the producer-consumer problem

$\{a, b\}$ where every word w in L is such that every prefix of it has at least as many a's as b's, which is not a regular language.

Now, LTL cannot express such non-regular behaviours and so the diagrams specified would have to exclude such behaviours. The second drawback is that a logic like LTL specifies how the global states of the system may evolve whereas it would be ideal to have a logic which specifies the effect of message passing in the system. This is more naturally done using an event based approach instead of considering sequentializations. Such an event based logic would be able to specify properties by using the partially ordered structure of a Lamport diagram and formulas of logic can then talk about properties like when a particular agent can send a message, what would an agent do while receiving a message etc.

As a first step towards defining such a logic, we consider a natural modal logic to reason about these diagrams. A modal logic LD_0 is defined in Chapter 4 with $\mathbf{X}$ (next), $\mathbf{Y}$ (previous), $\mathbf{F}$ (future) and $\mathbf{P}$ (past) modalities over Lamport diagrams. All the four modalities are *global* in their scope. For example, the formula $\mathbf{X}\alpha$ of the logic when asserted at a send event, talks about α being true at a corresponding receive event in another agent. Similarly, the modality $\mathbf{Y}$ interpreted at a receive event can be used to assert the truth of a formula at a send event. The $\mathbf{F}$ and $\mathbf{P}$ modalities also talk about paths that include events across agents in the Lamport diagram.

The satisfiability problem of this logic is the problem of checking if a given formula has a Lamport diagram as a model or not. It turns out that this problem is undecidable—the global $\mathbf{X}$ and $\mathbf{Y}$ modalities of this logic are expressive enough to describe Lamport diagrams that code up runs of non-deterministic 2-counter machines. The proof crucially uses the fact that the modalities $\mathbf{X}$ and $\mathbf{Y}$ are global in the sense that they can assert truth of formulas at other agents.

Since this logic turns out to be undecidable, the next step is to look for weaker logics with similar expressive power and check for their decidability. Requirements which specify conditions on when a particular agent may send a message can be specified using a very restricted version of the global $\mathbf{X}$ modality, namely, special *send* propositions, which is equivalent to the formula $\mathbf{X}\,\mathit{True}$. Similarly, requirements regarding what an agent would do upon receiving a message can be specified using special *receive* propositions which is again equivalent to $\mathbf{Y}\,\mathit{True}$. Notice that using such special send (receive) propositions, we can just specify requirements involving sending (receiving) a message and not actually the 'content' of the message. It turns out the satisfiability problem for both the weaker logics is also undecidable. This is mainly due to the reason that even the presence of one global modality along with the special proposition to represent a weaker version of the other one suffices to represent Lamport diagrams encoding runs of non-deterministic 2-counter machines.

A temporal logic over Lamport diagrams Faced with the fact that the modal logics proposed above are undecidable, we consider other possible restrictions to obtain decidable logics. We could consider a logic without any $\mathbf{X}$ or $\mathbf{Y}$ modalities, but, with only send and receive propositions. Such a logic would not be expressive enough to specify properties of message passing systems.

The other option is to consider logics wherein both the $\mathbf{X}$ and $\mathbf{Y}$ modalities are local. Such a logic again is not expressive enough to specify requirements involving sending or receiving messages. An intermediate option is to consider logics where exactly one of the modalities $\mathbf{X}$ or $\mathbf{Y}$ is global and the other is local. We believe that the satisfiability problem of such logics would also be undecidable.

As another intermediate option, we consider a logic with a local $\mathbf{X}$ modality and a *weakly* global $\mathbf{Y}$ modality in Chapter 5. The $\mathbf{Y}$ modality is indexed by the agent numbers and such a modality with an index j can assert the truth of a local formula of agent j at the *last* j-event in its past (which need not be the send event at agent j). To be precise, the logic m-LTL has the modalities of LTL (namely $\mathbf{X}$ and $\mathbf{U}$) for each agent in addition to a modality of the form $\oslash_j$ which is the weak $\mathbf{Y}$ modality that we discussed above. We show that the satisfiability problem of this logic is decidable by using the automata-theoretic approach to satisfiability. That is, we associate an SCA with every m-LTL formula such that the language accepted by the SCA is non-empty iff the formula is satisfiable. Since checking the former is decidable, we get decidability of the satisfiability problem for m-LTL. It turns out that m-LTL is also an expressive logic to reason about Lamport diagrams. We illustrate this fact by using an example involving a conference management system. The automata-theoretic approach to satisfiability can be used to define a natural model checking problem for m-LTL where the system is modelled as an SCA. We show that the decidability of the model checking problem.

Coming to related work, [Pel00] considers a similar logic with an $\mathbf{X}$ modality and without the $\mathbf{Y}$ modality interpreted over the partially ordered structure of MSCs (which is essentially a Lamport diagram) and shows that the satisfiability problem of this logic over the class of MSCs generated by MSGs is decidable. However, it is not clear if the satisfiability problem of this logic is decidable in general.

Logics over layered Lamport diagrams We explored various ways of syntactically restricting logics to obtain decidable and expressive logics over Lamport diagrams above. The other track to obtain decidable logics is to restrict the class of models considered. The first observation in this context is the fact that the satisfiability problem of the logic LD_0 is undecidable over models based on *finite* Lamport diagrams. We then place bounds on the size of Lamport diagrams we consider and then the satisfiability problem turns out to be decidable. This is not surprising as there are only a fixed number of models to check against once we place bounds on the size of the diagrams.

To use this result, we consider layered Lamport diagrams which were introduced earlier as models describing behaviours of systems which consist of repeated patterns of finite protocols. Since each layer is a finite Lamport diagram, the logic LD_0 is interpreted over the layers of an LLD. To structure the logic in tune with that of LLDs, we modify this logic over LLDs as follows. The logic λ-LTL has a two level syntax: Formulas of the logic LD_0 talk about the layers of an LLD and a temporal logic is built upon these formulas to talk about the sequence of layers that make up an LLD. The top level temporal logic has the usual linear time connectives, the modality $\bigcirc$ refers to the next layer and the modality $\mathbf{U}$ talks about sequence of layers of the LLDs. Such a logic can be used to write specifications which talk about a sequential composition of parallel processes which are described using LLDs.

It turns out that the satisfiability problem of this logic is again undecidable over the class of models based on bounded LLDs and over those based on (unbounded) communication closed LLDs as we show in Chapter 6. However, the logic becomes decidable when interpreted over the class of models based on communication closed and bounded LLDs and over the class of models based on channel bounded LLDs. We again use the automata theoretic approach to show decidability. Given a formula of λ-LTL over the class of communication closed and bounded LLDs, we associate a diagram automaton such that the language of LLDs accepted by the automaton is precisely those which are models of the formula. We similarly associate a fragment automaton with every λ-LTL formula interpreted over channel bounded LLDs. The results regarding decidability of the emptiness problem for these automata shown in Chapter 3 imply the decidability of the satisfiability problem for these logics.

Some results on MSCs and CMSGs We wind up the thesis by presenting a comparative study of Lamport diagrams and MSCs in the last chapter. We show that Lamport diagrams are partial orders representing the underlying causal structure of MSCs. They possess additional features like a single event being a send and a receive event simultaneously which are not modelled by MSCs. MSCs, on the other hand, can model messages explicitly which is not possible in Lamport diagrams.

We then extend these results and show that the class of communication closed and bounded LLDs represent strictly more general partial orders than those obtained from the MSCs generated from MSGs. Similarly, channel bounded LLDs strictly subsume the Lamport diagrams underlying the MSCs generated from CMSGs.

We also study the CMSG model in detail and show that it subsumes the model of MSGs and generates all of regular MSC languages as well. We finally present an algorithm for model checking CMSGs against MSO specifications.

Monadic second order logic over Lamport diagrams We then turn our attention to defining a suitable MSO over Lamport diagrams. We consider Lamport diagrams whose events are labelled with actions from a distributed alphabet and the MSO is defined on the partially ordered structure of the Lamport diagram. Not surprisingly, the satisfiability problem for this logic also turns out to be undecidable. The modalities of the logic LD_0 are all expressible in MSO. On the positive side, we show that the problem of checking if an MSO formula has a communication closed and bounded LLD as a model or not is decidable. We also show that the problem of checking if a given MSO formula has a channel bounded LLD as a model or not is decidable.

Chapter 2

Lamport diagrams

Throughout the thesis, we fix $n > 0$, and study distributed systems of n agents. We will follow the linguistic convention that the term 'system' will refer to composite distributed systems, and 'agent' to component processes in the system. We assume that the agents are sequential programs and communicate with each other by exchanging messages. We only consider systems whose components communicate by asynchronous message passing. The agents of the system do not share any common actions. For the present, we will make the following assumptions about the communication medium:

1. Every message is eventually delivered.

2. Messages are delivered in the order in which they were sent.

We refer to the agents by the indices i, $1 \leq i \leq n$ and use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$ of agents. The set of natural numbers will be denoted by $\mathbb{N}$ and the natural ordering on them will be denoted by $\leq_{\mathbb{N}}$.

2.1 Lamport diagrams

We know that behaviours of sequential systems can be described by finite or infinite words over a suitable alphabet of actions. The system has an underlying set of events and an alphabet of $\mathsf{actions}$. The actions can be thought of as labels of the $\mathsf{event\ occurrences}$. A word over such an alphabet represents a behaviour of the system as a totally ordered sequence of actions of the system and a set of such words represents various possible behaviours of the system. Extending this intuition, Lamport [Lam78] suggested that we can use certain partial orders to represent computations of distributed systems. Since events of a distributed system are unique to a particular agent and the agents themselves are concurrent, there might be event occurrences of different agents that are totally independent of each other. Partial orders are natural structures to represent the behaviour of such systems. Lamport diagrams constitute simple representations of behaviours of such systems and a diagram represents a single (non-sequential) run of the system. They are partial orders with the underlying set of events partitioned into those of n agents in such a way that the event occurrences of every agent form a linear order. The ordering relation captures the causal dependence of event occurrences. Since we have assumed that each agent is a sequential program, the event occurrences of each agent are totally ordered.

Definition 2.1.1 *A* **Lamport diagram** *is a tuple* $D = (E, \leq, \phi)$ *where*

- E *is an at most countable set of* events.
- $\leq \subseteq (E \times E)$ *is a partial order called the* causality relation *such that for every* $e \in E$, $\downarrow e \overset{\text{def}}{=} \{e' \in E \mid e' \leq e\}$ *is finite.*
- $\phi : E \rightarrow [n]$ *is a* labelling function *which satisfies the following condition:*

  Let $E_i \overset{\text{def}}{=} \{e \in E \mid \phi(e) = i\}$ *and* $\leq_i \overset{\text{def}}{=} \leq \cap (E_i \times E_i)$*. Then, for every* $i \in [n]$, $\leq_i$ *is a total order on* E_i.

In the above definition, the relation $\leq$ captures the causal dependence of events and the relations $\{\leq_i \mid i \in [n]\}$ capture the fact that event occurrences of each agent are totally ordered. Note that the labelling function ϕ assigns a unique agent to every event and hence rules out any synchronous communication in the underlying system.

For example, consider a distributed system where a fixed finite set of clients are registered with a server that provides them access to a database. A Lamport diagram representing a behaviour of the system is depicted in Figure 2.1. There are four agents: client1 and client2 are two clients registered with the server in order to access the database. Event e_1 is an event occurrence of the agent client1

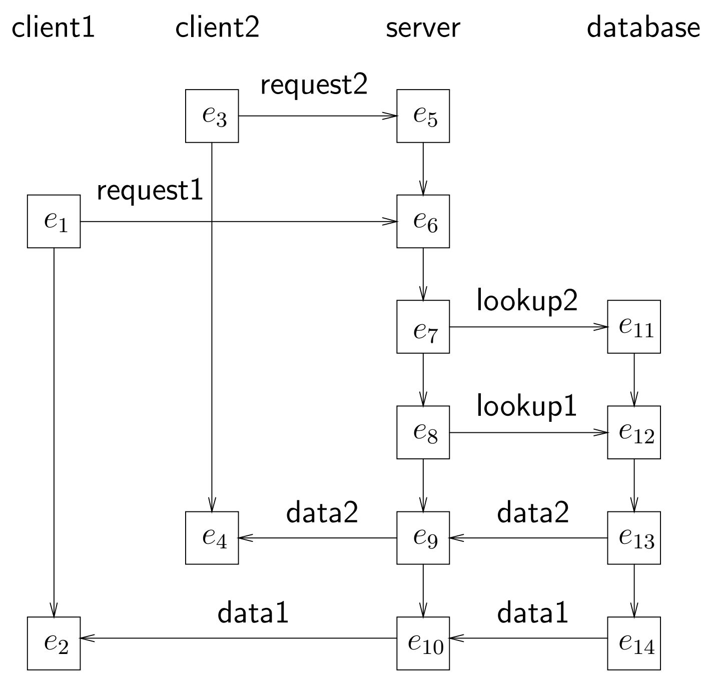


Figure 2.1: Lamport diagram representing a scenario of client—server system

corresponding to sending of the message request1 to the server. The receipt of this message by the server is represented by the event occurrence e_6. The server passes the requests to the database (represented by lookup1 and lookup2) and the response from the database (data1 and data2) is communicated back to the clients. Observe that event occurrences e_1 and e_3 corresponding to requests from client1 and client2 respectively are *concurrent*, i.e., they are not causally dependent on each other. On the other hand, the event occurrence e_5 corresponding to the receipt of the message request2 is causally dependent on e_3 corresponding to sending of that message. Similarly, for e_{13} to occur, e_7 and e_1 should have already occurred. It is in this sense that Lamport diagrams depict the causal dependence of various event occurrences.

To be precise, the relation $\leq$ is causal in the sense that whenever $e \leq e'$, we interpret this as the condition that, in any run of the system, e' cannot occur without e having occurred previously in that run. Since for all $e \in E$, $\downarrow e$ is finite, $\leq$ must be discrete. Hence there exists $\lessdot \subset \leq$, the immediate causality relation, which generates the causality relation; that is: for all e, e', e'', if $e \lessdot e'$ and $e \leq e'' \leq e'$ then $e'' \in \{e, e'\}$. We have $\leq = (\lessdot)^*$. Now consider $e \lessdot e'$. If $e, e' \in E_i$ for some $i \in [n]$, we see this as local causal dependence. However, if $e \in E_i$ and $e' \in E_j$, $i, j \in [n]$, $i \neq j$, we have remote causal dependence. For $e, e' \in E$, define $e <_c e'$ iff $e \in E_i$, $e' \in E_j$, $i \neq j$ and $e \lessdot e'$. In this case, we interpret e as the sending of a message by agent i and e' as its corresponding receipt by j. Accordingly, if $e <_c e'$ then, e will be referred to as a send event and e' will be its corresponding receive event. An event e will be interpreted as a local event if there exists no e' such that $e <_c e'$ or $e' <_c e$. With such an understanding, Lamport diagrams can be thought of as partial orders representing the underlying communication scenario of a system. Notice that the communication relation $<_c$ is derived from the Hasse diagram of the causal dependence relation which is a partial order. This rules out the presence of 'over-taking' send events. That is, there cannot be events $e_1, e_2 \in E_i$, $e'_1, e'_2 \in E_j$ where $i \neq j$ such that $e_1 \leq_i e_2$, $e'_1 \leq_j e'_2$ and $e_1 <_c e'_2$, $e_2 <_c e'_1$.

Note that given an event $e \in E$, there can be at most n events e' such that $e \lessdot e'$ and at most n events e' such that $e' \lessdot e$. In particular, if $e \in E_i$ and $e <_c e'$, $e <_c e''$ where $e' \in E_j$ and $e'' \in E_k$ for $j, k \in [n]$ such that $j \neq i$ and $k \neq i$, then e is a send event simultaneously to agents j and k. Such events can be thought of as representing broadcast type of communication where a common message is broadcast to all agents of the system. Similarly, there can be events which are simultaneous receive events from more than one agent. Also, an event e can be a send and a receive event simultaneously. For example, the events e_9 and e_{10} in the Lamport diagram given in Figure 2.1 are events which represent send and a receive actions simultaneously.

Given a Lamport diagram $D = (E, \leq, \phi)$ and $E' \subseteq E$, the Lamport diagram induced by E' is defined as $D_{E'} = (E', \leq \cap (E' \times E'), \phi \restriction E')$ where $\phi \restriction E'$ denotes the projection of ϕ on E'.

2.1.1 States of a Lamport diagram

The concept of global state in a Lamport diagram is given by the notion of a configuration, which is any downward closed set of events. That is, $c \subseteq E$ is a configuration iff for all $e \in c$, $\downarrow e \subseteq c$. For example, the set $c_1 = \{e_1, e_3, e_5, e_6\}$ is a configuration of the Lamport diagram given in Figure 2.1 whereas $c_2 = \{e_6, e_7\}$ is not a configuration as $e_1 \in \downarrow e_6$ but $e_1 \notin c_2$. Given a Lamport diagram D, let C_D^{fin} denote the set of all finite configurations of D. The empty configuration corresponds to the initial global state when no event has occurred and is denoted by $\emptyset$.

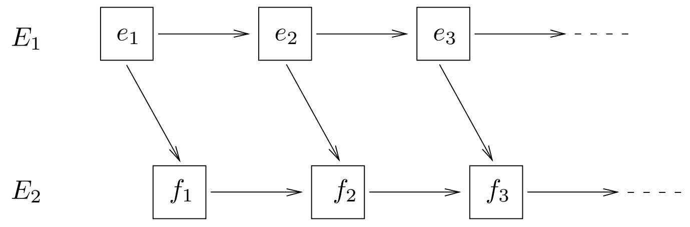


Figure 2.2: Lamport diagram of the producer — consumer problem

For each $i \in [n]$ and any finite configuration c, if $c \cap E_i \neq \emptyset$, then, there exists $e_i \in c \cap E_i$ which is the maximum with respect to $\leq$ (as $\leq_i$ is a total order on E_i). Hence, a finite configuration c can be represented by an n-tuple $(x_1, x_2, \ldots, x_n)$ where for $i \in [n]$, $x_i = e_i$ iff $c \cap E_i \neq \emptyset$ and e_i is the maximum event of E_i in c and $x_i = \perp$ otherwise. c is then given by $c = \cap_{i=1}^{n} {\downarrow} x_i$ where ${\downarrow}\perp = \emptyset$. For example, the configuration c_1 in the Lamport diagram of Figure 2.1 can be represented by the tuple $(e_1, e_3, e_6, \perp)$.

Let $e \in E_i$. Note that ${\downarrow}e$ is a configuration, and we can think of ${\downarrow}e$ as the local state of agent i when the event e has just occurred. This state contains the information that i has up till that instant in the computation, which contains it own local history and that of other agents according to the latest communication from them. The empty set corresponds to the local initial state, where no i-event has occurred, and is denoted by ϵ_i. Let the set of all local states of agent i be denoted $LC_i \overset{\text{def}}{=} \{\epsilon_i\} \cup \{{\downarrow}e \mid e \in E_i\}$ and let $LC \overset{\text{def}}{=} \bigcup_i LC_i$. We use d, d', d_1 etc to denote local states. We can extend the $\lessdot$ relation to local states as follows: let $d_1 \in LC_j$ and $d_2 \in LC_i$; we say $d_1 \lessdot d_2$ iff $d_1 \subset d_2$ and for all $d \in LC_j$, if $d \subseteq d_2$, then $d \subseteq d_1$ as well; that is, d_1 is the last j-local state seen by i at d_2. Therefore, $\epsilon_j \lessdot \epsilon_i$ for all $j \neq i$.

Sequentializations

Given a Lamport diagram $D = (E, \leq, \phi)$, a sequentialization of D is any sequence $\sigma = e_0 e_1 \ldots$ such that $E = \{e_0, e_1, \ldots\}$ and for all $k \geq 0$, ${\downarrow}e_k \subseteq \{e_0, \ldots, e_k\}$; that is, σ is a linear order that respects $\leq$.

For example, consider the Lamport diagram D corresponding to the producer-consumer problem given in Figure 2.2. The set of events E_1 of agent 1 represent the producer and the set E_2 of events of agent 2 represent the consumer and agent 1 keeps sending messages to agent 2. For all $k \geq 1$, e_k is a send event from agent 1 to agent 2 and f_k is its corresponding receive event. The sequence $\sigma_1 = e_1 f_1 e_2 f_2 \dots e_k f_k \dots$ is a sequentialization of D whereas the sequence $\sigma_2 = f_1 e_1 e_2 f_2 \dots$ is not a sequentialization as $\downarrow f_1 \not\subseteq \{f_1\}$.

We now argue that a Lamport diagram can be represented by the set of all its sequentializations. This defines a language of sequences of events of the Lamport diagram. Similarly, a collection of Lamport diagrams can be represented by the set of all sequentializations of each Lamport diagram in the collection.

Proposition 2.1.2 *A Lamport diagram can be represented by the set of all its sequentializations.*

Proof: Consider the set S of all sequentializations of some Lamport diagram $D = (E, \leq, \phi)$. Every sequentialization in the set S is an infinite string over E, so in order to show that D can be fully represented by S, it suffices to show that the causal order $\leq$ of D can be fully recovered from the set S. Fix a sequentialization $\sigma \in S$ and consider two distinct events e_1 and e_2 in σ such that e_1 occurs before e_2 in σ. If e_1 occurs before e_2 in every other sequentialization in S then it is easy to see that $e_1 \leq e_2$ in D. For otherwise, we know that either $e_2 \leq e_1$ or e_1 and e_2 are unordered in D. In the former case, it contradicts the fact that σ is a sequentialization of D and in the latter case it contradicts the fact that S is the set of *all* sequentializatons of D (as there would be a sequentialization in S where e_2 occurs before e_1 if they are unordered in D).

If there is at least one sequentialization in S (obviously different from σ) in which e_2 occurs before e_1, then we can again argue as above that e_1 and e_2 are not ordered by $\leq$ in D. Hence they are concurrent in the Lamport diagram D. $\square$

Sequentializations of a Lamport diagram induce the notion of a *buffer* between agents of the system. The buffer records the sequence of "pending sends" between every pair of agents for every prefix of the given sequentialization. For example, the sequentialization σ_1 of D given above is 1-*bounded* as there is at most one send event e_k without the corresponding receive event f_k in every prefix of σ_1. The sequentialization $\sigma_3 = e_1 e_2 f_1 f_2 \dots$ is at least 2-*bounded* as the prefix $e_1 e_2$ of σ_3 has two send events e_1 and e_2 without their corresponding receive events f_1 and f_2. We interpret e_1 and e_2 as *pending send events* at the prefix e_1e_2 of σ_3. A sequentialization where the number of pending send events grows unbounded as we consider prefixes of increasing length is *unbounded*.

The presence of such bounded and unbounded sequentializations of the same underlying Lamport diagram can lead to "non-regular" behaviours when we consider the behaviour of the diagram to be the set of all its sequentializations. For example, suppose the events of the Lamport diagram D given in Figure 2.2 were labelled by actions from a finite alphabet. Let all the event occurrences in E_1 be labelled a and all those in E_2 be labelled b. Then it is easy to see that the set of all finite sequentializations of D is the set of all words in which the number of a's greater than or equal to the number of b's, which is a context free language over the alphabet $\{a, b\}$.

We are interested in sequentializations which implicitly use a 'bounded buffer' with the hope that these will be "regular" languages. Let $\sigma = e_0e_1\ldots$ be a sequentialization of D. We say σ is b-bounded (for $b \in \mathbb{N}$) iff the following property holds:

Consider events $e_1, e_2, \ldots e_{b+1}$ and $f_1, f_2, \ldots f_{b+1}$, $e_k \in E_i$, $1 \leq_{\mathbb{N}} k \leq_{\mathbb{N}} (b+1)$ and $f_k \in E_j$, $1 \leq_{\mathbb{N}} k \leq_{\mathbb{N}} (b+1)$ where $i, j \in [n]$ with $i \neq j$ such that

1. $e_1 \leq_i e_2 \leq_i \cdots \leq_i e_{b+1}$,
2. $f_1 \leq_j f_2 \leq_j \cdots \leq_j f_{b+1}$,
3. $e_k <_c f_k$ for $1 \leq_{\mathbb{N}} k \leq_{\mathbb{N}} (b+1)$ and
4. for every k, $1 \leq_{\mathbb{N}} k \leq_{\mathbb{N}} b$, there exists no event $e \in E_i$ such that $e <_c f$ for some $f \in E_j$ such that $f_k \leq_j f \leq_j f_{k+1}$.

Then, f_b comes before e_{b+1} in σ.

That is, a b-bounded sequentialization is one in which for every pair of distinct agents i, j, at most b send events from i to j can occur in any prefix of the sequentialization without their corresponding receive events having occurred. In other words, there can be at most b send events from i to j in every prefix.

We now show that every Lamport diagram has at least one 1-bounded sequentialization.

Proposition 2.1.3 *Let $D = (E, \leq, \phi)$ be a Lamport diagram. Then, there exists at least one 1-bounded sequentialization of D.*

Proof: We will prove the existence of a 1-bounded sequentialization by constructing one. We know that E is a countable set, and so there is an enumeration of the events in E. Define a sequentialization $\sigma = e_0e_1\ldots$ as follows:

We know that there are at most n minimal events in E with respect to $\leq$. Define e_0 to be the minimum event which comes before all the other minimum events in the enumeration. Inductively, suppose we have defined $e_k, k \geq 0$ such that $\downarrow e_k \subseteq \{e_0, \ldots, e_k\}$. Consider the event e (say) such that e is the maximum i-event (with respect to $\leq_i$) in the sequence $e_0e_1\ldots e_k$ for some $i \in [n]$ and for every $j \neq i$, there exists no j-maximum event before e in $e_0e_1\ldots e_k$. (In general, for each agent, there might be a maximum event whose successor can be picked as e_{k+1}. We choose e_{k+1} to be the one which occurs before every other such event in the sequence $e_0e_1\ldots e_k$.)

As noted before, e has at most n $\lessdot$-successors.

Fix some $j \neq i$ such that $e' \in E_j$ and $e \lessdot e'$. Define $e_{k+1} = e'$. (That is, we 'schedule' a non-local successor if it is present). If there exists no such e', define e_{k+1} to be the local $\lessdot$-successor of e. Clearly, $\downarrow e_{k+1} = (\{e_{k+1}\} \cup \downarrow e_k) \subseteq \{e_0, \ldots, e_{k+1}\}$.

It follows from the inductive assumption that σ is a sequentialization. To show that σ is 1-bounded, consider events $e_1, e_2 \in E_i$, $e_1 \leq e_2$ and $e'_1, e'_2 \in E_j (j \neq i)$, $e'_1 \leq e'_2$ such that $e'_1 <_c e_1$ and $e'_2 <_c e_2$. Then, by the definition of σ, e_1 is scheduled before e'_2 and hence it is 1-bounded. $\square$

2.2 Layered Lamport diagrams

Behaviours of many distributed systems usually consist of repeated patterns of finite communication protocols. For example, consider the Lamport diagram depicting the behaviour of the producer-consumer problem in Figure 2.2. The pattern of producer sending a message (event occurrences e_i, $i \geq 0$) and the consumer receiving it (event occurrences f_i, $i \geq 0$) is repeated infinitely often. The Lamport diagram can be represented canonically by just the finite segment consisting of the send event e_i and its corresponding receive event f_i. This segment can be *composed* with itself infinitely often to generate the full Lamport diagram as in Figure 2.2.

In general, for systems whose communication patterns repeat as per a particular pattern, finite Lamport diagrams from an underlying set can be composed in a pre-defined way to obtain a full behaviour of the system. Such a composition also naturally defines a notion of *layering* in the resulting (infinite) Lamport diagram. The set of events belonging to a particular finite Lamport diagram from the underlying set constitute a *layer* and the entire Lamport diagram then becomes a *sequence of layers.*

Such a definition of composition and layering allows very general ways of structuring communications of some systems represented by Lamport diagrams. Each layer describes a finite episode of communications and a system behaviour is represented by concatenating/composing such layers.

Following this intuition, we define a class of Layered Lamport diagrams (LLDs) in such a way that every (countable) Lamport diagram can be thought of as a (countable) concatenation of finite Lamport diagrams. We refer to these finite Lamport diagrams as layers.

Definition 2.2.1 *A* **layered Lamport diagram** *is a tuple $D = (E, \leq, \phi, \lambda)$ where $(E, \leq, \phi)$ is a Lamport diagram and $\lambda : E \to \mathbb{N}$ is a* layering function *which satisfies the following conditions:*

- *for all $e \in E$, if $\lambda(e) = k$ then, for all $i \in [n]$, there exists $e' \in E_i$ such that $\lambda(e') = k$.*
- *for $e, e' \in E$, $e \leq e'$ implies $\lambda(e) \leq_{\mathbb{N}} \lambda(e')$.*
- *for every k, $\lambda^{-1}(k)$ is finite.*

Thus a layer is a finite set of events that includes at least one event of each agent, and the layering respects the causality relation.

For example, Figure 2.3 shows the Lamport diagram of the producer-consumer problem (event names are omitted) and two of its possible layerings. Notice that the first LLD in the figure is bounded (that is, every layer has boundedly many events) whereas the next one is, in general, unbounded (assuming that the number of events increases in each successive layer). Consequently, the underlying set of layers of the first LLD is also bounded but it is unbounded for the second one. We will make this precise later.

Given a layered Lamport diagram $D = (E, \leq, \phi, \lambda)$, $\lambda(E)$ is an infinite set and can be denoted by an increasing (infinite) sequence of natural numbers. More precisely, let ν_D denote the sequence of natural numbers $k_1, k_2, \ldots$ such that $\lambda(E) =$

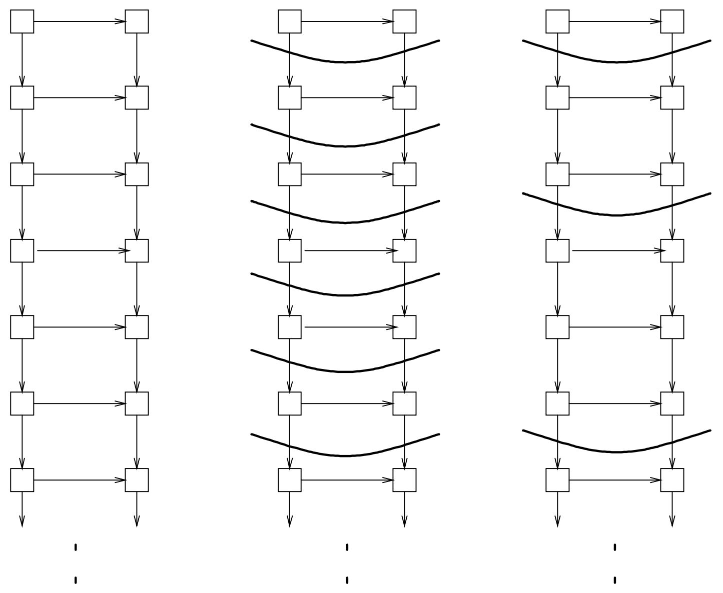


Figure 2.3: A Lamport diagram and its two possible layerings

$\{k_1, k_2, \ldots\}$ and $k_1 <_{\mathbb{N}} k_2 <_{\mathbb{N}} \ldots$. For $k \in \lambda(E)$, $\lambda^{-1}(k)$ induces a (finite) Lamport diagram which we call a layer of D and denote by D_k. Notice that a Lamport diagram which occurs as a layer has the additional property of having at least one event of each agent.

2.2.1 Types of layering

We will now look at various types of layerings that can occur in LLDs.

The first one we consider is that of communication closed LLDs. Layers in which for every send event in the layer, its corresponding receive event is also within the layer are called communication closed and we call the corresponding LLDs as communication closed LLDs.

Definition 2.2.2 *Consider a layered Lamport diagram $D = (E, \leq, \phi, \lambda)$. D is said to be* **communication closed** *if for every $e, e' \in E$ such that $e <_c e'$, $\lambda(e) = \lambda(e')$.*

For example, the second and the third LLDs described in Figure 2.3 are diagrams where the layering is communication closed. On the other hand, the layering of the

LLD representing the alternating bit protocol in Figure 2.4 is not communication closed.

When D is communication closed, the sequence of layers $D_{k_1}, D_{k_2}, \ldots$, where $\nu_D = k_1, k_2 \ldots$, completely specifies the diagram D, in the sense that D can be reconstructed as a countable *concatenation* of this sequence of diagrams. This can be important for system modelling—typical communication patterns are modelled as layers, and when the number of possible layers is finite, they constitute a finite alphabet of patterns. A layered diagram then is simply an infinite word over this finite alphabet. Just as we speak of the sequence of finite diagrams associated with D, we can conversely speak of the diagram D associated with a given countable sequence of finite diagrams. We make this precise below.

Definition 2.2.3 *Consider finite Lamport diagrams* $D_1 = (E_1, \leq_1, \phi_1)$ *and* $D_2 = (E_2, \leq_2, \phi_2)$ *with* $E_1 \cap E_2 = \emptyset$*. The* concatenation *of* D_1 *and* D_2 *is a Lamport diagram* D *denoted by* $D = D_1 \bullet D_2$ *which is defined as follows:*

$D = (E, \leq, \phi)$ *where*

- $E = E_1 \cup E_2$.
- $\phi = \begin{cases} \phi_1(e) & \textit{for } e \in E_1 \\ \phi_2(e) & \textit{for } e \in E_2 \end{cases}$
- $\leq = (\leq_1 \cup \leq_2 \cup \bigcup_{i=1}^{n} \{(e_1, e_2) \mid e_1 \in E_1 \cap E_i \textit{ and } e_2 \in E_2 \cap E_i\})^*$.

Note that the concatenation operation $\bullet$ is associative. Also, concatenation is *asynchronous* in the sense that a particular agent can execute events of the second Lamport diagram while another agent is still executing certain (concurrent) events of the first one. Now consider a communication closed LLD $D = (E, \leq, \phi, \lambda)$ with $\nu_D = k_1, k_2, \ldots$. Then D can be written as $D = D_{k_1} \bullet D_{k_2} \bullet \ldots$ where the concatenation is as defined above and the layering function λ is given by $\lambda(e) = k_i$ for all events e of D_{k_i}.

Communication closed layering of systems is a concept that has been studied at length in the context of process algebra ([EF82, PZ92]). These papers consider partial orders (representing distributed systems) that are more expressive than Lamport diagrams and layerings that may not include at least one event from every agent. They also consider *parallel composition* of layers (in addition to sequential composition presented above). This way, they model behaviours of distributed systems as partial orders with an unbounded number of agents (as the parallel composition operator concatenates the underlying partial orders in such a way that the two occur parallely, increasing the number of underlying agents). The bulk of work in these papers is in the context of representing distributed systems and in synthesizing their behaviours by composing some atomic/elementary patterns whereas we concentrate on the formal methods of such systems.

Another natural condition that can be imposed on a layered diagram is that of boundedness.

Definition 2.2.4 *Consider a layered Lamport diagram $D = (E, \leq, \phi, \lambda)$. Let $b \in \mathbb{N}$.*

1. *D is said to be b-**bounded**, if for all $k \in \lambda(E)$, $|\lambda^{-1}(k)| \leq_{\mathbb{N}} b$.*
2. *D is said to be **bounded** if there exists $b \in \mathbb{N}$ such that D is b-bounded.*

Note that when D is b-bounded, $b \geq n$, since every layer is assumed to contain at least one event per agent. The layering in the second diagram in Figure 2.3 is bounded as every layer has exactly 2 events but, the third diagram has two events in the first layer, four events in the second layer, six events in the third layer and in general, $2n$ events in the n^{th} layer and hence, is an unbounded layering.

When an LLD D is communication closed and bounded then, the number of different possible layers of D is also bounded and so D can be written as $D = D_{k_1} \bullet D_{k_2} \bullet \ldots$ where the layers D_{k_i} come from a finite alphabet of layers and $\bullet$ is as defined above.

We can also have layered diagrams with delays, whereby messages sent during one layer may be received later in another layer (i.e., the layers need not be communication closed). That is, there are events e_1, e_2 in an LLD D such that $e_1 <_c e_2$ and $\lambda(e_1) \neq \lambda(e_2)$. We call such a layering interleaved. Consider the first Lamport diagram in Figure 2.4 representing the alternating bit protocol. The layering of this diagram given in the figure is an interleaved layering. In the context of LLDs with interleaved layering, there is a different source of unboundedness, even if there is a uniform bound on the number of events in every layer. Let $e_1, e \in E_i$ such that $e_1 \leq e$. Call e_1 a pending send to j at e if there exists $e_2 \in E_j$ such that $e_1 <_c e_2$ and $\lambda(e) < \lambda(e_2)$. Now, for $e \in E_i$, and $j \neq i$, there need not be any uniform bound on the number of events $e' \leq e$ such that e' is a pending send to j. We can consider these pending sends to constitute the state of the communication *channel* from i to j

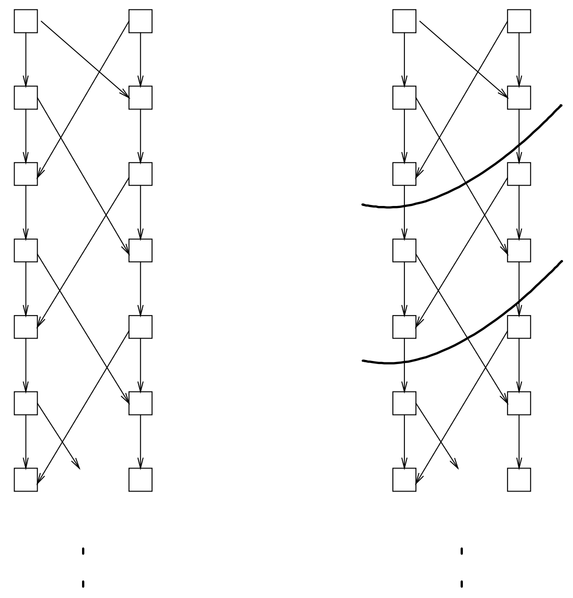


Figure 2.4: Lamport diagram of the alternating bit protocol

when the agent i is in the local state when e has just occurred. It is then reasonable to consider systems where the capacity of every such channel is uniformly bounded.

Definition 2.2.5 *Consider a layered Lamport diagram* $D = (E, \leq, \phi, \lambda)$. *Let* ν' *be a finite prefix of* ν_D *ending in* k_m*; let* $F_{\nu'} = \cup_{k_l \leq k_m} \lambda^{-1}(k_l)$*; for* $i, j \in [n]$, $i \neq j$, *define* $\delta_{\nu'}(i,j) = |\{e \in F_{\nu'} \cap E_i \mid \text{for some } e' \in (E_j - F_{\nu'}), e <_c e'\}|$. *We say that* D *is* **channel b-bounded***, if* D *is* b*-bounded and for all prefixes* ν' *of* ν_D *and* $i, j \in [n]$, $\delta_{\nu'}(i,j) \leq b$.

Thus, in a channel b-bounded LLD, for every pair of agents i, j such that $i \neq j$, at any layer, the number of send-events from i to j until this layer, for which corresponding receive-events by j have not been included yet, can be at most b. For example, the layering of the Lamport diagram representing the alternating bit protocol given in Figure 2.4 is channel-5-bounded. (Note that there are at most two pending send events at each layer. But, the first layer has 5 events. Hence, the term channel 5-bounded.)

We saw that communication closed diagrams can be presented as concatenation of finite diagrams which occur as its layers. However, in the case of diagrams which are not communication closed, a layer D_k is typically *incomplete.* 'Communication edges' across layers in the Hasse diagram of D are missing in the diagram given by

ν_D. Reconstructing D from ν_D for general diagrams requires a good deal of work. The layers are now *fragments*, which need to be tiled together, matching sends and receives appropriately to make up the original diagram. Moreover, the same event may be a pending send for several agents, and so also, a delayed receive may be from several agents. Therefore, the fragment must carry this information. However, it is easy to see that if e is a pending send for i to j in a layer, every later send event for i to j in that layer must be a pending send. Similarly, when e is a delayed receive for i from j in a layer, every preceding receive for i from j in that layer must be a delayed receive. These observations lead to the following way of reconstructing LLDs which have interleaved layering.

Definition 2.2.6 *A* **labelled fragment** *is a tuple* $F \overset{\text{def}}{=} (E, \leq, \phi, A, \eta)$ *where*

- $(E, \leq, \phi)$ *is a Lamport diagram,*
- A *is some (abstract) alphabet of labels, and*
- η *is a map that demarcates pending sends and receives in* E *as follows.*

  Let $T = (\{r, s\} \times [n] \times A)$. *Elements of* T *represent* tags. *The tag* (s, i, a) *denotes a pending send to* i *labelled* a, *and similarly* (r, j, a') *is a pending receive from* j *labelled* a'.

  Now, $\eta : E \longrightarrow (A \times 2^T)$ *associates with each event, a set of tags, subject to the following condition:*

  - *If* $\eta(e) = (a, X)$, *then* $(s, i, a') \in X$ *implies that* $e \notin E_i$ *and for every* $e' \in E$ *such that* $\phi(e) = \phi(e')$ *and* $e \leq e'$, *there does not exist* $e'' \in E_i$ *such that* $e' <_c e''$.
  - $(r, j, a') \in X$ *implies that* $e \notin E_j$ *and for every* $e' \in E$ *such that* $\phi(e) = \phi(e')$ *and* $e' \leq e$, *there does not exist* $e'' \in E_j$ *such that* $e'' <_c e$.
  - *Also,* $(s, i, a_1) \in X$ *and* $(s, i, a_2) \in X$ *implies* $a_1 = a_2$ *and similarly,* $(r, j, a_1) \in X$ *and* $(r, j, a_2) \in X$ *implies* $a_1 = a_2$.

Consider a channel b-bounded LLD D and let $\nu = k_1, k_2, \ldots$. We can now associate an alphabet A and a sequence of fragments $F_{k_1}, F_{k_2}, \ldots$ with D such that

- for any k_j, if there exists an event $e \in F_{k_j} \cap E_j$ such that $\eta(e) = (a, X)$ and $(s, i, a') \in X$, then there exists $e' \in F_{k_\ell} \cap E_i$ for some $k_\ell > k_j$ such that $e <_c e'$ and $\eta(e') = (a', Y)$ and $(r, j, a) \in Y$.

- A similar condition holds the other way also: for k_j and e as above, if $(r, i, a') \in X$, then there exists $e' \in F_{k_\ell} \cap E_i$ for some $k_\ell < k_j$ such that $e' <_c e$ and $\eta(e') = (a', Y)$ and $(s, j, a) \in Y$. (Strictly speaking, labels are not necessary for describing this correspondence, but we need them later on.)

In the case of b-bounded diagrams, a finite alphabet suffices above. We say that the sequence of fragments $F_{k_1}, F_{k_2}, \ldots$ is matched in D.

Clearly, every countable sequence of finite labelled fragments does not give rise to an LLD, since pending sends and delayed receives must be matched up appropriately. We make this precise in the definition of concatenation of fragments presented below.

Fix a finite alphabet A of labels. For an A-labelled fragment $F = (E, \leq, \phi, A, \eta)$ and $i, j \in [n]$, we define define two sequences F^s_{ij} and F^r_{ji} over A as follows:

- We know that E_i is a finite set linearly ordered by $\leq$; denote this sequence of events by $e_1 \cdots e_m$. Now $F^s_{ij} = a_{k_1} \cdots a_{k_\ell}$, where for all $p \in \{k_1, \ldots, k_\ell\}$, (s, j, a_p) is a tag in $\eta(e_p)$ and for all $p \in (\{1, \ldots, m\} \setminus \{k_1, \ldots, k_\ell\})$, there is no $a' \in A$ such that (s, j, a') is a tag in $\eta(e_p)$.

  The ordered sequence $e_{k_1}, e_{k_2}, \ldots e_{k_\ell}$ of events of E_i is called as the event sequence corresponding to F^s_{ij} and is denoted by $\mathsf{Ev}(F^s_{ij})$.

- Again, we know that E_j is a finite set linearly ordered by $\leq$; denote this sequence of events by $f_1 \cdots f_{m'}$. Now $F^r_{ji} = b_{k'_1} \cdots b_{k'_{\ell'}}$, where for all $p \in \{k'_1, \ldots, k'_{\ell'}\}$, $\eta(f_p) = (b_p, X)$ with $(r, i, b') \in X$ for some $b' \in A$ and for all $p \in (\{1, \ldots, m'\} \setminus \{k'_1, \ldots, k'_{\ell'}\})$, there is no $b' \in A$ such that (r, i, b') is a tag in $\eta(f_p)$.

  The ordered sequence $f_{k'_1}, f_{k'_2}, \ldots f_{k'_{\ell'}}$ of events of E_j is called as the event sequence corresponding to F^r_{ji} and is denoted by $\mathsf{Ev}(F^r_{ji})$.

$\mathsf{Ev}(F^s_{ij})$ records the sequence of pending send events agent i to agent j in F and $\mathsf{Ev}(F^r_{ji})$ records the sequence of unmatched receive events of agent j in F from agent i.

Consider A-labelled fragments $F_1 = (E_1, \leq_1, \phi_1, A, \eta_1)$ and $F_2 = (E_2, \leq_2, \phi_2, A, \eta_2)$. The concatenation of F_1 and F_2 is a fragment F denoted by $F = F_1 \circ F_2$ and is defined iff the following conditions are satisfied.

For all $i, j \in [n]$ such that $i \neq j$, we have

- $F^r_{1_{(ji)}} = \epsilon$ and
- $F^r_{2_{(ji)}}$ is a prefix of $F^s_{1_{(ij)}}$.

The first condition makes sure that the fragment F_1 has no unmatched receive events and the second condition ensures that the sequence of unmatched receive events of F_2 is a prefix of the sequence of unmatched send events of F_1 (so that they all can be matched up while concatenating F_1 and F_2).

Then, $F = F_1 \circ F_2$ is defined as $F = (E, \leq, \phi, A, \eta)$ where

- $E = E_1 \cup E_2$.
- $\phi = \begin{cases} \phi_1(e) & e \in E_1 \\ \phi_2(e) & e \in E_2 \end{cases}$
- $\leq = (\leq_1 \cup \leq_2 \cup \cup_{i=1}^{n} \{(e_1, e_2) \mid e_1 \in E_1 \cap E_i \text{ and } e_2 \in E_2 \cap E_i\}$
  $\cup \{(e_1, f_1), \ldots, (e_m, f_m) \mid \mathsf{Ev}(F^r_{2_{(ji)}}) = f_1, \ldots, f_m \text{ and } \mathsf{Ev}(F^s_{1_{(ij)}}) = e_1, \ldots, e_l\})^*$.
  (Since $F^r_{2_{(ji)}}$ is a prefix of $F^s_{1_{(ij)}}$, we know that $m \leq_{\mathbb{N}} l$).
- For $e \in E_k$, $k = 1, 2$, if $\eta_k(e) = (a, T)$ then $\eta(e) = (a, T')$, where $T' = \emptyset$ if $T = \emptyset$, and $T' = T - (\{t \mid t \text{ is a receive tag}\} \cup \{t \mid t = (s, j, a) \text{ where } e \in E_1,$ there exists $e' \in E_2$ such that $\phi_2(e') = j \neq \phi_1(e),\ e \leq e'\})$.

Observe that F is a fragment as well, and hence the operation of concatenation is well-defined. Note that $\circ$ is not associative. As illustrated in Figure 2.5, $(F_1 \circ F_1) \circ F_2$ is different from $F_1 \circ (F_1 \circ F_2)$. We follow the convention that $F_1 \circ F_2 \circ F_3$ denotes the fragment $((F_1 \circ F_2) \circ F_3)$, i.e., concatenation is done from left to right.

Now every interleaved LLD D can be written as $D = ((F_1 \circ F_2) \circ \ldots)$ where $F_1, F_2, \ldots$ are the fragments corresponding to the layers of D.

We now fix some notations to talk about collections of LLDs. A collection $\mathcal{L}$ of layered Lamport diagrams is said to be communication closed if every Lamport diagram in $\mathcal{L}$ is communication closed. $\mathcal{L}$ is said to be channel bounded if there exists $b \in \mathbb{N}$ such that for every $D \in \mathcal{L}$, D is channel b-bounded. Let $\mathcal{B}$ denote the

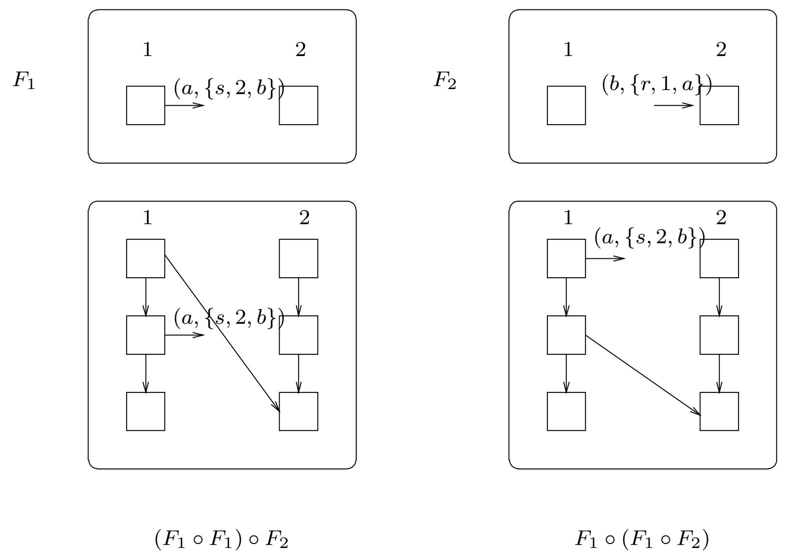


Figure 2.5: Example to show that $\circ$ is not associative

class of all bounded LLDs, $\mathcal{S}_b$ the class of channel-b-bounded LLDs and $\mathcal{C}_b$ the class of LLDs which are both b-bounded and communication closed. These classes will be of importance when we study suitable logics over layered diagrams in later chapters.

Chapter 3

Automata for distributed systems

In this chapter, we introduce some automata models for message-passing distributed systems. These automata accept Lamport diagrams and layered Lamport diagrams. We present results relating to their language emptiness problem and other closure properties with the aim of using them to solve the satisfiability problems of various logics on Lamport diagrams to be presented in later chapters. As we discuss in the chapter, the use of these automaton models as actual models of distributed message passing systems is yet to be fully answered. We present illustrative examples of systems being modelled using these automata and consider relevant model checking problems in later chapters.

3.1 System of communicating automata

The traditional high-level model for distributed systems has been that of communicating state machines. In such a model, there is a (sequential) automaton for each agent and the communicating automaton is obtained as a parallel composition of these sequential machines. Following this tradition, we propose an automaton model called System of Communicating Automata (SCA) for these systems.

As before, we fix $n > 0$ and focus our attention on n-agent systems. A distributed alphabet for such systems is an n-tuple $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, where for each $i \in [n]$, Σ_i is a finite non-empty alphabet of actions of agent i and for all $i \neq j$, $\Sigma_i \cap \Sigma_j = \emptyset$. The alphabet induced by $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ is given by $\Sigma = \bigcup_i \Sigma_i$. The set of system actions is the set $\Sigma' = \{\lambda\} \cup \Sigma$. The action symbol λ is referred to as the communication action. This is used as an action representing a communication constraint through which every receive action will be dependent on its corresponding send action. We use a, b, c etc to refer to elements of Σ and τ, τ' etc to refer to those of Σ'.

Definition 3.1.1 *A* **System of** n **Communicating Automata (SCA)** *on a distributed alphabet* $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ *is a tuple* $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ *where,*

1. *For* $j \in [n]$, Q_j *is a finite set of (local) states of agent* j. *For* $i \neq j$, $Q_i \cap Q_j = \emptyset$.
2. $G_j \subseteq Q_j$ *are the (local) good states of agent* j.
3. $Init \subseteq (Q_1 \times \ldots \times Q_n)$ *is the set of (global) initial states of the system.*
4. *Let* $Q = \bigcup\limits_i Q_i$. $\rightarrow \subseteq (Q \times \Sigma' \times Q)$ *such that if* $q \xrightarrow{\tau} q'$ *then either there exists* i *such that* $\{q, q'\} \subseteq Q_i$ *and* $\tau \in \Sigma_i$, *or there exist* $i \neq j$ *such that* $q \in Q_i, q' \in Q_j$ *and* $\tau = \lambda$.

Thus, SCAs are systems of n finite state automata with λ-labelled communication constraints between them. Note that $\rightarrow$ above is *not* a global transition relation, it consists of local transition relations, one for each agent, and communication constraints of the form $q \xrightarrow{\lambda} q'$, where q and q' are states of different agents. The latter define a coupling relation rather than a transition. The interpretation of local transition relations is standard: when the agent i is in state q_1 and reads input $a \in \Sigma_i$, it can move to a state q_2 and be ready for the next input if $(q_1, a, q_2) \in \rightarrow$. The interpretation of communication constraints is non-standard and depends only on automaton states, not on input. When $q \xrightarrow{\lambda} q'$, where $q \in Q_i$ and $q' \in Q_j$, it constraints the system behaviour as follows: whenever agent i is in state q, it puts a message whose content is q and intended recipient is j into the buffer; whenever agent j intends to enter state q', it checks its environment to see if a message of the form q from i is available for it, and waits indefinitely otherwise. If a system S has no λ constraints at all, the automata in it proceed asynchronously and do not wait for each other. We will refer to λ-constraints as 'λ-transitions' in the sequel for uniformity, but this explanation (that they are constraints not dependent on input) should be kept in mind.

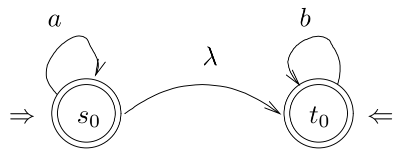


Figure 3.1: A simple SCA

We use the notation ${}^\bullet q \overset{\text{def}}{=} \{q' \mid q' \xrightarrow{\lambda} q\}$ and $q^\bullet \overset{\text{def}}{=} \{q' \mid q \xrightarrow{\lambda} q'\}$. For $q \in Q$, the set ${}^\bullet q$ refers to the set of all states from which q has incoming λ-transitions and the set $q^\bullet$ is the set of all states to which q has outgoing λ-transitions.

Global behaviour of an SCA will be defined using its set of global states. To refer to global states, we will use the set $\widetilde{Q} \overset{\text{def}}{=} (Q_1 \times \ldots \times Q_n)$. When $u = (q_1, \ldots, q_n) \in \widetilde{Q}$, we use the notation $u[i]$ to refer to q_i.

Figure 3.1 gives an SCA over the alphabet $\widetilde{\Sigma} = (\{a\}, \{b\})$. (We use $\Rightarrow$ to mark the initial states and circle the good states).

A state $q \in Q_i$ is terminal if $\{q' \mid q \xrightarrow{a} q'$ for some $a \in \Sigma_i\} = \emptyset$.

The language accepted by an SCA is a collection of Σ-labelled Lamport diagrams. We first define these labelled Lamport diagrams.

A Σ-labelled Lamport diagram is just a Lamport diagram whose events are labelled by actions from Σ.

Definition 3.1.2 *An n-agent* **Σ-labelled Lamport diagram** *is a tuple $D = (E, \leq, \phi, \Sigma)$, where*

1. *E is the set of event occurrences.*

2. *$\leq$ is a partial order on E called the causality relation such that for all $e \in E$, $\downarrow e \overset{\text{def}}{=} \{e' \mid e' \leq e\}$ is finite.*

3. *$\phi : E \to \Sigma$ is a labelling function such that the following holds:*

   Let $E_i = \{e \in E \mid \phi(e) \in \Sigma_i\}$. Then, for all $i \in [n]$, $\leq_i \overset{\text{def}}{=} \leq \cap (E_i \times E_i)$ is a total order.

Whenever Σ is evident from the context, we will refer to Σ-labelled Lamport diagrams as just labelled Lamport diagrams. Notice that the labelling function ϕ labels each event of a labelled Lamport diagram with an action instead of an agent name. The agent to which an event belongs to can be recovered from this label: for

$e \in E$, e is an event of agent i (i.e., $e \in E_i$) iff $\phi(e) \in \Sigma_i$. Since $\Sigma_i \cap \Sigma_j = \emptyset$ for all $i, j \in [n]$ such that $i \neq j$, it follows that $E_i \cap E_j = \emptyset$ for all $i \neq j$. Also notice that all the properties of Lamport diagrams mentioned in Chapter 1 hold for labelled Lamport diagrams too.

Though we will be working only with Σ-labelled Lamport diagrams in this section, we will continue to refer to them as Lamport diagrams when the context is clear.

3.1.1 Poset language of an SCA

We now formally define run of an SCA on a Lamport diagram and the poset language accepted by an SCA as the collection of Lamport diagrams on which the SCA has an accepting run.

Given an SCA S on $\widetilde{\Sigma}$, a run of S on a Lamport diagram $D = (E, \leq, \phi, \Sigma)$ is a map $\rho : C_D^{fin} \rightarrow \widetilde{Q}$ such that the following conditions are satisfied:

- $\rho(\emptyset) \in Init$.
- For $c \in C_D^{fin}$, suppose $\rho(c) = (q_1, q_2, \ldots, q_n)$. Consider $c' = (c \cup \{e\}) \in C_D^{fin}$, where $e \in E_i, e \notin c$ such that $\phi(e) = a$. Then,
  - $\rho(c') = (q'_1, q'_2, \ldots, q'_n)$ where $q'_j = q_j$ for all $j \neq i$ and $q_i \xrightarrow{a} q'_i$ in S.
  - For every $e' \in E_j, j \neq i$ such that $e' \lessdot e$, there exists $b \in \Sigma_j$ and there exists $q \in Q_j$ such that $q \xrightarrow{b} \rho(\downarrow e')[j]$ and $q \xrightarrow{\lambda} q'_i$ in S. In addition, we require that there exists a configuration $c'' \subseteq \downarrow e'$ such that $\rho(c'')[j] = q$ to make sure that a configuration including the event e' has already been labelled.
  - If $q_i{}^{\bullet} \cap Q_j \neq \emptyset$, then, there exists $e' \in E_j$ such that $e \lessdot e'$.

Thus, a run of S on D is a map from the set C_D^{fin} of configurations of D to the set of global states of S such that the following conditions hold: If c' is a configuration obtained by adding an event $e \in E_i$ (where $\phi(e) = a$) to a configuration c then, there is a transition on a from the local state of agent i in $\rho(c)$ to the local state of the same agent in $\rho(c')$ and all other local states are unaltered. In addition, if e is a receive event, we ensure that the corresponding send event has already occurred and that there is a λ-constraint into the resulting state. Similarly, if there are out-going

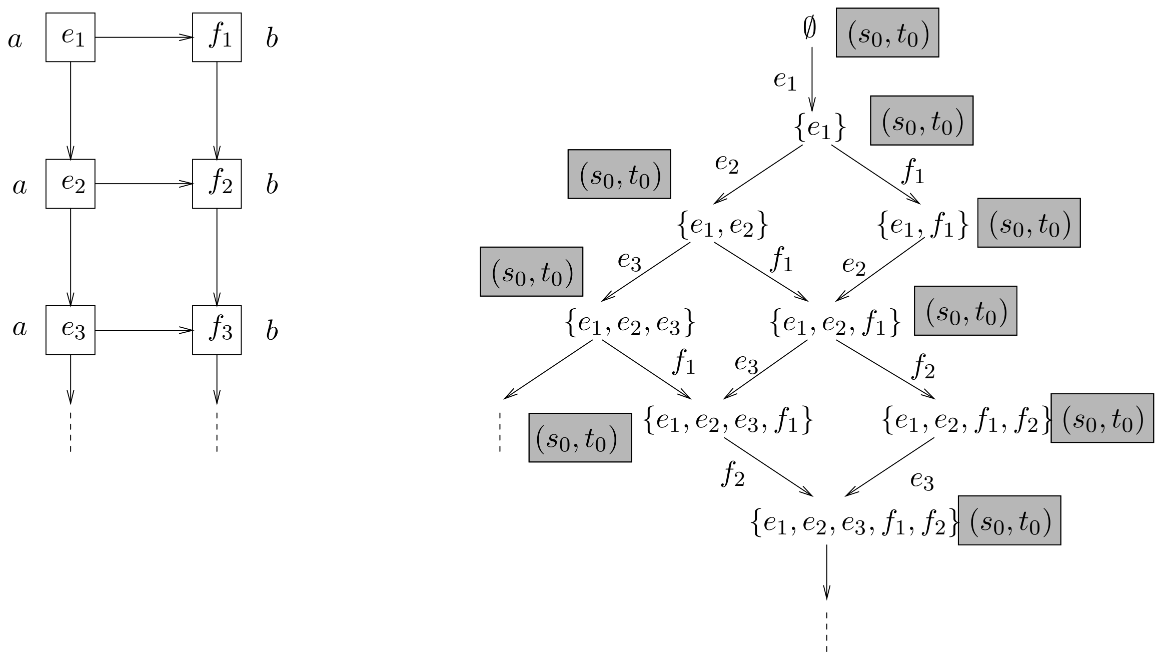


Figure 3.2: A Lamport diagram accepted by the SCA of Figure 3.1

λ-constraints from the enabling state, we make sure that the event e is a send event and that it has a matching receive event.

A run ρ is said to be good if $\forall i \in [n]$, E_i is finite implies $\rho(\downarrow e)[i]$ is a terminal state where e is the maximum event (with respect to $\leq$) in E_i.

Given a run ρ of S on D, define $Inf_i(\rho) = \{q \in Q_i \mid$ there exists infinitely many configurations $c \in C_D^{fin}$ such that $\rho(c)[i] = q\}$. ρ is said to be accepting iff ρ is good and $Inf_i(\rho) \cap G_i \neq \emptyset$ for all $i \in [n]$.

The poset language accepted by S is denoted by $\mathcal{L}^{po}(S)$ and is defined as

$\mathcal{L}^{po}(S) \overset{\text{def}}{=} \{D \mid D$ is a Σ-labelled Lamport diagram and S has an accepting run on $D\}$.

For example, Figure 3.2 gives a Lamport diagram in the poset language of the SCA given in Figure 3.1 along with its accepting run. The figure on the right hand side denotes the configuration space of the Lamport diagram. The state labels of ρ associated with each configuration (it turns out that (s_0, t_0) is the only global state) are given within shaded boxes adjacent to the configuration in the figure.

Figure 3.3 gives an SCA accepting the labelled Lamport diagram corresponding

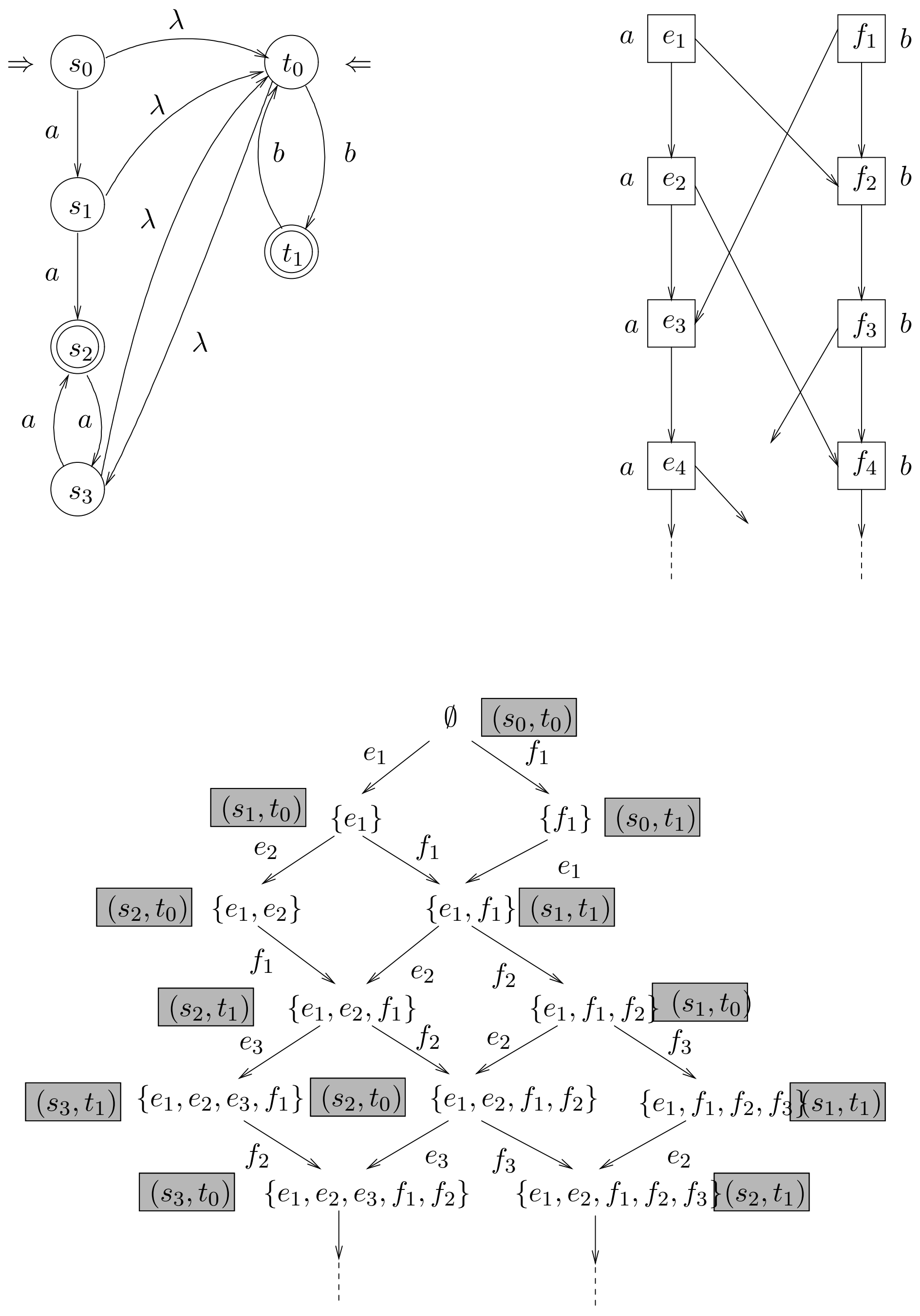


Figure 3.3: An SCA accepting the Lamport diagram corresponding to the alternating bit protocol

to the alternating bit protocol along with an accepting run. After the initial segment of the Lamport diagram (till the events e_3 and f_2 in the two agents), the labelling of the configurations by global states is such that the good states s_2 and t_1 occur infinitely often in agents 1 and 2 respectively. Hence the global run illustrated in the figure is an accepting run of the SCA on the Lamport diagram.

As mentioned before, we will be using SCAs to show decidability of the satisfiability problem of an appropriate logic to be defined later. Towards this, we now address the problem of checking if the poset language accepted by a given SCA is non-empty and show that it is decidable. A standard approach to solve the emptiness problem for sequential finite state automata is to look for strongly connected components containing a good state (in the graph of the automaton) which are reachable from one of the initial states. Towards using this approach for SCAs, we define the global automaton corresponding to an SCA by taking products of local states and including the states of buffers. But, the global automaton need not be finite-state in general as buffers can be unbounded. We then note that bounded buffers suffice; using the fact that Lamport diagrams have 1-bounded sequentializations (refer to Proposition 2.1.3), we can show that buffers of size 1 suffice for checking emptiness.

3.1.2 m-product of an SCA

Given an SCA, we define the corresponding global automaton with buffers of size at most m ($m \geq 1$) and then use the one with buffers of size 1 to show that the emptiness problem is decidable.

The global automaton corresponding to a given SCA is defined by taking the products of the local automata (represents parallel composition of sequential behaviour) and storing pending messages in buffers. We also ensure that actions corresponding to send events have appropriate actions which represent their matching receive events. We fix $m \geq 1$. Buffers will be represented as queues of length at most m and store *pending messages* between agents. There is a transition from one global state to another on an action of agent i iff there is a corresponding (local) transition on that action in the automaton of agent i. In addition, the buffers are updated depending on whether the action represents a send or a receive.

Definition 3.1.3 *Given an SCA* $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$*, the* m*-*

product *of the system is defined to be* $Pr_S^m = (X, \widetilde{I}, \widehat{G}, \Rightarrow)$ *where*

1. $X = \widetilde{Q} \times \mathcal{B}$ *where* $\mathcal{B}$ *is the set of buffers of the system defined as follows.*

   $\mathcal{B} = \{B \subseteq ([n] \times Q^*) \mid$ *if* $(i, q_1 \cdots q_l) \in B$ *then* $l \leq m$ *and* $q_1 \cdots q_l \in Q_j^*$ *for some* $j \neq i$ $\}$*. Further,* $\mathcal{B}$ *consists of exactly one pair of the form* $(i, q_1 \cdots q_l)$*,* $q_1 \cdots q_l \in Q_j^*$ *for each pair* $i, j \in [n]$ *such that* $i \neq j$*.*

2. $\widetilde{I} = (Init \times \{(i, \epsilon) \mid (i, epsilon) \in B\})$ *is the set of initial states,*

3. $\widehat{G} = (G_1, \ldots, G_n)$ *is the set of good states and*

4. *the transition relation* $\Rightarrow \subseteq (X \times \Sigma \times X)$ *is defined by:* $(q_1, \ldots, q_n, B) \overset{a}{\Rightarrow} (q'_1, \ldots, q'_n, B')$*,* $a \in \Sigma_i$*, iff*

   (a) $q_i \xrightarrow{a} q'_i$*, and for all* $j \neq i, q_j = q'_j$*.*

   (b) *If* $({}^\bullet q'_i \cap Q_j) = R \neq \emptyset$*, then there exists* $q \in R$ *and* $qw \in Q_j^*$*,* $|w| < m$ *such that* $(i, qw) \in B$ *and* $B' = (B - \{(i, qw)\}) \cup \{(i, w)\}$*.*

   (c) *If* $(q_i{}^\bullet \cap Q_j) \neq \emptyset$ *and for* $(j, w) \in B$*,* $|w| < m$ *, then* $B' = (B - \{(j, w)\}) \cup \{(j, wq_i)\}$*.*

$\mathcal{B}$ is the set of buffers of the system. There is a buffer between every distinct pair of agents i, j and hence there are totally $n(n-1)$ buffers in the system. The contents of the buffer corresponding to the pair (i, j) represents the sequences of local states of agent i which are messages to agent j. Since messages are assumed to be buffered in the FIFO order, we use sequences of local states (with the assumption that the leftmost element represents the top of the buffer) to represent buffers.

We use the notation $(i, q_1 \cdots q_l) \in B$, $q_1 \cdots q_l \in Q_j^*$, $j \neq i$ to mean that agent j has a sequence of messages $q_1 \cdots q_l$ for agent i in the buffer from j to i. Note that $l \leq m$ implies that each buffer can store at most m messages. Condition (2) ensures that whenever a local i-move is dependent on a message from agent j through a λ constraint, the particular state is there at the top of the buffer between i and j and it is utilized by the i-move. Condition (3) is to make sure agent i records its message for agent j, if any provided the corresponding buffer is not full.

Let $w = a_1 a_2 \ldots \in \Sigma^\omega$. We use the notation $w \lceil i$ to denote the restriction of w to Σ_i.

Computations of S are defined by runs of Pr_S^m on $w \in \Sigma^\omega$. An infinite run $\rho = x_0x_1\ldots$ on w is a sequence where for $k \geq 0$, $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$. For a state $x_k = (q_1, q_2, \ldots q_n, B) \in X$, we use the notation $x_k[i]$ to refer to q_i (for $i \in [n]$), $x_k[buf]$ to refer to B and for $(i, q_1 \ldots q_l) \in B$ where $q_1 \ldots q_l \in Q_j^*$, we use $x_k[buf](j,i)$ to refer to the sequence $q_1 \ldots q_l$.

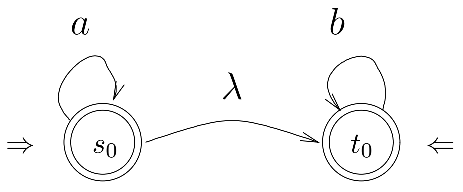
For a run ρ as above, we say that i terminates in ρ if there exists k such that $x_k[i]$ is terminal. ρ is said to be good if for all $i \in [n]$, either $w\lceil i$ is infinite or i terminates in ρ. Let $Inf_i(\rho) \overset{\text{def}}{=} \{q \in Q_i \mid$ for infinitely many $k \geq 0$, $x_k[i] = q,\ x_k[buf](i,j) = \emptyset$ for all $j \neq i\}$. The run ρ on w is said to be accepting iff ρ is good, $x_0 \in \widetilde{I}$, and for all i, $Inf_i(\rho) \cap G_i \neq \emptyset$. The string language accepted by Pr_S^m, denoted $\mathcal{L}^m(S) \overset{\text{def}}{=} \{w \in \Sigma^\omega \mid Pr_S^m$ has an accepting run on $w\}$.

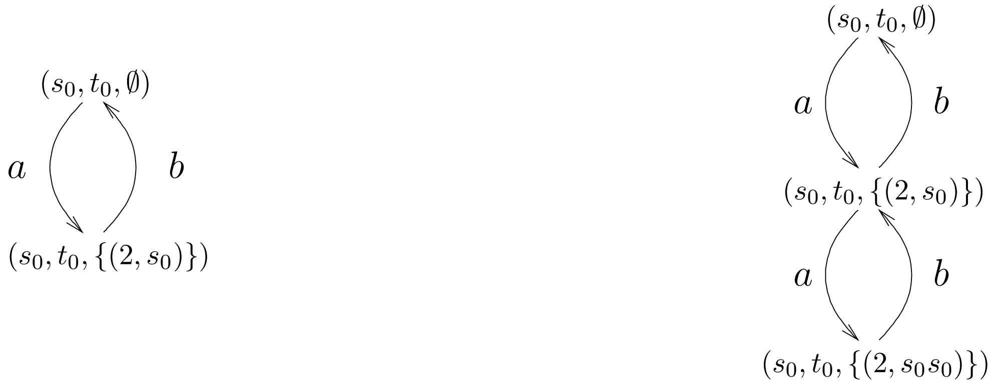
A consequence of the definition of accepting runs is that no agent gets stuck because of conditions imposed on buffers. In the definition of an accepting run, the condition that buffers between every pair of agents gets emptied infinitely often makes sure that every action representing a send event has an action representing the corresponding receive event. Figure 3.4, we present the SCA S_0 of Figure 3.1 along with its 1 and 2-products. The set of initial states in both the products is $\{(s_0, t_0, \emptyset)\}$ and the set of good states is $\{(s_0, t_0)\}$. The language accepted by the 1-product $Pr_{S_0}^1$ of S_0 is $\mathcal{L}^1(S_0) = \{(ab)^\omega\}$ and the language accepted by its 2-product $Pr_{S_0}^2$ is $\mathcal{L}^2(S_0) = (ab + aabb + abab)^\omega$.

Figure 3.5 gives another 2-agent SCA S_1 and its 1 and 2-products. The language of its 1-product is $\mathcal{L}^1(S_1) = (abb + bab)^\omega$ and its 2-product is $\mathcal{L}^2(S_1) = (abb + bab + aabbbb + ababbb + baabbb)^\omega$. Note that for both the SCAs the language accepted by the 1-product is contained in the language accepted by the 2-product.

3.1.3 Emptiness of m-product

In this section, we show that the problem of checking if the poset language accepted by a given SCA is non-empty is decidable. As mentioned earlier, given an SCA, we first construct its 1-product and using the fact that Lamport diagrams have 1-bounded sequentializations, we show that the poset language of the SCA is non-empty iff the language accepted by the 1-product of an SCA is non-empty. Since the 1-product of an SCA is a Büchi automaton whose language non-emptiness is decidable, we get decidability of the non-emptiness of the poset language accepted by the SCA.





$(s_0, t_0, \emptyset)$
a
b
$(s_0, t_0, \{(2, s_0)\})$
$(s_0, t_0, \emptyset)$
a
b
$(s_0, t_0, \{(2, s_0)\})$
a
b
$(s_0, t_0, \{(2, s_0 s_0)\})$

Figure 3.4: 1-product and 2-product of the SCA S_0 in Figure 3.1

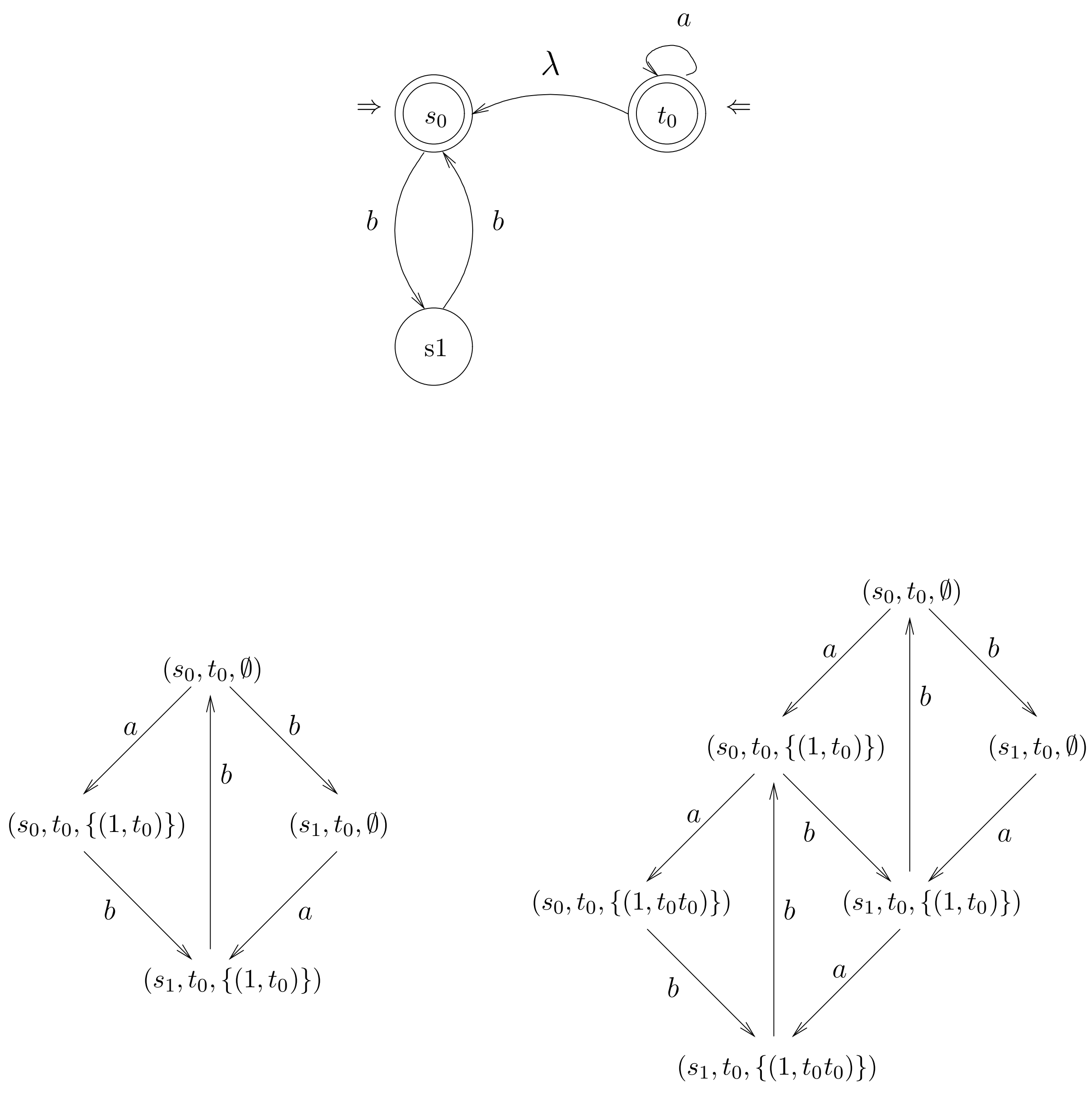

a
λ
s_0
t_0
b
b
s1
$(s_0, t_0, \emptyset)$
$(s_0, t_0, \{(1, t_0)\})$
$(s_1, t_0, \emptyset)$
$(s_1, t_0, \{(1, t_0)\})$
$(s_0, t_0, \{(1, t_0t_0)\})$
$(s_1, t_0, \{(1, t_0t_0)\})$

Figure 3.5: An SCA and its 1-product and 2-product

We first show that the emptiness problem of m-product of a given SCA is decidable. Since the m-product is basically a Büchi automaton, emptiness checking for the m-product of a given SCA can be solved by looking for strongly connected components in the underlying graph which contain a state from G_i for each $i \in [n]$ and which are reachable from an initial state. This can be done in time linear in the size of the product automaton. We thus have,

Lemma 3.1.4 *Given an SCA S of n automata, checking whether $\mathcal{L}^m(S) \overset{?}{=} \emptyset$ can be done in time $k^{O(mn)}$, where k is the maximum of $\{|Q_i| \mid i \in [n]\}$.*

Proof: Given an SCA S, consider its m-product $Pr_S^m = (X, \widetilde{I}, \widehat{G}, \Rightarrow)$. With S, we associate the directed graph $G_S = (V, E)$ with $V = X$ as the set of vertices and $E = \{(x, x') \mid \exists a \in \Sigma,\ x \overset{a}{\Rightarrow} x'\}$ as the set of edges.

A *good component* of G_S is a subset of vertices $V' \subseteq V$ which satisfies the following conditions:

1. There exists $q_0 \in \widetilde{I}$ and there exists $x \in V'$ such that x is reachable from q_0.
2. V' is a maximal strongly connected component.
3. For each $i \in [n]$, V' satisfies one of the following conditions:
   (a) There exists $x \in V'$ such that $x[i]$ is terminal and $x[i] \in G_i$ and $x[buf](j, i) = \emptyset$ for all $j \neq i$.
   (b) There exists $x, y \in V'$ and $a \in \Sigma_i$ such that $x \overset{a}{\Rightarrow} y$ and $y[i] \in G_i$ and $x[buf](j, i) = \emptyset$ for all $j \neq i$.

It is easy to check that $\mathcal{L}^m(S) \neq \emptyset$ iff G_S contains at least one good component. The maximal strongly connected components of G_S can be found in time $O(|V|^2)$. If we prove that $|V| = k^{O(mn)}$, we are done.

$|V|$ is the number of states in the m-buffered product which in turn is the product of the number of global states and the number of buffer states. We first estimate the number of buffer states. There are $n(n-1)$ buffers in the system, one for each pair (i, j), $i \neq j$, each containing at most m messages. Therefore, the buffer can be written as an $(n-1)n$-tuple; first $(n-1)$ elements are of the form $(2, x_2), (3, x_3), \ldots, (n, x_n)$, where x_j is a word over Q_1^* of length at most m. Similarly we have $(n-1)$-tuples for each of the agents. Let $|Q_i| = k_i$. Then the number of buffer states is $\Pi_i(1 + k_i + k_i^2 + \cdots + k_i^m)^{n-1} \leq n(1 + k + k^2 + \cdots + k^m)^{n-1}$, where k is the maximum of the k_i's. Therefore the total number of states is at most $(\Pi_i k_i).(n(1 + k + k^2 + \cdots + k^m)^{n-1}) = k^{O(mn)}$. $\square$

3.1.4 Emptiness of Poset Language accepted by an SCA

We now establish a 1-1 correspondence between runs of the 1-product of an SCA and Lamport diagrams in its poset language. Note that this will yield the decidability of emptiness of the poset language of an SCA.

From runs to Lamport diagrams

In this section, we show how to extract Lamport diagrams from computations of 1-products of SCAs. Consider an SCA S over $\widetilde{\Sigma}$ such that its 1-product Pr_S^1 has an (infinite) accepting run $\rho = x_0 x_1 \ldots$, on $w = a_1 a_2 \ldots \in \Sigma^\omega$, i.e., for $k \geq 0$, $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$ in Pr_S^1. We show how to associate a Lamport diagram D_ρ with ρ. Again, when $x_k = (q_1, \ldots, q_n, B)$, we use the notation $x_k[i]$ to refer to q_i, $i \in [n]$ and $x_k[buf]$ to refer to B. Further, when $(i, q) \in B, q \in Q_j, i \neq j$, we use the notation $x_k[buf](j, i) = q$. If no such q exists, then we say $x_k[buf](j, i) = \perp$. We use ρ to define a clock function $\chi : ([n] \times [n] \times \mathbb{N}) \to \mathbb{N}$ which records, for each pair of agents i, j and each instance k, the *latest* instant at which i last heard from the agent j at k. Define $\chi(i, j, k)$ by induction on k as follows:

1. For all $i \in [n]$, for all $k \in \mathbb{N}$: $\chi(i, i, k) = k$; for all $i, j \in [n]$, $\chi(i, j, 0) = 0$.
2. Let $k \geq 0$. Suppose $\chi(i, j, k)$ is defined. Let $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$, $a_{k+1} \in \Sigma_i$. Let $j \neq i$. For all $j' \in [n]$, $\chi(j, j', k + 1) = \chi(j, j', k)$. Let $\chi(i, j, k) = m$. If ${}^\bullet x_{k+1}[i] \cap Q_j = \emptyset$, then $\chi(i, j, k + 1) = m$. Otherwise, $x_k = (q_1, \ldots, q_n, B)$ and there exists $(i, q) \in B$ such that $q \in {}^\bullet x_{k+1}[i] \cap Q_j$.
   Claim: There exists a unique l such that $m < l \leq k$ and $x_l[j] = q$.
   Set $\chi(i, j, k + 1) = l$.

The claim is proved as follows. Let $(k, k+1)$ denote the r^{th} receive transition for i from j in ρ. Let $(k' - 1, k')$ similarly denote the $(r - 1)^{th}$ such transition; if $r = 1$, set $k' = 0$. By product construction, $x_{k'}[buf](j, i) = \perp$ and $x_k[buf](j, i) = q$. Let l be the least index such that $k' < l \leq k$ and $x_l[buf](j, i) \neq \perp$. Again by product construction, $x_l[j] = x_l[buf](j,i) = q' \in Q_j$, say. Now let $l \leq l' < k$; we can argue by induction on $l' - l$ that $x_{l'+1}[buf](j,i) = x_{l'}[buf](j,i)$, since no send is enabled when $x_{l'}[buf](j,i) \neq \perp$, by the product construction, and there is no receive by choice of indices and these are the only transitions that modify this component. We thus have a unique l such that $x_l[j] = q$, as required.

The following proposition follows from our choice of l for $\chi(i,j,k+1)$.

Proposition 3.1.5 *For all $k \geq 0$, for $i \neq j$, if $\chi(i,j,k) \leq k$.*

From (ρ, χ) we can extract a Σ-labelled Lamport diagram as follows. Recall that $\rho = x_0 x_1 \ldots$ and for all k, $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$, $a_{k+1} \in \Sigma$. The Lamport diagram is given by $D_\rho \overset{\text{def}}{=} (E, \preceq, \phi, \Sigma)$, where

1. $E = \{(k,k+1) \mid k \in \mathbb{N}\}$.

2. $\phi : E \to \Sigma$ is given by

   $\phi(e) = a_{k+1}$ iff $e = (k,k+1)$ and $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$ in ρ.

3. $\preceq = (\preceq_l \cup \lessdot_c)^*$, where

   (a) Let $E_i = \{e \in E \mid \phi(e) \in \Sigma_i\}$. Then,
   $\preceq_l = \bigcup_i ((E_i \times E_i) \cap \{((k,k+1),(l,l+1)) \mid k \leq l\})$.

   (b) $\lessdot_c = \{((m-1,m),(k,k+1)) \mid$ where $(m-1,m) \in E_j$, $(k,k+1) \in E_i$, $i \neq j$ and $\chi(i,j,k) < \chi(i,j,k+1) = m\}$.

It is easily seen that E_i is linearly ordered by $\preceq$ and that for all e, $\downarrow e \cap E_i$ is finite. Hence, for all e, $\downarrow e$ is finite as well. It only remains to show antisymmetry of $\preceq$. For this, first note that $\lessdot_c$ is asymmetric: whenever $(m-1,m) \lessdot_c (k,k+1)$, by the proposition above, $m < k+1$. Hence $\preceq$ cannot have any cycle that contains a $\lessdot_c$ edge; such a cycle must be composed of i-edges, for some i, violating the fact that E_i is linearly ordered by $\preceq$.

Thus, with each infinite run ρ of Pr_S^1, we can associate a Lamport diagram D_ρ. We use this below to establish a 1-1 correspondence between accepting runs of 1-product and Lamport diagrams in the poset language of S. The following result shows that every poset accepted by an SCA is generated by an accepting run of the 1-buffered product and vice versa.

Lemma 3.1.6 *Consider an SCA $S = ((Q_1, G_1), \dots, (Q_n, G_n), \rightarrow, Init)$ over the alphabet $\widetilde{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$. Then, $\mathcal{L}^{po}(S) = \{D_\rho \mid \rho$ is an accepting run of the 1-product of $S\}$.*

Proof: Consider D_ρ where $\rho = x_0 x_1 \dots$ is an accepting run of the 1-buffered product. To show that $D_\rho \in \mathcal{L}^{po}(S)$, we have to define a run $\rho' : C^{fin}_{D_\rho} \rightarrow \widetilde{Q}$ and show that it is accepting.

Define $\rho' : C^{fin}_{D_\rho} \rightarrow \widetilde{Q}$ inductively as follows: $\rho'(\emptyset) = x_0\lceil[n]$. (If $x = (q_1, \dots, q_n, B)$, we use the notation $x\lceil[n]$ to denote the tuple $(q_1, \dots q_n)$, the projection to the set of global states of S.)

Now, suppose $\rho'(c)$ is defined. Consider $c' = c \cup \{e\}$ where $e = (k, k+1) \in E_i$, such that $e \notin c$. Then, $\rho'(c') = x_{k+1}\lceil[n]$.

We now show that ρ' is an accepting run of S on D_ρ. Clearly, $\rho'(\emptyset) \in Init$ as $x_0 \in Init$. Suppose $\rho'(c) = (q_1, \dots, q_n)$ and $\rho'(c') = (q'_1, \dots q'_n)$ where c, c' are as before. Then, by the definition of ρ', we know that $q_i \overset{\phi(e)}{\rightarrow} q'_i$ and $q_j = q'_j$ for all $j \neq i$. Now, suppose there exists $e' = (m-1, m) \in E_j$ such that $e' \lessdot e$. By the definition of $\leq$ in D_ρ, it follows that $\chi(i, j, k) < \chi(i, j, k+1) = m$. Hence, there exists $q \in Q_j$ such that $q \overset{\lambda}{\rightarrow} q'_i$ and ρ being an accepting run of the 1-buffered product implies that $\rho(\downarrow e')[j] = q$. Now if $q_i{}^\bullet \cap Q_j \neq \emptyset$ then, since buffers are emptied infinitely often in accepting runs of the 1-product, we know that there exists an i-event f such that $e \leq f$. Define e' to be the event which is minimal (with respect to $\leq$) among all such events f. Clearly, $e \leq e'$ and ρ' is a run of S on D_ρ. Also, ρ' is accepting because ρ is.

Conversely, consider any $D \in \mathcal{L}^{po}(S)$. Let ρ' be an accepting run of S on D. Consider $w = \phi(e_1)\phi(e_2)\dots$ where $\sigma = e_1 e_2 \dots$ is any 1-bounded sequentialization of D. We define a run of the 1-buffered product of S and show that it is an accepting run on w.

Define $\rho = x_0 x_1 \dots$ where $x_i, i \geq 0$ is defined inductively as follows: $x_0 = (\rho'(\emptyset), \emptyset)$. Suppose $x_k = (\rho'(c), B)$ is defined. Consider the event e_{k+1} in σ. Let $c' = c \cup \{e_{k+1}\}$. c' is a configuration because σ is a sequentialization of events in D_ρ. Define $x_{k+1} = (\rho'(c'), B')$ where $B' = B \setminus (\{(i, \rho'(\downarrow e')[j]) \mid$ for all $e' \in E_j$, $(j \neq i)$ such that $e' \lessdot e_{k+1}\}) \cap \{(j, \rho'(c'))[j] \mid$ there exists $e \in E_j$, $(j \neq i)$ such that $e_{k+1} \leq e\}$. It can be easily proved that ρ is an accepting run of Pr^1_S on w. Clearly, the Lamport diagram D is generated from ρ. $\square$

Thus $\mathcal{L}^{po}(S) \neq \emptyset$ iff $\mathcal{L}^{1}(S) \neq \emptyset$. Hence, by Lemma 3.1.4, we have,

Theorem 3.1.7 *Given an SCA S of n automata, checking whether $\mathcal{L}^{po}(S) \neq \emptyset$ can be done in time $k^{O(n)}$, where k is the maximum of $\{|Q_i| \mid i \in [n]\}$.*

3.2 Closure properties of SCA

To investigate the use of SCAs as robust models of distributed message passing systems, it would be useful to prove that the class of string and poset languages they accept enjoy all the usual closure properties, namely being closed under union, intersection, complementation etc. In traditional automata theory, studies are also done in terms of defining suitable concatenation and iteration operations so that algebraic properties of the corresponding languages can be investigated.

It can be easily shown that languages accepted by SCAs are closed under the operations of union and intersection as we do in this section. However, proving/disproving closure under complementation seems non-trivial. A traditional approach to show closure under complementation is to first show that the class of automata are determinizable and then use the equivalent deterministic automaton to obtain an automaton that accepts the complement language. However, it is well-known that this technique does not work as such for the class of ω-regular languages accepted by non-deterministic Büchi automata.

The problem is little more difficult in the context of SCAs. When we consider the question of an SCA being deterministic, there seem to be various syntactic restrictions that we can impose on the structure of the transitions to obtain the semantic notion of determinism, namely that of there being an unique run on every input. Conditions like each of the local transitions being deterministic do not suffice as the λ-constraints can be exploited to violate determinism. In general, closure under complementation seems hard to prove.

A similar problem persists in the context of defining concatenation and iteration operations. The local behaviour of each component automaton can be captured using the usual Kleene operations of concatenation and iteration. To capture the notions of remote causal independence and concurrency (modelled using λ-transitions in SCAs), we investigated some *shuffle* (parallel composition) operations in vain. The main difficulty seems to stem from the fact that concurrency in SCAs is 'dynamic' in the sense that it is completely determined by the λ-transitions and not dictated by the underlying distributed alphabet.

We now prove that the class of string languages accepted by m-buffered products are effectively closed under union and intersection. A similar construction would work when we consider poset languages over SCAs. The proof follows the standard approach to show the corresponding results for ω-regular languages.

Theorem 3.2.1 *Let S_1 and S_2 be two SCAs over $\widetilde{\Sigma}$. Then, there exists an SCA S such that $\mathcal{L}^m(S) = \mathcal{L}^m(S_1) \cup \mathcal{L}^m(S_2)$ and the size of S is $O(n_1 + n_2)$ where n_l is the size of S_l for $l \in \{1, 2\}$.*

Proof: Let $S_l = ((Q_1^l, G_1^l), \ldots, (Q_n^l, g_n^l), \rightarrow_l, Init_l)$ for $l \in \{1, 2\}$ where $Q_i^1 \cap Q_i^2 = \emptyset$ for all i, $1 \leq i \leq n$. Define $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ where

- $Q_i = Q_i^1 \cup Q_i^2$ for $1 \leq i \leq n$,
- $G_i = G_i^1 \cup G_i^2$ for $1 \leq i \leq n$,
- $Init = Init_1 \cup Init_2$, and
- $\rightarrow = \rightarrow_1 \cup \rightarrow_2$.

It is easy to verify that the SCA S is such that $\mathcal{L}^m(S) = \mathcal{L}^m(S_1) \cup \mathcal{L}^m(S_2)$. Also, the SCA S has $O(n_1 + n_2)$ global states as the set of states of S is the union of those of S_1 and S_2. $\square$

Theorem 3.2.2 *Let S_1 and S_2 be two SCAs over $\widetilde{\Sigma}$. Then, one can effectively construct an SCA S such that $\mathcal{L}^m(S) = \mathcal{L}^m(S_1) \cap \mathcal{L}^m(S_2)$. Moreover, the size of S (the number of global states) is $O(2^n.n_1.n_2)$ where n_l is the size of S_l for $l \in \{1, 2\}$.*

Proof: Let $S_l = ((Q_1^l, G_1^l), \ldots, (Q_{n}^l{}_{,n}^{\,l}), \rightarrow_l, Init_l)$ for $l \in \{1, 2\}$. Define

$$S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$$

where

- $Q_i = Q_i^1 \times Q_i^2 \times \{1, 2\}$ for $1 \leq i \leq n$.
- $G_i = Q_i^1 \times G_i^2 \times \{2\}$ for $1 \leq i \leq n$.

- $Init \subseteq Q_1 \times Q_2 \times \cdots \times Q_n$ is defined as:

  $((q_1, p_1, x_1), \ldots, (q_n, p_n, x_n)) \in Init$ iff $(q_1, \ldots, q_n) \in Init_1$, $(p_1, \ldots, p_n) \in Init_2$ and $x_i = 1$ for each i.

- $\rightarrow \subseteq Q \times \Sigma \times Q$ is defined by: $(q, p, x) \xrightarrow{a} (q', p', y)$ iff

  1. $q \xrightarrow{a}_1 q'$ and $p \xrightarrow{a}_2 p'$.
  2. If $q, q' \in Q^1_i$ for some i then, $(x = 1 \Rightarrow y = 2)$ iff $q' \in G^1_i$.
  3. If $p, p' \in Q^2_j$ for some j then, $(x = 2 \Rightarrow y = 1)$ iff $q' \in G^2_j$.

It is easy to verify that the SCA S is such that $\mathcal{L}^m(S) = \mathcal{L}^m(S_1) \cap \mathcal{L}^m(S_2)$. Also, the SCA S has $O(2^n.n_1.n_2)$ global states as each product state carries an extra tag indicating whether the automaton is checking for a good state on the first or the second component. $\square$

3.3 Diagram automata

In Section 2.2, we defined layered Lamport diagrams as models representing the behaviour of systems which are represented as a sequential concatenation of parallel behaviour. We now define automata models for such systems. These automata take layered Lamport diagrams as input and run on their sequence of layers (finite Lamport diagrams) which constitute the finite alphabet of these automata. Note that these automata are 'dual' to SCAs—SCAs model parallel composition of sequential behaviour whereas automata over LLDs model sequential composition of parallel behaviour.

We define automata running over communication closed and bounded LLDs and over channel bounded LLDs. The former class of automata are called *diagram automata* and the latter are refered to as *fragment automata*. Communication closed and bounded LLDs have finite Lamport diagrams as their layers and these constitute the alphabet of diagram automata. Since fragment automata run over channel bounded LLDs, their alphabet is a set of fragments which are the layers of channel bounded LLDs.

We first introduce diagram automata and talk about their emptiness problem and closure properties. We fix $b \in \mathbb{N}$ for the rest of this section. Diagram automata take communication closed and b-bounded LLDs as input and run on their underlying set of layers. Let $\mathcal{LC}_b$ denote the set of b-bounded communication closed layers i.e., Lamport diagrams which have at most b events; it is clearly a finite set upto isomorphism.

A diagram automaton is nothing but a Büchi automaton whose alphabet is a finite set of b-bounded Lamport diagrams. We elaborate its definition below:

Definition 3.3.1 *A* **diagram automaton** *is given by a tuple* $\mathcal{A} = (Q, \mathcal{D}, \rightarrow, I, G)$ *where*

1. *Q is a finite set of states,*
2. *$\mathcal{D} \subseteq \mathcal{LC}_b$ is the alphabet of the automaton,*
3. *$I \subseteq Q$ is the set of initial states,*
4. *$G \subseteq Q$ is the set of good states and*
5. *$\rightarrow \subseteq (Q \times \mathcal{D} \times Q)$ is the transition relation.*

A diagram automaton takes a b-bounded and communication closed LLD as input. We know from Chapter 2 that such an LLD can be uniquely represented by its sequence of layers. These layers come from the alphabet of the diagram automaton and the automaton runs on the input sequence of layers like a Büchi automaton.

Consider a sequence of states of $\mathcal{A}$, $\rho = q_0, q_1, \ldots$ such that $q_0 \in I$ and $q_i \xrightarrow{D_i} q_{i+1}$ for all $i \geq 0$. Let D be the b-bounded communication closed LLD defined by the sequence $D_0, D_1, \ldots$. We say that ρ is a *run* of $\mathcal{A}$ on D. ρ is said to be *accepting* if $inf(\rho) \cap G \neq \emptyset$, where $inf(\rho) = \{q \mid q = q_i$ for infinitely many $i\}$. The *language of b-bounded communication closed LLDs* accepted by $\mathcal{A}$ is denoted by $L^b_C(\mathcal{A})$ and is defined as $L^b_C(\mathcal{A}) = \{D \mid$ there exists an accepting run of $\mathcal{A}$ on $D\}$.

For example, Figure 3.6 gives a diagram automaton and the 2-bounded and communication closed LLD (corresponding to the producer-consumer problem) accepted by it. The automaton has a single state q_0 which is also its initial and good state and its transition is labelled by the 2-bounded Lamport diagram which occurs as the layer of the producer-consumer LLD.

Since diagram automata are Büchi automata running on a finite alphabet of layers, it is not difficult to see they enjoy all the closure properties of Büchi automata.

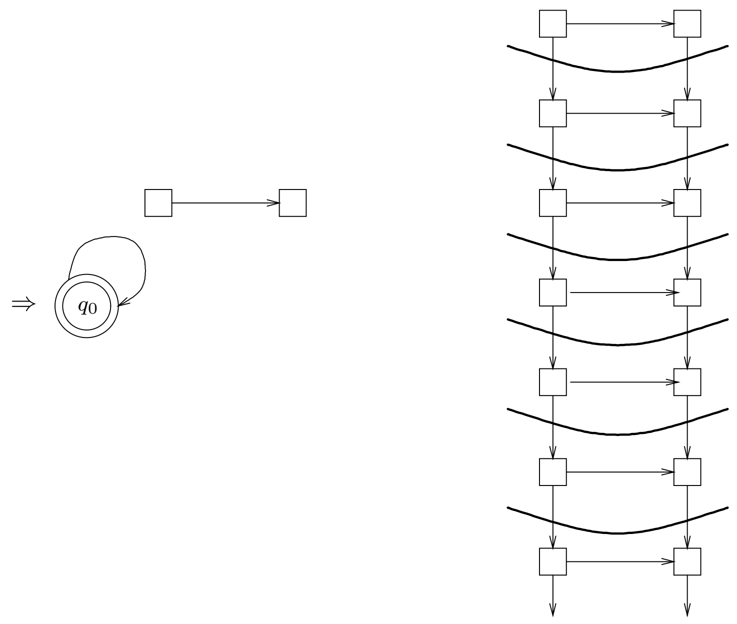


Figure 3.6: Diagram automaton over the LLD representing producer-consumer problem

We first show that the problem of checking if the language of communication closed and bounded LLDs accepted by a given diagram automaton is decidable. Not surprisingly, it turns out that the algorithm to check for emptiness is the same as that of Büchi automata: it suffices to check for the existence of a strongly connected component containing a good state which is reachable from one of the initial states. We present the details below.

Theorem 3.3.2 *Given a diagram automaton $\mathcal{A} = (Q, \mathcal{D}, \rightarrow, I, G)$, the problem of checking if $L^b_C(\mathcal{A}) \neq \emptyset$ is decidable in time $O(|Q|^2)$.*

Proof: We will show that $L^b_c(\mathcal{A}) \neq \emptyset$ iff there exists a strongly connected component in the graph of $\mathcal{A}$ such that the following conditions are satisfied:

- The strongly connected component contains a good state of $\mathcal{A}$.
- It is reachable from one of the initial states of $\mathcal{A}$.

Thus, checking if $L^b_c(\mathcal{A}) \neq \emptyset$ would amount to checking for the existence of a strongly connected component with the above properties and this can be done in time $O(|Q|^2)$.

Suppose $L^b_c(\mathcal{A}) \neq \emptyset$. Let $D = (E, \leq, \phi, \lambda) \in L^b_c(\mathcal{A})$ with $\nu_D = 0, 1, \ldots$. Also, let $\rho = q_0, q_1, \ldots$ be an accepting run of $\mathcal{A}$ on D. Since ρ acts as a run of the underlying Büchi automaton over the alphabet of b-bounded layers, it follows from the decidability of emptiness of Büchi automata that there exists a strongly connected component containing a good state in the graph of $\mathcal{A}$ such that it is reachable from an initial state.

Conversely, suppose the graph of $\mathcal{A}$ has a strongly connected component with the required properties. Considering the path of the strongly connected component from the initial state and unwinding the strongly connected component, we get an infinite sequence $D_0, D_1, \ldots$ of b-bounded Lamport diagrams and a sequence of states $q_0, q_1, \ldots$ such that $q_o \in I$ and $q_i \xrightarrow{D_i} q_{i+1}$ for all $i \geq 0$.

Define a layered Lamport diagram $D = (E, \leq, \phi, \lambda)$ by concatenating successive layers in the sequence, i.e., the underlying Lamport diagram D is given by $D \overset{\text{def}}{=} D_0 \bullet D_1 \bullet \ldots$ where $\bullet$ is as defined in Section 2.2 and the layering function λ is given by $\lambda(e) = i$ for every event e of D_i. The fact that D is a Lamport diagram follows from the definition of $\bullet$. Consider events e, e' in D such that $e \leq e'$. Then, either e, e' are events of D_i for some i or e is an event of D_i and e' is an event of D_j for some $j \geq i$. Either way $\lambda(e) \leq_{\mathbb{N}} \lambda(e')$. Also, for every i, $\lambda^{-1}(i)$ is finite and contains at least one event of every agent as each D_i is a finite layer. Hence λ is a layering and so, D is an LLD. Clearly, the sequence ρ as above is an accepting run of $\mathcal{A}$ on D and hence $D \in L^b_C(\mathcal{A})$. $\square$

3.3.1 Closure properties

We now show that the class of communication closed and bounded LLDs accepted by diagram automata are closed under boolean operations. As explained earlier, we exploit the fact that diagram automata are basically Büchi automata running over a finite alphabet of layers and use the fact that the class of ω-regular languages accepted by Büchi automata are closed under boolean operations.

It is easy to see that diagram automata are effectively closed under union and intersection. The techinque is exactly the same as that of Büchi automata.

Theorem 3.3.3 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two diagram automata. Then, there exists a diagram automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.*

Proof: Let $\mathcal{A}_i = (Q_i, D, \rightarrow_i, I_i, G_i)$ be such that $Q_1 \cap Q_2 = \emptyset$. Define $A = (Q, D, \rightarrow, I, G)$ where

- $Q = Q_1 \cup Q_2$,
- $I = I_1 \cup I_2$,
- $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ and
- $G = G_1 \cup G_2$.

It is easy to see that $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$. $\square$

Theorem 3.3.4 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two diagram automata. Then, there exists a diagram automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.*

Proof: Let $\mathcal{A}_i = (Q_i, D, \rightarrow_i, I_i, G_i)$ be such that $Q_1 \cap Q_2 = \emptyset$. Define $A = (Q, D, \rightarrow, I, G)$ where

- $Q = Q_1 \times Q_2 \times \{1, 2\}$,
- $I = I_1 \times I_2 \times \{1\}$,
- $G = Q_1 \times G_2 \times \{2\}$ and
- $\rightarrow$ is defined as follows: For $a \in \Sigma$, $q_1 \in Q_1$ and $q_2 \in Q_2$, we have
  - $(q_1, q_2, 1) \xrightarrow{a} (q_1', q_2', 1)$ iff $q_1 \xrightarrow{a} q_1'$, $q_2 \xrightarrow{a} q_2'$ and $q_1 \notin G_1$,
  - $(q_1, q_2, 1) \xrightarrow{a} (q_1', q_2', 2)$ iff $q_1 \xrightarrow{a} q_1'$, $q_2 \xrightarrow{a} q_2'$ and $q_1 \in G_1$,
  - $(q_1, q_2, 2) \xrightarrow{a} (q_1', q_2', 2)$ iff $q_1 \xrightarrow{a} q_1'$, $q_2 \xrightarrow{a} q_2'$ and $q_2 \notin G_2$ and
  - $(q_1, q_2, 2) \xrightarrow{a} (q_1', q_2', 1)$ iff $q_1 \xrightarrow{a} q_1'$, $q_2 \xrightarrow{a} q_2'$ and $q_2 \in G_2$.

It is easy to see that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. $\square$

The following functions will be useful in moving back and forth between diagram automata and Büchi automata. Let Σ be a finite alphabet and let $h : \mathcal{LC}_b \rightarrow \Sigma$ be a bijection. Let C_b denote the class communication closed and bounded LLDs. We first extend h to $\widehat{h} : C_b \rightarrow \Sigma^\omega$ as follows: For $D = D_0 \bullet D_1 \bullet \ldots \in C_b$, $\widehat{h}(D) = a_0 a_1 \ldots$

where $a_i = h(D_i)$ for $i \geq 0$. We further extend $\widehat{h}$ to $\widetilde{h} : 2^{C_b} \to 2^{\Sigma^\omega}$ as follows: For $L \in 2^{C_b}$, $\widetilde{h}(L) = \{\widehat{h}(D) \mid D \in L\}$. Using the fact that h is a bijection, it is easy to see that $\widehat{h}$ and $\widetilde{h}$ are also bijections and consequently, their inverse functions also exist. We will be using this fact below to show that we can move back and forth between diagram automata and Büchi automata.

Lemma 3.3.5 *1. Given a diagram automaton $\mathcal{A}$ over $\mathcal{LC}_b$, there exists a Büchi automaton $\mathcal{B}_\mathcal{A}$ over Σ such that $\widetilde{h}(L(\mathcal{A})) = L(\mathcal{B}_\mathcal{A})$.*

2. Given a Büchi automaton $\mathcal{B}$ over Σ, there exists a diagram automaton $\mathcal{A}_\mathcal{B}$ over $\mathcal{LC}_b$ such that $L(\mathcal{A}_\mathcal{B}) = \widetilde{h}^{-1}(L(\mathcal{B}))$.

Proof:

1. Let $\mathcal{A} = (Q, \mathcal{LC}_b, \rightarrow, I, G)$. Define $\mathcal{B}_\mathcal{A} = (Q, \Sigma, \Rightarrow, I, G)$ where $q \overset{a}{\Rightarrow} q'$ for $q, q' \in Q$ and $a \in \Sigma$ iff $q \overset{h^{-1}(a)}{\longrightarrow} q'$. It is then easy to see that $\widetilde{h}(L(\mathcal{A})) = L(\mathcal{B}_\mathcal{A})$.

2. Let $\mathcal{B} = (Q, \Sigma, \Rightarrow, I, G)$. Define $\mathcal{A}_\mathcal{B} = (Q, \mathcal{LC}_b, \rightarrow, I, G)$ where $q \overset{D}{\rightarrow} q'$ iff $q \overset{h(D)}{\Rightarrow} q'$. It is again easy to see that $L(\mathcal{A}_\mathcal{B}) = \widetilde{h}^{-1}(L(\mathcal{B}))$.

□

Note that the above lemma gives another proof of the decidability of emptiness of diagram automata. Given a diagram automaton $\mathcal{A}$, we fix a finite alphabet Σ and a bijection $h : \mathcal{LC}_b \to \Sigma$ as above. We can show that $L(\mathcal{A}) \neq \emptyset$ iff $L(\mathcal{B}_\mathcal{A}) \neq \emptyset$ and since emptiness problem of Büchi automata is decidable, it follows that emptiness problem of diagram automata is also decidable. The same lemma can also be used to show that the class of languages accepted by diagram automata are closed under boolean operations giving alternate proofs of Theorem 3.3.3 and Theorem 3.3.4. We can also use the above lemma to show that the class of languages accepted by diagram automata enjoy other closure properties as that of ω-regular languages like being closed under operations of projection, complementation etc. We will be using these results in Chapter 6.

We now show that the class of languages accepted by diagram automata are closed under complementation by using the above lemma.

Theorem 3.3.6 *Given a diagram automaton $\mathcal{A}$ over $\mathcal{LC}_b$, there exists a diagram automaton $\mathcal{A}^C$ over $\mathcal{LC}_b$ such that $L(\mathcal{A}^C) = L(\mathcal{A})^C$ where $L(\mathcal{A})^C$ denotes the complement of the language $L(\mathcal{A})$ over $\mathcal{LC}_b$.*

Proof: Fix a finite alphabet Σ and consider a bijection $h : \mathcal{LC}_b \to \Sigma$. We can now define the extension functions $\widehat{h}$ and $\widetilde{h}$ as above. Given $\mathcal{A}$, we first use Lemma 3.3.5 to construct the corresponding Büchi automaton $\mathcal{B}_{\mathcal{A}}$ such that $\widetilde{h}(L(\mathcal{A})) = L(\mathcal{B}_{\mathcal{A}})$. Since the class of ω-regular languages accepted by Büchi automata are closed under complement, we know that there exists a Büchi automaton, say, $\mathcal{B}^C_{\mathcal{A}}$ such that $L(\mathcal{B}_{\mathcal{A}})^C = L(\mathcal{B})^C$. We again use Lemma 3.3.5 and obtain a diagram automaton $\mathcal{A}_{\mathcal{B}^C_{\mathcal{A}}}$. We can then see that $L(\mathcal{A})^C = L(\mathcal{A}_{\mathcal{B}^C_{\mathcal{A}}})$. □

3.4 Fragment Automata

We can similarly define automata with fragments, rather than diagrams as input. Fragment automata take a channel b-bounded LLD as input and run on its sequence of fragments (which occur as layers of the LLD). Fix a finite alphabet A and $b \in \mathbb{N}$. Let $\mathcal{LF}_b(A)$ denote the set of b-bounded A-labelled fragments; again, it is a finite set. A fragment automaton is again basically a Büchi automaton which runs over an alphabet of fragments.

Definition 3.4.1 *A* **fragment automaton** *is given by a tuple $\mathcal{B} = (Q, A, \mathcal{F}, \to, I, G)$ where*

1. *A is a finite alphabet,*
2. *$\mathcal{F} \subseteq \mathcal{LF}_b(A)$ is the alphabet of the automaton,*
3. *Q is a finite set of states,*
4. *$\to \subseteq (Q \times \mathcal{F} \times Q)$ is the transition relation,*
5. *$I \subseteq Q$ is the set of initial states and*
6. *$G \subseteq Q$ is the set of good states.*

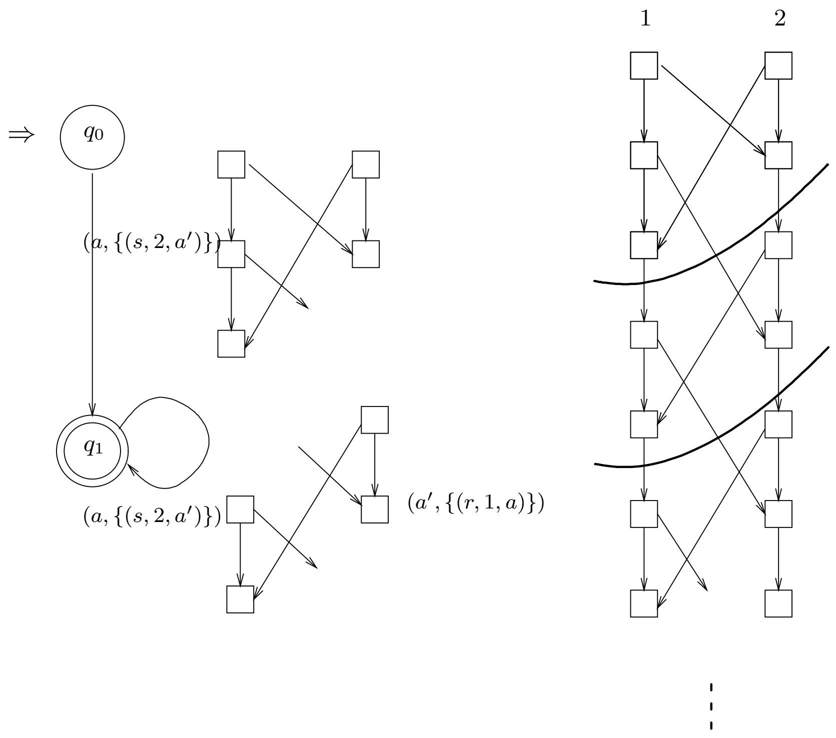


Figure 3.7: Fragment automaton accepting the LLD representing the alternating bit protocol

A *run* of $\mathcal{B}$ on a channel b-bounded LLD D is a sequence $\rho = q_0, q_1, \ldots$ such that $q_0 \in I$ and $q_i \xrightarrow{F_i} q_{i+1}$ for all $i \geq 0$ where $F_0, F_1, \ldots$ is the sequence of fragments associated with ν_D. ρ is said to be *accepting* if $inf(\rho) \cap G \neq \emptyset$, where $inf(\rho) = \{q \mid q = q_i$ for infinitely many $i\}$. The *language of channel b-bounded LLDs* accepted by $\mathcal{B}$ is denoted by $L_S^b(\mathcal{B})$ and is defined as $L_S^b(\mathcal{B}) = \{D \mid$ there exists an accepting run of $\mathcal{B}$ on $D\}$.

Given a channel b-bounded LLD as input, the automaton runs on its sequence of layers (which are fragments in the alphabet of the automaton) and accepts the LLD if it sees a good state infinitely often on the sequence of fragments (the layers of the LLD) it is reading.

For example, Figure 3.7 shows an example of a fragment automaton accepting the channel 5-bounded LLD corresponding to the alternating bit protocol. The automaton has two states and runs on the fragments corresponding to the layers of the LLD. In the figure, the fragments occuring as layers of the alternating bit protocol LLD are given as labels of the transitions of the automaton. The tags of the pending send and receive events of the fragments are given adjacent to the events as labels.

Again, we first consider the emptiness problem for fragment automata and show that it is decidable. Emptiness checking is no longer a simple search for a connected component containing a good state. We need to check in addition that every 'pending send' is matched with a 'hanging receive' with labels appropriately matched. The following theorem presents the details.

Theorem 3.4.2 *Given a fragment automaton $\mathcal{B} = (Q, A, \mathcal{F}, \rightarrow, I, G)$, the problem of checking if $L^b_S(\mathcal{B}) \neq \emptyset$ is decidable in time $O(|Q|^2 \times k^{b^2})$, where $k = |A|$.*

Proof: We build a larger automaton $\mathcal{C}$ based on $\mathcal{B}$ and show that $L^b_S(\mathcal{B}) \neq \emptyset$ iff there exists a strongly connected component in $\mathcal{C}$ containing good states of $\mathcal{C}$ reachable from one of the initial states of $\mathcal{C}$. For this, we first define a queue data structure that will be useful. Let $\overline{A} = \{x \in A^* \mid |x| \leq b\}$. We will be considering maps $\chi : ([n] \times [n]) \rightarrow (z \times \overline{A})$, where $z \in \{0, 1\}$ is a boolean flag to be used as a signal; let Ξ denote the set of all such maps. Let $\widetilde{Q} = (Q \times \Xi)$.

For any fragment $F = (E, \leq, \phi, A, \eta)$ (which corresponds to a layer of some channel b-bounded LLD) and $i, j \in [n]$, define two sequences F^s_{ij} and F^r_{ji} in $\overline{A}$ as done in Chapter 2: note that E_i is a finite set linearly ordered by $\leq$; denote this sequence of events by $e_1 \cdots e_m$. Now $F^s_{ij} = a_{k_1} \cdots a_{k_\ell}$, where for all $p \in \{k_1, \ldots, k_\ell\}$, $(s, j, a_p) \in \eta(e_{k_p})$ and for all $p \in (\{1, \ldots, m\} \setminus \{k_1, \ldots, k_\ell\})$, there is no $a' \in A$ such that $(s, j, a') \in \eta(e_{k_p})$. F^r_{ji} is defined similarly, using labels of the form (r, i, a). Note that both are b-bounded sequences.

Now define $\mathcal{C} = (\widetilde{Q}, A, \mathcal{F}, \Rightarrow, \widetilde{I}, \widetilde{G})$, where

- $\widetilde{Q}$ is defined as above,
- $\widetilde{I} = \{(q, \chi_0) \in \widetilde{Q} \mid q \in I, \chi_0(i, j) = (0, \epsilon) \text{ for all } i, j \in [n]\}$,
- $\widetilde{G} =< G_{ij} \mid i, j \in [n] >$, where $G_{ij} = \{(q, \chi) \in \widetilde{Q} \mid q \in G,\ \chi(i, j) = (0, x), x \in \overline{A}\}$ and
- $\Rightarrow$ is defined as follows:

  $(q, \chi) \overset{F}{\Rightarrow} (q', \chi')$ iff:

  - $q \xrightarrow{F} q'$ in $\mathcal{B}$;

– for $i, j \in [n]$, if $\chi(i,j) = (z, x)$ and $\chi'(i,j) = (z', x')$, then $x = F^r_{ji} \cdot y$, $x' = y \cdot F^s_{ij}$ and $z' = 0$ if ($z = 1$ and $y = \epsilon$) or ($z = 0$ and $x' = \epsilon$).

Call a connected component in $\mathcal{C}$ good only if it contains some state in G_{ij}, for each pair $i, j \in [n]$ and is reachable from $\widetilde{I}$. We can now show that $L^b_S(\mathcal{B}) \neq \emptyset$ iff $\mathcal{C}$ has a good connected component.

Suppose $L^b_S(\mathcal{B}) \neq \emptyset$. Let D be a channel b-bounded diagram accepted by $\mathcal{B}$. Then, there exists a sequence of fragments $F_1, F_2, \ldots$ such that the sequence matches D and there is an accepting run, say, $\rho = q_0 \xrightarrow{F_1} q_1 \xrightarrow{F_2} \ldots$ of $\mathcal{B}$ on D. We will show that $\mathcal{C}$ has a good component by inductively constructing a sequence of states of $\mathcal{C}$ below. Define $\rho' = (q_0, \chi_0), (q_1, \chi_1), \ldots$ where $q_0, q_1, \ldots$ is as in ρ and χ_i is defined inductively as follows:

For the base case, $\chi_0(i,j) = (0, \epsilon)$ for all $i, j \in [n]$. For $i, j \in [n]$ such that $i \neq j$, define $\chi_1(i,j) = (z_1, F^s_{1_{(ij)}})$ where $z_1 = 0$ iff $F^s_{1_{(ij)}} = \epsilon$. Now, define $\chi_2(i,j) = (z_2, x_2)$ where $x_2 = y \cdot F^s_{2_{(ij)}}$ where y is such that $F^s_{1_{(ij)}} = F^r_{2_{(ji)}} \cdot y$ and $z_2 = 0$ iff ($z_1 = 1$ and $y = \epsilon$) or ($z_1 = 0$ and $x_2 = \epsilon$). We claim that $F^s_{1_{(ij)}} = F^r_{2_{(ji)}} \cdot y$. Suppose not, that is, suppose $F^r_{2_{(ji)}}$ is not a prefix of $F^s_{1_{(ij)}}$. Then, there exists $wa \in \overline{A}$ such that $F^r_{2_{(ji)}} = wa$ and $F^s_{1_{(ij)}}$ is a prefix of w. This implies that there exists a sequence of events $e_1, e_2, \ldots, e_m$ (where $m = |wa|$) such that $(r, i, a) \in \eta(e_m)$. Since the sequence $F_1, F_2, \ldots$ matches D (refer to the previous chapter for the definition), we know that there exists $e \in E_i$ such that $e <_c e_m$ in D and in fact, $e \in F_1$ such that $(s, j, a) \in \eta(e)$. This is a contradiction to the fact that $F^s_{1_{(ij)}}$ is a prefix of w.

Inductively, suppose χ_k, $k \geq 2$ has been defined. We define χ_{k+1} as $\chi_{k+1}(i,j) = (z_{k+1}, x_{k+1})$ where $x_{k+1} = y \cdot F^s_{k+1_{(ji)}}$ where y is such that $x_k = F^r_{k+1_{(ij)}} \cdot y$, where $\chi(i,j) = (z_k, x_k)$ and $z_{k+1} = 0$ iff ($z_k = 1$ and $y = \epsilon$) or ($z_k = 0$ and $x_{k+1} = \epsilon$). Again, we can argue as done for the base case above that x_k is equal to $F^r_{k+1_{(ji)}} \cdot y$.

We now claim that the sequence $\rho' = (q_0, \chi_0), (q_1, \chi_1), \ldots$ defines a good connected component of $\mathcal{C}$. Clearly, by definition of ρ', $(q_0, \chi_0) \in \widetilde{I}$ and $(q_k, \chi_k) \xRightarrow{F_k} (q_{k+1}, \chi_{k+1})$ for all $k \geq 0$. Also, there exists infinitely many $k \geq 0$ such that $\chi(i,j) = (0, x)$ for all $i, j \in [n]$ such that $i \neq j$ because if not, then, this would imply that there is either a pending send event without a matching receive event or there is a pending receive event without a matching send event. Both these would contradict the fact that the sequence $F_1, F_2, \ldots$ matches D. Hence, ρ' defines a good connected component of $\mathcal{C}$ and we are done.

Conversely, suppose that there exists a good connected component of $\mathcal{C}$. We have to show that $L^b_S(\mathcal{B}) \neq \emptyset$. Let $\rho = (q_0, \chi_0) \overset{F_{k_1}}{\Rightarrow} (q_1, \chi_1) \overset{F_{k_2}}{\Rightarrow} \ldots$ be the sequence obtained by unwinding the good connected component of $\mathcal{C}$. That is $(q_0, \chi_0) \in \widetilde{I}$ and there exists infinitely many $k \geq 0$ such that (q_k, χ_k) contains some state in G_{ij} for each pair $i, j \in [n]$ such that $i \neq j$. We will inductively define a Lamport diagram $D = (E, \leq, \phi, \lambda)$ and show that

- D is channel b-bounded and the sequence $F_{k_1}, F_{k_2}, \ldots$ matches D.
- The run $\rho' = q_0 \overset{F_{k_1}}{\rightarrow} q_1 \overset{F_{k_2}}{\rightarrow} \ldots$ (obtained by projecting ρ to the states in Q) is an accepting run of $\mathcal{B}$ on D.

Let $F_{k_i} = (E_{k_i}, \leq_{k_i}, \phi_{k_i}, A, \eta_{k_i})$ for all k_i. Initially, $D_1 = (E_1, \leq_1, \phi_1, \lambda_1)$ where

- $E_1 = E_{k_1} \times \{1\}$.
- $\leq_1 = \{((e, 1), (e', 1)) \mid (e, e') \in \leq_{k_1}\}$.
- For $(e, 1) \in E_1$, $\phi_1((e, 1)) = \phi_{k_1}(e)$.
- $\lambda_1((e, 1)) = 1$ for all $(e, 1) \in E_1$.

Inductively, suppose that $D_l = (E_l, \leq_l, \phi_l, \lambda_l)$ has been defined. We define $D_{l+1} = (E_{l+1}, \leq_{l+1}, \phi_{l+1}, \lambda_{l+1})$ as follows:

- $E_{l+1} = E_l \cup \{(e, l+1) \mid e \in E_{k_{l+1}}\}$.
- $\leq_{l+1} = (\leq_l \cup \{((e, l+1), (e', l+1)) \mid (e, e') \in \leq_{k_{l+1}}\}$
  $\cup \cup_{i=1}^{n} \{((e, l'), (e', l+1)) \mid (e, l'), (e', l+1) \in E^i_{l+1}$ and $l' < l+1\}$ where E^i_{l+1} denotes the set of events of E_{l+1} local to agent i
  $\cup \cup_{i \neq j} \{((e_p, l_p), (e'_p, l+1)) \mid i \leq p \leq m', e'_1, e'_2, \ldots e'_{m'}$ is the sequence of j-events corresponding to $F^r_{k_{l+1}(ji)} = a_1 \ldots a_{m'}$ and $e_1 \ldots e_m, m \geq m'$ is the sequence of events corresponding to $y \cdot F^s_{k_{l(ij)}} = a_1 \ldots a_m$ and $e_p \in E_{k_{l_p}}, k_{l_p} < k_{l+1}\})^*$.
- For $(e, l') \in E_{l+1}$, $\phi_{k+1}((e, l')) = \phi_l((e, l'))$ for all (e, l') such that $l' \leq l$ and it is $\phi_{k_{l+1}}(e)$ if $l' = l+1$.
- $\lambda_{l+1}((e, l')) = \lambda_l((e, l'))$ if $l' \leq l$ and is $l+1$ if $l' = l+1$.

We claim that $D = (E, \leq, \phi, \lambda)$ as defined above is a channel b-bounded layered Lamport diagram. The argument that D is a layered Lamport diagram is similar that used in the proof of Theorem 3.3.2. Note that the same observation holds with respect to the $\leq$ defined here also. The fact that D is channel b-bounded follows from the observation that the states of $\mathcal{C}$ has sequence of labels of length at most b.

We now have to show that the sequence $F_{k_0}, F_{k_1}, \ldots$ matches D. From the definition of D, we know that $\nu_D = 1, 2, \ldots$. Consider the sequence of fragments represented by ν_D, say, $F_0, F_1, \ldots$. Again, from the definition of D, it is clear that the sequence $F_1, F_2, \ldots$ is isomorphic to $F_{k_1}, F_{k_2}, \ldots$, that is, there exists a sequence of isomorphisms $h_i : F_i \to F_{k_i}$ for all $i \geq 1$. To show that the sequence $F_{k_1}, F_{k_2}, \ldots$ matches D, consider k_l and $e \in F_{k_l}$ such that $(s, i, a) \in \eta_{k_l}(e)$. Now, $e \in F_{k_l}$ implies $(e, l) \in D$, and $\lambda((e, l)) = l$. Hence, a is present in the sequence $F^s_{l_{(ji)}}$. From the definition of ρ, we know that $\chi_{l+1}(i, j) = (z_{l+1}, x_{l+1})$ where $x_{l+1} = y \cdot F^s_{l_{(ji)}}$. Let $m > l$ be the least index such that $\chi_m(i, j) = (z_{m+1}, x_{m+1})$ where $z_{m+1} = 0$ (we know that such an m exists as ρ is accepting). By the definition of $\Rightarrow$, we know that $z_{m+1} = 0$ only if there are no pending sends or all the pending receives gets matched up (depending on the value of z_m). In either case, by the definition of $\leq$ in D, we know that $(e, l) \leq (e', m)$ where e' is the event such that $(r, j, a) \in \eta_{k_m}(e')$ (that is a is in the sequence $F^r_{m_{(ij)}}$). Infact, $(e, l) <_c (e', m)$ and hence we are done. Since the sequence $F_{k_1}, F_{k_2}, \ldots$ matches D, it follows that ρ' is an accepting run of $\mathcal{B}$ on D. $\square$

3.4.1 Closure Properties

As done for diagram automata, we now show that the class of LLDs accepted by fragment automata are closed under boolean operations. It is again based on the fact that fragment automata are basically Büchi automata running over fragments. Closure under union and intersection are exactly as done for diagram automata and so, we just state the results. Closure under complementation involves an additional check to ensure that the complement language is a language of channel bounded LLDs. We again prove results which help us to move back and forth between Büchi automata and fragment automata and use these to show that the class of languages accepted by fragment automata is closed under boolean operations and under complementation.

Theorem 3.4.3 • *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two fragment automata. Then, there exists a fragment automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.*

- *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two fragment automata. Then, there exists a fragment automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.*

As done in the previous section, we first define bijections back and forth between a finite alphabet of words and between fragments so that we can exploit the closure of Büchi automata under complementation to show that fragment automata are also closed under complementation.

Fix $b \in \mathbb{N}$ and a finite alphabet A of labels of fragments. Let $\mathcal{LF}_b(A)$ denote the set of A-labelled b-bounded fragments. Let Σ be a finite alphabet and let $g : \Sigma \rightarrow \mathcal{LF}_b(A)$ be a bijection. We extend g to $\widehat{g} : \Sigma^\omega \rightarrow \mathcal{LF}_b(A)^\omega \times \{0, 1\}$. For $\sigma = a_0 a_1 \ldots \in \Sigma^\omega$, $\widehat{g}(\sigma) = (D, k)$ where $D = g(a_0) \circ g(a_1) \circ \ldots$ and $k = 1$ iff D is channel b-bounded. The second component k is to filter out those concatenations which result in LLDs that are not channel b-bounded as not every concatenation of an arbitrary sequence of fragments would result in channel b-bounded LLDs. We further extend $\widehat{g}$ to $\tilde{g} : 2^{\Sigma^\omega} \rightarrow 2^{(\mathcal{LF}_b(A)^\omega \times \{0,1\})}$ by defining $\tilde{g}(L) = \{\widehat{g}(\sigma) \mid \sigma \in L\}$ for $L \subseteq \Sigma^\omega$. It is easy to see that $\widehat{g}$ and $\tilde{g}$ are also bijections on the appropriate domains. The bijections defined above would help us to construct fragment automata from Büchi automata.

To do the reverse construction, we start with a bijection $f : \mathcal{LF}_b(A) \rightarrow \Sigma$ and extend f to $\widehat{f} : S_b \rightarrow \Sigma^\omega$ as follows: for a channel b-bounded LLD $D \in S_b$ given by $D = D_0 \circ D_1 \circ \ldots$, we define $\widehat{f}(D) = a_0 a_1 \ldots$ where $a_i = f(D_i)$ for all $i \geq 0$. We further extend $\widehat{f}$ to $\tilde{f} : 2^{S_b} \rightarrow 2^{\Sigma^\omega}$ as $\tilde{f}(L) = \{\widehat{f}(D) \mid D \in L\}$. We now establish a back and forth correpondence between Büchi automata and fragment automata.

Lemma 3.4.4 *1. Given a fragment automaton $\mathcal{A}$ over $\mathcal{LF}_b(A)$, there exists a Büchi automaton $\mathcal{B}_\mathcal{A}$ over Σ such that $\tilde{f}(L(\mathcal{A})) = L(\mathcal{B}_\mathcal{A})$.*

2. Given a Büchi automaton $\mathcal{B}$ over Σ, there exists a fragment automaton $\mathcal{A}_\mathcal{B}$ over $\mathcal{LF}_b(A)$ such that $L(\mathcal{A}_\mathcal{B}) = \{D \mid (D, 1) \in \widehat{g}(\sigma)$ for some $\sigma \in L(\mathcal{B})$ and D is channel b-bounded $\}$.

Proof:

1. Let $\mathcal{A} = (Q, A, \mathcal{LF}_b, \rightarrow, I, G)$ be the given fragment automaton. We define the Büchi automaton as $\mathcal{B}_{\mathcal{A}} = (Q, \Sigma, \Rightarrow, I, G)$ where $q \overset{a}{\Rightarrow} q'$ iff $q \xrightarrow{D} q'$ in $\mathcal{A}$, where $D \in \mathcal{LF}_b$ is such that $f(D) = a$.

2. Let $\mathcal{B} = (Q, \Sigma, \Rightarrow, I, G)$ by the given Büchi automaton. We first define a fragment automaton $\mathcal{A}'_{\mathcal{B}}$ as $\mathcal{A}'_{\mathcal{B}} = (Q, A, \mathcal{LF}_b, \rightarrow, I, G)$ where the transition relation $\rightarrow$ is given by $q \xrightarrow{D} q'$ iff $D = g(a)$ for some $a \in \Sigma$ and $q \overset{a}{\Rightarrow} q'$ in $\mathcal{B}$. The automaton $\mathcal{A}'_{\mathcal{B}}$ accepts all LLDs of the form $\widehat{g}(\sigma)$ where $\sigma \in L(\mathcal{B})$. We now restrict the automaton $\mathcal{A}'_{\mathcal{B}}$ to accept channel b-bounded LLDs among those of the form $\widehat{g}(\sigma)$ for $\sigma \in L\mathcal{B}$ as follows: We know that the class of languages accepted by fragment automata are effectively closed under intersection. We now construct a fragment automaton $\mathcal{A}_b$ which accepts precisely those LLDs which are channel b-bounded. Then, the required fragment automaton $\mathcal{A}_{\mathcal{B}}$ is obtained by taking the intersection of $\mathcal{A}'_{\mathcal{B}}$ and $\mathcal{A}_b$. Since $\mathcal{A}'_{\mathcal{B}}$ accepts all LLDs of the form $\widehat{g}(\sigma)$ and $\mathcal{A}_b$ accepts an input LLD iff it is channel b-bounded, the automaton $\mathcal{A}_{\mathcal{B}}$ constructed to accept the intersection of the two languages is the required one.

   We now define the automaton $\mathcal{A}_b$ to complete the proof. The automaton $\mathcal{A}_b$ is given by $\mathcal{A}_b = (Q, A, \mathcal{LF}_b, \rightarrow, I, G)$ where

   - $Q = \{(w, i, j) \mid w \in A^*, |w| \leq b \text{ and } i, j \in [n] \text{ such that } i \neq j\}$,
   - $I = \{(\epsilon, i, j) \mid i, j \in [n] \text{ such that } i \neq j\}$,
   - $G = Q$ and
   - $\rightarrow$ is given by: $(w, i, j) \xrightarrow{F} (w', i', j')$ iff $w = F^r_{ji} \cdot y$ and $w' = y \cdot F^s_{ij}$, where F^r_{ji} and F^s_{ij} are as defined in Chapter 2.

   It is easy to see that $D \in L(\mathcal{A}_b)$ iff D is channel b-bounded.

□

We now show that the class of languages accepted by fragment are closed under complementation.

Theorem 3.4.5 *Given a fragment automaton $\mathcal{A}$ over $\mathcal{LF}_b(A)$, there exists a fragment automaton $\mathcal{A}^C$ over $\mathcal{LF}_b(A)$ such that $L(\mathcal{A}^C) = L(\mathcal{A})^C$ where $L(\mathcal{A})^C$ denotes* *the complement of the language $L(\mathcal{A})$ over $\mathcal{LF}_b(A)$ i.e., those channel b-bounded LLDs which are not in the set $L(\mathcal{A})$.*

Proof: From the given fragment automaton $\mathcal{A}$, we can construct a Büchi automaton $\mathcal{B}_\mathcal{A}$ such that $\tilde{f}(L(\mathcal{A})) = L(\mathcal{B}_\mathcal{A})$. We know that the class of ω-regular languages accepted by Büchi automata are effectively closed under complementation. Hence, there exists a Büchi automaton $\mathcal{B}^C_\mathcal{A}$ which accepts $L(\mathcal{B}_\mathcal{A})^C$. Again, from the previous lemma, we can construct a fragment automaton $\mathcal{A}_{\mathcal{B}^C_\mathcal{A}}$ such that $L(\mathcal{A}_{\mathcal{B}^C_\mathcal{A}})$ is the set of all channel b-bounded LLDs in $\tilde{g}(\mathcal{B}^C_\mathcal{A})$. It is now straightforward to see that $L(\mathcal{A}_{\mathcal{B}^C_\mathcal{A}}) = L(\mathcal{A})^C$. □

Note that the previous lemma can also be used to show that the class of languages accepted by fragment automata enjoy other closure properties as those of ω-regular languages accepted by Büchi automata.

Chapter 4

Modal logics over Lamport diagrams

In the earlier two chapters, we introduced Lamport diagrams as partial orders describing causality and communication in distributed message passing systems and discussed various automata models over them. With the overall aim of the thesis being developing methods to reason about Lamport diagrams, we now concentrate on defining suitable logics to describe properties of Lamport diagrams. Temporal logics and monadic second order logics have always been two natural choices of logics to reason about systems. We first concentrate on developing decidable temporal logics over Lamport diagrams. We will define a monadic second order logic in one of the later chapters.

Temporal logics are modal logics which are tuned to reason about systems whose behaviours evolve with time. Modal logics have been well known as expressive logics to specify properties of arbitrary Kripke structures and partial orders [HC96]. When it comes to defining logics as specification languages, the decidability of satisfiability problem of the logic is an important qualifying point. In the context of formal verification, the algorithm for satisfiability also aids in proving the decidability of a suitable model checking problem. The satisfiability problem of modal logics is decidable over the general class of partial orders [Kra99]. Also, it is well known that the satisfiability problem of temporal logics is decidable over arbitrary linear orders [HR04].

Keeping in mind the fact that Lamport diagrams are basically specialized partial orders describing causality between events of a distributed system, we first define a natural modal logic over these diagrams. The modalities of the logic are tuned to describe typical properties of message passing systems apart from causality; namely properties specifying when an agent would send a message, what an agent would do upon receiving a message etc. We introduce two modalities to refer to an immediate successor and an immediate predecessor of an event to talk about send and receive events. The logic, called LD_0, has four modalities—$\mathbf{X}$ (immediate successor), $\mathbf{Y}$ (immediate predecessor), $\mathbf{F}$ (future), $\mathbf{P}$ (past) and formulas of the logic are interpreted over Lamport diagrams. The satisfiability problem of this logic is the problem of checking if a given formula has a Lamport diagram as a model or not. The problem turns out to be undecidable as the logic is expressive enough to describe computations of 2-counter machines. In fact, we show that the satisfiability problems of weaker versions of LD_0 (with restricted $\mathbf{X}$ and $\mathbf{Y}$ modalities) are also undecidable. We look at decidable temporal logics with further restrictions in the syntax in the next chapter.

4.1 A modal logic on Lamport diagrams

Fix countable sets of *propositional letters* $(P_1, P_2, \ldots P_n)$ where P_i consists of atomic local properties of agent i. Let $P \overset{\text{def}}{=} \cup_i P_i$. The set of i-local propositions P_i also includes a special *type proposition* τ_i which is true at all the events of agent i. Note that P_i and P_j need not be disjoint when $i \neq j$, since we can always use τ_i to disambiguate, if needed.

We first consider a modal logic LD_0, whose syntax is given below:

$$LD_0 ::= p \in P \mid \tau_i \mid \neg \alpha \mid \alpha_1 \vee \alpha_2 \mid \mathbf{X}\, \alpha \mid \mathbf{Y}\, \alpha \mid \mathbf{F}\, \alpha \mid \mathbf{P}\, \alpha$$

As mentioned in the beginning, the logic LD_0 is a standard propositional modal logic over any labelled discrete partial order. In the context of Lamport diagrams, the $\mathbf{X}$ modality refers to an immediate successor event and the $\mathbf{Y}$ modality refers to an immediate predecessor event. $\mathbf{F}$ stands for 'future' and $\mathbf{P}$ for 'past' respectively.

We will use indexed modalities as abbreviations: for instance, $\mathbf{X}_i\alpha$ denotes the formula $\mathbf{X}(\tau_i \wedge \alpha)$; other abbreviations for $\mathbf{Y}_i$, $\mathbf{F}_i$ and $\mathbf{P}_i$ are defined similarly.

The dual of $\mathbf{F}$ is denoted $\mathbf{G}$ and is defined as $\mathbf{G}\alpha = \neg\mathbf{F}\neg\alpha$ the dual of $\mathbf{P}$ is denoted $\mathbf{H}$ and is defined as $\mathbf{H}\alpha = \neg\mathbf{P}\neg\alpha$. $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$, the duals of $\mathbf{X}$ and $\mathbf{Y}$ respectively are defined similarly. Note that the abbreviations denote implications for the dual modalities; for instance $\mathbf{G}_i\alpha$ denotes $\mathbf{G}(\tau_i \supset \alpha)$.

The formulas are interpreted on Lamport diagrams. We will denote models by $M = (D, V_E)$ where $D = (E, \leq, \phi)$ is a Lamport diagram, $V_E : E \rightarrow 2^P$ is a valuation function such that $V_E(E_i) \subseteq P_i$ for all $i \in [n]$ and for all $e \in E$, $\tau_i \in V_E(e)$ iff $\phi(e) = i$.

Let $\alpha \in LD_0$ and $e \in E$. The notion that α holds at e in M is denoted $M, e \models \alpha$ and is defined inductively as follows:

- $M, e \models p$ iff $p \in V_E(e)$.
- $M, e \models \neg\alpha$ iff $M, e \not\models \alpha$.
- $M, e \models \alpha \vee \beta$ iff $M, e \models \alpha$ or $M, e \models \beta$.
- $M, e \models \mathbf{X}\alpha$ iff there exists $e' \in E$ such that $e \lessdot e'$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{F}\alpha$ iff there exists e' such that $e \leq e'$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{Y}\alpha$ iff there exists $e' \in E$ such that $e' \lessdot e$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{P}\alpha$ iff there exists e' such that $e' \leq e$ and $M, e' \models \alpha$.

We say that α is *satisfiable* iff there exists a model $M = (D, V_E)$, where $D = (E, \leq, \phi)$ and $e \in E$ such that $M, e \models \alpha$. The satisfiability problem for LD_0 is the problem of checking if a given formula is satisfiable.

If α has a model $M = (D, V_E)$ where E is finite, we say α is *finitely satisfiable.* When $\mid E \mid \leq b$ for some $b \in \mathbb{N}$, we say α is *b-satisfiable.* The finite satisfiability (b-satisfiability) problem for LD_0 is the problem of checking if a given formula is finitely satisfiable (b-satisfiable).

Except for the presence of special propositions τ_i, the logic is seen to be a standard tense logic on partial orders. The logic may further be strengthened with $\mathbf{U}$ (until) and $\mathbf{S}$ (since) modalities and the technical results which follow still hold for such a logic as well. We will use the abbreviations *True* and *False* to denote the formulas $p_0 \vee \neg p_0$ and $p_0 \wedge \neg p_0$ respectively, where $p_0 \in P$. The modalities $\mathbf{X}_i$ and $\mathbf{Y}_i$ act as *local* modalities when asserted at events of agent i and as *global* modalities when asserted at events of some agent j where $j \neq i$. Consider an event e of agent

i. The formula $\mathbf{X}_i\alpha$ asserted at e says that α holds at the immediate local successor of e and the formula $\mathbf{X}_j\alpha$ (where $j \neq i$) asserted at e says that e is a send event to agent j and α holds at the corresponding receive event in agent j. Hence, the formula $\tau_1 \wedge \mathbf{X}_2 \mathit{True}$ may be seen as asserting the sending of a message by agent 1 to 2, and similarly $\tau_1 \wedge \mathbf{Y}_2 \mathit{True}$ asserts the receipt of a message by 1 from 2.

Formulas of LD_0 can be used to specify various natural properties of message passing systems. For example, the formula $\mathbf{Y}_1(\tau_2 \wedge req) \supset \mathbf{F}_2(\mathbf{X}_1 ack)$ specifies that agent 2 sends an acknowledgment (ack) in reply to a request (req) from agent 1.

Theorem 4.1.1 *The (finite) satisfiability problem for LD_0 is undecidable.*

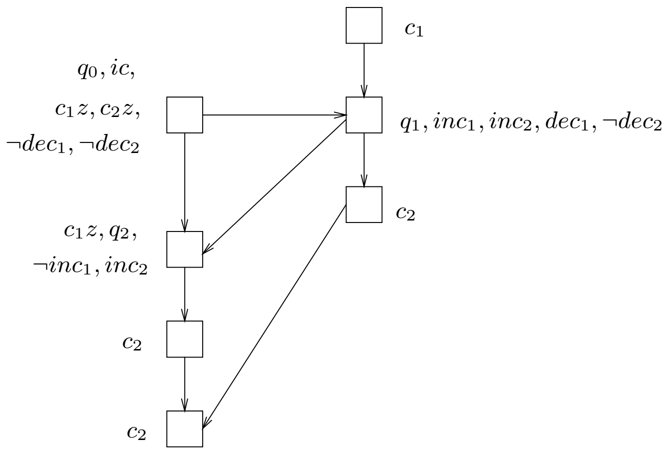
The negative result here mainly stems from the fact that runs of non-deterministic 2-counter machines can be described using Lamport diagrams. We first describe non-deterministic 2-counter machines.

Non-deterministic 2-counter machines A non-deterministic 2-counter machine has a finite number of states and two counters. The transitions of the machine increment or decrement the values of each counter while changing from one state to another. A non-deterministic 2-counter machine is given by a tuple $M = (Q, \delta, q_0, q_f)$ where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state and
- $\delta \subseteq T$ is the transition relation where $T = (Q \times \{0,1\}^2 \times Q \times \{-1,0,1\}^2)$

When $(q, x_1, x_2, q', y_1, y_2) \in \delta$ and the machine is in state q, the transition is enabled depending on whether the two counters are zero or non-zero as given by x_1 and x_2 and the machine changes state to q', and the counter values are decremented or incremented or left unchanged, depending on y_1 and y_2. We assume that if $(q, x_1, x_2, q', y_1, y_2) \in \delta$ then $y_i = -1$ implies $x_i = 1$ for $i = 1, 2$ (only a positive counter can be decremented) and $q \neq q_f$ (there are no transitions out of the final state). A *configuration* of M is a triple (q, n_1, n_2) where $q \in Q$ and $n_i \in \mathbb{N}$ are the values of the two counters. We say that $(q, n_1, n_2) \rightarrow (q', n_1 + y_1, n_2 + y_2)$ if

Figure 4.1: Lamport diagram corresponding to a run of M



$(q, x_1, x_2, q', y_1, y_2) \in \delta$ and $n_i = 0$ iff $x_i = 0$ for $i = 1, 2$. A *run* ρ of M is any sequence $(q_0, 0, 0) \rightarrow (q_1, n_1^1, n_2^1) \rightarrow (q_2, n_1^2, n_2^2) \rightarrow \ldots$. A configuration (q, n_1, n_2) is *reachable* if there exists a run of M from $(q_0, 0, 0)$ to (q, n_1, n_2). A configuration is *final* if $q = q_f$. Given a 2-counter machine, the problem of checking if a final configuration is reachable is undecidable.

Runs and Lamport diagrams Given a run ρ of M, we will define a 2-agent Lamport diagram that 'represents' ρ. For example, the Lamport diagram corresponding to the run $\rho = (q_0, 0, 0) \rightarrow (q_1, 1, 1) \rightarrow (q_2, 0, 2)$ is depicted in Figure 4.1. The propositions labelling an event are the ones that are satisfied at the event. The proposition ic represents the initial configuration. The propositions c_iz represent the fact that the value of counter $i = 0$. The (non-zero) values of counter i are coded up using propositions c_i for $i = 1, 2$ respectively which are repeated as many times as the value of the counter. A configuration (q, n_1, n_2) with $n_i \neq 0$ for $i = 1, 2$ is represented by a sequence of n_1 events labelled by c_1 followed by the event labelled by the proposition corresponding to q followed by a sequence of n_2 events labelled by c_2. If $n_i = 0$, then proposition c_iz is holds at the event labelled by a state proposition.

Consider a configuration (q, n_1, n_2) represented in the above way say, as a sequence of events of agent 1. A transition of M from (q, n_1, n_2) to (q', n_1', n_2') is coded as follows. There is an immediate successor (to the event labelled by q) in agent 2 which is labelled by q'. The propositions inc_i and dec_i represent incrementing and decrementing counter i for $i = 1, 2$ respectively. The proposition inc_1 is satisfied at the q'-labelled event whenever counter 1 is incremented and is implemented by adding a new event in agent 2 labelled with c_1 just before the q'-labelled event. Similarly, proposition inc_2 is satisfied at the q'-labelled event whenever counter 2 is incremented and is implemented by adding a new event in agent 2 labelled with c_2 just after the q'-labelled event. For decrementing, the proposition dec_1 is satisfied at the q-labelled event whenever counter 1 is decremented and is implemented by not copying the last c_1-labelled event (just before the q-labelled event) from agent 1 to agent 2. Similarly, dec_2 holds at the q-labelled event whenever counter 2 is decremented and is implemented by not copying the first c_2-labelled event (just after the q-labelled event) from agent 1 to agent 2.

Each run of M is represented by a Lamport diagram in this way. We will now show that such runs can be described by a formula of the logic LD_0. That is, given a 2-counter machine M, we will define a formula α_M such that α_M is satisfiable iff a final configuration of M is reachable.

It turns out that we can define α_M using a logic (called LD_1) with a more restricted syntax than that of LD_0. Consequently, the undecidability of LD_0 follows from the undecidability of the logic LD_1 which is presented next.

4.2 Variations on the theme

The global power of $\mathbf{X}$ and $\mathbf{Y}$ modalities will be crucially used in the formula α_M to express Lamport diagrams coding up runs of 2-counter machines. It turns out that the satisfiability problem becomes undecidable even if we consider a weaker logic where one of the $\mathbf{X}$ or $\mathbf{Y}$ modalities is global and the other is local along with special propositions to talk about receiving or sending messages respectively. In fact, we obtain undecidability without using the $\mathbf{P}$ modality and by using only a local $\mathbf{F}$ modality. The logics LD_1 and LD_2 are defined below with these restrictions in mind. We show that the satisfiability problems of these logics are also undecidable by constructing corresponding formulae α_M in these logics. The undecidability of LD_0 follows from these proofs.

Logic LD_1

Let r_i^j, $i, j \in [n]$, $i \neq j$ be special 'receive' propositions in P_i which code up the fact that a particular event (of agent i) is a receive event from agent j. The syntax of LD_1 is given by

$$LD_1 ::= p \ \in \ P \mid \tau_i \mid r_i^j \mid \neg\, \alpha \mid \alpha_1 \ \vee \ \alpha_2 \mid \mathbf{X}_i \ \alpha \mid \mathbf{Y}_i \ \alpha \mid \mathbf{F}_i \ \alpha$$

The semantics is defined inductively as done for LD_0. For the sake of clarity, we only mention the cases which are different below.

- $M, e \models r_i^j$ iff $e \in E_i$ and there exists $e' \in E_j$ such that $e' \lessdot e$.
- $M, e \models \mathbf{X}_i\alpha$ iff there exists $e' \in E_i$ such that $e \lessdot e'$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{Y}_i\alpha$ iff $e \in E_i$ and there exists $e' \in E_i$ such that $e' \lessdot e$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{F}_i\alpha$ iff $e \in E_i$ and there exists $e' \in E_i$ such that $e \leq e'$ and $M, e' \models \alpha$.

The modality $\mathbf{X}_i$ above is a global modality as the event e at which it is asserted need not be an event of agent i. The modalities $\mathbf{Y}_i$ and $\mathbf{F}_i$ interpreted at events of agent i require that the immediate predecessor and the future event respectively belong to the same agent and hence are local.

Logic LD_2

We can also define the logic LD_2, a symmetric version of LD_1 with a global $\mathbf{Y}$ modality and a local $\mathbf{X}$ modality and special send propositions. Let s_i^j, $i, j \in [n]$, $i \neq j$ be special 'send' propositions in P_i which code up the fact that a particular event (of agent i) is a send event to agent j.

$$LD_2 ::= p \ \in \ P \mid \tau_i \mid s_i^j \mid \neg\, \alpha \mid \alpha_1 \ \vee \ \alpha_2 \mid \mathbf{X}_i \ \alpha \mid \mathbf{Y}_i \ \alpha \mid \mathbf{F}_i \ \alpha$$

The semantics is again defined inductively as done for LD_0. We note the changes below.

- $M, e \models s_i^j$ iff $e \in E_i$ and there exists $e' \in E_j$ such that $e \lessdot e'$.
- $M, e \models \mathbf{X}_i\alpha$ iff $e \in E_i$ and there exists $e' \in E_i$ such that $e \lessdot e'$ and $M, e' \models \alpha$.
- $M, e \models \mathbf{Y}_i\alpha$ iff there exists $e' \in E_i$ such that $e' \lessdot e$ and $M, e' \models \alpha$.

Observe that the modality $\mathbf{X}_i$ is local and $\mathbf{Y}_i$ is global here.

Note that for $i, j \in [n]$, $i \neq j$, the proposition r_i^j can be expressed in LD_0 as $\tau_i \wedge \mathbf{Y}_j\, True$ and s_i^j can be expressed as $\tau_i \wedge \mathbf{X}_j\, True$. Also, $\mathbf{X}_i$, $\mathbf{Y}_i$ and $\mathbf{F}_i$ modalities in LD_1 and LD_2 are all present in LD_0 as well. The logic LD_0 also has global $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{P}$ modalities in addition to the above.

Note that by using the propositions r_i^j (s_i^j), a particular event of agent i can just assert that a message has been received (sent) to agent j and nothing regarding the content of the message.

Theorem 4.2.1 *The (finite) satisfiability problem for LD_1 is undecidable.*

Proof: As explained earlier, given a non-deterministic 2-counter machine M, we construct a formula α_M such that α_M is satisfiable iff a final configuration of M is reachable. Let $M = (Q, \delta, q_0, q_f)$.

The set of propositions P_i of agent i include the set $\{c_k z, c_k, inc_k, dec_k, ic\} \cup Q$ for $k = 1, 2$ in addition to special propositions τ_i and r_i^j, ($j \neq i$) for $i = 1, 2$. As mentioned earlier, $c_k z$ codes up the fact that counter k is zero. The proposition c_k contributes a value of 1 to counter k, inc_k and dec_k represent incrementing and decrementing the value of counter k by 1 respectively and ic represents the initial configuration $(q_0, 0, 0)$ of M.

In the definition of the formula α_M and in the proofs, we will use the following notation. Given $i \in \{1, 2\}$, $i^{co} = 1$ if $i = 2$ and $i^{co} = 2$ if $i = 1$.

The formula α_M is given by $\alpha_M \overset{\text{def}}{=} \mathsf{init} \wedge \mathsf{inv} \wedge \mathsf{fin}$ where init, inv and fin are defined as follows.

- The formula
  $\mathsf{init} \overset{\text{def}}{=} \tau_1 \wedge q_0 \wedge c_1 z \wedge c_2 z \wedge \overline{\mathbf{Y}}_1 False \wedge \neg r_1^2$
  codes up the initial configuration of $(q_0, 0, 0)$ of M.

- The formula $\mathsf{fin} \overset{\text{def}}{=} \mathbf{F}_1 \mathsf{fin}_1 \vee \mathbf{X}_2 \mathbf{F}_2 \mathsf{fin}_2$ where $\mathsf{fin}_1 \overset{\text{def}}{=} q_f \wedge \mathbf{G}_1 \overline{\mathbf{X}}_2 False$ and $\mathsf{fin}_2 \overset{\text{def}}{=} q_f \wedge \mathbf{G}_2 \overline{\mathbf{X}}_1 False$ asserts that a final configuration is reachable and there are no moves out of it.

- The transitions of M and other facts about the two counter values encoded by c_j are defined by the following formulas.
  $\mathsf{inv} \overset{\text{def}}{=} \mathbf{G}_1 \mathsf{inv}_1 \wedge \overline{\mathbf{X}}_2 \mathbf{G}_2 \mathsf{inv}_2$ where $\mathsf{inv}_i \overset{\text{def}}{=} (\mathsf{state} \vee \mathsf{ctr}_1 \vee \mathsf{ctr}_2) \wedge \mathsf{trans}_i \wedge \mathsf{consis}_i$ for $i = 1, 2$ and these formulas are defined below.

- $\mathsf{state} \overset{\text{def}}{=} \bigvee_{q \in Q} (q \wedge \bigwedge_{q' \in Q, q \neq q'} \neg q') \wedge \neg c_1 \wedge \neg c_2.$
- $\mathsf{ctr}_1 \overset{\text{def}}{=} c_1 \wedge (\bigwedge_{q \in Q} \neg q) \wedge \neg c_2 \wedge \neg c_1 z.$
- $\mathsf{ctr}_2 \overset{\text{def}}{=} c_2 \wedge (\bigwedge_{q \in Q} \neg q) \wedge \neg c_1 \wedge \neg c_2 z.$

  The above formulas ensure that an event labelled by a state or c_1 or c_2 is not labelled by any of the other two propositions.
- The transitions of M are coded up by trans_i as follows.

  For a tuple $t = (q, x, y, q', x', y') \in T$ and $q \neq q_f$, we first define a formula θ_t which codes up the move defined by t.

$$\theta_t \overset{\text{def}}{=} q \wedge (\gamma_x^1 \wedge \gamma_y^2 \wedge \mathbf{X}_{i^{co}} q' \wedge \xi^1_{x',i} \wedge \xi^2_{y',i})$$

  where the various sub-formulas are given by

$$\gamma_0^j = c_j z \qquad \gamma_1^j = \neg c_j z \qquad \xi^j_{0,i} = \neg dec_j \wedge \mathbf{X}_{i^{co}} \neg inc_j$$
$$\xi^j_{1,i} = \neg dec_j \wedge \mathbf{X}_{i^{co}} inc_j \qquad \xi^j_{-1,i} = dec_j \wedge \mathbf{X}_{i^{co}} \neg inc_j$$

  Notice that the $\mathbf{X}_{i^{co}}$ is used as a global modality (to specify what is true at the other agent). Also note that the 'decrement j' decision of a transition causes dec_j to hold at the same event where q holds, whereas the 'increment j' causes inc_j to hold at the event satisfying the successor q' in the other agent i^{co}.

  Now, the transitions of M are coded up as

  $\mathsf{trans}_i \overset{\text{def}}{=} (\bigvee_{q \in Q, q \neq q_f} q \supset \bigvee_{t=(q,x,y,q',x',y') \in \delta} \theta_t) \wedge \bigwedge_{t' \notin \delta} \neg \theta_{t'}.$
- $\mathsf{consis}_i \overset{\text{def}}{=} \mathsf{local\text{-}consis}_i \wedge \mathsf{across\text{-}consis}_i$ where

  $\mathsf{local\text{-}consis}_i \overset{\text{def}}{=} \mathsf{seq}_i \wedge \mathsf{z\text{-}consis}_i \wedge \mathsf{inc\text{-}consis}_i \wedge \mathsf{dec\text{-}consis}_i \wedge \mathsf{z\text{-}dec\text{-}consis}_i \wedge \mathsf{state\text{-}inc\text{-}dec\text{-}consis}_i$ where

  * $\mathsf{seq}_i \overset{\text{def}}{=} (c_1 \supset \overline{\mathbf{X}}_i(c_1 \vee \mathsf{state})) \wedge (c_2 \supset \overline{\mathbf{X}}_i(c_2 \vee c_1 \vee (\mathsf{state} \wedge c_1 z))) \wedge (\mathsf{state} \supset \overline{\mathbf{X}}_i(c_2 \vee c_1 \vee (\mathsf{state} \wedge c_1 z)))$

    ensures that in a configuration, the propositions c_1 coding the value of the first counter occur before an event labelled by a state and the propositions c_2 encoding the value of the second counter occur after an event labelled by a state.

- $\mathsf{z\text{-}consis}_i \overset{\text{def}}{=} (\mathsf{state} \supset (c_1z \equiv \overline{\mathbf{Y}}_i\neg c_1) \wedge (c_2z \equiv \overline{\mathbf{X}}_i\neg c_2))$
  ensures that the value of counter 1 (2) is zero iff there are no events labelled with c_1 (c_2) in the past (future) of an event labelled by a state.
- $\mathsf{inc\text{-}consis}_i \overset{\text{def}}{=} (((\mathsf{state} \wedge inc_1) \supset \mathbf{Y}_i(c_1 \wedge \neg r_i^{i^{co}})) \wedge ((\mathsf{state} \wedge inc_2) \supset \mathbf{X}_i(c_2 \wedge \neg r_i^{i^{co}})))$
  ensures that a new event labelled by c_j (which is not a copy of some event labelled by c_j from the other agent) precedes/succeeds the event corresponding to a state whenever the value of counter j is incremented for $j = 1, 2$.
- $\mathsf{dec\text{-}consis}_i \overset{\text{def}}{=} (((\mathsf{state} \wedge dec_1) \supset \mathbf{Y}_i(c_1 \wedge \overline{\mathbf{X}}_{i^{co}}\mathit{False})) \wedge ((\mathsf{state} \wedge dec_2) \supset \mathbf{X}_i(c_2 \wedge \overline{\mathbf{X}}_{i^{co}}\mathit{False})))$
  ensures that the counter proposition c_j at an event of agent i is not copied to agent i^{co} whenever counter j is decremented for $j = 1, 2$.
- $\mathsf{z\text{-}dec\text{-}consis}_i \overset{\text{def}}{=} (c_1z \supset \neg dec_1) \wedge (c_2z \supset \neg dec_2)$
  ensures that decrementing a counter is not possible if its value is zero.
- $\mathsf{state\text{-}inc\text{-}dec\text{-}consis}_i \overset{\text{def}}{=} \mathsf{state} \supset (\mathsf{dec\text{-}options} \wedge (q_f \vee \mathbf{X}_{i^{co}}\mathsf{inc\text{-}options}))$
  where
  $\mathsf{dec\text{-}options} \overset{\text{def}}{=} (dec_1 \wedge dec_2) \vee (dec_1 \wedge \neg dec_2) \vee (\neg dec_1 \wedge dec_2) \vee (\neg dec_1 \wedge \neg dec_2)$ and
  $\mathsf{inc\text{-}options} \overset{\text{def}}{=} (inc_1 \wedge inc_2) \vee (inc_1 \wedge \neg inc_2) \vee (\neg inc_1 \wedge inc_2) \vee (\neg inc_1 \wedge \neg inc_2)$
  ensures that the counters are either decremented or incremented or left unchanged at every transition.

– $\mathsf{across\text{-}consis}_i \overset{\text{def}}{=} \mathsf{state} \supset ((q_f \vee \mathbf{X}_{i^{co}}\mathsf{state}) \wedge (ic \vee r_i^{i^{co}})) \wedge$
$c_1 \supset ((\mathbf{X}_{i^{co}}c_1 \vee \mathbf{X}_i(\mathsf{state} \wedge dec_1)) \wedge (r_i^{i^{co}} \vee \mathbf{X}_i(\mathsf{state} \wedge inc_1))) \wedge$
$c_2 \supset ((\mathbf{X}_{i^{co}}c_2 \vee \mathbf{Y}_i(\mathsf{state} \wedge dec_2)) \wedge (r_i^{i^{co}} \vee \mathbf{Y}_i(\mathsf{state} \wedge inc_2)))$
ensures that the counter values and the states are updated appropriately from one agent to the other, implementing a transition of M.

We now show that α_M is satisfiable iff a final configuration of M is reachable, which implies that (finite) satisfiability of LD_1 is undecidable.

Suppose there exists a run of M in which a final configuration is reachable, say $\rho = (q^0, n_1^0, n_2^0) \to \ldots (q^k, n_1^k, n_2^k) \to \ldots \to (q^m, n_1^m, n_2^m)$ where $q^0 = q_0$, $n_1^0 = n_2^0 = 0$

and $q^m = q_f$. We will show that α_M is satisfiable by defining a Lamport diagram D_M and a valuation function V_M and then by proving that $M' = (D_M, V_M)$ is a model of α_M.

Define $D_M = (E, \leq, \phi)$ as follows:

- $E = \cup_{0 \leq k \leq m} E_k$ where $E_k = \{c_1e(j,k) \mid 1 \leq_{\mathbb{N}} j \leq_{\mathbb{N}} n_1^k\} \cup \{se_k\} \cup \{c_2e(j,k) \mid 1 \leq_{\mathbb{N}} j \leq_{\mathbb{N}} n_2^k\}$.

  For $0 \leq k \leq m$, define

  $$emax_k = \begin{cases} se_k & \text{if } n_2^k = 0 \\ c_2e(n_2^k, k) & \text{otherwise} \end{cases}$$

  and

  $$emin_k = \begin{cases} se_k & \text{if } n_1^k = 0 \\ c_1e(1, k) & \text{otherwise} \end{cases}$$

- $$\phi(e) = \begin{cases} 1 & \text{if } e \in E_{2l},\ 0 \leq_{\mathbb{N}} 2l \leq_{\mathbb{N}} m \\ 2 & \text{otherwise} \end{cases}$$

- $\leq \overset{\text{def}}{=} (\mathsf{local} \cup \mathsf{comm})^*$ where

  $\mathsf{local} = \{(emax_k, emin_{k+2}) \mid 0 \leq_{\mathbb{N}} k <_{\mathbb{N}} (m-1)\}$
  $\cup_{0 <_{\mathbb{N}} k \leq_{\mathbb{N}} m} (\cup\{(c_1e(j,k), c_1e(j+1,k)) \mid 1 \leq_{\mathbb{N}} j <_{\mathbb{N}} n_1^k\}$
  $\cup \{(c_1e(n_1^k, k), se_k) \mid n_1^k >_{\mathbb{N}} 0\}$
  $\cup \{(se_k, c_2e(1,k)) \mid n_2^k >_{\mathbb{N}} 0\}$
  $\cup \{(c_2e(j,k), c_2e(j+1,k)) \mid 1 \leq_{\mathbb{N}} j <_{\mathbb{N}} n_2^k\})$

  and

  $\mathsf{comm} = \{(se_k, se_{k+1}) \mid 0 \leq_{\mathbb{N}} k <_{\mathbb{N}} m\}$
  $\cup \{(c_1e(j,k), c_1e(j,k+1)) \mid 1 \leq_{\mathbb{N}} j \leq_{\mathbb{N}} min\{n_1^k, n_1^{k+1}\}, 0 \leq_{\mathbb{N}} k <_{\mathbb{N}} m\}$
  $\cup \{(c_2e(l_1(j),k), c_2e(l_2(j),k+1)) \mid 1 \leq_{\mathbb{N}} j \leq_{\mathbb{N}} min\{n_2^k, n_2^{k+1}\}, 0 \leq_{\mathbb{N}} k <_{\mathbb{N}} m\}$
  where

  - $l_1(j) = l_2(j) = j$ if $n_2^k = n_2^{k+1}$,
  - $l_1(j) = j+1, l_2(j) = j$ if $n_2^k > n_2^{k+1}$ and
  - $l_1(j) = j, l_2(j) = j+1$ if $n_2^k < n_2^{k+1}$.

The valuation function V_M is defined as $V_M(e) = V_k(e)$ for all $e \in E_k$ where V_k is defined as follows:

- For $l = 1, 2$,
  $$V_k(c_le(j,k)) = \begin{cases} \{c_l, \tau_1\} & \text{if } k \bmod 2 = 0 \\ \{c_l, \tau_2\} & \text{otherwise} \end{cases}$$

- $V_k(se_k) = \{q^k, \tau_{f(k)}\} \cup \{ic \mid k = 0\} \cup \cup_{l\in\{1,2\}}(\{c_lz \mid n_l^k = 0\}$
  $\cup \{inc_l \mid n_l^k >_{\mathbb{N}} n_l^{k-1}, 0 <_{\mathbb{N}} k \leq_{\mathbb{N}} m\})$
  $\cup \{dec_l \mid n_l^k >_{\mathbb{N}} n_l^{k+1}, 0 <_{\mathbb{N}} n_l^k, 0 \leq_{\mathbb{N}} k <_{\mathbb{N}} m\}$ where
  $$f(k) = \begin{cases} 1 & \text{if } k \bmod 2 = 0 \\ 2 & \text{otherwise} \end{cases}$$

We now show that M' is a model of α_M.

- We can easily show that M' satisfies the formulas $\mathsf{init}, \mathsf{fin}, \mathsf{state}, \mathsf{ctr}_1$ and ctr_2 by using the definition of V_M.

- We now show that M' satisfies trans_i. To show that M' satisfies $\bigvee_{t\in\delta} \theta_t$, consider an event $se_k \in E_k$ such that $V_k(se_k) = q^k$ where $q^k \neq q_f$. Then, by the definition of D_M, we know that the event $se_{k+1} \in E_{k+1}$ is such that $q^{k+1} \in V_{k+1}(se_{k+1})$ where $(q^k, n_1^k, n_2^k) \to (q^{k+1}, n_1^{k+1}, n_2^{k+1})$ in ρ (say, through the transition $t' \in \delta$). Using the definition of V_{k+1}, we can show that $M', se_k \models \theta_{t'}$. Let $t' = (q^k, x_1, y_1, q', x_1', y_1')$. Suppose $M', se_k \models \theta_{t''}$ for some $t'' \in T \setminus \delta$. Let $t'' = (q^k, x_2, y_2, q', x_2', y_2')$. By the definition of θ_t and from the fact that se_{k+1} is the unique i^{co}-successor to se_k (where se_k is an event of agent i), we know that $M', se_{k+1} \models q'$. But, $M', se_{k+1} \models q^{k+1}$ as well, and since $M', se_{k+1} \models \mathsf{state}$, we get $q' = q^{k+1}$. Similarly, using $\mathsf{z\text{-}consis}_i$, we can show that $x_1 = x_2$ and $y_1 = y_2$, and using $\mathsf{state\text{-}inc\text{-}dec\text{-}consis}_i$, we can show that $x_1' = x_2'$ and $y_1' = y_2'$. But then $t'' = t'$ contradicting the assumption that $t'' \notin \delta$.

- We now prove that M' satisfies consis_i. From the first definition on the local ordering of events in E_k and from the definition of V_k it follows that seq_i is satisfied by M'.

- To show that M' satisfies $\mathsf{z\text{-}consis}_i$, we know by the definition of V_M that c_jz is true at an event se_k only if state is true. In addition, we also know that c_jz is true at e_k iff $n_j^k = 0$ for $j = 1, 2$. If $n_1^k = 0$, then, from the definition of D_M, we see that there is no event of the form $c_1e(j,k)$ immediately preceding e_k.

Since c_1 holds only at events of the form $c_1e(j,k)$ we can conclude that $\overline{\mathbf{Y}}_i\neg c_1$ holds at se_k. Similarly, we can show that c_2z holds iff $\overline{\mathbf{X}}_i\neg c_2$ holds at se_k.

- We can show that M' satisfies $\mathsf{inc\text{-}consis}_i$, $\mathsf{dec\text{-}consis}_i$, $\mathsf{z\text{-}dec\text{-}consis}_i$ and $\mathsf{state\text{-}inc\text{-}dec\text{-}consis}_i$ by using the definition of V_M.

- Finally, to show that M' satisfies $\mathsf{across\text{-}consis}_i$, consider an event se_k such that $M', se_k \models \mathsf{state}$. If $k = m$, we know that $q_f \in V_m(se_k)$ and so $M', se_k \models q_f$. Also, if $k = 0$, we know that $ic \in V_k(se_k)$ and so $M', se_k \models ic$. For $0 < k < m$, by the definition of D_M, we know that $se_{k+1} \in E_{k+1}$ is such that $se_k \lessdot se_{k+1}$ and $q^{k+1} \in V_{k+1}(se_{k+1})$. Also, $\phi(se_{k+1}) = i^{co}$ if $\phi(se_k) = i$. This implies $M', se_k \models \mathbf{X}_{i^{co}}\mathsf{state}$. Similarly, $se_{k-1} \in E_{k-1}$ is such that $se_{k-1} \lessdot se_k$ and $\phi(se_{k-1}) = i^{co}$ if $\phi(se_k) = i$. This implies $M', se_k \models r_i^{i^{co}}$.

  Now, consider an event e such that $M', e \models c_1$ where e is of the form $c_1e(j,k)$. If $j < n_1^k$ then, we know from the definition of M' that $c_1e(j,k+1)$ is such that $c_1e(j,k) \lessdot c_1e(j,k+1)$, $M', c_1e(j,k+1) \models c_1$ and $\phi(c_1e(j,k+1)) = i^{co}$ if $\phi(c_1e(j,k)) = i$. Hence $M', c_1e(j,k) \models \mathbf{X}_{i^{co}}c_1$. Suppose $n_1^{k+1} = n_1^k - 1$. Then, we know from the definition of V_M that dec_1 holds at se_k and hence $M', c_1e(n_1^k,k) \models \mathbf{X}_i(\mathsf{state} \wedge dec_1)$.

  Since $k > 0$, for $j < n_1^k$, the event $c_1e(j,k-1)$ is such that $c_1e(j,k-1) \lessdot c_1e(j,k)$ and $\phi(c_1e(j,k-1)) = i^{co}$ if $\phi(c_1e(j,k)) = i$. Hence $M', c_1e(j,k) \models r_i^{i^{co}}$. When $n_1^{k+1} = n_1^k + 1$, we can show that $\mathbf{X}_i(\mathsf{state} \wedge inc_1)$ holds at $c_1e(n_1^k,k)$.

  We can argue as above using the definition of $\leq$ in D_M and the definition of the valuation function to show that the sub-formula involving c_2 also is satisfied.

Conversely, suppose α_M is satisfiable. We have to show that a final configuration of M is reachable. We will prove the existence of a run ρ of M such that ρ terminates at a final configuration. Let $M' = (D, V)$ where $D = (E, \leq, \phi)$ be a model of α_M. Since M' satisfies init, we know that there exists $e_0 \in E$ such that $M', e_0 \models \mathsf{init}$ and since M' satisfies fin, we know that there exists an event $e_m \in E$ such that $e_0 \leq e_m$ and $M', e_m \models q_f$.

Claim: There exists a set $X = \{f_0, f_1, \ldots f_{m'}\}$ such that $e_0 = f_0 \lessdot f_1 \lessdot \ldots \lessdot e_m = f_{m'}$, where $\phi(f_0) = 1$ and if $\phi(f_k) = i$ then $\phi(f_{k+1}) = i^{co}$ for $0 \leq_{\mathbb{N}} k <_{\mathbb{N}} m$. Also, $M', f_k \models \mathsf{state}$ for all k and for $k \geq 2$, there exists no event $e \in E$ such that $f_j < e < f_{j+2}$, $0 \leq_{\mathbb{N}} j <_{\mathbb{N}} (k-1)$ and $M', e \models \mathsf{state}$.

Proof of claim: We prove this claim by induction on k. Initially, set $f_0 = e_0$. $\phi(e_0) = 1$ since $M', e_0 \models \mathsf{init}$. If $q_0 = q_f$, then $e_0 = e_m$ and we are done.

Otherwise, inductively suppose $f_0 \lessdot f_1 \lessdot \ldots \lessdot f_k$ $(k \geq 0)$ has been defined such that $f_0, f_1, \ldots f_k$ satisfy the above properties.

We define f_{k+1} as follows. Suppose $\phi(f_k) = i$ for $i \in \{1, 2\}$. Since $M', f_k \models \mathsf{state}$ we know that $M', f_k \models \mathsf{across\text{-}consis}_i$ and so there exists $e' \in E$ such that $f_k \lessdot e'$, $\phi(e') = i^{co}$ and $M', e' \models \mathsf{state}$. Set $f_{k+1} = e'$.

We now show that there exists no event $e_1 \in E$ such that $f_{k-1} < e_1 < f_{k+1}$ $(k \geq 1)$ such that $M', e_1 \models \mathsf{state}$. Suppose there exists such an event e_1. Then, $M', e_1 \models r^i_{i^{co}}$ (since $\mathsf{across\text{-}consis}_{i^{co}}$ holds at e_1) and so there exists an event $e_2 \in E$ such that $e_2 \lessdot e_1$ and $\phi(e_2) = i$. Now $f_k \not\leq e_2$ as it would contradict the fact that $f_k \lessdot f_{k+1}$. Hence $e_2 \leq f_k$. We can show that $M', e_2 \models \mathsf{state}$ using the fact that state holds at e_1 and that e_2 is the unique i-predecessor of e_1. This contradicts the inductive assumption for the events f_{k-2} and f_k.

If the above sequence of events terminates at some $f_{m'}$ such that $f_{m'} = e_m$, we are done. Otherwise, since $e_0 \leq e_m$, we know that there exists a sequence of events $e_0 \leq e_1 \leq \ldots e_m$ in D. Let $f_0, f_1, \ldots, f_\ell$ $(\ell > 0)$ be the sequence of events such that $e_j = f_j$ for all j, $0 \leq_{\mathbb{N}} j \leq_{\mathbb{N}} \ell$ and $e_{\ell+1} \neq f_{\ell+1}$. That is, $f_0, f_1, \ldots, f_\ell$ is the largest common prefix of the sequences $e_0, e_1, \ldots, e_m$ and $f_0, f_1, \ldots, f_{m'}$. We will now show that there exists a path $f_0 \lessdot f_1 \lessdot \ldots f_\ell \lessdot f_{\ell+1} \leq e_m$ in D.

Suppose $\phi(f_\ell) = \phi(e_m) = i$. Now, since $M', e_m \models \mathsf{state}$, we know from $\mathsf{across\text{-}consis}_i$ that there exists an event e' such that $e' \lessdot e_m$, $\phi(e') = i^{co}$. We can again argue that $M', e' \models \mathsf{state}$. Since $\phi(e') = \phi(f_{\ell+1}) = i^{co}$, and $f_\ell \lessdot f_{\ell+1}$, we know that $f_{\ell+1} \leq e'$. Hence $f_0 \lessdot f_1 \lessdot \ldots f_\ell \lessdot f_{\ell+1} \leq e' \lessdot e_m$ in D. On the other hand, if $\phi(f_\ell) \neq \phi(e_m)$ then, we know that $\phi(f_{\ell+1}) = \phi(e_m)$ and so $f_0 \lessdot f_1 \lessdot \ldots f_\ell \lessdot f_{\ell+1} \leq e_m$ in D.

We can repeat the above argument for the events $f_{\ell+2}, f_{\ell+3}$ and so on till we reach an event $f_{m'}$ such that $f_{m'} = e_m$. Hence induction on k is complete and we have: $X = \{f_0, f_1, \ldots f_{m'}\}$ as above and the claim is proved.

To complete the proof of the theorem, we define a sequence
$\rho = (q_0, n^0_1, n^0_2), (q_1, n^1_1, n^1_2), \ldots, (q_m, n^m_1, n^m_2)$

where $q_m = q_f$, q_k is such that $M', f_k \models q_k$ and n^k_1, n^k_2 are defined as follows. Initially, $n^0_1 = n^0_2 = 0$. To define n^k_1 and n^k_2 for $k > 0$, consider the event f_k. If $M', f_k \models c_i z$, we define $n^k_i = 0$ for $i = 1, 2$. Otherwise, we know that there exists at least one event immediately before f_k satisfying c_1. Let $e_1, e_2, \ldots, e_m$ be the maximum sequence of events in E such that $e_m \lessdot \ldots \lessdot e_1 \lessdot f_k$, $\phi(f_k) = \phi(e_j)$ and $V(e_j) = c_1$ for all j, $1 \leq j \leq m$. Define $n_1^k = m$. Similarly, we define $n_2^k = m'$ where $e'_1, e'_2, \ldots e'_{m'}$ is the maximum sequence of events in E such that $f_k \lessdot e'_1 \lessdot \ldots \lessdot e'_{m'}$, $\phi(f_k) = \phi(e'_j)$ and $V(e'_j) = c_2$ for all j, $1 \leq j \leq m'$.

We now show that $(q_k, n_1^k, n_2^k) \rightarrow (q_{k+1}, n_1^{k+1}, n_2^{k+1})$ for all k, $0 \leq k \leq m-1$ by using the fact that $M' \models \mathsf{trans}_i$ and that $M' \models \mathsf{consis}_i$ for $i = 1, 2$. Consider the event f_k which satisfies q_k. Suppose $\phi(f_k) = i$. We know that f_{k+1} is such that $\phi(f_{k+1}) = i^{co}$ and $M', f_{k+1} \models q_{k+1}$. Since $M', f_k \models \mathsf{trans}_i$ and f_{k+1} is the unique immediate successor of f_k in i^{co}, it follows that there exists a transition $t = (q_k, x, y, q_{k+1}, x', y') \in \delta$ such that $M', f_k \models \theta_t$. Let $\theta_t = r_x^1 \wedge r_x^2 \wedge \mathbf{X}_{i^{co}} q_{k+1} \wedge \xi^1_{x',i} \wedge \xi^2_{y',i}$. We have to show that $n_1^{k+1} = n_1^k + x'$ and $n_2^{k+1} = n_2^k + y'$. Suppose $x' = -1$. Then, since $M', f_k \models \xi^1_{-1,i}$, we know that $M', f_k \models dec_1$ and there exists an event e' such that $f_k \lessdot e'$, $\phi(e') = i^{co}$ and $M', e' \models \neg inc_1$. Now using the fact that M' satisfies $\mathsf{dec\text{-}consis}_i$, we know that there exists an event e'' satisfying c_1 which is immediately before f_k in i and it does not have an immediate successor in i^{co}. Also, $\mathsf{across\text{-}consis}_i$ holds and so all the events in the past of e'' satisfying the proposition c_1 (representing n_1^k in agent i) have successors in agent i^{co}. So, all the events satisfying c_1 representing the counter value n_1^k get copied to agent i^{co} except for the event e''. Hence we can conclude that $n_1^{k+1} = n_1^k - 1$. We can similarly argue for the other values of x' and y' using the fact that M' satisfies $\mathsf{inc\text{-}consis}_i \wedge \mathsf{z\text{-}dec\text{-}consis}_i$ (when x' or y' is 1) to show that $n_1^{k+1} = n_1^k + x'$ and $n_2^{k+1} = n_2^k + y'$. Hence ρ as defined above is a run of M and since $q_m = q_f$ it follows that ρ is a run of M ending in a final configuration.

This completes our demonstration that the reduction of the halting problem for 2-counter machines to finite satisfiability of LD_1 is correct. $\Box$

The proof of Theorem 4.1.1 follows from the above proof as the formulas of LD_1 are also formulas of LD_0 with the occurrences of the formula $\neg r_j^i$ replaced by the formula $\tau_j \wedge \overline{\mathbf{Y}}_i False$ and the formula r_j^i by $\tau_j \wedge \mathbf{Y}_i True$ for $i, j \in \{1, 2\}$.

Theorem 4.2.2 *The (finite) satisfiability problem for LD_2 is undecidable.*

Proof: The proof proceeds similarly as above, defining a formula α_M of LD_2 for a given 2-counter machine M. We have to re-write some of the formulas used in the proof of Theorem 4.2.1 using the global $\mathbf{Y}$ modality and special send propositions instead of the global $\mathbf{X}$ modality. In addition, we create a new event preceding the event corresponding to the initial configuration. This helps us to re-write some of formulas in a way that is symmetric to the corresponding ones in LD_1. For the sake of clarity, we only mention the formulas that are re-written below. The others will be as in the proof of Theorem 4.2.1.

- $\mathsf{init} \overset{\text{def}}{=} \tau_1 \wedge ic \wedge q_0 \wedge c_1 z \wedge c_2 z \wedge \overline{\mathbf{Y}}_1 False \wedge \overline{\mathbf{X}}_1 \mathbf{G}_1 \neg ic \wedge \mathbf{Y}_2((q_0 \wedge c_1 z \wedge c_2 z) \wedge \overline{\mathbf{Y}}_2 False \wedge \overline{\mathbf{X}}_2 \mathbf{G}_2 \neg ic)$.
- $\mathsf{fin} \overset{\text{def}}{=} \mathbf{F}_1 \mathsf{fin}_1 \vee \mathbf{Y}_2 \mathbf{F}_2 \mathsf{fin}_2$ where $\mathsf{fin}_1 \overset{\text{def}}{=} q_f \wedge \mathbf{G}_1 \neg s_1^2$ and $\mathsf{fin}_2 \overset{\text{def}}{=} q_f \wedge \mathbf{G}_2 \neg s_2^1$.
- $\mathsf{inv} \overset{\text{def}}{=} \mathbf{G}_1 \mathsf{inv}_1 \wedge \overline{\mathbf{Y}}_2 \mathbf{G}_2 \mathsf{inv}_2$.
- For $t = (q, x, y, q', x', y') \in T$ and $q \neq q_f$, we define

  $$\theta_t \overset{\text{def}}{=} (q' \wedge \neg ic) \wedge (\mathbf{Y}_{i^{co}}(\gamma_x^1 \wedge \gamma_y^2 \wedge q) \wedge \xi_{x',i}^1 \wedge \xi_{y',i}^2))$$

  where

  $$\gamma_0^j = c_j z \qquad \gamma_1^j = \neg c_j z \qquad \xi_{0,i}^j = \neg inc_j \wedge \mathbf{Y}_{i^{co}} \neg dec_j$$
  $$\xi_{1,i}^j = inc_j \wedge \mathbf{Y}_{i^{co}} \neg dec_j \qquad \xi_{-1,i}^j = \neg inc_j \wedge \mathbf{Y}_{i^{co}} dec_j$$

  Now, $\mathsf{trans}_i \overset{\text{def}}{=} (q' \wedge \neg ic) \supset \bigvee_{t=(q,x,y,q',x',y') \in \delta} \theta_t \wedge \bigwedge_{t \notin \delta} \neg \theta_t$.
- $\mathsf{dec\text{-}consis}_i \overset{\text{def}}{=} (((\mathsf{state} \wedge dec_1) \supset \mathbf{Y}_i(c_1 \wedge \neg s_i^{i^{co}})) \wedge ((\mathsf{state} \wedge dec_2) \supset \mathbf{X}_i(c_2 \wedge \neg s_i^{i^{co}})))$.
- $\mathsf{inc\text{-}consis}_i \overset{\text{def}}{=} (((\mathsf{state} \wedge inc_1) \supset \mathbf{Y}_i(c_1 \wedge \overline{\mathbf{Y}}_{i^{co}} False)) \wedge ((\mathsf{state} \wedge inc_2) \supset \mathbf{X}_i(c_2 \wedge \overline{\mathbf{Y}}_{i^{co}} False)))$
- $\mathsf{across\text{-}consis}_i \overset{\text{def}}{=} (\mathsf{state} \supset (ic \vee \mathbf{Y}_{i^{co}} \mathsf{state}) \wedge (q_f \vee s_i^{i^{co}})) \wedge$
  $c_1 \supset ((s_i^{i^{co}} \vee \mathbf{X}_i(\mathsf{state} \wedge dec_1)) \wedge (\mathbf{Y}_{i^{co}} c_1 \vee \mathbf{X}_i(\mathsf{state} \wedge inc_1))) \wedge$
  $c_2 \supset ((s_i^{i^{co}} \vee \mathbf{Y}_i(\mathsf{state} \wedge dec_2)) \wedge (\mathbf{Y}_{i^{co}} c_2 \vee \mathbf{Y}_i(\mathsf{state} \wedge inc_2)))$.
- $\mathsf{state\text{-}inc\text{-}dec\text{-}consis}_i \overset{\text{def}}{=} \mathsf{state} \supset ((ic \vee \mathbf{Y}_{i^{co}} \mathsf{dec\text{-}options}) \wedge \mathsf{inc\text{-}options})$ where dec-options and inc-options are as defined for LD_1.

It is easy to see that relating models of α_M and halting runs of M can be carried out as in the undecidability proof for LD_1. Given a run ρ of M ending in a final configuration, we define a model of α_M as done in the above proof. In addition, we add the set E'_0 to E and extend the valuation function suitably to take care of the extra initial event satisfying ic.

That is, define a model $M' = (D', V')$ where $D' = (E', \leq', \phi')$ is given by

- $E' = E'_0 \cup E$ where $E'_0 = \{e'_0\}$ and E is as defined in the above proof.
- $\phi'(e'_0) = 2$ and $\phi'(e) = \phi(e)$ for every $e \in E$.
- $\leq' = (\leq' \cup \{(e'_0, e'_0) \cup (e'_0, se_0)\})^*$.

Finally, the valuation function V' is given by $V'(e'_0) = \{\tau_1, q_0, c_1z, c_2z, ic\}$ and $V'(e) = V(e)$ for all $e \in E$.

We now show that M' is a model of α_M by arguing that M' satisfies all the formulas which were re-defined for α_M in this theorem.

To start with, $M', se_0 \models \mathsf{init}$ as $V_0 = \{q_0, c_1z, c_2z, ic\}$ and $e'_0 <_c se_0$ is such that $\phi'(e'_0) = 2$ and $M', e'_0 \models (q_0 \wedge c_1z \wedge c_2z \wedge ic)$. We can show that the formulas fin, trans_i, $\mathsf{inc\text{-}consis}_i$ and $\mathsf{dec\text{-}consis}_i$ are all satisfied by M' as done in the proof of Theorem 4.2.1 above. To show that M' satisfies $\mathsf{across\text{-}consis}_i$, we first note that $\mathbf{X}_{i^{co}}\beta \supset s_i^{i^{co}}$ for any formula β which is not *False* and the truth of sub-formulas involving the $\mathbf{Y}_{i^{co}}$ modality in $\mathsf{across\text{-}consis}_i$ can be proved in a way similar to the proof of Theorem 4.2.1.

Conversely, suppose α_M is satisfiable. We construct a run ρ of M such that ρ terminates at a final configuration as done in the proof of Theorem 4.2.1 above. The set X of events satisfying states is defined in the same way and we prove that events in X can be arranged in a sequence by working backwards from the event e_m which satisfies q_f. We can again define ρ and show that it defines a run of M to a configuration containing q_f. $\square$

4.3 Coping with undecidability

The results in this chapter show that our first attempts to define logics which are expressive and decidable are not successful. It turns out that we get undecidability even if one of the next or previous modalities is local in the presence of special send or receive propositions.

With the aim of obtaining decidable and expressive logics over Lamport diagrams, we now consider the various restricting approaches. Two natural techniques can be followed to possibly obtain decidable logics— one way is to further restrict the expressive power of modalities (and omit/restrict the special send and receive propositions) and the other is consider restricted or more structured sub-classes of Lamport diagrams as models of formulas.

Restricted logics

The first approach we consider is to syntactically restrict the logics that we define over Lamport diagrams. We consider one such logic called m-LTL in the next chapter. This is a local temporal logic over Lamport diagrams—the syntax includes local temporal modalities ($\mathbf{X}$ and $\mathbf{U}$) for each of the n agents along with a non-local previous ($\mathbf{Y}$) modality. The previous modality is also less expressive in the sense that it asserts the truth of a formula only at the "last" event of a particular agent in its past and not at the send event. It turns out that this logic is expressive enough to specify interesting properties of Lamport diagrams and the satisfiability problem for this logic is also decidable.

Restricted models

The second approach of considering Lamport diagrams with more structure as models is taken in subsequent chapters. We consider layered Lamport diagrams as models. The syntax of the logic λ-LTL that we consider is also tuned to the structure of LLDs. The logic has a two-level syntax: a temporal logic is defined at the top level assuming an LLD to be a sequence of layers. This top level temporal logic is built on formulas from a logic like LD_0 which serves to zoom into each layer of the LLD and talk about it structure. We then show that the satisfiability problem of this logic is undecidable over the class of models based on bounded LLDs and over the class of models based on communication closed LLDs. However, imposing additional structure results in the satisfiability problem being decidable. The logic becomes decidable over the class of models based on communication closed and bounded LLDs and over the class of models based on channel bounded LLDs.

The following table summarizes the various logics that we consider in this thesis along with the results regarding decidability of the satisfiability problem. The stated results for the logics m-LTL and λ-LTL are presented in the next two chapters.

Logic	*Syntax*	*Models*	*Satisfiability*
LD_0	Global $\mathbf{X}$ and $\mathbf{Y}$	Lamport diagrams	Undecidable
LD_1	Global $\mathbf{X}$, local $\mathbf{Y}$ and special receive propositions	Lamport diagrams	Undecidable
LD_2	Local $\mathbf{X}$, global $\mathbf{Y}$ and special send propositions	Lamport diagrams	Undecidable
m-LTL	Local $\mathbf{X}$, weakly global $\mathbf{Y}$, local $\mathbf{U}$	Lamport diagrams	Decidable
λ-LTL	Global $\mathbf{X}$ and $\mathbf{Y}$	Bounded LLDs	Undecidable
λ-LTL	Global $\mathbf{X}$ and $\mathbf{Y}$	Communication closed LLDs	Undecidable
λ-LTL	Global $\mathbf{X}$ and $\mathbf{Y}$	Bounded and communication closed LLDs	Decidable
λ-LTL	Global $\mathbf{X}$ and $\mathbf{Y}$	Channel bounded LLDs	Decidable

Chapter 5

A temporal logic over Lamport diagrams

In the previous chapter we saw that the main reason behind the undecidability of the logics LD_i (for $i \in \{1, 2, 3\}$) is the presence of global $\mathbf{X}$ and $\mathbf{Y}$ modalities. We now consider a restricted temporal logic where both the $\mathbf{X}$ and $\mathbf{Y}$ modalities are indexed, but the $\mathbf{X}$ modality is local and the $\mathbf{Y}$ modality is *weakly global*. Using such a modality, a particular state of agent i in a Lamport diagram can assert the truth of a formula at the *last* j-local state (for some $j \neq i$) seen in its past (which need not be through a direct communication between i and j). We call this temporal logic **m-LTL**. This logic is used to reason about local assertions on Lamport diagrams. It is 'locally linear time' in the sense of [Ram96]; locally, it is linear time temporal logic and in addition, it includes a weakly global past modality to refer to communications in the past.

The main aim of this chapter is to show that the satisfiability problem of m-LTL is decidable. We will solve this problem using the so-called automata-theoretic approach to satisfiability, i.e., given an m-LTL formula ψ, we will construct an SCA S_ψ such that the poset language accepted by S_ψ is the set of all models of ψ. Then, ψ is satisfiable iff the SCA S_ψ accepts a non-empty poset language. Since the latter problem is decidable, we also get decidability of the satisfiability problem.

The intuitive reason behind obtaining decidability of this logic (unlike the ones in the previous chapter) is the fact that a particular agent cannot assert the sending or a receipt of a particular message using just the local $\mathbf{X}$ modality and the weakly global $\mathbf{Y}$ modality. In other words, the communication modality is asymmetric, i.e., the receiver of a message gets information about the sender's past, whereas the sender cannot access the receiver's future.

Interestingly, it turns out that this logic is expressive enough to act as a specification language to talk about local properties of Lamport diagrams. We present an example of a conference management system and illustrate the usefulness of m-LTL as a specification language by writing properties of this system using formulas from m-LTL. Many interesting safety and liveness properties involving distributed systems can be specified using formulas from m-LTL. The fact that m-LTL acts as a natural specification language also motivates us to consider the model checking problem for m-LTL. The automata-theoretic approach to prove that satisfiability problem is decidable also yields a proof of the decidability of the model checking problem.

5.1 The logic m-LTL

Fix countable sets of *propositional letters* $(P_1, P_2, \ldots, P_n)$, where P_i consists of the atomic local properties of agent i. We assume that $P_i \cap P_j = \emptyset$ for $i \neq j$. Let $P \stackrel{\text{def}}{=} \bigcup\limits_i P_i$.

Let $i \in [n]$. The syntax of *i-local formulas* is given below:

$$\Phi_i ::= p \ \in \ P_i \mid \neg\ \alpha \mid \alpha_1 \ \vee \ \alpha_2 \mid \bigcirc\ \alpha \mid \alpha_1 \ \mathbf{U}\ \alpha_2 \mid \oslash_j\ \alpha,\ j \neq i,\ \alpha \ \in \ \Phi_j$$

Global formulas are obtained by boolean combination of local formulas:

$$\Psi ::= \alpha@i,\ \alpha \ \in \ \Phi_i \mid \neg\ \psi \mid \psi_1 \ \vee \ \psi_2$$

The propositional connectives $(\wedge, \supset, \equiv)$ and derived temporal modalities $(\Diamond, \Box)$ are defined as usual. The dual of $\oslash_j\alpha$ is given by $\otimes_j\alpha \stackrel{\text{def}}{=} \neg\oslash_j\neg\alpha$.

The formulas are interpreted on Lamport diagrams. For technical convenience, we consider only infinite behaviours. Formally, models are 2^P-labelled Lamport diagrams over a countable set of events. We will use the notation $M = (E, \leq, \phi, 2^P)$ to denote such diagrams. The labelling of the events of M by subsets of P can be thought of as associating a valuation function with the Lamport diagram. We choose to consider the valuation function as a label as it would be easier to associate an SCA with every formula later.

While defining the semantics and in the decidability proof, we borrow notations related to Lamport diagrams from Chapter 2.

Let $\alpha \in \Phi_i$ and $d \in LC_i$. The notion that α *holds in the local state* d of agent i in model M is denoted $M, d \models_i \alpha$, and is defined inductively as follows:

- $M, d \models_i p$ iff $p \in \phi(e)$ where $d = \downarrow e$.
- $M, d \models_i \neg\alpha$ iff $M, d \not\models_i \alpha$.
- $M, d \models_i \alpha \vee \beta$ iff $M, d \models_i \alpha$ or $M, d \models_i \beta$.
- $M, d \models_i \bigcirc\alpha$ iff there exists $d' \in LC_i$ such that $d \lessdot d'$ and $M, d' \models_i \alpha$.
- $M, d \models_i \alpha \mathbf{U} \beta$ iff $\exists d' \in LC_i$: $d \subseteq d', M, d' \models_i \beta$ and $\forall d'' \in LC_i : d \subseteq d'' \subset d'$: $M, d'' \models_i \alpha$.
- $M, d \models_i \oslash_j \alpha$ iff there exists $d' \in LC_j$ such that $d' \lessdot d$ and $M, d' \models_j \alpha$.

The modalities $\bigcirc$ (next) and $\mathbf{U}$ (until) are as in standard linear time temporal logic and are interpreted at the local future of a state. The new weakly global modality $\oslash_j \alpha$, asserted by i, says that α held in the last j-local state visible to i. Notice that this need not be through a 'direct edge' from j to i, i.e., there need not be any communication between i and j directly.

Global satisfaction is defined in terms of local satisfaction at the initial events:

- $M \models \alpha @ i$ iff $M, \epsilon_i \models_i \alpha$.
- $M \models \neg\psi$ iff $M \not\models \psi$.
- $M \models \psi_1 \vee \psi_2$ iff $M \models \psi_1$ or $M \models \psi_2$.

We say that ψ is *satisfiable* iff there exists a model M such that $M \models \psi$. We say that ψ is *valid* if for every model M, we have $M \models \psi$. Let *Models*$(\psi) = \{M \mid M$ is a model of $\psi\}$. The *satisfiability problem* of m-LTL is to check if a given formula ψ is satisfiable or not.

A typical specification in the logic has the form: $(\Box(p \wedge \oslash_2 \neg$ 'OK' $\supset \bigcirc(q \wedge \oslash_2$ 'OK')))@1, which asserts that agent 1 can make a transition from a state satisfying p into a state in which q holds only after hearing an 'OK' from agent 2, and must block otherwise. We will see more examples of specifications using this logic later in the chapter.

5.2 m-LTL Satisfiability

In this section we show that the satisfiability problem for m-LTL can be settled using SCAs. We now show that one can effectively associate an SCA S_ψ with each m-LTL formula ψ in such a way that $\mathcal{L}^{po}(S_\psi) = \mathit{Models}(\psi)$.

Theorem 5.2.1 *Let ψ be a m-LTL formula of length m. Satisfiability of ψ over n-agent Lamport diagrams can be checked in time $2^{O(mn)}$.*

The theorem is proved by associating an SCA S_ψ over the distributed alphabet $(2^{P_1}, \ldots, 2^{P_n})$ with every formula ψ such that $\mathcal{L}^{po}(S_\psi) = \mathit{Models}(\psi)$.

As mentioned earlier, we take the well-known automata-theoretic approach to deciding satisfiability [VW86]. For linear time temporal logic, given a formula ψ, Vardi and Wolper show how to associate a Büchi automaton which accepts precisely the models of ψ. The automaton has atoms corresponding to ψ as its states, the transition relation is defined to capture the $\bigcirc$ requirements and the $\mathbf{U}$ requirements are checked through accepting states. We adopt the same idea in this setting. The construction is extended to define local automata, one for each agent and we also have to do additional work to capture the $\oslash_j$ requirements through λ-constraints of the SCA.

As usual, we begin with the definition of sub-formula closure. We can define, for any global formula ψ, the sets of sub-formulas $CL(\psi)$ and CL_i for $i \in [n]$, by simultaneous induction in such a way that:

- $\psi \in CL(\psi)$.
- $\alpha@i \in CL(\psi)$ iff $\alpha \in CL_i$.
- if $\psi' \in CL(\psi)$ then $\neg\psi' \in CL(\psi)$; a similar condition holds for CL_i and here $\neg\neg$ is treated as identity.
- if $\psi_1 \vee \psi_2 \in CL(\psi)$ then $\psi_1, \psi_2 \in CL(\psi)$.
- if $\beta_1 \vee \beta_2 \in CL_i$ then $\beta_1, \beta_2 \in CL_i$.
- if $\bigcirc\beta \in CL_i$ then $\beta \in CL_i$.
- if $\beta_1\mathbf{U}\beta_2 \in CL_i$ then β_1, β_2, $\bigcirc(\beta_1\mathbf{U}\beta_2) \in CL_i$.

- if $\oslash_j\beta \in CL_i$ then $\beta \in CL_j$.

It can be checked that $|CL(\psi)|$ is linear in the size of ψ. For the rest of this section, fix a global formula $\psi_0 \in \Psi$. We will refer to $CL(\psi_0)$ simply as CL and CL_i will refer to the associated sets of i-local formulas. We also use $U_i \overset{\text{def}}{=} \{\alpha\mathbf{U}\beta \mid \alpha\mathbf{U}\beta \in CL_i\}$, and $L_i^j \overset{\text{def}}{=} \{\oslash_j\alpha \mid \oslash_j\alpha \in CL_i\}$.

We say that $A \subseteq CL_i$ is an *i-atom* iff it satisfies the following conditions:

- for every formula $\alpha \in CL_i$, either $\alpha \in A$ or $\neg\alpha \in A$ but not both.
- for every formula $\alpha \vee \beta \in CL_i$, $\alpha \vee \beta \in A$ iff $\alpha \in A$ or $\beta \in A$.
- for every formula $\alpha\mathbf{U}\beta \in CL_i$, $\alpha\mathbf{U}\beta \in A$ iff $\beta \in A$ or $\{\alpha, \bigcirc(\alpha\mathbf{U}\beta)\} \subseteq A$.

Let AT_i denote the set of all i-atoms. Let $AT \overset{\text{def}}{=} \bigcup_i AT_i$. Let $\widetilde{AT}$ denote the set $AT_1 \times \ldots \times AT_n$. We let $\widetilde{X}, \widetilde{Y}$ etc to range over $\widetilde{AT}$, and $\widetilde{X}[i]$ to denote the i-atom in the tuple.

Let ψ be a global formula. We define the notion $\psi \in \widetilde{X}$ as follows: if $\alpha \in \Phi_i$, then $\alpha@i \in \widetilde{X}$ iff $\alpha \in \widetilde{X}[i]$; $\neg\psi \in \widetilde{X}$ iff $\psi \notin \widetilde{X}$; $\psi_1 \vee \psi_2 \in \widetilde{X}$ iff $\psi_1 \in \widetilde{X}$ or $\psi_2 \in \widetilde{X}$.

Define $\rightsquigarrow$ on AT as follows: $B \rightsquigarrow A$ iff for some $i \neq j$, $A \in AT_i$, $B \in AT_j$ and for all $\oslash_j\alpha \in CL_i$, $\oslash_j\alpha \in A$ iff $\alpha \in B$. Let $\widetilde{X} \in \widetilde{AT}$. The relation $\rightsquigarrow$ will be used to capture the $\oslash_j$-requirement at agent i through λ-transitions. We say $\widetilde{X}$ is *self-contained*, iff for all $i \neq j$, $\widetilde{X}[j] \rightsquigarrow \widetilde{X}[i]$.

We are now ready to associate an SCA with the given formula in the standard manner. For $i \in [n]$, $\Sigma_i \overset{\text{def}}{=} 2^{P_i}$ constitute the distributed alphabet over which the SCA is defined.

Definition 5.2.2 *Given any formula ψ_0, the* **SCA associated with** ψ_0 *is defined by:*

$$S_{\psi_0} \overset{\text{def}}{=} ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$$

where:

- $Q_i = \{(A, u, I, J) \mid A \in AT_i, u \subseteq (U_i \cap A), \{I, J\} \subseteq 2^{([n]\setminus\{i\})}$ *such that* $I \subseteq J\}$.
- $G_i = \{(A, \emptyset, ([n] \setminus \{i\}), ([n] \setminus \{i\})) \mid A \in AT_i$ *such that* $\bigcirc\beta \notin A$ *for all* $\bigcirc\beta \in CL_i\}$.

- *Init* $= \{((A_1, \emptyset, \emptyset, \emptyset), \ldots, (A_n, \emptyset, \emptyset, \emptyset)) \mid \psi_0 \in (A_1, \ldots, A_n)$ *and* $(A_1, \ldots, A_n)$ *is self-contained* $\}$.

- $(A, u, I, J) \xrightarrow{P'} (B, v, I', J')$, *where* $A, B \in AT_i$, *iff*

  1. $P' = A \cap P_i$.
  2. *there exists a formula* $\bigcirc\alpha \in CL_i$ *such that* $\bigcirc\alpha \in A$.
  3. *for every* $\bigcirc\beta \in CL_i$, $\bigcirc\beta \in A$ *iff* $\beta \in B$.
  4. *The set* v *is defined as follows:*

$$v = \begin{cases} \{\alpha \mathbf{U} \beta \in B \mid \beta \notin B\} & \text{if } u = \emptyset \\ \{\alpha \mathbf{U} \beta \in u \mid \beta \notin B\} & \text{otherwise} \end{cases}$$

  5. $J' = \{j \mid L_i^j \cap A \neq L_i^j \cap B\}$.
  6. *for all* $j \in I'$, *there exists* $C \in AT_j$ *such that* $C \rightsquigarrow B$.

- $(A, u, I, J) \xrightarrow{\lambda} (B, v, I', J')$ *iff for some* i, j *such that* $i \neq j$, $A \in AT_i$, $B \in AT_j$ *and*

  1. $A \rightsquigarrow B$.
  2. $i \in I' \cap J'$.
  3. *for all* $k \in J' \setminus I'$, $\oslash_k\gamma \in A$ *iff* $\oslash_k\gamma \in B$.

We will denote S_{ψ_0} by S_0 from now on.

The two components I and J in an i-local state $q = (A, u, I, J)$ are used to take care of the $\oslash_j\alpha$ requirements at q. The set J has all the indices $j \neq i$ such that $\oslash_j\alpha$ is asserted at q but not in the previous i-local state and its subset I is the set of all indices k for which the corresponding $\oslash_k\beta$ assertion is via a 'direct edge'. That is, whenever some $\oslash_k\beta$ is asserted at q where $k \in I$ then, the definition of λ-constraints makes sure that ${}^{\bullet}q \cap Q_k \neq \emptyset$.

Lemma 5.2.3 $\mathcal{L}^{po}(S_0) = Models(\psi_0)$.

Proof: Consider $M = (E, \leq, \phi, 2^P) \in \mathcal{L}^{po}(S_0)$ and let ρ be an accepting run of S_0 on M. We have to show that M is a model of ψ_0.

Let d be an i-local configuration of M. We associate an i-atom A_d with d as follows: for all d, $\rho(d)[i]$ is a tuple (A, u, I, J), set $A_d = A$.

The following assertion can be proved by induction on the structure of formulas in CL_i.

Claim: For all $\alpha \in CL_i$, for all $d \in LC_i$, $M, d \models_i \alpha$ iff $\alpha \in A_d$.

Proof: As mentioned above, we prove the claim by induction on the structure of α.

($\alpha = p \in P_i$) $M, d \models_i p$ iff $p \in \phi(e)$, (where $d = \downarrow e$) iff $p \in A_d$.

($\alpha = \neg\beta$) $M, d \models_i \neg\beta$ iff $M, d \not\models_i \beta$ iff (by the induction hypothesis) $\beta \notin A_d$ iff $\neg\beta \in A_d$ (by the definition of an atom).

($\alpha = \beta \vee \gamma$) $M, d \models_i \beta \vee \gamma$ iff $M, d \models_i \beta$ or $M, d \models_i \gamma$ iff (by the induction hypothesis) $\beta \in A_d$ or $\gamma \in A_d$ iff $\beta \vee \gamma \in A_d$ (by the definition of an atom).

($\alpha = \bigcirc\beta$) Suppose $M, d \models_i \bigcirc\beta$. We must show that $\bigcirc\beta \in A_d$. Since $M, d \models_i \bigcirc\beta$, there exists $d' \in LC_i$ such that $d \lessdot d'$ and $M, d' \models_i \beta$. By the induction hypothesis, $\beta \in A_{d'}$. Let $d = \downarrow e$ and $d' = \downarrow e'$ for some $e' \in E_i$. If we show that $\rho(d)[i] \overset{A_d \cap P_i}{\rightarrow} \rho(d')[i]$ then, by the definition of $\rightarrow$, we will have $\bigcirc\beta \in A_d$ as required. Now, $d \lessdot d'$ and $d, d' \in LC_i$ implies that $e \lessdot e'$ in M. Now, $e' \in E_i$ implies $\phi(e') \in \Sigma_i$ and since ρ is a run of S on M, we have $\rho(d)[i] \overset{A_d \cap P_i}{\rightarrow} \rho(d')[i]$ as required.

Conversely, suppose $\bigcirc\beta \in A_d$. We must show that $M, d \models_i \bigcirc\beta$. By the induction hypothesis and by the semantics of the modality $\bigcirc$, it suffices to prove that there exists $d' \in LC_i$ such that $d \lessdot d'$ and $\beta \in A_{d'}$. Again, suppose $d = \downarrow e$ for some $e \in E_i$. Consider the state $\rho(d)[i]$. Since $\bigcirc\beta \in A_d$, we claim that $\rho(d)[i]$ is not a terminal state. We prove the claim by contradiction. Suppose $\rho(d)[i]$ is a terminal state. Since ρ is an accepting run, $inf_i(\rho) \cap G_i \neq \emptyset$ and so it follows that $\rho(d)[i] \in G_i$. But this is a contradiction as $\bigcirc\alpha \in A_d$. Hence $\rho(d)[i]$ is not a terminal state. Now, since ρ is a good run, there exists $e' \in E_i$ such that $e \lessdot e'$. Let $d' = \downarrow e'$. It follows that $\rho(d)[i] \overset{\phi(e')}{\rightarrow} \rho(d')[i]$ in S_0. Therefore, $\bigcirc\beta \in A_d$ implies $\beta \in A_{d'}$. Also, by the choice of e', we know that $d \lessdot d'$. Hence $M, d \models_i \bigcirc\beta$ as required.

($\alpha = \beta\mathbf{U}\gamma$) Suppose $M, d \models_i \beta\mathbf{U}\gamma$. We must show that $\beta\mathbf{U}\gamma \in A_d$. Since $M, d \models_i \beta\mathbf{U}\gamma$, there exists $d' \in LC_i$ such that $d \subseteq d'$, $M, d' \models_i \gamma$ and for all $d'' \in LC_i$:

$d \subseteq d'' \subset d' : M, d'' \models_i \beta$. We show that $\beta \mathbf{U} \gamma \in A_d$ by a second induction on $l = |d'| - |d|$.

Base case: $(l = 0)$.

Then, $d = d'$ and so $M, d \models_i \gamma$. By the main induction hypothesis, $\gamma \in A_d$ and (by the definition of atom), $\beta \mathbf{U} \gamma \in A_d$.

Induction step: $(l > 0)$.

By the semantics of the modality $\mathbf{U}$, $M, d \models_i \beta$ and $M, d_1 \models_i \beta \mathbf{U} \gamma$ for d_1 such that $d \subset d_1 \subseteq d''$. Therefore, by the secondary induction hypothesis, $\beta \mathbf{U} \gamma \in A_{d_1}$. From the definition of $\rightarrow$, we have $\bigcirc(\beta \mathbf{U} \gamma) \in A_d$ (recall that if $\beta \mathbf{U} \gamma \in CL_i$ then $\bigcirc(\beta \mathbf{U} \gamma) \in CL_i$ as well). By the main induction hypothesis, we have $\beta \in A_d$ as well. Combining these facts and using the definition of an atom, we see that $\beta \mathbf{U} \gamma \in A_d$ as required.

Conversely, suppose $\beta \mathbf{U} \gamma \in A_d$. We must show that $M, d \models_i \beta \mathbf{U} \gamma$. Since ρ is an accepting run of S_0, there exists $d' \in LC_i$ such that $d \subseteq d'$ and $\gamma \in A_{d'}$. Once again, we do a second induction on $|d'| - |d|$ to show that $M, d \models_i \beta \mathbf{U} \gamma$.

Base case: $((|d'| - |d|) = 0)$.

Then, $d = d'$ and so $\gamma \in A_d$. Then, by the main induction hypothesis it follows that $M, d \models_i \gamma$ and so $M, d \models_i \beta \mathbf{U} \gamma$.

Induction step: $((|d'| - |d|) > 0)$.

Now, $\gamma \notin A_d$. From the definition of atoms, both β and $\bigcirc(\beta \mathbf{U} \gamma)$ must be in A_d. By the definition of $\rightarrow$, $\beta \mathbf{U} \gamma \in A_{d''}$ where $d'' \in LC_i$ such that $d \lessdot d''$. By the secondary induction hypothesis, $M, d'' \models_i \beta \mathbf{U} \gamma$. Simultaneously, by the main induction hypothesis, $M, d \models_i \beta$. Therefore, by the semantics of the modality $\mathbf{U}$, $M, d \models_i \beta \mathbf{U} \gamma$ as required.

$(\alpha = \oslash_j \beta)$ We first prove the following claim by induction on $|d|$.

Claim: For all i, for all $d \in LC_i$, for all $\oslash_j \beta \in CL_i$,

$\oslash_j \beta \in A_d$ iff $\beta \in A_{d'}$ where $d' \in LC_j$ such that $d' \lessdot d$.

Proof: We prove the claim by induction on $|d|$.

Base case: $|d| = 0$. Then, $d = \epsilon_i$. Here $d' = \epsilon_j$ and since $Init$ is self-contained, $\oslash_j \beta \in A_d$ iff $\beta \in A_{d'}$.

Induction step: $|d| > 0$. Then, $d = \downarrow e$ for some $e \in E_i$.

Case 1: $d' = \epsilon_j$. That is, ϵ_j is the j-maximal event in d. Hence $L_i^j \cap A_d = L_i^j \cap A_{\epsilon_i}$. Hence, $\oslash_j \beta \in A_d$ iff $\oslash_j \beta \in A_{\epsilon_i}$ iff $\beta \in A_{\epsilon_j}$ (by the base case above).

Case 2: $d' = \downarrow e'$ where $e' \in E_j$ such that e' is the j-maximal event in d.

Sub case 2.1: Suppose $e' \lessdot e$.

Since ρ is a run of S_0 on M, it should be the case that $\rho(d')[j] \xrightarrow{\lambda} \rho(d)[i]$ in S_0. Now, by the definition of $\rightarrow$, it follows that $\oslash_j \beta \in A_d$ iff $\beta \in A_{d'}$.

Sub case 2.2: Suppose $e' \not\lessdot e$.

Since d and d' are non-trivial configurations such that $d' \lessdot d$, it should be the case that there exists an event e'' such that $e'' \lessdot e$ and e' is the j-maximal event in $\downarrow e''$. Let $d'' = \downarrow e''$. We have $d' \lessdot d''$.

If $e'' \in E_i$ then, $L_i^j \cap A_d = L_i^j \cap A_{d''}$. Hence, $\oslash_j \beta \in A_d$ iff $\oslash_j \beta \in A_{d''}$ iff (by induction hypothesis, since $|d''| < |d|$) $\beta \in A_{d'}$.

On the other hand, suppose $e'' \in E_l$, for some $l \in [n]$ such that $l \neq i$. Again, since $e'' \lessdot e$, we know that $\rho(d'')[l] \xrightarrow{\lambda} \rho(d)[i]$ in S_0. Now, by the definition of $\rightarrow$, $\oslash_j \beta \in A_d$ iff $\oslash_j \beta \in A_{d''}$ iff (by the induction hypothesis) $\beta \in A_{d'}$.

Hence, induction on $|d|$ is complete and the claim is true.

Now getting back to the proof through the main induction, $M, d \models_i \oslash_j \beta$ iff there exists $d' \in LC_j$ such that $d' \lessdot d$ and $M, d \models_j \beta$ iff by the induction hypothesis $\beta \in A_{d'}$ iff $\oslash_j \beta \in A_d$ (by the claim above).

From this it is easy to see that $M \models \psi_0$ iff $\psi_0 \in (A_{\epsilon_1}, \ldots, A_{\epsilon_n})$. But this follows from the definition of $Init$, the set of initial states of S_0. Hence M is a model for ψ_0.

Conversely, suppose $M \models \psi_0$, where $M = (E, \leq, \phi, 2^P)$. To show that M is a member of $\mathcal{L}^{po}(S_0)$, we have to construct an accepting run of S_0 on M. For any i-local configuration d of M, let $\nu_i(d) \overset{\text{def}}{=} \{\alpha \in CL_i \mid M, d \models_i \alpha\}$. It is easy to see that $\nu_i(d) \in AT_i$. Consider $e_1, e_2 \in E_i$ such that $e_1 \lessdot e_2$. Then, by the semantics of $\bigcirc$ modality, for any $\bigcirc\alpha \in CL_i$, $\bigcirc\alpha \in \nu_i(\downarrow e_1)$ iff $\alpha \in \nu_i(\downarrow e_2)$. Similarly, if $\oslash_j \alpha \in \nu_i(\downarrow e)$, then there exists e' j-maximal in $\downarrow e$ such that $\alpha \in \nu_j(\downarrow e')$.

We now define a map $\rho : C_M^{fin} \rightarrow \widetilde{Q}$, the set of global states of S_0 inductively as follows: $\rho(\emptyset) = ((\nu_1(\emptyset), \emptyset, \emptyset, \emptyset), \ldots, (\nu_n(\emptyset), \emptyset, \emptyset, \emptyset))$.

Now, suppose that $\rho(c) = ((\nu_1(d_1), u_1, I_1, J_1), \ldots, (\nu_n(d_n), u_n, I_n, J_n))$ is defined for some finite configuration c. Consider $c' = c \cup \{e\}$ where $e \in E_i$ such that $e \notin c$. Define $\rho(c') = ((\nu_1(d'_1), v_1, I'_1, J'_1), \ldots, (\nu_n(d'_n), v_n, I'_n, J'_n))$, where for $j \neq i$, $d'_j = d_j$, $v_j = u_j$, $I'_j = I_j$, $J'_j = J_j$ and for all j', $buf(j, j') = buf'(j, j')$. Let $A = \nu_i(d_i)$ and $B = \nu_i(d'_i)$ where $d'_i = \downarrow e$. If $u_i = \emptyset$ then $v_i = \{\alpha \mathbf{U} \beta \in B \mid \beta \notin B\}$; otherwise, $v_i = \{\alpha \mathbf{U} \beta \in u_i \mid \beta \notin B\}$. Also, $J'_i = \{j \mid L^j_i \cap A \neq L^j_i \cap B\}$ and $I'_i = \{j \mid$ there exists $C \in AT_j$ such that $C \rightsquigarrow B\}$.

We now show that ρ is an accepting run of S_0 on M. First to show that $\rho(c) \in Init$, we have to show that $\psi_0 \in (\nu_1(\emptyset), \ldots, \nu_n(\emptyset))$ and that $(\nu_1(\emptyset), \ldots, \nu_n(\emptyset))$ is self-contained. But, this follows from the fact that $\nu_i(\emptyset) = \{\alpha \in CL_i \mid M, \emptyset \models_i \alpha\}$ and the assumption that M is a model of ψ_0. Also, by the definition of $\nu_i(\emptyset)$, $\oslash_j \alpha \in \nu_i(\emptyset)$ for some $j \neq i$, iff $\alpha \in \nu_j(\emptyset)$. Therefore, $(\nu_1(\emptyset), \ldots, \nu_n(\emptyset))$ is self-contained.

Suppose $\rho(c) = (q_1, \ldots, q_n)$ and $\rho(c') = (q'_1, \ldots, q'_n)$ where c and c' are as before. From the definition of ρ, we know that $q_i \xrightarrow{\phi(e)} q'_i$ where $\phi(e) = \nu_i(c) \cap P_i$. Suppose ${}^\bullet e \cap E_j \neq \emptyset$. Let e' be the j-maximal event in $\downarrow e$. By the inductive definition of ρ, $(\nu_j(\downarrow e'), u, I, J)$ is defined already and since e' is j-maximal in$\downarrow e$ we have, $\rho(\downarrow e)[j] \xrightarrow{\lambda} q'_i$ in S_0.

Clearly, if $\bigcirc(\alpha) \in \rho(c)$ for some $\bigcirc(\alpha) \in CL_i$ then, $\rho(c)$ is not a terminal state. Hence ρ is a good run. We can show that $inf_i(\rho) \cap G_i \neq \emptyset$ for all i by using the fact that M is a model. Therefore, ρ defines a run of S_0 on M, and indeed an accepting run at that. Hence $M \in \mathcal{L}^{po}(S)$. $\Box$

From the above lemma, it follows that deciding satisfiability of ψ_0 amounts to checking emptiness of the SCA S_0. From Theorem 3.1.7, it follows that emptiness of the poset language accepted by an SCA can be checked in time $k^{O(n)}$, where k is the maximum of $\{|Q_i| \mid i \in [n]\}$. Now the time bound stated in Theorem 5.2.1 follows by observing that each component in S_0 has a maximum of $2^{O(m)}$ states, where m is the size of ψ.

5.3 Model Checking

The goal of this section is to formulate the model checking problem for m-LTL and show that it is decidable. We again solve this problem using the so-called automata-theoretic approach to model checking. In such a setting, the program is modelled as an SCA S the specification is given by a formula ψ in m-LTL. The model checking problem is to check if the system satisfies the specification i.e, to check if every "behaviour" of S "satisfies" ψ. To do this, we construct the system S_ψ accepting the models of ψ and check if the poset language of S is a subset of the class of models of ψ. But, the system S, in general is given over some arbitrary alphabet $\widetilde{\Sigma}$ and the system associated with ψ runs over 2^P. So, we have to "interpret" the system S as running over 2^P.

To make this precise, we define an *interpreted system* to be a pair $\mathcal{S} = (S, Val)$, where $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ on $\widetilde{\Sigma}$, $Val : Q \rightarrow 2^P$ such that for all $q \in Q_i$, $Val(q) \subseteq P_i$. Consider any Lamport diagram $D = (E, \leq, \phi, \Sigma) \in \mathcal{L}^{po}(S)$. Let ρ be an accepting run of S on D. We define the *associated model* as $M = D'$, where $D' = (E, \leq, \phi', 2^P)$ where $\phi' : E \rightarrow 2^P$ is defined as follows: For $e \in E$, $\phi'(e) = Val(\rho(\downarrow e)[i])$, if $e \in E_i$.

We say that an interpreted system $\mathcal{S} = (S, Val)$ *satisfies* a formula ψ of m-LTL iff $\{D' \mid D \in \mathcal{L}^{po}(S)\} \subseteq Models(\psi)$. We denote this by $S \models \psi$.

Theorem 5.3.1 *Let ψ be an m-LTL formula of length m and S be an interpreted SCA with k being the maximum of $\{Q_i \mid i \in [n]\}$. Then the question $S \models^? \psi$ can be answered in time $k^{O(n)} 2^{O(mn)}$.*

Proof: To check if $S \models \psi$, we have to check if $\mathcal{L}^{po}(S) \subseteq Models(\psi)$. From the proof of Theorem 5.2.1, it follows that $S \models \psi$ iff $\mathcal{L}^{po}(S) \subseteq \mathcal{L}^{po}(S_\psi)$. But then, this is equivalent to checking if $\mathcal{L}^{po}(S) \cap \mathcal{L}^{po}(S_{\neg\psi}) = \emptyset$. We know from Theorem 3.2.2 that the class of poset languages accepted by SCAs are effectively closed under intersection and from Theorem 3.1.7 that the emptiness of the language accepted by the resulting SCA is decidable. Hence the theorem. $\Box$

5.4 System specification example

In this section, we illustrate the use of m-LTL as a specification language by considering as an example, a fragment of the "Conference Management on the Internet" case study from the COORDINA working group [CNT98, MS99]. The example involves a conference management system design developed to conduct a conference where the organisers, the authors and the conference committee members are in geographically different locations. The various components of the system design talk about how interactions between the involved members happen during the process of organising a conference.

We concentrate on the part of the system design which involves the sequence of messages interchanged between the relevant members during the process of reviewing research papers submitted to a conference. Here again, to keep the presentation simple, we consider a restricted version of the system that has only five components—the first two members are two authors (A_1 and A_2), a moderator (M) is the third member and the reviewers (R_1 and R_2) are two more members. We view these members as agents of the system. Both the authors are interested in publishing their (individual) papers and both the reviewers review each submitted paper. The authors submit their papers to the moderator who passes it on to the reviewers. The reviewers pass on the result of the review to the moderator who communicates it to the authors. There is no direct communication between the authors and the reviewers. Papers are considered one at a time for review and a paper is accepted only if both the reviewers choose to accept it.

The system is modelled as an SCA given in Figure 5.1. For simplification, we have just shown the automata corresponding to A_1, M and R_1. The component automata representing the author A_2 and the reviewer R_2 are similar to A_1 and R_1 respectively. The set of actions corresponding to each local component is irrelevant in this context. They are assumed to be a singleton set and omitted here for brevity. The set of good states is empty for each component. The λ-constraints across the system are shown by dotted-directed lines pointing to the respective sending/receiving states. For example, $q_0 \xrightarrow{\lambda} p_1$, $p_1 \xrightarrow{\lambda} s_1$ and so on. The notation $s_3(R_2)$ in the figure refers to the state corresponding to s_3 in the automaton for R_2 (which is not depicted in the figure). The λ-constraints associated with the states p'_i and s'_j (of the agents M and R_1 respectively) are symmetric to those associated with the states p_i and s_j respectively and hence the corresponding lines are not labelled.

A_1 submits at a state q_0 (λ-constraint to state p_1 of M). When the paper submitted by A_1 is accepted (λ-constraint from state p_7 of M), it enters the local state q_2 (similarly, reject is state q_3). M has two symmetric sub-components, one for A_1 and another for A_2. When it receives a submission from A_1 (local state p_1), it sends it to R_1 and R_2 and waits (in local state p_2) for one of the four possible outcomes, one of which results in the corresponding paper being accepted (local state p_3); the results are accordingly communicated in states p_7 and p_8. The automaton

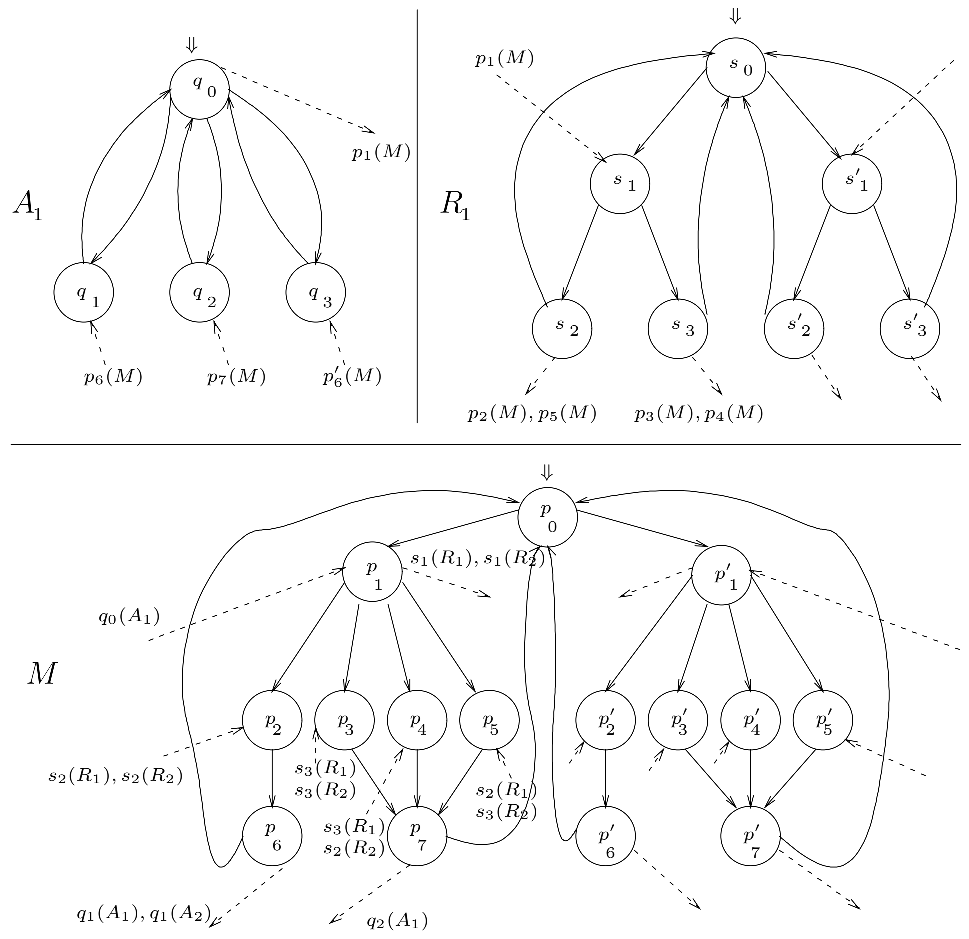


Figure 5.1: SCA representing a paper review system

for R_1 is simple; it gets a submission, makes a binary choice (accept or reject) and reverts to waiting.

Various formulas specifying properties of the system can be given in m-LTL. We begin by giving the set of propositions associated with each component:

- The set of propositions associated with the component A_1 is the set $P_1 = \{wait_{A_1}, result_{A_1} = nil/acc/rej\}$.
- The set of propositions associated with the component A_2 is the set $P_2 = \{wait_{A_2}, result_{A_2} = nil/acc/rej\}$.
- The set of propositions associated with the component M is the set $P_3 = \{ready_M\} \cup \{proc_{A_i} \mid i = 1,2\} \cup \{FB_i = x \mid x \in Dec, i = 1,2\} \cup \{R = x \mid x \in Dec\}$ where $Dec = \{acc, rej, nil\}$.
- The set of propositions associated with the component R_1 is the set $P_4 = \{ready_{R_1}, review_1(i), decided_1(i), OK_1(i) \mid i = 1,2\}$.
- The set of propositions associated with the component R_2 is the set $P_5 = \{ready_{R_2}, review_2(i), decided_2(i), OK_2(i) \mid i = 1,2\}$.

In the description above, the intended meaning of each proposition is implied by its name. For example, $result_{A_1}$ refers to the result of a submission by A_1. The variable FB_i stands for the feedback from the reviewer R_i, for $i = 1,2$. The feedback can take any of the three values from the set $\{acc, rej, nil\}$ depending on the decision of the reviewer. Similarly, the result of the review is represented by the variable R. The value nil of these variables can be thought of a shorthand notation to denote the fact that the reviews are pending.

To describe the system completely, we need to specify the set of propositions that hold in each of the local states. They are given in the table below. For simplicity, we just mention the set of propositions that are true in each state. The valuation for the primed states can be obtained from the corresponding mappings for unprimed ones. For instance, the set of propositions true in the state s'_1 is $\{review_1(2)\}$.

A_1:q_0	$\{result_{A_1} = nil\}$	R_1:s_0	$\{ready_{R_1}\}$
q_1	$\{wait_{A_1}, result_{A_1} = nil\}$	s_1	$\{review_1(1)\}$
q_2	$\{result_{A_1} = acc\}$	s_2	$\{decided_1(1), OK_1(1)\}$

q_3 $\{result_{A_1} = rej\}$	s_3 $\{decided_1(1), \neg OK_1(1)\}$

$M{:}\,p_0$ $\{ready_M, FB_1 = FB_2 = R = nil\}$

p_1 $\{FB_1 = FB_2 = R = nil\}$

p_2 $\{proc_{A_1}, FB_1 = FB_2 = R = nil\}$

p_3 $\{proc_{A_1}, FB_1 = FB_2 = acc, R = nil\}$

p_4 $\{proc_{A_1}, FB_1 = FB_2 = rej, R = nil\}$

p_5 $\{proc_{A_1}, FB_1 = rej, FB_2 = acc, R = nil\}$

p_6 $\{proc_{A_1}, FB_1 = acc, FB_2 = rej, R = nil\}$

p_7 $\{proc_{A_1}, FB_1 = FB_2 = nil, R = acc\}$

p_8 $\{proc_{A_1}, FB_1 = FB_2 = nil, R = rej\}$

We are now ready to specify local properties of this system using formulas from m-LTL. The following are some examples of such properties.

1. Moderator reacts to a publication request from A_1 by accepting or rejecting the submitted paper (local formula of agent 3).

   $\Box((FB_i = nil \wedge R = nil) \wedge \oslash_1(result_{A_1} = nil \wedge \neg wait_{A_1})) \supset ((proc_{A_1} \wedge \neg ready_M)\mathbf{U}(R = acc \vee R = rej))$

2. A paper submitted by A_1 is accepted by the moderator for publication only if both the reviewers choose accept the paper (local formula of agent 3).

   $\Box((proc_{A_1} \wedge (R = acc)) \supset (\oslash_4(decided_1(1) \wedge OK_1(1)) \wedge \oslash_5(decided_2(1) \wedge OK_2(1))))$

3. A paper is reviewed by R_1 only if there was a submission (to the moderator) from the corresponding author (local formula of agent 4).

   $\Box(review_1(1) \supset \oslash_1(\neg wait_{A_1} \wedge result_{A_1} = nil))$

Notice that R_1 can assert the last formula in the list above despite having no direct correspondence with A_1. This is because R_1 can assert $\oslash_3\oslash_1(\neg wait_{A_1} \wedge result_{A_1} = nil)$ above, and by the semantics of the $\oslash_j$ modality, the use of $\oslash_1$ suffices in the above formula.

We now intuitively argue that the system S given in Figure 5.1 satisfies these properties. We present such an argument for the first formula (of agent 3) in the list above. Consider any Lamport diagram D accepted by the interpreted system of S. In order to show that D satisfies the formula (1) above, we have to show that from every state of D in which the moderator is processing a request from A_1 (local state of M is p_1 and local state of A_1 is q_0), we eventually reach a state in which the result is known (local state of M is p_7 or p_8). It is easy to see that this holds: when the local state of M is p_1, the components R_1 and R_2 can start reviewing (local states $s_1(R_1)$ and $s_1(R_2)$) the submitted paper and upon deciding, they inform M (λ-constraints to states p_j $(3 \leq j \leq 6)$ of M). Depending on the decision of the reviewers, M enters one of the states p_j $(3 \leq j \leq 6)$ after which the decision on acceptance/rejection of the paper is taken (local states p_7 or p_8 of M) accordingly.

Chapter 6

Logics over layered Lamport diagrams

Towards the end of Chapter 4 we had mentioned two approaches to get decidable logics over Lamport diagrams. The previous chapter reflected the first approach—to restrict the expressive power of the $\mathbf{X}$ and $\mathbf{Y}$ modalities. We consider the second approach in this chapter—to restrict the class of models (Lamport diagrams) considered and look at possible ways of obtaining decidable logics over these restricted Lamport diagrams.

From Chapter 4 we know that undecidability results hold for models based on *finite* Lamport diagrams as well. But, these finite diagrams are *unbounded* in general and hence we could place bounds on the size of diagrams and look for decidability. The satisfiability of LD_0 becomes decidable over bounded diagrams as there are only finitely many different diagrams with bounded number of events. Given a formula, we just check if it is true against each of the Lamport diagrams one by one to see if it is satisfiable in one of them. We use this fact and consider logics over layered Lamport diagrams as these have layers which are finite Lamport diagrams. We define a temporal logic with two types of formulas over layered Lamport diagrams as follows: *Layer formulas* which are defined based on the logic LD_0 are used to describe properties of layers and temporal formulas built from these using the usual linear time connectives talk about the sequence of layers that make up an LLD. Specifications written in such a logic describe a sequential composition of parallel processes reflecting the structure of LLDs.

We show that the satisfiability problem of this logic is again undecidable even when we restrict the size of layers to be uniformly bounded. However, the logic turns out to be decidable over the class of communication closed and bounded diagrams and also over the class of channel bounded diagrams. We again illustrate the usefulness of such a temporal logic as a natural specification language of systems that are described using sequential composition of parallel components by using examples.

6.1 Logic LD_0 over bounded models

In this section, we consider the logic LD_0 defined in Chapter 4 over models based on bounded Lamport diagrams and show that satisfiability becomes decidable.

To recall the logic LD_0 from Chapter 4, it is a standard modal logic over Lamport diagrams. Formulas are built from propositions (which include special propositions τ_i, $i \in [n]$) using boolean combinations and four modalities ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{F}$ and $\mathbf{P}$). Models are Lamport diagrams equipped with a valuation function and all the four modalities are global.

A bounded Lamport diagram is one which has only boundedly many events. Fix $b \in \mathbb{N}$. A Lamport diagram $D = (E, \leq, \phi)$ is said to be b-**bounded** if $|E| \leq b$.

Theorem 6.1.1 *The satisfiability of a formula of length m in LD_0 over b-bounded diagrams is decidable in time $m \cdot 2^{O(b^2)}$.*

Proof: The set of b-bounded Lamport diagrams is a finite set, call it $\mathcal{L}_b$. Given a formula $\alpha \in LD_0$, the problem is to check if there exists a model for α among the diagrams in $\mathcal{L}_b$.

Consider a model $M = (D, V_E)$ where $D \in \mathcal{L}_b$ and a formula $\alpha \in LD_0$. To check whether $M, e_{min} \models \alpha$ where e_{min} is a minimal event in M, we consider the following simple labelling algorithm which labels each event with a set of sub-formulas of α such that it terminates, and on termination, for every sub-formula β and every event e in M, we have $M, e \models \beta$ iff $\beta \in \mathsf{label}(e)$.

The set $SF(\alpha)$ of sub-formulas of α is defined to be the least set of formulas such that the following conditions hold:

- $\alpha \in SF(\alpha)$.

- $\neg\beta \in SF(\alpha)$ implies $\beta \in SF(\alpha)$.
- $\beta \vee \gamma \in SF(\alpha)$ implies $\beta, \gamma \in SF(\alpha)$.
- $\mathbf{X}\beta \in SF(\alpha)$ implies $\beta \in SF(\alpha)$.
- $\mathbf{Y}\beta \in SF(\alpha)$ implies $\beta \in SF(\alpha)$.
- $\mathbf{F}\beta \in SF(\alpha)$ implies $\beta \in SF(\alpha)$.
- $\mathbf{P}\beta \in SF(\alpha)$ implies $\beta \in SF(\alpha)$.

Let $\{\alpha_1, \dots \alpha_m\}$ be the sub-formulas of α. With each event e of M, we define the label of e by associating an array of length $m = |SF(\alpha)|$ such that the entry corresponding to index i in the array is 1 if $\alpha_i \in \mathsf{label}(e)$ and is 0 otherwise. Also, if α_i is a sub-formula of α_j then, $i \leq j$.

We can define such a labelling by induction on i. For the base case, we have $\alpha_i = p$ for some $p \in P$ and we can fill up the corresponding entry of the array by looking at the valuation function associated with M. For $i > 1$, if $\alpha_i = \neg\alpha_j$, then, $j < i$. We just look at the entry corresponding to j and flip it to obtain the entry for i. Similarly, for $\alpha_i = \alpha_j \vee \alpha_k$, we take the maximum of the entries corresponding to j and k. For $\alpha_i = \mathbf{X}\alpha_j$, we look at the entries corresponding to α_j for all the immediate successors of e in M and if one of them has entry 1, we put the same in the entry for i; otherwise, we put 0. The entries corresponding to $\alpha_i = \mathbf{Y}\alpha_j$ are filled up similarly. For $\alpha_i = \mathbf{F}\alpha_j$, we do a depth first search starting from e and if one of the events reachable from e has entry 1 corresponding to α_j, we fill up the $\mathbf{F}\alpha_j$ entry of e with 1. If none of the events reachable from e have entry 1 corresponding to α_j, we put 0 in the $\mathbf{F}\alpha_j$ entry of e. The case when $\alpha_i = \mathbf{P}\alpha_j$ is handled similarly.

It is easy to see that this labelling algorithm runs in time $O(k^2 \cdot m)$, where the size of M (that is the number of events in M) is k , and the length of α is m.

Thus, for checking satisfiability of layer formulas against the finite set $\mathcal{L}_b$ of Lamport diagrams, we can consider the diagrams in $\mathcal{L}_b$ one at a time and check if α holds at a minimal event. From the argument above, given a model M of size at most b, we know that the time taken to check the satisfiability of a formula α of length m against M is less than or equal to $O(m.b^2)$. $\mathcal{L}_b$ consists of all Lamport diagrams of size $\leq b$, and the number of Lamport diagrams of size k is at most 2^{k^2}. Therefore,

$|\mathcal{L}_b| \leq 2^{n^2} + 2^{(n+1)^2} + \ldots + 2^{b^2}$ (since $b \geq n$) $\leq b \cdot 2^{b^2}$. Hence the total time taken to check the satisfiability of a formula of α against all the models in $\mathcal{L}_b$ is $\leq O(m \cdot 2^{b^2})$.

□

Since the logics LD_1 and LD_2 are translatable into LD_0, the corresponding theorem for them results as a corollary of the theorem above.

6.2 A temporal logic over layered Lamport diagrams

In the previous section, we saw that the logic LD_0 was decidable over the set of bounded Lamport diagrams. This motivates us to consider extensions of LD_0 over Layered Lamport Diagrams (LLDs) as LLDs are obtained by concatenating finite or bounded diagrams which occur as their layers. We define a logic, which we call λ-**LTL** with a two-level syntax. We have *layer formulas* which come from LD_0 and *temporal formulas* which are built from the layer formulas using standard linear time connectives. As we know, LLDs describe behaviours of systems which work with a fixed finite set of communication patterns between agents where each such pattern would be given by a finite diagram (layer) and the system would be described by choosing patterns dynamically and composing them. The layer and temporal formulas of λ-LTL also follow this structuring. The layer formulas describe properties of a layer and temporal formulas describe the properties of the sequence of layers which make up the LLD.

We fix a finite set of *layer names*, Γ.

The syntax of layer formulas is given as in LD_0.

$$\Phi_l ::= p \ \in \ P \mid \tau_i \mid \neg\, \alpha \mid \alpha_1 \ \vee \ \alpha_2 \mid \mathbf{X}_j\, \alpha \mid \mathbf{Y}_j\, \alpha \mid \mathbf{F}\, \alpha \mid \mathbf{P}\, \alpha$$

where $i, j \in [n]$. Note that the $\mathbf{X}$ and $\mathbf{Y}$ modalities are global indexed modalities. The formula $\mathbf{X}_j\alpha$ interpreted at an event e of agent i asserts that there exists an immediate successor of e in agent j at which α holds. Similarly, the formula $\mathbf{Y}_j\alpha$ interpreted at an event e of agent i asserts that there exists an immediate predecessor of e in agent j at which α holds.

Temporal formulas are given by the following syntax:

$$\Psi ::= \alpha @ i,\ \alpha \in \Phi_l\ ,\ i \in [n] \mid a,\ a \in \Gamma \mid \neg\,\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\,\varphi \mid \varphi_1\, \mathbf{U}\, \varphi_2$$

This is the same as the standard propositional temporal logic of linear time, but built up from layer formulas and layer names. The propositional connectives ($\wedge, \supset, \equiv$) and the derived temporal modalities ($\Box, \Diamond$) are defined as usual.

The formulas are interpreted on *countable* layered Lamport diagrams. Formally, *models* are layered Lamport diagrams equipped with two valuation functions: one maps the events to a subset of propositions and the other maps a particular layer to some layer name from Γ. We will denote models by $M = (D, V_E, V_\lambda)$ where $D = (E, \leq, \phi, \lambda)$ is a countable layered Lamport diagram, $V_E : E \to 2^P$ is such that for all $e \in E$ and $i \in [n]$, $\tau_i \in V_E(e)$ iff $\phi(e) = i$ and $V_\lambda : \lambda(E) \to \Gamma$. V_E and V_λ are valuation functions which define the set of propositions true at each event and identify a layer name with each layer respectively,

We will first define the semantics of layer formulas, over the layers of a given model. For this, it matters crucially whether the layer is communication closed or not, since the $\mathbf{X}_j$ and $\mathbf{Y}_j$ modalities are constrained to be satisfied within the layer or not, appropriately. For technical convenience, we always interpret $\mathbf{F}$ and $\mathbf{P}$ modalities within layers, and locally for every agent. As we will see later, many interesting specifications of message behaviours can be written even with this restriction. Also, it turns out that the classes of LLDs over which this logic is decidable have layers which are bounded (apart from other properties) and hence $\mathbf{F}$ and $\mathbf{P}$ modalities interpreted within layers can be written using a "sequence" of local $\mathbf{X}_j$ and $\mathbf{Y}_j$ modalities respectively.

Let $\alpha \in \Phi_l$ and $e \in E$. We define two notions below: the first notion that α holds at e in M is denoted $M, e \models_l \alpha$ and is defined inductively as done before. The base case for propositions and the case when α is a boolean combination of formulas are as done for the logic LD_0. The semantics of the various modalities are given below. We first consider LLDs where the layers need not be communication closed and define the semantics of the various modalities.

- $M, e \models_l \mathbf{X}_j \alpha$ iff there exists $e' \in E_j$ such that $e \lessdot e'$ and $M, e' \models_l \alpha$.
- $M, e \models_l \mathbf{F} \alpha$ iff there exists e' such that $e \leq e'$, $\phi(e) = \phi(e')$, $\lambda(e) = \lambda(e')$ and $M, e' \models_l \alpha$.

- $M, e \models_l \mathbf{Y}_j\alpha$ iff there exists $e' \in E_j$ such that $e' \lessdot e$ and $M, e' \models_l \alpha$.
- $M, e \models_l \mathbf{P}\alpha$ iff there exists e' such that $e' \leq e$, $\phi(e) = \phi(e')$, $\lambda(e) = \lambda(e')$ and $M, e' \models_l \alpha$.

The second notion is defined on models with communication closed layers. Let M be a model based on a communication closed LLD. The notion that α holds at e in M is denoted $M, e \models_{cl} \alpha$ and is defined inductively as above. The only changes are:

- $M, e \models_{cl} \mathbf{X}_j\alpha$ iff there exists $e' \in E_j$ such that $e \lessdot e'$, $\lambda(e) = \lambda(e')$ and $M, e' \models_{cl} \alpha$.
- $M, e \models_{cl} \mathbf{Y}_j\alpha$ iff there exists $e' \in E_j$ such that $e' \lessdot e$, $\lambda(e) = \lambda(e')$ and $M, e' \models_{cl} \alpha$.

Temporal formulas are interpreted at layers of a layered Lamport diagram. Given a model $M = (D, V_E, V_\lambda)$ and $\varphi \in \Psi$, the notion that φ holds in the layer k of D is denoted $M, k \models \varphi$ and is defined inductively as follows:

- $M, k \models \alpha@i$ iff $M, e \models_l \alpha$ where e is the i-minimum event of the layer D_k.
- $M, k \models a,\ a \in \Gamma$ iff $V_\lambda(k) = a$.
- $M, k \models \neg\varphi$ iff $M, k \not\models \varphi$.
- $M, k \models \varphi \vee \psi$ iff $M, k \models \varphi$ or $M, k \models \psi$.
- $M, k \models \bigcirc\varphi$ iff $M, k' \models \varphi$ where k' is the successor of k in ν_D (where ν_D denotes the sequence of indices representing layers of M as in Chapter 2).
- $M, k \models \varphi\mathbf{U}\psi$ iff there exists $k' \in l(E)$: $k \leq_{\mathbb{N}} k', M, k' \models \psi$ and for all $k'' \in l(E) : k \leq_{\mathbb{N}} k'' <_{\mathbb{N}} k' : M, k'' \models \varphi$.

When M is a model based on a communication closed LLD, we define $M, k \models_c \varphi$ exactly as above, except for the base case: $M, k \models_c \alpha@i$ iff $M, e \models_{cl} \alpha$ where e is the i-minimum event of D_k.

For a model M and $\varphi \in \Psi$, we say that $M \models \varphi$ iff $M, k_{min} \models \varphi$ where k_{min} denotes the first element in the sequence ν_D. The notion $M \models_c \varphi$ is defined similarly for a model based on a communication closed LLD. φ is *satisfiable* iff there exists a model M such that $M \models \varphi$. φ is *C-satisfiable* iff there exists a model M based on a communication closed LLD such that $M \models \varphi$.

We will be interested in other restricted satisfiability notions as well. We say φ is *B-satisfiable* if there exists a model M based on a bounded LLD such that $M \models \varphi$. The notion of S_b-satisfiability is defined similarly using a model based on a channel b-bounded LLD, and C_b-satisfiability using a model based on an LLD that is both communication closed and b-bounded. The various notions of validity are also defined similarly.

6.3 System specification example

Before proceeding with the proofs of the various undecidability and decidability results, we present an example of system specification using λ-LTL in this section. We consider the distributed system corresponding to an Automatic Teller Machine (ATM). At the level of abstraction we consider, the system has three agents—**User**, **Bank** and **ATM**. The ATM provides options for the user to check the balance in his/her account and to withdraw cash after validating the balance. The interactions between the agents can be naturally described using Lamport diagrams. For example, Lamport diagrams representing the possible computations for withdrawing cash are depicted in Figure 6.1. As depicted in the diagrams, we again focus on the sequence of messages exchanged between the components and not on the local actions of a particular agent. The propositions that are true at events of various agents are specified as message labels in the figure. For example, the proposition Withdraw holds at the initial event of the agent **User** (which is a send event) and at the initial event of the agent **ATM** (the corresponding receive event).

Figure 6.1 also shows the layering of the Lamport diagram representing the computations. The initial layer contains the scenario corresponding to validating the bank balance and is repeated in both the Lamport diagrams. These can be combined and represented in a concise way. The resulting system can actually be presented as a diagram automaton and is depicted in Figure 6.2. The automaton represents the ATM system and accepts the set of computations of this system which are represented using communication closed and bounded LLDs. Note that these LLDs are obtained by concatenating the individual layers (finite Lamport diagrams)

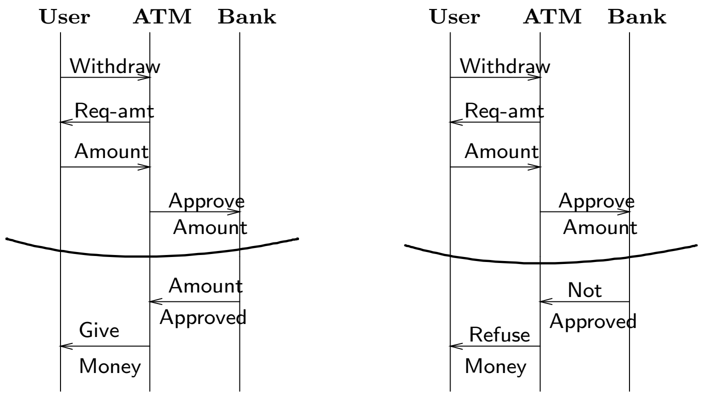


Figure 6.1: Lamport diagrams representing computations of an ATM

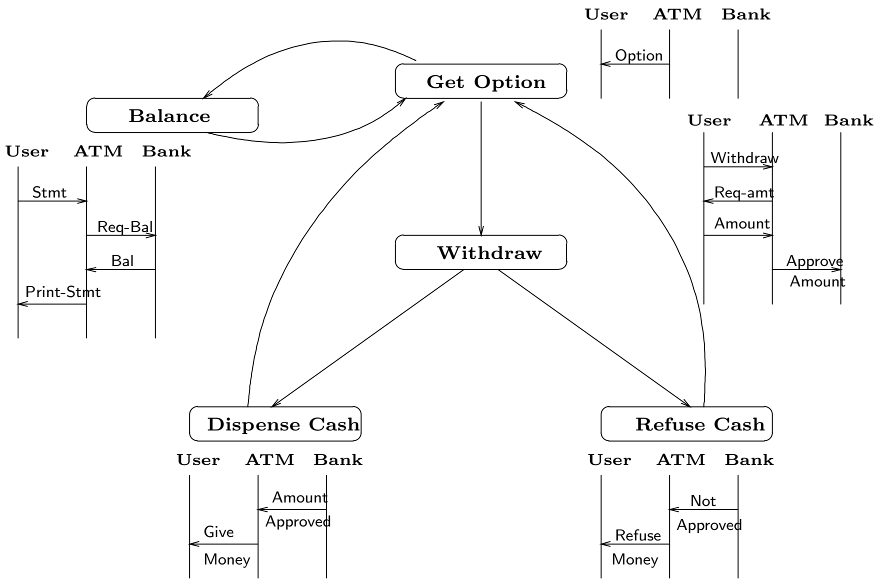


Figure 6.2: A diagram automaton representing an ATM

that occur while tracing paths in the underlying automaton.

The logic λ-LTL can be used to specify various properties of the above system. The agents are refered to by their names, which are **User**, **ATM** and **Bank**. The set of propositions is given by { Option, Withdraw, Req-Amt, Amount, Approve-Amount, Amount-Approved, Give-Money, Not-Approved, Refuse-Money, Stmt, Req-Bal, Bal, Print-Stmt }. As mentioned before, the propositions that are true at various events of the system are presented as labels of messages (that the events constitute) in the figure for the sake of clarity.

The following properties can be specified using formulas from λ-LTL.

1. ATM accepts options to either check balance or to withdraw cash.

   $\Box(((\mathsf{Option} \wedge \tau_{\mathbf{ATM}} @\mathbf{ATM}) \supset$
   $\bigcirc((\tau_{\mathbf{User}} \wedge \mathsf{stmt}@\mathbf{User} \vee (\tau_{\mathbf{User}} \wedge \mathsf{Withdraw})@\mathbf{User}))))$

2. Every request for withdrawal from the user is approved or rejected by the ATM depending on whether the bank approves or rejects the amount.

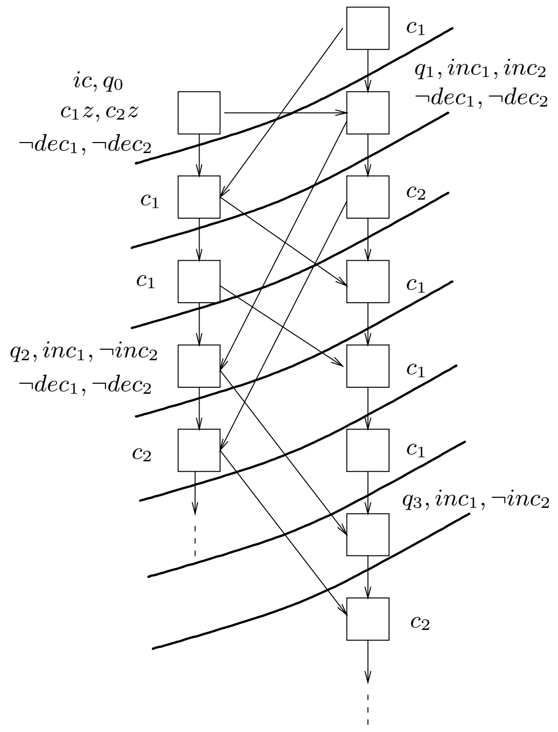
   $\Box((\tau_{\mathbf{User}} \wedge \mathsf{Withdraw} \wedge \mathbf{X}_{\mathbf{ATM}}(\tau_{\mathbf{ATM}} \wedge \mathsf{Withdraw})@\mathbf{User}) \supset$
   $\bigcirc((\tau_{\mathbf{Bank}} \wedge \mathsf{Amount-Approved} \wedge$
   $\mathbf{X}_{\mathbf{ATM}}(\tau_{\mathbf{ATM}} \wedge \mathsf{Amount-Approved} \wedge \mathbf{X}_{\mathbf{ATM}}\mathsf{Give-Money})@\mathbf{Bank} \vee$
   $\tau_{\mathbf{Bank}} \wedge \mathsf{Not-Approved} \wedge$
   $\mathbf{X}_{\mathbf{ATM}}(\tau_{\mathbf{ATM}} \wedge \mathsf{Not-Approved} \wedge \mathbf{X}_{\mathbf{ATM}}\mathsf{Refuse-Money})@\mathbf{Bank})))$

6.4 Undecidable problems

We now show that B-satisfiability and C-satisfiability are undecidable. In case of B-satisfiability we obtain undecidability (in spite of the models having layers with a uniform bound) due to the fact that in a bounded LLD a particular message can get delivered after unboundedly many layers. Also, there the number of pending send events can grow unboundedly as we go down the LLD. Consequently, the number of pending $\mathbf{X}$ requirements which have to be checked also grows unboundedly.

C-satisfiability becomes undecidable mainly because of the fact that the layers of a communication closed LLD can have unboundedly many events. Consequently, even though $\mathbf{X}$ and $\mathbf{Y}$ modalities are interpreted within a layer, they could be matched after an unbounded number of events.

Figure 6.3: LLD representing a run of M



Theorem 6.4.1 *B-satisfiability is undecidable.*

Proof: The result is proved in the same way as we did in Chapter 4. We again consider non-deterministic 2-counter machines and use the fact that the problem of checking if a final configuration is reachable or not is undecidable. Given a non-deterministic 2-counter machine, we construct a 2-agent Lamport diagram which represents a run of the machine. Of course, the diagram constructed there was a finite diagram, whereas we need countable diagrams as models here. This is easily achieved by appending an infinite sequence of dummy local events to each process. In addition, we have to define a layering which is bounded.

To show that B-satisfiability is undecidable, consider the layering which picks exactly one event from each agent for a layer. Clearly this is a bounded layering and it is not communication closed. Also, such a layering is not always channel bounded. For example, consider the Lamport diagram representing a run of the 2-counter machine M (as defined in Chapter 4) where counter 1 is incremented at every transition. This implies that a new event labelled by c_1 is added immediately before every event labelling the resulting state of a transition. For example, a prefix of the LLD corresponding to the run $\rho = (q_0, 0, 0) \rightarrow (q_1, 1, 1) \rightarrow (q_2, 2, 1) \rightarrow (q_3, 3, 1) \ldots$ is depicted in Figure 6.3. Consider the prefixes of this LLD ending at events labelled by states q_0 and q_2. There is only one pending send (from agent 1 to agent 2) at the prefix ending at q_0-labelled event whereas there are two pending sends (from agent 1 to agent 2) at the prefix ending at q_2-labelled event. Using the fact that we consider layers having exactly one event per agent, we can argue that the number of pending sends from agent 1 to agent 2 will grow unboundedly as we consider successive layers ending at state-labelled events. That is, such an LLD is not channel b-bounded for any $b \in \mathbb{N}$.

We can now show that such a layering as well as the diagram itself can be described by a formula φ_M of the logic. The various formulas comprising φ_M are basically the same as the one used in the proof of Theorem 4.2.1. We replace the $\mathbf{G}$ and $\mathbf{F}$ modalities by $\Box$ and $\Diamond$ appropriately. In addition, we add two formulas to describe the layering and to add dummy events at the end.

To start with, we define the sets P_1 and P_2 of propositions of agent 1 and 2 as done in the proof of Theorem 4.2.1. In addition, we add a proposition d to represent dummy events and define the set of layer names $\Gamma = \{a\}$.

The formula φ_M is given by $\varphi_M \overset{\text{def}}{=} \varphi_b \wedge \varphi_d \wedge \varphi_{\mathsf{init}} \wedge \varphi_{\mathsf{fin}} \wedge \varphi_{\mathsf{inv}}$ and these formulas are defined below.

- The formula $\varphi_b \overset{\text{def}}{=} \Box(a \equiv (\tau_1@1 \wedge \tau_2@2))$ ensures that every layer includes exactly one event of each agent.

- The formula $\varphi_d \overset{\text{def}}{=} \Box(\vee_{i=1,2}(q_f@i) \supset (c_2@i\mathbf{U}d@i)) \wedge \bigcirc\Box(((d \wedge \tau_i)@i) \wedge (d \wedge \tau_{i^{co}})@i^{co})$ appends dummy events at the end of the finite diagram representing a run of M.

- The formula $\varphi_{\mathsf{init}} \overset{\text{def}}{=} \mathsf{init}@1$ where init is as defined in the proof of Theorem 4.2.1 specifies the initial configuration.

- The formula $\varphi_{\mathsf{fin}} \overset{\text{def}}{=} \Diamond(q_f@1 \vee q_f@2)$ asserts that a final configuration is reachable.

- The formula $\varphi_{\mathsf{inv}} \overset{\text{def}}{=} \Box(\wedge_{i=1,2}(\tau_i \supset \mathsf{inv}_i)@i)$ where inv_i are defined below.

  $\mathsf{inv}_1 \overset{\text{def}}{=} (\mathsf{state} \vee \mathsf{ctr}_1 \vee \mathsf{ctr}_2 \vee \mathsf{dummy}) \wedge \mathsf{trans}_1 \wedge \mathsf{consis}_1$ and $\mathsf{inv}_2 \overset{\text{def}}{=} (\mathsf{state} \vee \mathsf{ctr}_1 \vee \mathsf{ctr}_2 \vee \mathsf{dummy}) \wedge \mathsf{trans}_2 \wedge \mathsf{consis}_2$ where trans_i and consis_i are as defined in the proof of Theorem 4.2.1 and the other sub-formulas are defined below:

– $\mathsf{state} \stackrel{\text{def}}{=} \bigvee_{q \in Q} (q \wedge \bigwedge_{q' \in Q, q \neq q'} \neg q') \wedge \neg c_1 \wedge \neg c_2 \wedge \neg d.$

– $\mathsf{ctr}_1 \stackrel{\text{def}}{=} c_1 \wedge (\bigwedge_{q \in Q} \neg q) \wedge \neg c_2 \wedge \neg c_1 z \wedge \neg d.$

– $\mathsf{ctr}_2 \stackrel{\text{def}}{=} c_2 \wedge (\bigwedge_{q \in Q} \neg q) \wedge \neg c_1 \wedge \neg c_2 z \wedge \neg d.$

– $\mathsf{dummy} \stackrel{\text{def}}{=} d \wedge (\bigwedge_{q \in Q} \neg q) \wedge \neg c_1 \wedge \neg c_2 \wedge \neg c_1 z \wedge \neg c_2 z.$

– The formulas trans_i and consis_i are the same as the one used in the proof of Theorem 4.2.1 where we replace the special proposition r_i^j by $\tau_i \wedge \mathbf{Y}_j \mathit{True}$ and $\neg r_i^j$ by $\tau_i \wedge \overline{\mathbf{Y}}_j \mathit{False}$.

We show that φ_M is 2-satisfiable iff a final configuration of M is reachable, thus establishing the undecidability of B-satisfiability. Again, the proof is very similar to that of Theorem 4.2.1.

Suppose there exists a run ρ of M in which a final configuration is reachable. We will show that φ_M is satisfiable by inductively defining a model M' of φ_M. Let $\rho = (q^0, n_1^0, n_2^0) \to \ldots (q^k, n_1^k, n_2^k) \to \ldots (q^m, n_1^m, n_2^m)$ where $q^0 = q_0$, $n_1^0 = n_2^0 = 0$ and $q^m = q_f$. We define the Lamport diagram D_M and valuation function V_M exactly as done in the proof of Theorem 4.2.1. Now define $M' = (D, V)$ where $D = (E, \leq, \phi, \lambda)$ is given by

- $E = E_M \cup \{e_i^1, e_i^2 \mid i \in \mathbb{N}\}$ where $e_i^j \notin E_M$ for all $i \in \mathbb{N}$ and $j = 1, 2$.
- $\phi(e) = \phi_M(e)$ for $e \in E_k$ and $\phi(e_i^j) = j$ for all $i \in \mathbb{N}$.
- $\leq = (\leq_M \cup \{(e_i^1, e_{i+1}^1) \mid i \in \mathbb{N}\} \cup \{(e_i^2, e_{i+1}^2) \mid i \in \mathbb{N}\})^*$.
- It is easy to see that $\leq$ is a total order on the events of agent j of E. Let $e_1, e_2, \ldots$ be an enumeration of the events of agent 1 (with respect to $\leq_1$) and let $f_1, f_2, \ldots$ be an enumeration of events of agent 2 (with respect to $\leq_2$). Define $\lambda(e_i) = \lambda(f_i) = i$ for all i.

and $V(e) = V_M(e)$ for all $e \in E_M$ and $V(e_i^j) = \{\tau_j, d\}$ for all $i \in \mathbb{N}$ and $j = 1, 2$. It is easy to see that D is a layered Lamport diagram. The rest of the proof goes along the lines of the proof of Theorem 4.2.1. $\square$

Note: Since we get 2-agent Lamport diagrams in the coding and the layering is such that it includes exactly one event per agent, the above result actually shows that the problem of checking if a given formula has a model based on an n-bounded LLD (where n is the number of agents) is undecidable.

Theorem 6.4.2 *C-satisfiability is undecidable.*

Proof: The result is again proved using non-deterministic 2-counter machines. Dummy local events are appended to the finite diagram which represents a run of M that ends in a final configuration. Now consider the part of the diagram representing a run of M which ends in a final configuration as the first layer, and each subsequent layer as containing exactly one event (which is a dummy local event, without any communications) per agent. This is a communication closed (but unbounded) layering, and this can be specified using a formula φ_M as well. Hence C-satisfiability is also undecidable. The details are sketched below.

The set of propositions are as defined for the undecidability of B-satisfiability. We use the formula α_M as defined for the proof of undecidability of LD_1. In addition, we need to define a layering with the finite Lamport diagram representing a run of M ending in a final configuration as the first layer and each subsequent layer as containing exactly one event per agent. The following formulas capture this layering.

- The formula $\alpha_M@1$ where α_M is the formula defined in the proof of Theorem 4.2.1 specifies that the first layer is the diagram representing a run of M ending in a final configuration.

- The formulas $\Box(\bigcirc(\bigwedge_{p_1 \in P_1 \setminus \{d,\tau_1\}} (\neg p_1 \wedge d \wedge \tau_1)@1))$ and $\Box(\bigcirc(\bigwedge_{p_2 \in P_2 \setminus \{d,\tau_2\}} (\neg p_2 \wedge d \wedge \tau_2)@2))$ specify that the events of agent i from the second layer onwards are labelled only by the proposition d.

- The formula $\Box(\bigcirc(\bigwedge_i (\tau_i \wedge \overline{\mathbf{X}}_i \neg \mathit{False})@i))$ ensures that every layer (from the second layer onwards) includes exactly one event of each agent.

Let φ_M denote the conjunction of all the above formulas. If M has a halting run, then we inductively define a model $M' = (D, V)$ where $D = (E, \leq, \phi, \lambda)$ and E, $\leq$, ϕ and V are as defined in the proof of Theorem 6.4.1 and λ is given by $\lambda(e) = 1$ for all $e \in E_M$ and $\lambda(e_i^j) = i + 1$ for all $i \in \mathbb{N}$. The rest of the proof is along the lines of that of Theorem 4.2.1. □

6.5 C_b-Satisfiability

Given that both B-satisfiability and C-satisfiability became undecidable, we now turn our attention to further restricting the class of models considered. The first such restriction we look at is C_b-satisfiability where the models are based on LLDs which are both communication closed and b-bounded. In this section we show that C_b-satisfiability is decidable. The proof is again using the automata-theoretic approach to satisfiability. Diagram automata were introduced in Chapter 3 and accept communication closed and bounded LLDs. We show that C_b-satisfiability is decidable by associating a diagram automaton with every λ-LTL formula. The automaton accepts precisely the communication closed and bounded LLDs which are models of the formula.

Theorem 6.5.1 *Given a temporal formula φ_0 of length m, the satisfiability of φ_0 over b-bounded communication closed diagrams can be checked in time $2^{O(m+b^2)}$.*

Proof: We associate a diagram automaton $\mathcal{A}_0$ (over the alphabet of models based on b-bounded layers) with φ_0 such that $L^b_C(\mathcal{A}_0) = \{M \mid M$ is a model of φ_0 based on a b-bounded communication closed LLD$\}$. Hence, φ_0 is satisfiable iff $L^b_C(\mathcal{A}_0) \neq \emptyset$. Since the emptiness of the language accepted by a diagram automaton is decidable (Theorem 3.3.2), it follows that checking if φ_0 is satisfiable is also decidable. Note that the states and transitions are the same as what we would use for LTL as λ-LTL is basically LTL and diagram automata are Büchi automata running over LLDs; we only need to decide the diagram alphabet carefully, and this relies on decidability of the logic LD_0 over b-bounded diagrams (Theorem 6.1.1).

We start by defining the sub-formula closure $CL(\varphi_0)$ of φ_0. Define $CL'(\varphi_0)$ to be the least set of formulas such that the following conditions are satisfied:

- $\varphi_0 \in CL'(\varphi_0)$.
- $\neg\psi \in CL'(\varphi_0)$ implies $\psi \in CL'(\varphi_0)$.
- $\psi_1 \vee \psi_2 \in CL'(\varphi_0)$ implies $\psi_1, \psi_2 \in CL'(\varphi_0)$.

- $\bigcirc\psi \in CL'(\varphi_0)$ implies $\psi \in CL'(\varphi_0)$.
- $\psi_1 \mathbf{U} \psi_2 \in CL'(\varphi_0)$ implies $\psi_1, \psi_2, \bigcirc(\psi_1 \mathbf{U} \psi_2) \in CL'(\varphi_0)$.

Now define $CL(\varphi_0) \overset{\text{def}}{=} CL'(\varphi_0) \cup \{\neg\psi \mid \psi \in CL'(\varphi_0)\} \cup \{a_0, \neg a_0\}$ with $\neg\neg$ treated as identity and $a_0 \in \Gamma$. The presence of a_0 is to ensure that at least one layer name is included, even if none is mentioned in φ_0.

A set $A \subseteq CL(\varphi_0)$ is said to be an atom iff it satisfies the following conditions:

- for every formula $\neg\psi \in CL(\varphi_0)$, $\neg\psi \in A$ iff $\psi \notin A$.
- for every formula $\psi_1 \vee \psi_2 \in CL(\varphi_0)$, $\psi_1 \vee \psi_2 \in A$ iff $\psi_1 \in A$ or $\psi_2 \in A$.
- for every formula $\psi_1 \mathbf{U} \psi_2 \in CL(\varphi_0)$, $\psi_1 \mathbf{U} \psi_2 \in A$ iff either $\psi_2 \in A$ or $\psi_1, \bigcirc(\psi_1 \mathbf{U} \psi_2) \in A$.
- $\mid A \cap \Gamma \mid = 1$.

Let $AT(\varphi_0)$ denote the set of atoms of φ_0. Let P_0 denote the set of propositions that appear in φ_0 and $\Gamma_0 = \Gamma \cap CL(\varphi_0)$. The alphabet of the automaton $\mathcal{A}_0$ is the set $\mathcal{LC}_b^{P_0} = \{M = (D, V_E, a) \mid D = (E, \leq, \phi) \in \mathcal{LC}_b$ and $V_E : E \to 2^{P_0}$ and $a \in \Gamma_0\}$. That is, the alphabet of the diagram automaton is the set of all b-bounded Lamport diagrams along with their valuations. Notice that given a model $M = (D, V_E, V_\lambda)$ where D is a communication closed and b-bounded LLD with $\nu_D = 0, 1, \ldots$ (without loss of generality), we can view M as an infinite word over $\mathcal{LC}_b^{P_0}$ given by $M = M_0 \bullet M_1, \bullet \ldots$ where $M_i = (D_i, V_E \restriction D_i, a_i)$ for all i, D_i is the Lamport diagram corresponding to layer i of D and $V_\lambda(i) = a_i$.

Define a diagram automaton $\mathcal{A}_0$ over $\mathcal{LC}_b^{P_0}$ as follows:

$\mathcal{A}_0 = (Q, \mathcal{LC}_b^{P_0}, \to, I, F)$ where

1. $Q = AT(\varphi_0) \times U$ where $U = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in CL(\varphi_0)\}$.
2. $I = \{(A, \emptyset) \mid A \in AT(\varphi_0)$ such that $\varphi_0 \in A\}$.
3. $F = \{(A, \emptyset) \mid A \in AT(\varphi_0)\}$, and
4. $\to$ is given by $(A, u) \xrightarrow{M} (B, v)$ for $A, B \in Q$ and $M = (D, V_E, a)$ iff

   (a) $a \in A$,

(b) $\alpha @ i \in A$ iff $M, e \models_{cl} \alpha$ where e is the i-minimum event in D,

(c) for every $\bigcirc\psi \in CL(\varphi_0)$, $\bigcirc\psi \in A$ iff $\psi \in B$, and

(d) if $u \neq \emptyset$ then, $v = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in u$ and $\psi_2 \notin B\}$. Otherwise, $v = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in B$ and $\psi_2 \notin B\}$.

In order to construct the set Q of states, we need to check for the satisfiability of layer formulas present in atoms over b-bounded diagrams. The proof of Theorem 6.1.1 gives us an algorithm to check this. Since the complexity of checking the satisfiability of the layer formulas against b-bounded diagrams is $m \cdot O(2^{b^2})$, it follows that Q is constructible in time $2^{O(m+b^2)}$. The proof follows from the following lemma.

Lemma 6.5.2 $L^b_C(\mathcal{A}_0) = \{M \mid M$ *is a model of* φ_0 *based on a b-bounded communication closed LLD* $\}$.

Let $M = M_0 \bullet M_1 \bullet \ldots \in L^b_C(\mathcal{A}_0)$. We show that $M, k_0 \models_c \varphi_0$. Let $\rho = (A_0, u_0), (A_1, u_1), \ldots$ be an accepting run of $\mathcal{A}_0$ on M. We first prove the following claim.

Claim: For all $\psi \in CL(\varphi_0)$, for all $i \in \mathbb{N}$, $M, i \models_c \psi$ iff $\psi \in A_i$.

Proof of claim: The proof is by induction on the structure of ψ.

$\psi = \alpha @ j$ $M, i \models_c \alpha @ j$ iff $M, e \models_{cl} \alpha$ where e is the j-minimum event of D_i iff $\alpha @ j \in A_i$ (by definition of the transition relation).

$\psi = a$**,** $a \in \Gamma$ $M, i \models_c a$ iff $V_\lambda(i) = a$ iff $a \in A_i$.

$\psi = \neg\psi'$ $M, i \models_c \neg\psi'$ iff $M, i \not\models_c \psi'$ iff $\psi' \notin A_i$ (by induction hypothesis) iff $\neg\psi' \in A_i$ (since A_i is an atom).

$\psi = \psi_1 \vee \psi_2$ $M, i \models_c \psi_1 \vee \psi_2$ iff $M, i \models_c \psi_1$ or $M, i \models_c \psi_2$ iff $\psi_1 \in A_i$ or $\psi_2 \in A_i$ (by induction hypothesis) iff $\psi_1 \vee \psi_2 \in A_i$ (since A_i is an atom).

$\psi = \bigcirc\psi'$ Suppose $M, i \models_c \bigcirc\psi'$. Then, $M, i{+}1 \models_c \psi'$ where $i{+}1$ is the successor of i in M. By induction hypothesis, it follows that $\psi' \in A_{i+1}$ which implies $\bigcirc\psi' \in A_i$ (since $(A_i, u_i) \rightarrow (A_{i+1}, u_{i+1})$ in $\mathcal{A}_0$). Conversely, suppose $\bigcirc\psi' \in A_i$. We have to prove that $M, i \models_c \bigcirc\psi'$. Since $\bigcirc\psi' \in A_i$ and $(A_i, u_i) \rightarrow (A_{i+1}, u_{i+1})$, we have $\psi' \in A_{i+1}$. By induction hypothesis it follows that $M, i+1 \models_c \psi'$ and so $M, i \models_c \bigcirc\psi'$.

$\psi = \psi_1 \mathbf{U} \psi_2$ Suppose $M, i \models_c \psi_1 \mathbf{U} \psi_2$. We have to prove that $\psi_1 \mathbf{U} \psi_2 \in A_i$. Now, since $M, i \models_c \psi_1 \mathbf{U} \psi_2$, there exists $k \geq i$ such that $M, k \models_c \psi_2$ and for all j, $i \leq j \leq k$, $M, j \models_c \psi_1$. We will show that $\psi_1 \mathbf{U} \psi_2 \in A_i$ by a second induction on $k - i$.

Base case: $(k - i = 0)$. Then, $k = i$ and $M, i \models_c \psi_2$. By the main induction hypothesis, $\psi_2 \in A_i$ and from the definition of atoms it follows that $\psi_1 \mathbf{U} \psi_2 \in A_i$.

Induction step: $(k - i > 0)$. By the semantics of the $\mathbf{U}$ modality, $M, i \models_c \psi_1$ and $M, i+1 \models_c \psi_1 \mathbf{U} \psi_2$. Now, by the secondary induction hypothesis, $\psi_1 \mathbf{U} \psi_2 \in A_{i+1}$. From the definition of $\rightarrow$, we have $\bigcirc(\psi_1 \mathbf{U} \psi_2) \in A_i$. Also, by the main induction hypothesis, $M, i \models_c \psi$ implies $\psi \in A_i$. Since A_i is an atom, we have $\psi_1 \mathbf{U} \psi_2 \in A_i$.

Conversely, suppose $\psi_1 \mathbf{U} \psi_2 \in A_i$. We must show that $M, i \models_c \psi_i \mathbf{U} \psi_2$. Since ρ is an accepting run of $\mathcal{A}_0$ on M, there exists $k \geq i$ such that $\psi_2 \in A_k$. We will show that $M, i \models_c \psi_1 \mathbf{U} \psi_2$ by using a second induction on $k - i$.

Base case: $(k - i = 0)$. Then, $k = i$ and $\psi_2 \in A_i$. By the main induction hypothesis, $M, i \models_c \psi_2$ and so $M, i \models_c \psi_1 \mathbf{U} \psi_2$ as well.

Induction step: $(k - i > 0)$. Now, $\psi_2 \notin A_i$ and $\psi_1 \mathbf{U} \psi_2 \in A_i$ implies $\psi_1, \bigcirc(\psi_1 \mathbf{U} \psi_2) in A_i$. Since $(A_i, u_i) \rightarrow (A_{i+1}, u_{i+1})$, we have $\psi_1 \mathbf{U} \psi_2 \in A_{i+1}$. By the secondary induction hypothesis, $M, i+1 \models_c \psi_1 \mathbf{U} \psi_2$ implies $M, i \models_c \bigcirc(\psi_1 \mathbf{U} \psi_2)$. Simultaneously, by the main induction hypothesis, $M, i \models_c \psi_1$ implies $M, i \models_c \psi_1 \mathbf{U} \psi_2$.

Thus induction on the structure of ψ is complete and the claim is true. From the claim it follows that $M, k_0 \models_c \varphi_0$ as required.

Conversely, to show that every model of φ_0 is in $L^b_C(\mathcal{A}_0)$, let $M = M_0 \bullet M_1 \bullet \ldots$ be such that $M, 0 \models_c \varphi_0$. We have to exhibit an accepting run of $\mathcal{A}_0$ on M. For $i \geq 0$, define $A_i = \{\psi \in CL(\varphi_0) \mid M, i \models_c \psi\}$. We can show that each A_i is an atom. Now, define a sequence $\rho = (A_0, u_0), (A_1, u_1), \ldots$ where the A_i's are as given before. We define u_i by induction on i. $u_0 = \emptyset$ and for $i > 0$, define $u_{i+1} = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in A_{i+1}, \psi_2 \notin A_{i+1}\}$ if $u_i = \emptyset$ and $\{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in u_i, \psi_2 \notin A_{i+1}\}$ otherwise. We can show that ρ is an accepting run of $\mathcal{A}_0$ on M and hence $M \in L^b_C(\mathcal{A}_0)$. $\square$

6.6 S_b-satisfiability

In this section we consider another restriction on the class of layered Lamport diagrams considered as models and show that λ-LTL is again decidable. The class of models we consider are those based on channel b-bounded LLDs (for a fixed $b \in \mathbb{N}$) i.e, S_b-satisfiability. We again consider the automata theoretic approach to solve the satisfiability problem and show that S_b-satisfiability is decidable by associating a fragment automaton with every formula such that every model of the formula corresponds to an accepting run of the associated automaton, and vice versa. Then, the given formula is satisfiable iff the associated fragment automaton accepts a non-empty language and since the emptiness problem of fragment automata is decidable (Theorem 3.4.2), we get decidability of S_b-satisfiability.

Note that this proof cannot simply mimic the proof of C_b-satisfiability, since labelling events of fragments with layer atoms does not give models per se; we get models only when the fragments are 'tiled' suitably, yielding a layered Lamport diagram. Also, the complexity of checking for S_b-satisfiability of a given formula turns out to be slightly higher than that of C_b-satisfiability as the time complexity to check emptiness of fragment automata is higher that that of diagram automata.

Theorem 6.6.1 *Given a temporal formula φ_0 of length m, the satisfiability of φ_0 over channel b-bounded diagrams can be checked in time $2^{O(m \cdot b^2)}$.*

We prove this theorem by associating a fragment automaton $\mathcal{A}_0$ (over an alphabet of fragments which occur as layers of channel b-bounded LLDs) with every formula φ_0 such that $L^b_S(\mathcal{A}_0) = \{M \mid M$ is a model of φ_0 based on a channel b-bounded LLD $\}$. Hence, φ_0 is satisfiable iff $L^b_S(\mathcal{A}_0) \neq \emptyset$. Since the emptiness of the language accepted by a fragment automaton is decidable, it follows that checking if φ_0 is satisfiable is also decidable.

The sub-formula closure $CL(\varphi_0)$ and the set of *temporal* atoms $AT(\varphi_0)$ are defined as in the proof of Theorem 6.5.1. However, since $\mathbf{X}_j$ and $\mathbf{Y}_j$ modalities need not be satisfied within the same layer, we have to do extra work to take care of requirements of the form $\alpha@i$. We define *layer atoms* which are atoms corresponding to formulas of LD_0 and carry pending $\mathbf{X}_j$ and $\mathbf{Y}_j$ with transitions of the automaton till they are met. For every formula of the form $\alpha@i \in CL(\varphi_0)$, the set $SL(\alpha)$ of layer sub-formulas of α is defined below. Let SF_l denote the set of layer sub-formulas associated with φ_0. That is, $SF_l = \cup_{\alpha@i \in CL(\varphi_0)} SL(\alpha)$. The definition of when $A \subseteq SF_l$ is called a layer atom is also given below. We use the notation AT_l to denote the set of layer atoms associated with φ_0.

The sub-formula closure $SL(\alpha_0)$ of a layer formula $\alpha_0 \in \Phi_l$ is defined as follows. Define $SL'(\alpha_0)$ to be the least set of formulas such that the following conditions are satisfied:

- $\alpha_0 \in SL'(\alpha_0)$.
- $\alpha_1 \vee \alpha_2 \in SL'(\alpha_0)$ implies $\alpha_1, \alpha_2 \in SL'(\alpha_0)$.
- if $C\alpha \in SL'(\alpha_0)$ then $\alpha \in SL'(\alpha_0)$, where $C \in \{\neg, \overline{\mathbf{X}}_i, \overline{\mathbf{Y}}_i, \mathbf{F}, \mathbf{G}, \mathbf{P}, \mathbf{H}\}$, $i \in [n]$.
- if $\mathbf{X}_i\alpha \in SL'(\alpha_0)$ then $\overline{\mathbf{X}}_i\alpha \in SL'(\alpha_0)$, $i \in [n]$.
- if $\mathbf{Y}_i\alpha \in SL'(\alpha_0)$ then $\overline{\mathbf{Y}}_i\alpha \in SL'(\alpha_0)$, $i \in [n]$.
- $\mathbf{X}_i \mathit{True} \in SL'(\alpha_0)$ for all $i \in [n]$, and $\mathbf{Y}_i \mathit{True} \in SL'(\alpha_0)$ for all $i \in [n]$.
- for all $i \in [n]$, $\tau_i \in SL'(\alpha_0)$.

Now define $SL(\alpha_0) \overset{\text{def}}{=} SL'(\alpha_0) \cup \{\neg\alpha \mid \alpha \in SL'(\alpha_0)\}$ with $\neg\neg$ treated as identity.

A set $A \subseteq SF_l$ is said to be a layer atom iff it satisfies the following conditions:

- for every formula $\neg\alpha \in SL(\alpha_0)$, $\neg\alpha \in A$ iff $\alpha \notin A$.
- for every formula $\alpha_1 \vee \alpha_2 \in SL(\alpha_0)$, $\alpha_1 \vee \alpha_2 \in A$ iff $\alpha_1 \in A$ or $\alpha_2 \in A$.
- if $\mathbf{X}_i\alpha \in A$ then $\mathbf{X}_i \mathit{True} \in A$ and $\overline{\mathbf{X}}_i\alpha \in A$.
- if $\mathbf{Y}_i\alpha \in A$ then $\mathbf{Y}_i \mathit{True} \in A$ and $\overline{\mathbf{Y}}_i\alpha \in A$.
- if $\mathbf{G}\alpha \in A$ then $\alpha \in A$.
- if $\mathbf{H}\alpha \in A$ then $\alpha \in A$.
- $\mid \{\tau_i \mid i \in [n], \tau_i \in A\} \mid = 1$.

Define a relation $\preceq$ on layer atoms as follows: Let A and B be layer atoms such that $\tau_i \in A$ and $\tau_j \in B$. Then $A \preceq B$ iff:

- if $\overline{\mathbf{X}}_j\alpha \in A$ then $\alpha \in B$.
- if $\overline{\mathbf{Y}}_i\alpha \in B$ then $\alpha \in A$.

Call an n-tuple of layer atoms $(A_1, \ldots, A_n)$ coherent if, for all $i \in [n]$, $\tau_i \in A_i$. Given two coherent n-tuples of layer atoms $(A_1, \ldots, A_n)$ and $(B_1, \ldots, B_n)$, we say $(A_1, \ldots, A_n) \preceq (B_1, \ldots, B_n)$ if for all $i \in [n]$, $A_i \preceq B_i$. Let $\widetilde{AT_l}$ denote the set of coherent n-tuples of layer atoms. Below, we will also need the set $A_{tup} = \widetilde{AT_l} \cup \{(\perp_1, \ldots, \perp_n)\}$, where the symbol $\perp_i$ will be used to denote that no layer atom is associated with i. By convention, set $(\perp_1, \ldots, \perp_n) \preceq X$, for every $X \in \widetilde{AT_l}$.

We now move on to computing the alphabet of the fragment automaton $\mathcal{A}_0$ to be associated with φ_0. For this first define the set $\mathcal{LF}_b(AT_l)$ of b-bounded fragments which are labelled by layer atoms. Define $LF = (F, a)$ where $F = (E, \leq, \phi, AT_l, \eta)$ and $a \in \Gamma_0$ ($\Gamma_0 = \Gamma \cap CL(\varphi_0)$). Define $\nu : E \rightarrow AT_l$ by: $\nu(e) = A$, where $\eta(e) = (A, T)$. We require that ν satisfies the following *consistency* conditions:

- for all $e \in E$, $\phi(e) = i$ iff $\tau_i \in \nu(e)$.
- for all $e, e' \in E$, if $e \lessdot e'$ then $\nu(e) \preceq \nu(e')$.
- for all $e \in E_i$, for all $\mathbf{F}\alpha \in SF_l$, $\mathbf{F}\alpha \in \nu(e)$ iff there exists $e' \in E_i$ such that $e \leq e'$ and $\alpha \in \nu(e')$.
- for all $e \in E_i$, for all $\mathbf{P}\alpha \in SF_l$, $\mathbf{P}\alpha \in \nu(e)$ iff there exists $e' \in E_i$ such that $e' \leq e$ and $\alpha \in \nu(e')$.
- for all $e \in E_i$, if $(s, j, A) \in \eta(e)$ then $\mathbf{X}_j\,True \in \nu(e)$ and $\nu(e) \preceq A$.
- for all $e \in E_i$, if $(r, j, A) \in \eta(e)$ then $\mathbf{Y}_j\,True \in \nu(e)$ and $A \preceq \nu(e)$.

By $\nu_{min}(F)$, we denote the (coherent) n-tuple of layer atoms $(\nu(e_1), \ldots, \nu(e_n))$, where e_i is the i-minimum event in F. Similarly, by $\nu_{max}(F)$, we denote the (coherent) n-tuple of layer atoms $(\nu(f_1), \ldots, \nu(f_n))$, where f_i is the i-maximum event in F.

We are now ready to define the required fragment automaton working on the alphabet $\mathcal{LF}_b(AT_l)$ of fragments. The automaton construction follows the one used for C_b-satisfiability, but states are now enriched with coherent tuples of layer atoms (except initial states, where we resort to the use of $\perp_i$). Such a tuple corresponds to the layer atoms labelling the maximal events that the automaton has seen thus far. Hence the transition relation checks that this tuple is $\preceq$-related to the minimal ones of the fragment labelling the transition, thus ensuring that the fragment may be joined to the diagram being constructed by the accepting run of the automaton.

Formally, the automaton $\mathcal{A}_0$ associated with φ_0 is defined as follows:

$$\mathcal{A}_0 = (Q, \mathcal{LF}_b(AT_l), \rightarrow, I, F)$$

where

- $Q = AT(\varphi_0) \times U \times A_{tup}$

  where $U = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in CL(\varphi_0)\}$.
- $I = \{(T, \emptyset, (\bot_1, \ldots, \bot_n) \mid T \in AT(\varphi_0)$ such that $\varphi_0 \in T\}$.
- $F = \{(T, \emptyset, X) \mid T \in AT(\varphi_0), X \in \widetilde{AT_l}\}$.
- $\rightarrow$ is given by

  $(T, u, (A_1, \ldots, A_n)) \xrightarrow{LF} (T', u', (A'_1, \ldots, A'_n))$ (where $LF = (F, a)$, iff

  1. $a \in T$.
  2. For every $\bigcirc\psi \in CL(\varphi_0)$, $\bigcirc\psi \in T$ iff $\psi \in T'$.
  3. If $u \neq \emptyset$ then, $u' = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in u$ and $\psi_2 \notin T'\}$. Otherwise, $u' = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in T'$ and $\psi_2 \notin T'\}$.
  4. $\nu_{max}(F) = (A'_1, \ldots, A'_n)$.
  5. $(A_1, \ldots, A_n) \preceq \nu_{min}(F)$.
  6. for $i \in [n]$, $\alpha@i \in T$ iff $\alpha \in B_i$, where $\nu_{min}(F) = (B_1, \ldots, B_n)$.

Lemma 6.6.2 $L^b_S(\mathcal{A}_0) = \{M \mid M$ *is a model of* φ_0 *based on a channel b-bounded LLD* $\}$.

Proof:

Let $D = ((LF_0 \bullet LF_1) \bullet \ldots) \in L^b_S(\mathcal{A}_0)$, where $LF_k = (F_k, a_k)$. Suppose $D = (E, \leq, \phi, \lambda)$. Note that, from the way concatenation of fragments has been defined, we can define the map $\nu : E \rightarrow AT_l$ (uniquely), based on the map ν_k. Now define $M = (D, V_E, V_\lambda)$ by: for all $e \in E$, $V_E(e) = \nu(e) \cap P$ and $V_\lambda(k) = a_k$.

Note that D has a crucial property: for all $e \in E_i$, if e comes from $F_k, k \geq 0$ and $(s, j, A) \in \eta_k(e)$, then there exists $\ell > k$ and $e' \in E_j$ such that e' comes from F_ℓ, $(r, i, B) \in \eta_\ell(e')$, $e \lessdot e'$, $\nu(e') = A$ and $\nu(e) = B$. A similar assertion holds the other way as well: for e, k as above, if $(r, j, A) \in \eta_k(e)$, then there exists $\ell < k$ and $e' \in E_j$ such that e' comes from F_ℓ, $(s, i, B) \in \eta_\ell(e')$, $e' \lessdot e$, $\nu(e') = A$ and $\nu(e) = B$. In addition, if e_1 is the i-maximum event of F_k and e_2 the i-minimum event of F_{k+1}, then by definition of the transition relation $\rightarrow$, we have that $\nu(e_1) \preceq \nu(e_2)$. From these, and the property of ν_k for each fragment F_k, we have the following, which we call the *successor labelling property*:

- for all $e, e' \in E$, if $e \lessdot e'$, then $\nu(e) \preceq \nu(e')$.
- for all $e \in E$, if $\mathbf{X}_i \mathit{True} \in \nu(e)$, then there exists $e' \in E_i$ such that $e \lessdot e'$.
- if $\mathbf{Y}_i \mathit{True} \in \nu(e)$, then there exists $e' \in E_i$ such that $e' \lessdot e$.

We first prove the following claim.

Claim: For all $e \in E$, $\alpha \in SF_l$, $M, e \models_l \alpha$ iff $\alpha \in \nu(e)$.

Proof of Claim: The proof proceeds by induction on the structure of α. The base case, when $\alpha \in P$, is seen to hold by the way V_E was defined. The cases when α is of the form $\neg\beta$ or of the form $\beta_1 \vee \beta_2$ routinely follow by application of the induction hypothesis and using the properties of an atom.

Now suppose $\alpha = \mathbf{X}_i\beta$ and let $e \in E$. If $M, e \models_l \mathbf{X}_i\beta$, then there exists $e' \in E_i$ such that $e \lessdot e'$ and $M, e' \models_l \beta$. We then have, by successor labelling property, $\nu(e) \preceq \nu(e')$. By induction hypothesis, $\beta \in \nu(e')$. Then, by definition of $\preceq$, $\mathbf{X}_i\beta \in \nu(e)$, as required.

On the other hand, if $\mathbf{X}_i\beta \in \nu(e)$, then by definition of layer atoms, $\mathbf{X}_i \mathit{True} \in \nu(e)$. By successor labelling property, there exists $e' \in E_i$ such that $e \lessdot e'$ and hence $\nu(e) \preceq \nu(e')$. Since $\nu(e)$ is an atom, $\overline{\mathbf{X}}_i\beta \in \nu(e)$ as well, and hence, by definition of $\preceq$, $\beta \in \nu(e')$. But then, by induction hypothesis, $M, e' \models_l \beta$ and hence we get: $M, e \models_l \mathbf{X}_i\beta$, as required.

The case when $\alpha = \mathbf{Y}_i\beta$ is similar. The cases when $\alpha = \mathbf{F}\beta$ or $\alpha = \mathbf{P}\beta$ are proved easily using the consistency condition on ν_k where $k = \lambda(e)$ and the induction hypothesis. This completes the induction, proving the Claim.

We now show that $M, 0 \models \varphi_0$. Let $\rho = (T_0, u_0, X_0), (T_1, u_1, X_1), \ldots$ be an accepting run of $\mathcal{A}_0$ on D. For this, it suffices to show that for every temporal formula

$\psi \in CL(\varphi_0)$ and every layer k, $M, k \models \psi$ iff $\psi \in T_k$. The proof of this assertion, by induction on the structure of ψ, is exactly as that of the corresponding one in Lemma 6.5.2 except for the base case when $\psi = \alpha @ i$.

For the base case, observe that $M, k \models \alpha@i$ iff $M, e \models_l \alpha$ where e is the i-minimum event of the fragment F_k. But, from the claim above we know that $M, e \models_l \alpha$ iff $\alpha \in \nu(e)$. By definition of $\rightarrow$, $\alpha \in \nu(e)$ iff $\alpha@i \in T_k$, when e is the i-minimum event of F_k. The other base case, when $\psi \in \Gamma$, is also easily seen to follow from the definition of $\rightarrow$.

Conversely, to show that every model of φ_0 is in $L^b_S(\mathcal{A}_0)$, let $M = (D, V_E, V_\lambda)$ be such that such that $M, 0 \models \varphi_0$. Let $F_0, F_1, \ldots$ be the sequence of fragments associated with D. We show that $L^b_S(\mathcal{A}_0) \neq \emptyset$ by exhibiting an accepting run of $\mathcal{A}_0$.

For an event e of M, define $A_e = \{\alpha \in SF_l \mid M, e \models_l \alpha\}$. It can be easily checked that A_e is a layer atom. Now let $k \geq 0$. Define $T_k = \{\psi \in CL(\varphi_0) \mid M, k \models \psi\}$. We can show that T_k is a temporal atom. For $j \in [n]$, define A^k_j as follows: $A^0_j = \bot_j$; $A^{k+1}_j = A_{e^j_k}$ where e^j_k is the j-maximum event of F_k.

Let $LF_k = (F_k, a)$ where for all e in the fragment F_k, $\nu_k(e) = A_e$, and $a \in \Gamma$ such that $M, k \models a$. We now construct a sequence

$\rho = (T_0, u_0, (A^0_1, \ldots A^0_n)), (T_1, u_1, (A^1_1, \ldots A^1_n)), \ldots$

of states of $\mathcal{A}_0$ and show that ρ is a accepting run of $\mathcal{A}_0$ on $LF = LF_0, LF_1, \ldots$. The sets u_k are defined by induction on k. Define $u_0 = \emptyset$ and suppose u_k has been defined. Define $u_{k+1} = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in T_{k+1}, \psi_2 \notin T_{k+1}\}$ if $u_k = \emptyset$ and $u_{k+1} = \{\psi_1 \mathbf{U} \psi_2 \mid \psi_1 \mathbf{U} \psi_2 \in u_k, \psi_2 \notin T_{k+1}\}$ otherwise.

We can then easily show that ρ is an accepting run of $\mathcal{A}_0$ on LF. □

Chapter 7

Message Sequence Charts

In this chapter we present other related models for describing behaviours of distributed systems whose agents communicate by exchanging messages across buffers and compare the expressive power of these formalisms with those of Lamport diagrams and layered Lamport diagrams. The model discussed in this chapter is that of Message Sequence Charts (MSCs). We again talk about n-agent distributed systems and their behaviours are now described as MSCs. MSCs focus on message-passing (i.e., the sequence of messages exchanged between agents) happening in the distributed system unlike Lamport diagrams which focus on causality.

We introduce MSCs, various models to represent collections of MSCs and compare them against relevant classes of Lamport diagrams. We work only with MSCs over a finite number of events (describing only finite behaviours of distributed message passing systems) as some of the results discussed here will use this fact crucially. Consequently, the Lamport diagrams are also finite, including layered Lamport diagrams.

We also present results related to the expressive power of collections of MSCs with respect to each other and with relevant classes of finite LLDs. We introduce a Monadic Second Order (MSO) logic over MSCs and over Lamport diagrams. In the context of MSCs, we present an algorithm to model check CMSGs against MSO specifications. In the context of Lamport diagrams, it is not surprising to see that satisfiability problem is undecidable. We show that checking the satisfiability of MSO formulas against communication closed and bounded LLDs and against channel bounded LLDs is decidable.

7.1 Message Sequence Charts

Message Sequence Charts (MSCs) are a standard notion [ITU97, RGG96] widely used to capture system requirements in the early stages of design of communication protocols. The graphical layout of an MSC describes how the components (agents) of a distributed system communicate by exchanging messages. In its simplest form, an MSC depicts a finite pattern describing the exchange of messages between agents of a system and represents a single partially-ordered and finite execution of the events of the system.

The events of an MSC are labelled by actions from a finite alphabet which identifies the event occurrence as send, receive or a local event. Towards defining this alphabet, let us fix a finite set of messages Γ_m and a finite set of local actions Γ_l for the rest of this chapter (unless they are defined explicitly). For $i, j \in [n]$ such that $i \neq j$, let $\Sigma_i = \{(i!j, a), (i?j, a), (i, l) \mid a \in \Gamma_m, l \in \Gamma_l\}$ denote the set of actions that agent i participates in. The action $(i!j, a)$ should be read as "i sends the message a to j" while the action $(i?j, a)$ means "i receives the message a from j". (i, l) represents i doing the local action l. The alphabet of the system which denotes the set of all actions is defined by $\Sigma = \bigcup_{i \in [n]} \Sigma_i$. Note that this alphabet is different from the distributed alphabet we considered in earlier chapters. Earlier, the distributed alphabet was made up of a set of abstract actions whereas the one considered here is made up of actions with a structure—they represent send, receive or local event occurrences.

Definition 7.1.1 *A* **Message Sequence Chart** *(MSC) over $[n]$ is a tuple*

$$M = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta)$$

where

- *E is a finite set of events.*
- *$\ell : E \to \Sigma$ is the* labelling function *which identifies for each event an action. Let $E_i = \{e \in E \mid \ell(e) \in \Sigma_i\}$ denote the set of events of E which i participates in. Also, let $S_E = \{e \in E \mid \ell(e) = (i!j, a)$ for some $i, j \in [n], a \in \Gamma_m\}$ denote the set of* send events *and $R_E = \{e \in E \mid \ell(e) = (i?j, a)$ for some $i, j \in [n], a \in \Gamma_m\}$ denote the set of* receive events *of E.*

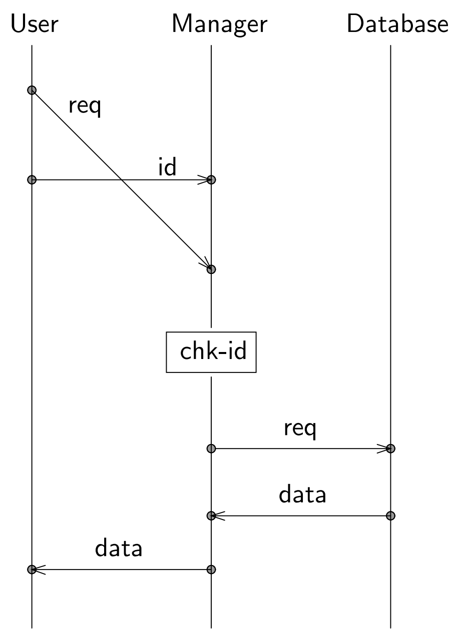


Figure 7.1: A Message Sequence Chart

- *$\eta : S_E \rightarrow R_E$ is the* matching function *which associates with each send event, its corresponding receive event. We require η to be a bijection and for every $e, e' \in E$, if $\eta(e) = e'$, then $\ell(e) = (i!j, a)$ and $\ell(e') = (j?i, a)$ for some $i, j \in [n], a \in \Gamma_m$.*

- *$\leq_i$ is a total order on E_i for each $i \in [n]$.*

- *Let $\widehat{\lessapprox} = (\bigcup_{i\in[n]} \leq_i) \cup \{(e, e') \mid e, e' \in E$ and $\eta(e) = e'\}$. Let $\leq = (\widehat{\lessapprox})^*$ be the reflexive, transitive closure of $\widehat{\lessapprox}$. Then $\leq$ denotes the* causal ordering *of events in the MSC and we require it to be a partial order on E.*

For technical reasons, we require MSCs to satisfy an additional non-degeneracy condition [AEY00]. We say an MSC is degenerate if there is an agent which sends two identical messages that are received out of order. More formally, an MSC is degenerate if there are events e_1, e_2, e'_1, e'_2 such that $\ell(e_1) = \ell(e_2) = (i!j, a)$, $\eta(e_1) = e'_1$, $\eta(e_2) = e'_2$ and $e_1 <_i e_2$ but $e'_2 <_j e'_1$.

Figure 7.1 shows an MSC over three agents (User, Manager and Database). req, id and data are the message labels and chk-id represents a local action of the agent Manager. The labelling function ℓ and the matching function η are easy to extract from the diagram, for instance, the label of the first event of agent User is (User!Manager, req). Note that the message id overtakes the message req between the agents User and Manager. The local total orders of every agent are also illustrated in the figure: the events of every agent are given along a line in order of their occurrence from top to bottom.

7.1.1 MSCs and Lamport diagrams

We have introduced MSCs and Lamport diagrams as models describing behaviours of n-agent systems. In this section, we compare the expressive power of these formalisms.

As the above definition describes, an MSC is a special kind of graph over an underlying set of events such that the ordering relation defined by the edges of the graph is a partial order. The set of events is partitioned into set of send, receive and local events (according to their labels). The messages and the local total order of the events of each agent define the edges of the MSC. The reflexive and transitive closure of the edges gives the causal order of the events. Notice that the causal order gives all possible dependencies between events among which a few edges are explicitly identified as messages by the matching function. On the other hand, Lamport diagrams mainly describe the causal dependence of the various events (which is also required to be a partial order) and the underlying communication pattern is derived from this order. With this understanding, Lamport diagrams can be seen as finite partial orders generated by MSCs (as done in [AHP96], for example, even though they do not use the term Lamport diagrams explicitly). Thus, Lamport diagrams describe causality rather than messages as in MSCs. Also, Lamport diagrams can have simultaneous send and/or receive events, an additional feature not modelled by MSCs. We formalize these concepts below.

Given an MSC $M = (E, \{\leq_i\}_{i\in[n]}, \ell, \eta)$, define the structure $D_M \overset{\text{def}}{=} (E, \leq, \phi)$ where for $e \in E$, $\phi(e) = i$ iff $e \in E_i$. It is easily seen that D_M is a Lamport diagram over a finite set of events. We call D_M as the Lamport diagram associated with M. Apart from being a Lamport diagram over a finite set of events, it satisfies the following conditions:

- It has no simultaneous send events: for all $e \in E$, there exists at most one $e' \in E$ such that $e <_c e'$.

- It has no simultaneous receive events: for all $e \in E$, there exists at most one

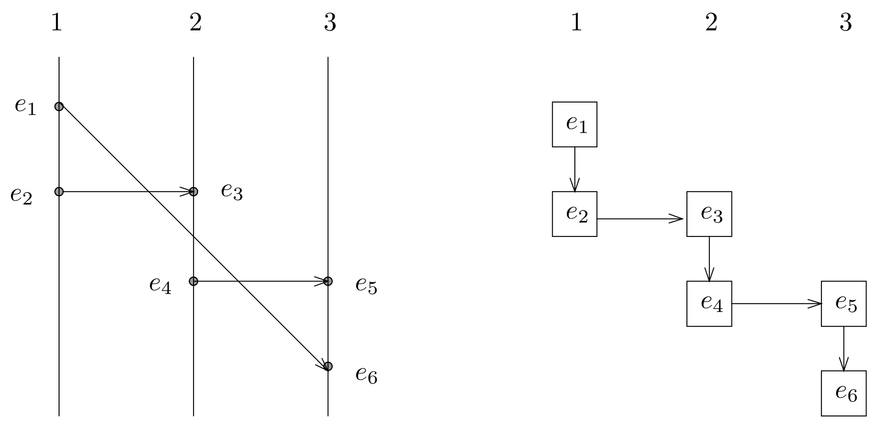


Figure 7.2: An MSC and its associated Lamport diagram

$e' \in E$ such that $e' <_c e$.

- It has no event which is a send and a receive simultaneously: for all $e \in E$, there exists at most one $e' \in E$ such that either $e <_c e'$ or $e' <_c e$.

For example, Figure 7.2 gives an MSC and the Hasse diagram of the Lamport diagram associated with it. The labels of the event occurrences of the MSC are omitted for the sake of clarity. Note that $\eta(e_1) = e_6$ in the MSC but, $e_1 \not<_c e_6$ in the corresponding Lamport diagram. Since e_6 is causally dependent on e_1 (as shown in the Hasse diagram on the right side), the message sent by e_1 and received by e_6 is not modelled explicitly in the Lamport diagram.

Conversely, not every finite Lamport diagram can be represented by an MSC as such. The main reason is the presence of simultaneous send and/or receive events in a Lamport diagram, a feature not allowed in MSC. Given a Lamport diagram $D = (E, \leq, \phi)$ (with E finite) which has no simultaneous send and receive events as above, we define the corresponding MSC as follows:

Fix $\Gamma_m = \{a\}$, $\Gamma_l = \{b\}$ and for $i \in [n]$, $\Sigma_i = \{(i!j, a), (i?j, a), (i, b)\}$. Define the structure $M_D = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta)$, where $\leq_i$ is the local ordering induced by D, $(e, e') \in \eta$ iff $e <_c e'$ in D and ℓ is defined as follows:

- $\ell(e) = (i!j, a)$ iff $e \in E_i$ and there exists $e' \in E_j$ such that $e <_c e'$ in D.
- $\ell(e) = (i?j, a)$ iff $e \in E_i$ and there exists $e' \in E_j$ such that $e' <_c e$ in D.
- $\ell(e) = (i, b)$ iff there exists no event $e' \in E$ such that $e <_c e'$ or $e' <_c e$.

It is easily seen that M_D is an MSC and it has no 'over-taking' messages.

The following proposition summarizes the above-mentioned correspondence.

Proposition 7.1.2 *A finite Lamport Diagram D with no multiple send events and no multiple receive events is isomorphic to M_D, whereas for an MSC M, D_M is isomorphic to M_{D_M}.*

Proof: The identity function on the set of events would be the required isomorphism from a Lamport diagram D to its corresponding MSC M_D and also from D_M to M_{D_M} as defined above. □

7.2 Collections of MSCs

A single MSC just describes one partially-ordered behaviour of the underlying system. The ITU standard [ITU97] also prescribes ways for representing collections of MSCs to express a set of behaviours of a system. The most fundamental model among such representations is the model of message sequence graphs. In addition to those prescribed by the standard, various other models have been proposed to represent collection of MSCs. The notion of regular MSC languages was introduced in [HMKT00a] as a class of languages suited for specifying and verifying collections of MSCs. In [GMP01], compositional message sequence graphs were introduced as a generalization of message sequence graphs. We introduce all these models in this section and keeping in mind the relationship between MSCs and Lamport diagrams, we also compare some of these models with relevant classes of layered Lamport diagrams.

7.2.1 Message Sequence Graphs

According to the ITU standard ([ITU97]), collections of MSCs can be described using Message Sequence Graphs (MSGs) which are also sometimes known as high-level MSCs. MSGs allow MSCs to be combined using the operations of choice, concatenation and repetition. An MSG is a finite graph with designated sets of initial and final vertices and is usually presented as a finite state automaton. The vertices are labelled by MSCs and the language of finite MSCs represented by an

MSG is obtained by **asynchronously concatenating** the MSCs labelling the paths of the MSG starting from an initial vertex and ending in one of the final vertices.

We will first define concatenation of two MSCs. Concatenation of two MSCs is done by asynchronously concatenating the events of each agent in the two MSCs.

Definition 7.2.1 *Let $M_k = (E_k, \{\leq_i^k\}_{i\in[n]}, \ell_k, \eta_k)$, $k = 1, 2$ be two MSCs with $E_1 \cap E_2 = \emptyset$. The* **concatenation** *of M_1 and M_2 is the MSC M denoted by $M = M_1 \bullet M_2$*[1] *and defined as $M = (E, \{\leq_i\}_{i\in[n]}, \ell, \eta)$ where*

- $E = E_1 \cup E_2$,
- *for* $e \in E$, $\ell(e) = \begin{cases} \ell_1(e) & e \in E_1 \\ \ell_2(e) & e \in E_2 \end{cases}$,
- *for* $e \in E$, $\eta(e) = \begin{cases} \eta_1(e) & e \in E_1 \\ \eta_2(e) & e \in E_2 \end{cases}$ *and*
- *for* $i \in [n]$, $\leq_i = \leq_i^1 \cup \leq_i^2 \cup \{(e_1, e_2) \mid e_1 \in E_1 \cap E_i, e_2 \in E_2 \cap E_i\}$.

Note that concatenation is associative. Figure 7.3 shows two MSCs M_1 and M_2 and their concatenated MSC $M_1 \bullet M_2$. Asynchronous concatenation implies that a particular agent can proceed with the execution of events in the second MSC while another agent is still executing events of the first MSC. Notice that this operation is similar to the concatenation of finite Lamport diagrams (that occur as layers of an LLD) defined in Chapter 2.

We now fix some notation for describing automata in order to define MSGs. Let A be a finite alphabet. A deterministic finite automaton (DFA) over A is a tuple $\mathcal{A} = (S, s_{in}, \delta, F)$ where S is a finite set of states, $s_{in} \in S$ is the initial state, $\delta : S \times A \to S$ is the transition function and $F \subseteq S$ is the set of final states. δ can be extended to $\delta' : S \times A^* \to S$ by defining $\delta'(s, \epsilon) = s$ and $\delta'(s, x.d) = \delta(\delta'(s, x), d)$ where $x \in A^*$ and $d \in A$. We say that a word $x \in A^*$ is accepted by the automaton $\mathcal{A}$ if $\delta'(s_{in}, x) \in F$. The language of $\mathcal{A}$, $L(\mathcal{A})$ is the set of words in A^* accepted by $\mathcal{A}$.

We are now ready to define MSGs. Let us fix a finite set of MSCs $\mathcal{M}$ over $[n]$. We will refer to the MSCs in $\mathcal{M}$ as *atomic* MSCs.

[1]Note that we use the same notation for concatenation of Lamport diagrams in Chapter 1

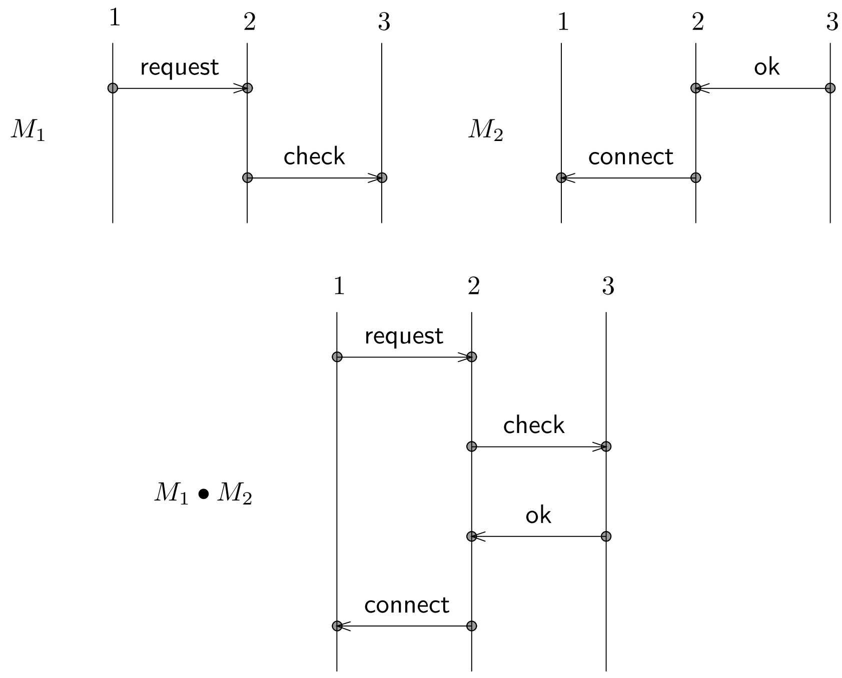


Figure 7.3: Concatenation of two MSCs

Definition 7.2.2 *A* **Message Sequence Graph** *(MSG) is a tuple* $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ *where* Π *is a finite alphabet,* $\mathcal{M}$ *is the finite set of MSCs fixed above,* $h : \Pi \rightarrow \mathcal{M}$ *is a bijection that identifies an MSC* $M \in \mathcal{M}$ *with each symbol in* Π *and* $\mathcal{A}$ *is a DFA over* Π*.*

For $x = d_0 d_1 \ldots d_k \in \Pi^*$, let $msc(x) = h(d_0) \bullet h(d_1) \bullet \ldots \bullet h(d_k)$. The language of MSCs represented by the MSG G is denoted by $L(G)$ and is defined as $L(G) = \{msc(x) \mid x \in L(\mathcal{A})\}$.

For example, Figure 7.4 gives the MSG corresponding to the producer-consumer problem where the producer p sends the message a continuously to the consumer c. The figure shows the MSG and a typical MSC represented by it.

MSGs and communication closed and bounded LLDs

MSCs represented by an MSC are obtained by concatenating a fixed set of atomic MSCs. This gives rise to a natural notion of “layering” where the events of each atomic MSC constitute one “layer”. For example, the MSC corresponding to the producer-consumer problem can be layered with each $h(d)$ constituting one layer. In the previous section, we had established a precise correspondence between MSCs

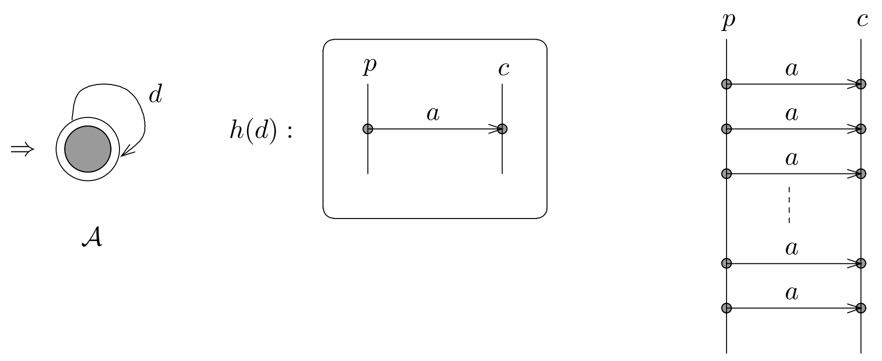


Figure 7.4: A message sequence graph

and Lamport diagrams. Along similar lines, we compare Lamport diagrams corresponding to MSCs represented by an MSG with classes of layered Lamport diagrams. Since we consider only finite MSCs represented by an MSG, the corresponding Lamport diagrams (as defined in Section 7.1.1) are also finite. We show that Lamport diagrams corresponding to the MSCs represented by a given MSG form a communication closed and bounded collection of finite LLDs. Note that the notion of layering holds for finite Lamport diagrams too and so, we can work with finite LLDs. However, it is not always possible to pull out a set of finite layered Lamport diagrams from the MSCs generated by a given MSG. It is because MSCs labelling the states of an MSG need not have at least one event per agent, but, the layers of an LLD defined in Chapter 1 have to include at least one event from every agent. So, we consider MSGs with the requirement that all the MSCs labelling the states of the underlying automaton include at least one from every agent.

Proposition 7.2.3 *Given an MSG $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ where every $M = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta) \in \mathcal{M}$ is such that $E_i \neq \emptyset$ for all $i \in [n]$, the set of Lamport diagrams defined by $\mathcal{L}_G = \{D_M \mid M \in L(G)\}$ is a collection of communication closed and bounded LLDs over a finite set of events.*

Proof: Consider $M \in L(G)$. We know that $M = msc(x)$ for some $x \in L(\mathcal{A})$. Let $x = d_0 d_1 \ldots d_k$ and let $msc(x) = M_0 \bullet M_1 \bullet \ldots \bullet M_k$ where $M_i = h(d_i)$ for $0 \leq i \leq k$. For M as above, consider the associated Lamport diagram D_M as defined in Section 7.1.1. D_M is a finite Lamport diagram and it can be layered as follows: For an event e of D_M, define $\lambda(e) = i$ iff e is an event of M_i. We now show that λ defines a layering of $D(M)$. From the assumption that each $M \in \mathcal{M}$ has at least one event from every agent, it follows that every layer includes at least one event from every agent. Consider $e, e' \in D_M$ such that $e \leq e'$. Then, $e \in M_i$ and $e' \in M_j$ for some i, j such that $i \leq_{\mathbb{N}} j$ and so $\lambda(e) \leq_{\mathbb{N}} \lambda(e')$. Since each M_i is a finite MSC, every layer includes only finitely many events.

We now claim that $\{D_M \mid M \in L(G)\}$ is a collection of communication closed and bounded (finite) LLDs. Consider D_M as above and events e, e' in D_M such that $e <_c e'$. Then, $e, e' \in M_i$ for some i and so $\lambda(e) = \lambda(e') = i$. Hence D_M is a communication closed LLD. Also, D_M is b-bounded where $b = max\{|M| \mid M \in \mathcal{M}\}$ ($|M|$ denotes the number of events in M). □

On the other hand, not every collection of communication closed and bounded (finite) LLDs can be represented by an MSG. For example, consider the Lamport diagram $h(d)$ given in Figure 7.4. For $k \in \mathbb{N}$, let $h(d)^k = \underbrace{h(d) \bullet h(d) \bullet \ldots \bullet h(d)}_{k \text{ times}}$. Then, the language L given by $L = \{h(d)^p \mid p \text{ is a prime number }\}$ is a collection of communication closed and bounded LLDs, but it cannot be represented by an MSG as the underlying language $\{d^p \mid p \text{ is a prime number }\}$ is not a regular language over the finite alphabet $\{d\}$ and hence cannot be accepted by any finite state automaton.

7.2.2 Regular MSC languages

In this section, we present regular MSC languages which were introduced in [HMKT00b] and recall some results on them. A collection of MSCs comprising a regular MSC language can be represented using a class of distributed automata. We will introduce compositional MSGs as yet another model to represent collections of MSCs in the next section and show that they subsume regular MSC languages and MSGs. We start with defining linearizations of MSCs as regular MSC languages will be defined using linearizations.

Linearizations of MSCs

Since an MSC is a partial order over a set of events, another natural way of representing an MSC is by considering linearizations of the underlying partial order.

Consider an MSC $M = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta)$. An event linearization of M is a linear order on E which extends $\leq$, the partial order of M. We represent event linearizations as sequences over E: a sequence $e_1, \ldots, e_k$ represents the event linearization

$\sqsubseteq$ given by $e_i \sqsubseteq e_j$ iff $i \leq j$, for $1 \leq_{\mathbb{N}} i, j \leq_{\mathbb{N}} k$.

A linearization of an MSC M is any possible sequence of actions which the MSC describes. Formally, $lin(M)$, the set of linearizations of M, is the set of all sequences $\ell(e_1), \ldots, \ell(e_k)$ where $e_1, \ldots, e_k$ is an event linearization of M. Linearizations are hence represented as words in Σ^*. For a collection L of MSCs, let $lin(L) = \{lin(M) \mid M \in L\}$.

While an MSC defines a non-empty set of linearizations, one can associate a unique MSC (upto isomorphism) with a given linearization by defining an appropriate set of events and matching the sends and receives using the fact that the MSC is non-degenerate. For a linearization $w \in \Sigma^*$, we can define the MSC corresponding to w as $M_w = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta)$, where

- E is the set of all non-epsilon prefixes of w,
- $\ell(x.r) = r$, where $x.r \in E$, $x \in \Sigma^*$, $r \in \Sigma$,
- for $i \in [n]$, $x.r \leq_i y.r'$ iff $x.r$ is a prefix of $y.r'$ and $r, r' \in \Sigma_i$ and
- $\eta(x.r) = y.r'$ iff $r = (i!j, a)$, $r' = (j?i, a)$ for some $i, j \in [n]$ and the number of occurrences of r in x is equal to the number of occurrences of r' in y.

We now identify words in Σ^* which correspond to linearizations of MSCs. Consider $w \in \Sigma^*$. If for every $i, j \in [n]$, $a \in \Gamma_m$, and every prefix y of w, the number of occurrences of $(i!j, a)$ in y is at most the number of occurrences of $(j?i, a)$ in y, and the number of occurrences of $(i!j, a)$ in w is the same as the number of occurrences of $(j?i, a)$ in w, then one can associate an MSC M (as above) for which w is a linearization. We call such words well-formed words.

Let $w \in \Sigma^*$ be a well-formed word and $b \in \mathbb{N}$. w is said to be b-bounded if for every prefix x of w, the difference between the number of send events in x of the type $(i!j, a)$ and the number of receive events of the type $(j?i, a)$ is at most b. A language $L \subseteq \Sigma^*$ of well-formed words is said to be b-bounded if each word in L is b-bounded.

Now, there are two ways of representing collections of MSCs. One way is to represent a collection of MSCs through a set of representative linearizations as follows. Let $L \subseteq \Sigma^*$ be a set of well-formed words. Then L represents the class of MSCs $msc(L) = \{M_w \mid w \in L\}$. Note that we do not require L to have *all* the linearizations of the MSCs it represents, i.e. L need not be equal to $lin(msc(L))$. L is said to be a representative linearization of a collection $\mathcal{L}$ of MSCs if $\mathcal{L} = \{M_w \mid w \in L\}$.

On the other hand, we can represent a collection L of MSCs by taking *all* the linearizations of each MSC in the collection (as done in [HMKT00b]). That is, L can be represented by $lin(L)$.

Definition 7.2.4 *A collection L of MSCs over $[n]$ is said to be a* regular MSC language *if $lin(L)$ is a regular subset of Σ^*.*

Message Passing Automata

In [HMKT00b],an automaton model for regular MSC languages is presented in terms of bounded Message Passing Automata (MPA). We briefly describe MPAs and state relevant results that we will be using later. An MPA, like an SCA, is a collection of local automata, one for every agent in $[n]$. These automata communicate with each other by exchanging messages over FIFO channels. The global automaton corresponding to an MPA is constructed by taking products of local automata along with buffers which store actions corresponding to send events. Transitions are defined by using local transitions of the corresponding automata and a transition on a send (receive) event adds (removes) the corresponding action from the buffer. A run of an MPA on a linearization of an MSC is defined by a run of its global automaton. Some linearizations give rise to unboundedly many messages being stored in buffers and MPAs accept regular languages when buffers are bounded. In [HMKT00b], it is also shown that b-bounded regular MSC languages are precisely those accepted by b-bounded MPAs (i.e., MPAs where the size of the buffers is bounded by b).

MPAs are very similar to SCAs—transitions of component automata in MPAs add or remove messages from buffers depending on the actions (send or receive) and messages in MPAs are taken from a message alphabet. On the other hand, contents of local states of component automata in SCAs constitute the messages exchanged and they are added/removed from buffers as dictated by the λ-transitions.

MSGs and regular MSC languages

In another paper ([HMKT00a]), it is shown that the class of MSC languages represented by an MSG is incomparable with the class of regular MSC languages. They go ahead and characterize the class of regular MSC languages which can be represented by MSGs. It turns out that the class of *finitely generated* regular MSC languages is precisely the class of languages defined by a restricted class of MSGs called *locally synchronized* MSGs (which were also studied in [AY99] and [MP99]). Finitely generated regular MSC languages are those which can be obtained by the operations of union, asynchronous concatenation and its corresponding closure over a fixed finite set of atomic MSCs.

7.3 Compositional Message Sequence Graphs

A weakness of the MSG model was demonstrated in [GMP01], where the authors present the model of Compositional Message Sequence Graphs (CMSGs) as a solution. They illustrate how some standard protocols like the alternating bit protocol cannot be represented by an MSG. A CMSG is a finite graph like an MSG but, the vertices of this graph are labelled by Compositional Message Sequence Charts (CMSCs). A CMSC is basically an MSC which, in addition to send and receive events in an MSC, can also have unmatched send and receive events. These are special send (receive) events which do not have corresponding receive (send) events in the same CMSC and they will be matched up later using corresponding unmatched receive and send events respectively, when the CMSC is composed with another. The collection of MSCs represented by an CMSG again consists of all the MSCs obtained by tracing a path in the CMSG from an initial vertex to a terminal vertex and concatenating the CMSCs that are encountered along the path. While concatenating, we are required to match up the unmatched send and receive events to generate an MSC.

Definition 7.3.1 *A* **Compositional Message Sequence Chart** *(CMSC) over* $[n]$ *is a tuple* $M = (E, \{\leq_i\}_{i\in[n]}, \ell, \eta)$ *where* E, $\leq_i$ *and* ℓ *are as in an MSC and* $\eta : S_E \rightharpoonup R_E$ *is a* partial *matching function which associates with some send events, their corresponding receive events. We require the function* η *to be injective and for every* $e, e' \in E$, *if* $\eta(e) = e'$, *then* $\lambda(e) = (i!j, a)$ *and* $\lambda(e') = (j?i, a)$ *for some* $i, j \in [n], a \in \Gamma_m$.

We also require that the CMSC be non-degenerate with respect to the messages matched by the matching function. We say that $e \in S_E$ is an unmatched send event if η is not defined on e and $e \in R_E$ is an unmatched receive event if $e \notin \eta(S_E)$. Hence a CMSC is like an MSC but, it could have unmatched send and receive events. A CMSC without any unmatched send and receive events is just an MSC.

We say that a CMSC has no unmatched sends if η is a function on S_E and we say that it has no unmatched receives if η is surjective. For a CMSC M and $(i!j,a) \in \Sigma$, let

$$\#^{unm}_{(i!j,a)}(M) = |\{e \in E \mid \ell(e) = (i!j,a) \text{ and } \eta(e) \text{ is not defined }\}|$$

denote the number of unmatched sends of the type $(i!j,a)$.

We now present the model of a Compositional Message Sequence Graph (CMSG) introduced in [GMP01]. CMSGs are basically finite state automata like MSGs but, the actions here are mapped to CMSCs. We consider accepting paths in the underlying automaton of the CMSG and concatenate the CMSCs labelling adjacent vertices in a path.

Concatenation of two CMSCs is done by (asynchronously) concatenating the events of each agent, and by matching up some unmatched sends of the resulting CMSC with corresponding unmatched receives.

Definition 7.3.2 *Let $M_k = (E_k, \{\leq_i^k\}_{i\in[n]}, \ell_k, \eta_k)$, $k = 1,2$ be two CMSCs, with $E_1 \cap E_2 = \emptyset$. Also, let M_1 have no unmatched receives. Then the* **concatenation** *of M_1 and M_2 is $M = M_1 \circ M_2$*[2] *$= (E, \{\leq_i\}_{i\in[n]}, \ell, \eta)$ where*

- *$E = E_1 \cup E_2$,*
- *for $e \in E$, $\ell(e) = \begin{cases} \ell_1(e) & e \in E_1 \\ \ell_2(e) & e \in E_2 \end{cases}$,*
- *for $i \in [n]$, $\leq_i = \leq_i^1 \cup \leq_i^2 \cup \{(e_1, e_2) \mid e_1 \in E_1 \cap E_i, e_2 \in E_2 \cap E_i\}$ and*
- *η is defined as follows:*

  (C1) *If $e \in S_{E_k}$ and $\eta_k(e)$ is defined, where $k \in \{1,2\}$, then $\eta(e) = \eta_k(e)$.*

  (C2) *Let $e \in S_E$, η_1 and η_2 be undefined on e, and $\ell(e) = (i!j,a)$. If there is an event $e' \in R_{E_2}$ such that η_2 is undefined on e', $\ell(e') = (j?i,a)$ and $|\{f \in E \mid f \leq_i e, \lambda(f) = (i!j,a)\}| = |\{f' \in E \mid f' \leq_i e', \lambda(f') = (j?i,a)\}|$, then we set $\eta(e) = e'$. Clearly, if such an e' exists, it is unique.*

  (C3) *η is undefined on all other events of S_E.*

We require that the CMSC M be non-degenerate — otherwise concatenation is not defined.

[2] Note that we use the same notation for concatenation of fragments in Chapter 1.

When we write a series of concatenations $M_1 \circ M_2 \circ \ldots$ we always mean a left-to-right application, i.e. the term $((M_1 \circ M_2) \circ M_3) \ldots$. Note that concatenation is not associative and is hence sensitive to the order in which it is made. The example given in Figure 2.5 in Chapter 1 also applies here as the Lamport diagrams given there are basically MSCs (with labels appropriately defined).

We are now ready to define CMSGs. Let us fix a finite set of *atomic* CMSCs $\mathcal{M}$ for the rest of this section.

We say that a sequence of CMSCs $M_1, M_2, \ldots M_k$ from $\mathcal{M}$ is well-defined iff the concatenation of the CMSCs $M_1 \circ M_2 \circ \ldots \circ M_k$ is defined. We say that it is complete if the concatenated CMSC has no unmatched send or receive events — i.e. it is an MSC.

Let Π be a finite alphabet and $h : \Pi \rightarrow \mathcal{M}$ be a bijection. Given a word $x = d_0 d_1 \ldots d_k \in \Pi^*$, if $h(d_0), h(d_1), \ldots, h(d_k)$ is well-defined, then x defines the CMSC $cmsc(x) = h(d_0) \circ h(d_1) \circ \ldots \circ h(d_k)$. We say that x is well-defined if $h(d_0), h(d_1), \ldots, h(d_k)$ is well-defined and say x is complete if $cmsc(x)$ is an MSC.

Definition 7.3.3 *A* **compositional message sequence graph** *(CMSG) is a tuple $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ where Π is a finite alphabet, $\mathcal{M}$ is a finite set of CMSCs, $h : \Pi \rightarrow \mathcal{M}$ is a bijection, and $\mathcal{A}$ is a DFA over Π.*

We require that every $x \in L(\mathcal{A})$ is well-defined and complete. The CMSG represents *the set of MSCs $msc(G) = \{cmsc(x) \mid x \in L(\mathcal{A})\}$.*

For example, consider the CMSG in Figure 7.5. It depicts a variant of the producer-consumer example where the producer p sends messages continuously to the consumer c. At some point, c sends the "abort" signal to p requesting it to stop sending messages but this message takes an arbitrarily long time to reach p. The figure shows a typical behaviour and a CMSG which represents such scenarios.

MSGs are just like CMSGs except that the atomic CMSCs are in fact MSCs. By definition, the class of CMSGs extend that of MSGs — in fact, the extension is strict. The behaviour illustrated in Figure 7.5 cannot be represented by any MSG as the set of MSCs generated by this CMSG have unbounded continuous sequences of the message $(p!q, a)$ in between the send event of *abort* and its corresponding receive. Consequently, the underlying set of atomic MSCs also has to be an infinite set.

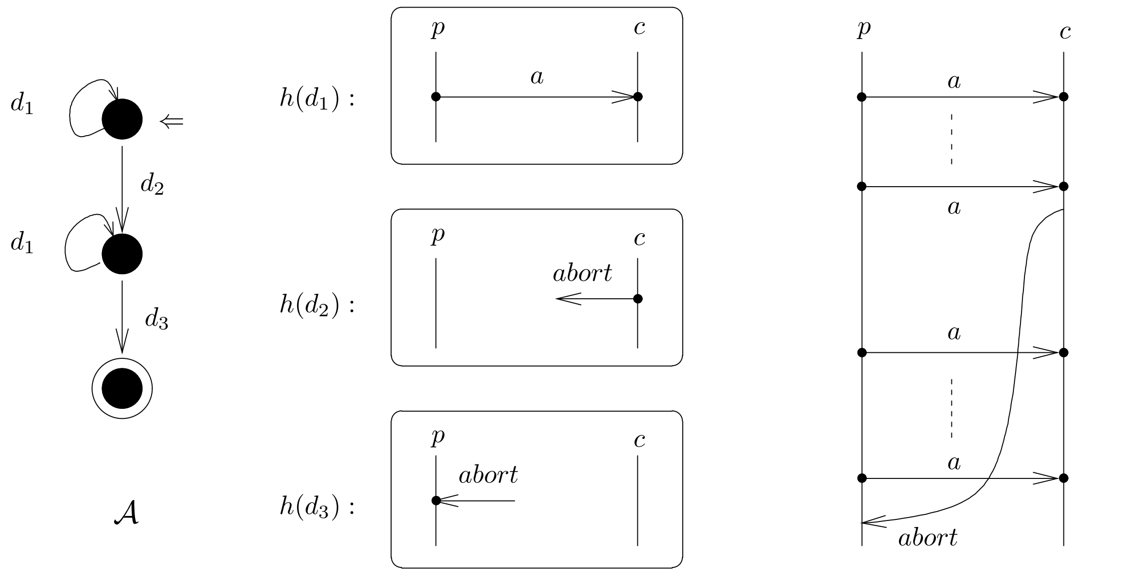


Figure 7.5: A compositional message sequence graph

CMSGs and channel bounded LLDs

MSCs generated by CMSGs can also be "layered" naturally where a layer comprises of all the events of the CMSC labelling a particular vertex, i.e., a layer is an atomic CMSC. As done in Section 7.2.1 for MSGs, we compare the language of Lamport diagrams corresponding to MSCs generated by CMSGs with the class of channel bounded LLDs. Again, keeping in mind the fact that a layering has to include at least one event per agent, we work with CMSGs whose underlying CMSCs have at least one event per agent. It turns out that CMSGs represent channel bounded LLDs.

We know that a CMSC can have unmatched send and receive events. While concatenating the CMSCs labeling a path in the underlying automaton of an MSC, a few send events might remain unmatched in the concatenated MSC. We can define a notion of buffer which stores these unmatched events. It turns out that the size of such a buffer in every CMSG is bounded as we assume that CMSGs generate only finite MSCs, and hence we can relate the MSCs generated by a CMSG to channel bounded LLDs over a finite set of events. We formalise this notion below.

Definition 7.3.4 *A CMSC $M = (E, \{\leq_i\}_{i\in[n]}, \ell, \eta)$ is said to be b-***memory bounded***, where $b \in \mathbb{N}$, if the number of unmatched sends of any type is at most b, i.e.,*

$\#^{unm}_{(i!j,a)}(M) \leq b$ for every $(i!j, a) \in \Sigma$.

A sequence of CMSCs $M_1, M_2, \ldots, M_k$ is b-memory bounded if for all its prefixes $M_1, M_2, \ldots, M_l$, the CMSC $M_1 \circ M_2 \ldots \circ M_l$ is b-memory bounded, where $l \leq k$.

A CMSG $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ is b-memory bounded if for every $d_1 d_2 \ldots d_k \in L(\mathcal{A})$, the sequence of CMSCs $h(d_1), h(d_2), \ldots, h(d_k)$ is b-memory bounded.

The fact that a CMSG defines a collection of finite MSCs forces it to be b-memory bounded as every send event has to be matched up with a receive event within the finite MSC.

Proposition 7.3.5 *For every CMSG G, there exists $b \in \mathbb{N}$ such that G is b-memory bounded.*

Proof: Let $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ be a CMSG and let $\mathcal{A} = (S, s_{in}, \delta, F)$. We say that a state $s \in S$ is live if there exist words $w, w' \in \Pi^*$ such that $\delta'(s_{in}, w) = s$ and $\delta'(s, w') \in F$. With each live state $s \in S$, we associate a memory-capacity function $f_s : \{(i!j, a) \in \Sigma\} \to \mathbb{N}$ is such that for a word $w = d_1 \ldots d_n$, if $\delta'(s_{in}, w) = s$ and $M = cmsc(w) = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta)$ then, for every $(i!j, a) \in \Sigma$, $f_s((i!j, a)) = \#^{unm}_{(i!j,a)}(M)$. That is, f_s gives the number of unmatched send events of each type in any CMSC obtained when tracing a path from the initial state to s.

We first show that each f_s is well-defined, i.e., for every $w, w' \in \Pi^*$ such that $\delta'(s_{in}, w) = s = \delta'(s_{in}, w')$, we first show that $\#^{unm}_{(i!j,a)}(cmsc(w)) = \#^{unm}_{(i!j,a)}(cmsc(w'))$. Since s is live, let $w'' \in \Pi^*$ be such that $\delta'(s, w'') \in F$. Since $ww'', w'w'' \in L(\mathcal{A})$ and every word in $L(\mathcal{A})$ is complete, it must be the case that $\#^{unm}_{(i!j,a)}(cmsc(w)) = \#^{unm}_{(i!j,a)}(cmsc(w'))$

We can now define $f_s((i!j, a)) = \#^{unm}_{(i!j,a)}(cmsc(w))$ for some w such that $\delta'(s_{in}, w) = s$. We choose b to be the maximum value of $f_s(r)$ where s is live and $r = (i!j, a) \in \Sigma$. G is b-memory bounded by the choice of b. $\Box$

We are now ready to relate CMSGs with channel bounded LLDs.

Proposition 7.3.6 *Given a CMSG $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ where every CMSC $M = (E, \{\leq_i\}_{i \in [n]}, \ell, \eta) \in \mathcal{M}$ is such that $E_i \neq \emptyset$ for all $i \in [n]$, the set of Lamport diagrams defined by $\mathcal{L}_G \overset{\text{def}}{=} \{D_M \mid M \in L(G)\}$ is a channel bounded collection of finite layered Lamport diagrams.*

Proof: We know from Proposition 7.3.5 above that there exists $b' \in \mathbb{N}$ such that G is b'-memory bounded. Let $b'' = max\{|M_i| \mid M_i \in \mathcal{M}\}$ ($|M|$ denotes the number of events in M).

For an MSC $M \in L(G)$, consider the associated Lamport diagram D_M as defined in Section 7.1.1. We can layer D_M as done in the proof of Proposition 7.2.3 and show that $\mathcal{L}_G = \{D_M \mid M \in L(G)\}$ defines a collection of finite layered Lamport diagrams.

We now claim that $\mathcal{L}_G$ is channel b-bounded where $b = max\{b', b''\}$. Every D_M in $\mathcal{L}_G$ is b-bounded as every layer in D_M includes at most b'' events. To show that $\mathcal{L}_G$ is channel b-bounded, consider $D_M \in \mathcal{L}_G$ and a prefix ν' of ν_{D_M}. We have to show that $\delta_{\nu'}(i,j) \leq_{\mathbb{N}} b$ (where $\delta_{\nu'}(i,j)$ is as given in Definition 2.2.5). Let $\nu_{D_M} = 1, 2, \ldots, k$ and $\nu' = 1, 2, \ldots, l$. Then, $\delta_{\nu'}(i,j) = |M_1 \circ M_2 \circ \ldots \circ M_l|$ and since G is b'-memory bounded, we have $\delta_{\nu'}(i,j) \leq_{\mathbb{N}} b$. □

7.3.1 CMSGs and regular representative linearizations

We now show that every MSC-language represented by a CMSG is definable using a regular language of representative linearizations and conversely, every regular language of representative linearizations is represented by a CMSG. This would help us to conclude that MSC languages represented by CMSGs subsume regular MSC languages.

Theorem 7.3.7 *Let $\mathcal{L}$ be a collection of MSCs over $[n]$. There exists a CMSG $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ that represents $\mathcal{L}$ iff there exists a regular set $L \subseteq \Sigma^*$ of well-formed words which is a representative linearization of $\mathcal{L}$.*

Proof: Let $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ be a CMSG such that $msc(G) = \mathcal{L}$. For every $M \in \mathcal{M}$, let us fix one linearization of M and call it $lin(M)$. Let $L = \{lin(h(d_1)) \ldots lin(h(d_k)) \mid d_1 \ldots d_k \in L(\mathcal{A})\}$. Clearly $lin(h(d_1)) \ldots lin(h(d_k))$ respects the $\leq_i$ orderings of the concatenated MSC $h(d_1) \circ \ldots \circ h(d_k)$ (as we finish with the events of an atomic CMSC before moving to the next). It also respects the ordering defined by η — the send-events are always ordered before the corresponding receive events since at any point the definition of concatenation requires that the CMSC formed till then has no unmatched receives. It follows from the definition of L that it is a representative linearization of $\mathcal{L}$. L is regular as it is obtained by

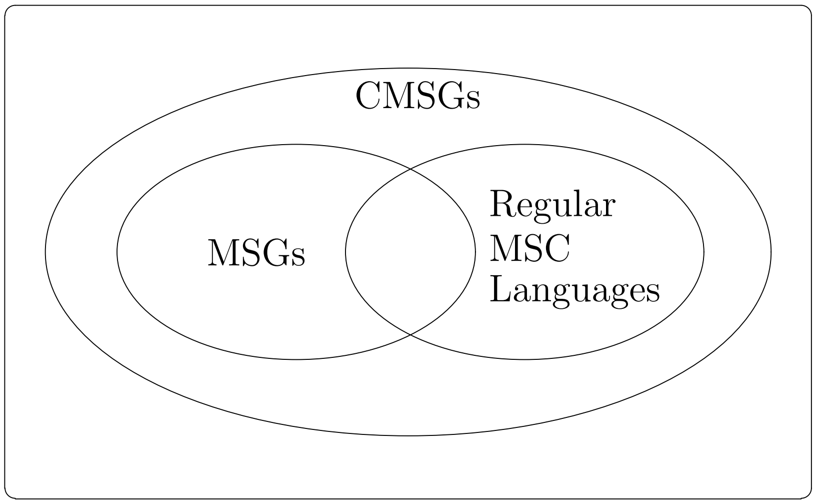


Figure 7.6: Classes of MSC languages

a homomorphism of the regular language $L(\mathcal{A})$ where each letter d is replaced by $lin(h(d))$ (and regular languages are closed under homomorphism [HU79]).

The other direction follows from [GMP01]. Let $L \subseteq \Sigma^*$ be a representative linearization of $\mathcal{L}$ and L be regular. Take a DFA $\mathcal{B} = (S, s_{in}, \delta, F)$ over Σ which accepts L. We now define a CMSG $G = (Pi, \mathcal{M}, h, \mathcal{A})$ such that $\mathcal{L} = msc(G)$. For every $r \in \Sigma_j$, let $\mathcal{M}$ have the CMSC $M_r = (\{e_r\}, \{\leq_i^r\}_{i\in[n]}, \ell_r, \eta_r)$ where

- $\leq_j^r = \{(e_r, e_r)\}$, $\leq_i^r = \emptyset$ for each $i \in [n] \setminus \{j\}$,
- $\ell_r(e_r) = r$, and
- η_r is not defined on e_r.

Let Π be an alphabet and $h : \Pi \to \mathcal{M}$ be a bijection. Let $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ where $\mathcal{A} = (S, s_{in}, \widehat{\delta}, F)$ is a DFA on Π with $\widehat{\delta}(s, d) = \delta(s, r)$ where $h(d) = M_r$. It is easy to see that G is a CMSG. Now, if $w = r_1 \ldots r_k \in L$, then $msc(G)$ has the MSC $M_{r_1} \circ \ldots \circ M_{r_k}$, which is clearly $msc(w)$. Also, for any MSC $M = M_{r_1} \circ \ldots \circ M_{r_k}$ in $msc(G)$, it is clear that $w = r_1 \ldots r_k \in L$ and w is a linearization of M — hence $M \in msc(L)$. Therefore, $\mathcal{L} = msc(G)$. □

Since a regular MSC-language L (as defined in Section 7.2.2) is such that all the linearizations of all the MSCs in L form a regular set, it follows from the above theorem that MSC languages represented by CMSGs subsume the class of regular

MSC-languages as well. Figure 7.6 summarizes the relationship between MSGs, regular MSC languages and CMSGs (we again note the fact that MSGs and regular MSC languages are incomparable classes of MSC languages [HMKT00a]).

7.4 MSO over MSCs

We use monadic second order logic (MSO) as the specification language to describe properties of MSCs. Various results regarding model checking classes of MSC languages against MSO specifications are known. We will introduce MSO over MSCs in this section and also provide brief details regarding the decidable problems involving model checking classes of MSCs against MSO specifications. We will finally show that model checking CMSGs against MSO specifications is decidable in the next section.

From the previous sections, it is clear that there are two ways of representing MSCs—as graphs on events where the underlying relation is a partial order or as the set of all/some linearizations of the underlying partial order. Accordingly, the MSO can be defined on the graph representing an MSC or on set of words which occur as linearizations of MSCs respectively.

We define MSO on the partially-ordered structure of the MSC. We have at our disposal a countable number of first-order variables $\{x, y, z, \ldots\}$ and second-order variables $\{X, Y, Z, \ldots\}$. The atomic formulas are of the kind $(x \rightarrow y)$, $(x \leq_i y)$ (for each $i \in [n]$), $Q_r(x)$ (for $r \in \Sigma$) and $(x \in X)$. Other formulas are formed using the boolean connectives $\vee$ and $\neg$ and using quantification over first-order and second-order variables. Notice that the MSO is parametrized by $[n]$ and Σ. We fix Σ and $[n]$ for the rest of the chapter and consider MSCs over $[n]$ agent systems whose actions are labelled from Σ.

Let φ be a formula. φ is interpreted over an MSC M as follows: Let $M = (E, \{\leq_i\}_{i\in[n]}, \lambda, \eta)$. An interpretation I of a set of first-order and second-order variables V is a function that assigns to each first-order variable in V, an event $e \in E$ and assigns a set of events to each second-order variable in V. For φ in MSO, let V_φ denote the set of variables which occur free in φ. In the following, given an interpretation I, we will use the notation $I[e/x]$ to denote an interpretation allots an event e to the variable x and is the same as I for every variable apart from x. Similarly, $I[E'/X]$ denotes an interpretation that allots a set of events E' to X and coincides with I on every other variable. We define the notion of when an MSC M satisfies φ under an interpretation I of V_φ, which we denote by $M \models_I \varphi$, inductively as follows:

- $M \models_I (x \rightarrow y)$ iff $I(x) \in S_E$ and $\eta(I(x)) = I(y)$
- $M \models_I (x \leq_i y)$ iff $I(x), I(y) \in E_i$ and $I(x) \leq_i I(y)$
- $M \models_I Q_r(x)$ iff $\lambda(I(x)) = r$
- $M \models_I (x \in X)$ iff $I(x) \in I(X)$
- $M \models_I \varphi \vee \psi$ iff $M \models_I \varphi$ or $M \models_I \psi$
- $M \models_I \neg\varphi$ iff $M \not\models_I \varphi$
- $M \models_I \exists x \varphi(x)$ iff there exists an event e such that $M \models_{I[e/x]} \varphi$
- $M \models_I \exists X \varphi(X)$ iff there exists a set of events E such that $M \models_{I[E/X]} \varphi$.

Hence $(x \rightarrow y)$ means that x is a send-event matched with the receive-event y, $x \leq_i y$ says that the events x and y are causally ordered in the event-sequence of agent i and $Q_r(x)$ says that the label of event assigned to x is r.

As usual, φ is a sentence if there are no free first-order and second-order variables in φ. Given an MSO sentence φ, the language of MSCs defined by φ is denoted by L_φ and is defined as $L_\varphi = \{M \mid M \models \varphi\}$. We say that an MSC language L is MSO-definable iff there exists an MSO sentence φ such that $L = L_\varphi$.

Model checking regular MSC languages against MSO specifications

In [HMKT00b], the authors provide an MSO characterization of regular MSC languages.

Theorem 7.4.1 ([HMKT00b]) *A language of MSCs L over $[n]$ is regular iff L is MSO-definable.*

This theorem is proved by using MSO defined on words over Σ and using the fact that a language over Σ is regular iff it is definable in this MSO ([Tho90, Tho97]). Given an MSO sentence φ, they show that L_φ is a regular MSC language by constructing a sentence φ_{lin} in MSO over words such that $lin(L_\varphi) = \{w \in \Sigma^* \mid$

$w \models \varphi_{lin}\}$. Conversely, given a regular MSC language L, using the fact that $lin(L)$ is a regular language of strings over Σ, we know that by Büchi's theorem ([Büc60, Elg61, Tho90, Tho97]) that there exists a sentence φ in MSO over words such that $L = \{w \in \Sigma^* \mid w \models \varphi\}$. They then construct a sentence $\hat{\varphi}$ in MSO over MSCs and show that an MSC M is a model of $\hat{\varphi}$ iff w is a model of φ where w is a linearization of M.

From the proof of the above theorem, it follows that the model checking problem for regular MSC languages against MSO specifications is decidable.

Model checking MSGs against MSO specifications

It is also known that the problem of checking if the set of all MSCs represented by an MSG satisfy a property given as an MSO formula is decidable.

Theorem 7.4.2 ([Mad01]) *Given an MSG G and an MSO formula φ over MSCs, the problem of checking whether all the MSCs represented by G satisfy φ is decidable.*

This proof works by viewing MSGs as automata which run over the finite alphabet of MSCs (labelling the transitions of the automaton). An interpretation of formulas is also encoded along with the MSCs into a finite alphabet. Unlike the previous theorem, MSO here is defined on the partially-ordered graph of an MSC. Given an MSO formula φ, the main idea behind the proof is to construct a finite automaton $\mathcal{A}_\varphi$ (over such an alphabet) such that $\mathcal{A}_\varphi$ accepts a word iff the MSC obtained by concatenating the MSCs labelling the letters in the word, under the interpretation defined by it, satisfies φ. Now, the set of all MSCs represented by the given MSG G satisfy φ iff the language of MSCs accepted by the underlying automaton of G is a subset of the language of MSCs accepted by $\mathcal{A}_\varphi$.

7.5 Model checking CMSGs against MSO specifications

As we have mentioned in the previous section, the problem of model checking MSGs against MSO specifications was shown to be decidable in [Mad01]. From the MSO characterization of regular MSC languages given in [HMKT00a], it follows that model checking MSO specifications against regular MSC languages is also decidable. Now, given that CMSGs subsume both these classes, it will be interesting to explore the possibility of model checking MSCs generated by a CMSG against MSO specifications.

The model checking problem for CMSGs against MSO specifications is defined as follows:

Given a CMSG G (or a representative linearization L), and a MSO sentence φ, do all the MSCs represented by G satisfy φ?

We show that this problem is decidable. We also exhibit two proofs of this theorem, one which extends the proof of decidability of model-checking MSGs [Mad01] and the other which extends the proof of decidability of model-checking for regular MSC-languages [HMKT00b].

We first present the former proof of the above result. The following lemma will be used to show that the model checking problem for CMSGs is decidable.

Lemma 7.5.1 *Let Π be a finite alphabet, $\mathcal{M}$ be a set of CMSCs and $h : \Pi \to \mathcal{M}$ be a bijection. Let φ be an MSO formula and $b \in \mathbb{N}$. Then the collection of words $w = d_1 \ldots d_n \in \Pi^*$ such that $h(d_1), \ldots h(d_n)$ is well-defined, complete and b-memory bounded, and $cmsc(w) \models \varphi$, is a regular subset of Π^*.*

Proof: The proof follows the corresponding proof for MSCs in [Mad01]. We have to do extra work for the formula $X \to Y$ due to the presence of unmatched send and receive events wherein we exploit the fact that we are working with b-memory sequences. First, we can work with MSO formulas over a restricted syntax where only second-order quantification is allowed and where we have as atomic formulas $X \subseteq Y$ (X is a subset of Y), $Singleton(X)$ (X is a singleton set), $X \leq_i Y$ and $X \to Y$ (which mean that X and Y are singletons with $X = \{x\}, Y = \{y\}$ and $x \leq_i y$ or $x \to y$ respectively) and $Q_r(X)$ (which is true if $X = \{x\}$ and $Q_r(x)$). It is easy to see that this syntax is exactly as expressive as the original one [Tho90].

Let φ be an MSO formula with free second-order variables $V_\varphi = \{X_1, \ldots X_k\}$. We augment the alphabet Π to get a new alphabet Π_φ which has letters of the form (d, I) where $d \in \Pi$ and $I : V_\varphi \to 2^E$ (where E is the set of events of $h(d)$) is an interpretation of the free variables over the events of the CMSC $h(d)$ corresponding to d. The interpretation I can be encoded as follows: Let $h(d)$ have m events and without loss of generality, we can assume that these events are ordered. Then, a matrix say, Z with 0 or 1 entries and with k rows and m columns can represent I—the j^{th} event of $h(d)$ belongs to the i^{th} variable iff the entry on the i^{th} row and j^{th} column in Z is 1.

The idea is to construct an automaton $\mathcal{A}_\varphi$ which will accept a word $(d_1, I_1) \dots (d_n, I_n)$ iff the sequence $h(d_1), h(d_2), \dots, h(d_n)$ is well-defined, complete and b-memory bounded, and the MSC $M = cmsc(d_1 d_2 \dots d_n)$ under the combined interpretation defined by $I_1, \dots I_n$ on the events of M, satisfies φ. This is done inductively on the structure of the formula.

First, the set of all well-defined and complete b-memory bounded sequences is regular [HMKT00b]. We run an automaton accepting this in parallel with the automaton we construct.

We now sketch the details of the inductive construction of the automaton. For the sake of readability, we just provide a textual description of the automaton instead of presenting the precise details. The various checks can be easily coded up into automata.

$Singleton(X)$ The automaton checks if the interpretation assigned to X is a singleton and rejects otherwise.

$X \subseteq Y$ The automaton checks if the interpretation assigned to X is a subset of the interpretation assigned to Y.

$Q_r(X)$ The automaton first checks if the interpretation assigned to X is a singleton and if the corresponding event is labelled by r and rejects otherwise.

$X \leq_i Y$ The automaton first checks if X and Y are singletons and if they both are events of agent i and rejects otherwise. Later, it checks if the event assigned to Y is reachable from the event assigned to X.

$X \rightarrow Y$ The atomic formula $X \rightarrow Y$ is the hardest to handle. The automaton checks if they are singletons — assume they are, with $X = \{x\}$ and $Y = \{y\}$ and let $\lambda(x) = (i!j, a)$, $\lambda(y) = (j?i, a)$ for some $i, j \in [n]$ with $i \neq j$. The automaton rejects the input string if any of the above conditions is false.

The automaton now has to check if y is the corresponding receive event of x. This is done as follows. Let the CMSC-sequence corresponding to the word the automaton is reading be $M_1, \dots M_i, \dots M_j \dots M_p$, where the interpretation of x is an event e_x in M_i, and y an event e_y in M_j, $i \leq j$ with $\lambda(e_x) = (i!j, a)$ and $\lambda(e_y) = (j?i, a)$. Since this sequence is b-memory bounded, we can associate at each point k $(k \leq p)$, a number t_k which is the number of unmatched sends of the kind $(i!j, a)$ in the CMSC $M_1 \cdot M_2 \cdot \ldots M_k$ and we know that $t_k \leq b$. Now, let s be the number of unmatched send events of the kind $(i!j, a)$ in M_i before e_x and let r be the number of unmatched receive-events of the kind $(j?i, a)$ in M_j before e_y. It is easy to see that e_y is the corresponding receive event of e_x iff the number of receive events of the kind $(j?i, a)$ in all MSCs M_k, where $i \leq k < j$ is l and $t_i + s = l + r$. Note that $t_i + s$ is bounded by some $b' \in \mathbb{N}$.

We can check this property by the following automaton. Let $r_{(j?i,a)}(M)$ denote the number of receive messages of the kind $(j?i, a)$ in M for each atomic CMSC M. Then, we can define an automaton with states u where $u \leq t_i + s - r$ (note that the range of u is bounded). From a state u, there is a transition on d to u' iff $u' = u + r_{(j?i,a)}(h(d))$. The initial state is 0 and the final state is then $(t_i + s - r)$. It is easy to see that this automaton verifies whether the sequence $M_{i+1} \ldots M_{j-1}$ is such that in the CMSC $M_1 \cdot \ldots M_p$, y is the matching receive event of x.

Other cases The formulas obtained by disjunction, negation and existential quantification can be handled by using the fact that automata are closed under union, complement and projection respectively [HU79].

Now, given a sentence φ, we can construct an automaton over Π such that the automaton accepts a word $w = d_1 d_2 \ldots d_n$ iff $h(d_1), \ldots h(d_n)$ is well-defined, complete and b-memory bounded, and $cmsc(w) \models \varphi$. Hence the proof. □

We can now solve the model checking problem for CMSGs against MSO specifications.

Theorem 7.5.2 *The model checking problem for CMSGs against MSO-formulas is decidable.*

Proof: Let $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ be a CMSG and φ be an MSO-formula. From Proposition 7.3.5, it follows that there exists $b \in \mathbb{N}$ (which can be computed) such that G is b-memory bounded. Now, for this b, using Lemma 7.5.1, we construct an automaton $\mathcal{A}^b_\varphi$ over Π^* that accepts a word $w = d_1 d_2 \ldots d_k$ iff $h(d_1), h(d_2), \ldots, h(d_k)$ is well-defined, complete and b-memory bounded, and $cmsc(w) \models \varphi$. Clearly, all MSCs in $msc(G)$ satisfy φ iff $L(\mathcal{A}) \subseteq L(\mathcal{A}^b_\varphi)$, which can be checked. □

We now turn to linearizations and show that they can also be used to solve the model checking problem. The following lemma will be used towards the proof.

Lemma 7.5.3 *Given an MSO-formula φ and $b \in \mathbb{N}$, the set of all well-formed words $w \in \Sigma^*$ such that w is b-bounded and $msc(w) \models \varphi$ is regular.*

Proof: From φ, using the technique given in [HMKT99, Lemma 4.1], we get an MSO formula φ^b_{str} over finite words in Σ^* such that for $w \in \Sigma^*$, $w \models \varphi^b_{str}$ iff w well-formed, b-bounded and $msc(w) \models \varphi$. The required conclusion follows since the strings described by MSO formulae on words form a regular set ([Tho90, Tho97]). $\Box$

We can now give another proof of the result that model checking CMSGs against MSO specifications is decidable using linearizations.

Alternative proof of Theorem 7.5.2:

In view of Theorem 7.3.7, let the CMSG be presented as a regular representative linearization L via a DFA $\mathcal{B}$ accepting L. Words in L are well-formed words as they are linearizations of MSCs. Since any regular language of well-formed words is b-bounded (see [HMKT00b]), where b can be computed, we can find a b such that L is b-bounded. Construct using Lemma 7.5.3, an automaton $\mathcal{B}'$ that accepts all b-bounded words which represent MSCs that satisfy φ. Clearly, all the MSCs represented by L satisfy φ iff $L(\mathcal{B}) \subseteq L(\mathcal{B}')$, which is decidable.

Restricted Satisfiability

We can extend the proof of Theorem 7.5.2 to show the decidability of a restricted satisfiability problem for MSO over MSCs.

Theorem 7.5.4 *Given a finite set $\mathcal{M}$ of CMSCs, $b \in \mathbb{N}$ and an MSO formula φ, the problem of checking if there exists a finite MSC M formed by the concatenation of CMSCs in $\mathcal{M}$, which is b-memory bounded and satisfies φ is decidable.*

Proof: Choose an alphabet Π and a bijection h between Π and $\mathcal{M}$. Using Lemma 7.5.1, construct a DFA over Π which accepts a word w iff w is well-defined, complete and b-memory bounded and $cmsc(w) \models \varphi$. The problem of checking if φ is satisfiable then reduces to checking emptiness of this automaton. $\Box$

The general problem of satisfiability of an MSO formula over the class of *all* MSCs is known to be undecidable [Thi01]. Also, in the above restricted satisfiability problem, if b is not given, the problem turns out to be undecidable.

Theorem 7.5.5 *Given a finite set $\mathcal{M}$ of CMSCs, and an MSO formula φ, the problem of checking if there exists an MSC M, formed by the concatenation of CMSCs in $\mathcal{M}$, which satisfies φ is undecidable.*

The proof again follows from a reduction from the halting problem of non-deterministic 2-counter machines. It is not difficult to see that the Lamport diagram depicting a run of a 2-counter machine (as done in Chapter 4, for example) can also be presented as an MSC. We can define an MSO formula describing this MSC and the class of MSCs can be made to include all the MSCs witnessing runs of 2-counter machines using the flexibility of unmatched send and receive events of the CMSG.

7.5.1 Linearization model checking

In [AY99], the authors study the model checking problem for finite MSCs and MSGs. Given an MSG and a regular set of finite sequences of actions as the specification, the model checking problem is to check whether for every MSC represented by the MSG, *all* the linearizations of the MSC are included in the specification. While the problem is easily seen to be decidable given a single MSC (describing the behaviour of the system) and the specification given by a finite-state automaton, it becomes undecidable when the behaviour of the system is given by an MSG. They then identify a subclass of MSGs called *locally synchronized (or bounded)* MSGs for which problem is decidable.

One important, though simple, consequence of dealing with representative linearizations is the decidability of the above linearization model checking problem for *linearization-closed* specifications. The linearization model checking problem for CMSGs is: Given a CMSG G and a regular language L of finite words over Σ, is $lin(msc(G)) \subseteq L$?

As we have mentioned above, this problem is known to be undecidable even for MSGs [AY99]. However, if the specification L is linearization-closed, i.e., $lin(msc(L)) = L$, then the problem turns out to be decidable.

Theorem 7.5.6 *The linearization model checking problem for CMSGs against regular linearization-closed specifications is decidable.*

Proof: Let $G = (\Pi, \mathcal{M}, h, \mathcal{A})$ be the given CMSG and $L \subseteq \Sigma^*$ be the regular linearization-closed specification given as a DFA $\mathcal{B}$. Using Theorem 7.3.7, construct L' which is a regular representative linearization of $msc(G)$. Now, we claim that all linearizations of MSCs represented by G belong to L iff $L' \subseteq L$. For, if there is an MSC $M \in msc(G)$ and a linearization w of M which is not in L, then all linearizations of M will not be in L. One such linearization will be in L' and hence $L' \nsubseteq L$. Also, if all linearizations of MSCs represented by G are in L, then clearly $L' \subseteq L$. □

Note that the above procedure works in polynomial time. If the CMSG G is such that the underlying automaton $\mathcal{A}$ has n states and each atomic CMSC has at most m events, and if $\mathcal{B}$ has l states, then we can model-check in time $O(n \cdot m^2 \cdot l)$.

7.6 MSO over Lamport diagrams

We define an MSO over Lamport diagrams in this section. We consider a distributed alphabet $\widetilde{\Sigma}$ as in Chapter 3 and work with Σ-labelled Lamport diagrams. The logic is also parametrized by Σ and is denoted by MSO(Σ). Not surprisingly, the problem of checking if a given MSO(Σ) formula is satisfiable (i.e., whether it has a Lamport diagram as a model or not) is undecidable. We then show that checking for satisfiability of MSO formulas is decidable over the class of communication closed and bounded LLDs and over the class of channel bounded LLDs.

To recap, a distributed alphabet is an n-tuple $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, where for each $i \in [n]$, Σ_i is a finite non-empty alphabet of actions of agent i and for all $j \in [n]$ such that $i \neq j$, $\Sigma_i \cap \Sigma_j = \emptyset$. The alphabet induced by $\widetilde{\Sigma}$ is given by $\Sigma = \cup_{i=1}^{n} \Sigma_i$. Σ-labelled Lamport diagrams are then as defined in Chapter 3.

MSO parametrized by Σ, denoted MSO(Σ), is defined on the partially ordered structure of Lamport diagrams. We have at our disposal a countable number of first order variables which are denoted by $x, y, \ldots$ and a countable number of second order variables which are denoted by $X, Y, \ldots$. The atomic formulas are of the kind $Q_a^i(x)$, for $i \in [n]$ and $a \in \Sigma_i$, $x \lessdot_i y$ for $i \in [n]$, $x <_c y$ and $x \in X$. Other formulas are formed by using the boolean connectives $\vee$ and $\neg$ and by using existential quantification over first order and second order variables. More precisely, MSO formulas are defined by the following syntax:

$$MSO(\Sigma) ::= Q_a^i(x), i \in [n], a \in \Sigma \mid x \lessdot_i y, i \in [n] \mid x <_c y \mid x \in X \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists x \phi$$

The formulas are interpreted over labelled Lamport diagrams with an interpretation function which assigns an event of the Lamport diagram to every first order variable and a set of events to every second order variable. Intuitively, the semantics of the atomic formula $Q^i_a(x)$ says that the label of the event assigned to the variable x is a, $a \in \Sigma_i$ in the labelled Lamport diagram, $x \lessdot_i y$ says that the event assigned to x is the immediate predecessor of the event assigned to y in the local total order of agent i in the labelled Lamport diagram and $x <_c y$ codes up the fact that the event assigned to x is a send event whose corresponding receive event is the one assigned to y. Finally, $x \in X$ says that the event assigned to x belongs to the set of events assigned to X.

Models of formulas of $MSO(\Sigma)$ are given by $M = (D, I)$ where D is a Σ-labelled Lamport diagram and I is an interpretation function which assigns an event of D to every individual variable and a set of events of D to every set variable. Given a model M and an $MSO(\Sigma)$ formula φ, we write $M \models_I \varphi$ to denote the fact that M satisfies φ under an interpretation I that maps variables in φ to events or set of events in D and define it inductively as follows. We just present the semantics of atomic formulas. The semantics of other formulas is done as in Section 7.4.

- $M \models_I Q^I_a(x)$ iff $a \in \phi(I(x)) \cap \Sigma_i$.
- $M \models_I x \leq_i y$ iff $I(x) \leq_i I(y)$ in D.
- $M \models_I x <_c y$ iff $I(x) <_c I(y)$ in D.
- $M \models_I x \in X$ iff $I(x) \in I(X)$.

A sentence is a formula without any free variables. An MSO(Σ) sentence φ is said to be it satisfiable if there exists a model M based on a Σ-labelled Lamport diagram such that $M \models \varphi$.

We first show that the satisfiability problem of this logic is undecidable.

Theorem 7.6.1 *Given an MSO(Σ) sentence φ, the problem of checking if there exists a model M based on a labelled Lamport diagram which satisfies φ is undecidable.*

The proof is again by a reduction from the halting problem of non-deterministic 2-counter machines used to show the undecidability of the satisfiability problem of the logic LD_1 in Chapter 4. The underlying idea is exactly the same—to define

Lamport diagrams representing runs of 2-counter machines. These diagrams will now be described using MSO formulas instead of formulas from LD_1. It is easy to see that all the formulas of $LD - 1$ used in Chapter 4 can be described using MSO formulas as all the modalities used in the logic LD_1 can be described in MSO.

We wind up the chapter by considering layered Lamport diagrams and interpreting MSO over LLDs. The syntax of the logic is the same as above, there is no special construct in the logic to reflect the structure of the layering of an LLD. The formulas are just interpreted on LLDs. We obtain the following decidability results.

Theorem 7.6.2 *1. Given an MSO formula φ and $b \in \mathbb{N}$, the problem of checking if φ is satisfiable over the class of communication closed and b-bounded LLDs is decidable.*

2. Given an MSO formula φ and $b \in \mathbb{N}$, the problem of checking if φ is satisfiable over the class of channel b-bounded LLDs is decidable.

The proofs are done by associating diagram automata (fragment automata) with φ such that the automaton accepts precisely those communication closed and b-bounded (channel b-bounded) LLDs that satisfy φ. We define the alphabet layers/fragments of the automaton in such a way that it codes up an interpretation of the free variables in the formula along with the layers/fragments as in the proof of Lemma 7.5.1. We can then define the automaton inductively as done in the proof of the lemma.

Alternately, we can also define a homomorphism between a finite alphabet Π and the alphabet coding up the interpretation above and use the results of Lemma 3.3.5 (and that of Lemma 3.4.4) in Chapter 3 to move back and forth between diagram (fragment) automata and Büchi automata over Π. Then, in a way similar to [HMKT99], we can define equivalent MSO formula interpreted over strings in Π^* and reduce checking satisfiability of MSO over LLDs to MSO over strings. Since the latter problem is decidable [Tho90, Tho97], we obtain the decidability of this problem too.

Chapter 8

Conclusion

We summarize the work done in this thesis below.

- In Chapter 2, we started with the model of Lamport diagrams to represent causal behaviours of distributed message passing systems. Notions of local and global states of Lamport diagram were introduced along with a few properties about them. Layered Lamport diagrams (LLDs) were introduced to describe behaviours of distributed systems which consist of repeated patterns of finite protocols. Various types of layerings were also discussed.

- Automata models for distributed systems were introduced in Chapter 3. System of Communicating Automata (SCA) served as the automaton model for systems whose behaviour is a collection of Lamport diagrams. Dually, diagram and fragment automata accepted LLDs. We also proved the decidability of emptiness problem for these automata and showed some closure properties.

- Modal logics tuned to talk about message passing systems were introduced in Chapter 4. The logic LD_0 had next, previous, future and past modalities and it was shown that the satisfiability problem is undecidable. In fact, we could show that undecidability pertains even if we consider restricted versions of the logic where one of next or previous modalities was restricted to special send or receive propositions respectively.

- A temporal logic to reason about local assertions on Lamport diagrams was introduced in Chapter 5 and it was shown that the satisfiability problem for this logic is decidable using the automata theoretic approach. Given a formula, an SCA accepting the models of the formula was defined and decidability of the emptiness problem for SCAs was used to show decidability of satisfiability here. An appropriate model checking problem was also shown to be decidable.

- In Chapter 6, we introduced temporal logics over layered Lamport diagrams. The logic was built on top of formulas from LD_0 which were used to reason about layers and the temporal modalities were used to talk about sequence of layers that make up the LLD. We showed that the satisfiability problem is undecidable, even if we restricted the size of layers to be uniformly bounded. However, the problem was shown to be decidable over the class of models based on communication closed and bounded LLDs and over the class of models based on channel bounded LLDs. We again showed decidability using the automata-theoretic approach, by using diagram automata and fragment automata.

- Message Sequence Charts (MSCs) were considered as alternate models of behaviours of message passing systems in Chapter 7. We compared Lamport diagrams with MSCs and also showed that LLDs in general, are more expressive than the models of Message Sequence Graphs (MSGs) and Compositional Message Sequence Graphs (CMSGs). We also considered MSO interpreted over Lamport diagrams and over MSCs. It was shown that satisfiability problem for MSO is undecidable in general, but is decidable when we restrict our attention to models based on communication closed and bounded LLDs and to models based on channel bounded LLDs. We also showed that model checking CMSGs against MSO specifications is decidable.

Future Work

In this thesis, we presented a study of various possible modal and temporal logics as specification languages of distributed message passing systems. It was shown that most of the natural choices of modalities resulted in undecidable satisfiability problems. However, exploiting the partially ordered nature of Lamport diagrams, we could define expressive temporal logics like m-LTL which are decidable. It would be useful to come up with more expressive logics like m-LTL that are also decidable. For example, m-LTL could itself be extended to include global next and until modalities. It is not clear if such an extension would be decidable. The presence of a weakly global previous modality (without receive propositions) does not suffice to code up runs of non-deterministic 2-counter machines. The decidability of satisfiability problem of logics where exactly one of the next or previous modalities is global and the other is local (without any special send/receive propositions) is also open.

In the context of distributed systems whose behaviours are described by LLDs, we again showed that natural choices of temporal logics resulted in undecidable satisfiability problems. On the positive side, we could obtain decidability when the formulas are interpreted over models based on communication closed and bounded LLDs and over models based on channel bounded LLDs. The bounds considered in both the cases were externally imposed irrespective of the specification. It would be nice to consider models whose structure is dictated by the specification and investigate the satisfiability problem over such models.

With reference to the decidability results presented in the thesis, questions regarding completeness of the algorithm also remain open.

There are many open questions on the automata-theoretic part. As mentioned in Chapter 3, most of the automata-theoretic questions related to SCAs including closure under complementation remain unanswered. It would be useful to develop a full automata theory of these models. If we succeed in showing that these automata define a robust class of behaviours of distributed systems, these can be used as good models of distributed message passing systems.

Publications

[MR00] B. Meenakshi and R. Ramanujam. Reasoning about message passing in finite state environments In *Proceedings of ICALP'00 (International Colloquium on Automata, Languages and Programming)*, volume 1853 of *Lecture Notes in Computer Science*, pages 487–498, 2000.

[MM01] P. Madhusudan and B. Meenakshi. Beyond message sequence graphs In *Proceedings of FST & TCS'01 (Foundations of Software Technology and Theoretical Computer Science)*, volume 2245 of *Lecture Notes in Computer Science*, pages 256–267, 2001.

[MR03] B. Meenakshi and R. Ramanujam. Reasoning about layered message passing systems In *Proceedings of VMCAI'03 (Verification, Model Checking and Abstract Interpretation)*, volume 2575 of *Lecture Notes in Computer Science*, pages 268–282, 2003.

[MR04] B. Meenakshi and R. Ramanujam. Reasoning about layered message passing systems To appear in *Computer Languages, Systems and Structures.*

Bibliography

[AEY00] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. 22nd International Conference on Software Engineering*, pages 304–313, 2000.

[AHP96] R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. In *Software Concepts and Tools*, volume 17(2), pages 70–77, 1996.

[AMP98] R. Alur, K. L. McMillan, and D. Peled. Deciding global partial order properties. In *Proc. ICALP 1998*, volume 1443 of *LNCS*, pages 41–52. Springer-Verlag, 1998.

[APP95] R. Alur, D. Peled, and W. Penczek. Model checking of causality properties. In *Proc. 10th LICS*, pages 90–100. IEEE Computer Society, 1995.

[AY99] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. CONCUR 1999*, volume 1664 of *LNCS*. Springer-Verlag, 1999.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. In *Z. Math. Logik Grundl. Math.*, volume 6, pages 66–92, 1960.

[CGP00] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking.* MIT Press, 2000.

[CNT98] P. Ciancarini, O. Neirstrasz, and R. Tolksdorf. A case study in coordination: Conference management on the internet. 1998. Available at:

`http://malvasia.di.fct.unl.pt/activity/coordina/working/case-studies`.

[CW96] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 1996.

[EF82] T. Elrad and N. Francez. Decomposition of distributed programs into communication closed layers. In *Science of Computer Programming*, volume 2, pages 155–173, 1982.

[Elg61] C.C. Elgot. Decision problems of finite automata and related arithmetics. In *Trans. Amer. Math. Soc.*, volume 98, pages 21–52, 1961.

[GMP01] E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence graphs. In *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 496–511. Springer-Verlag, 2001.

[HC96] G. E. Hughes and M. J. Cresswell. *A new introduction to modal logic.* Routledge, 1996.

[HMKT99] J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Towards a theory of regular MSC languages. *BRICS Report RS-99-52, Department of Computer Science, Aarhus University, Denmark*, 1999.

[HMKT00a] J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. ICALP 2000*, volume 1853 of *LNCS*, pages 675–686. Springer-Verlag, 2000.

[HMKT00b] J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Regular collections of message sequence charts. In *Proc. MFCS 2000*, volume 1893 of *LNCS*, pages 405–414. Springer-Verlag, 2000.

[HNW99] M. Huhn, P. Neibert, and F. Wallner. Model checking logics for communicating sequential agents. In *Proceedings of FOSSACS 1999*, volume 1578 of *LNCS*, pages 227–242. Springer-Verlag, 1999.

[HR04] M. Huth and M. D. Ryan. *Logic in Computer Science.* Cambridge University Press, 2 edition, 2004.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[ITU97] ITU-TS Recommendation Z.120. Message sequence chart (MSC). *ITU-TS*, 1997.

[Kra99] M. Kracht. *Tools and techniques in modal logic.* Elsevier, 1999.

[Lam78] L. Lamport. Time, clocks and ordering of events in a distributed system. In *Communications of ACM*, volume 21(7), pages 558–565, 1978.

[LL90] L. Lamport and N. Lynch. Distributed computing: Models and methods. *Handbook of Theoretical Computer Science: Volume B*, pages 1157–1199, 1990.

[LPRT95] K. Lodaya, R. Parikh, R. Ramanujam, and P. S. Thiagarajan. A logical study of distributed transition systems. *Information and Computation*, 119(1):91–118, 1995.

[LRT92] K. Lodaya, R. Ramanujam, and P. S. Thiagarajan. Temporal logics for communicating sequential agents. *International Journal of Foundations of Computer Science*, 3(2):117–159, 1992.

[Mad01] P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. ICALP 2001*, volume 2076 of *LNCS*, pages 809–820. Springer-Verlag, 2001.

[Maz87] A. Z. Mazurkiewicz. Trace theory. In *Advances in Petri nets*, volume 255 of *LNCS*, pages 279–324. Springer-Verlag, 1987.

[MP91] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification.* Springer-Verlag, 1991.

[MP99] A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. MFCS 1999*, volume 1672 of *LNCS*, pages 81–91. Springer-Verlag, 1999.

[MR02] S. Mohalik and R. Ramanujam. Distributed automata in an assumption—commitment framework. *Sadhana, Special issue on Formal Methods in Verification*, 27(2):209–250, 2002.

[MS99] C. Montangero and L. Semini. Composing specifications for coordination. volume 1594 of *LNCS*. Springer-Verlag, 1999.

[Pel00] D. Peled. Specification and verification of message sequence charts. In *Proc. IFIP FORTE/PSTV 2000*, volume 183, pages 139–152. Kluwer, 2000.

[Pel01] D. Peled. *Software reliability methods*. Springer-Verlag, 2001.

[Pra86] V. Pratt. Modelling concurrency with partial orders. *IJPP*, 15(1), 1986.

[PZ92] M. Poel and J. Zwiers. Layering techniques for development of parallel systems. In *Proc. CAV 1992*, LNCS, pages 16–29. Springer-Verlag, 1992.

[Ram96] R. Ramanujam. Locally linear time temporal logic. In *Proc. 11th LICS*, pages 118–127. IEEE Computer Society, 1996.

[RGG96] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC*, volume 28, 1996.

[Rus96] J. Rushby. Mechanized formal methods: Progress and prospects. In *16th FST & TCS*, volume 1180 of *LNCS*, pages 45–51. Springer-Verlag, 1996.

[Thi94] P. S. Thiagarajan. A trace based extension of propositional linear time temporal logic. In *Proc. 9th LICS*, pages 438–447. IEEE Computer Society, 1994.

[Thi01] P. S. Thiagarajan. Personal communication, 2001.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science: Volume B*, pages 165–191, 1990.

[Tho97] W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.

[TW97] P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proc. 12th LICS*. IEEE Computer Society, 1997.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332–345. IEEE Computer Society, 1986.

[Win87] G. Winskel. Event structures. In *Advances in Petri nets*, volume 255 of *LNCS*, pages 325–392. Springer-Verlag, 1987.

[WN95] G. Winskel and M. Nielson. Models for concurrency. *Handbook of Logic in Computer Science*, pages 1–148, 1995.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.