Formal Verification of Communicating Automata

Thèse de doctorat de l'université Paris-Saclay

Amrita SURESH

Composition du jury

Mihaela Sighireanu Professeure, ENS Paris-Saclay	Présidente
Ahmed Bouajjani Professeur, Univ. Paris Diderot	Rapporteur
Rupak Majumdar Directeur de recherche, Max Planck Institute for Software Systems	Rapporteur
Thierry Jéron Directeur de recherche, IRISA / INRIA Rennes	Examinateur
Nobuko Yoshida Professeure, University of Oxford	Examinatrice
Alain Finkel Professeur, ENS Paris-Saclay	Directeur de thèse
Benedikt Bollig Directeur de recherche, CNRS, ENS Paris-Saclay	Co-encadrant de thèse

Abstract

Distributed systems involve processes that run independently and communicate asynchronously. While they capture a wide range of use cases and are hence, ubiquitous in our world, it is also particularly difficult to ensure their correctness. In this thesis, we model such systems using mathematical and logical formulation, and try to verify them algorithmically. In particular, we focus on FIFO (First-In First-Out) machines, with one or more finite-state machines communicating via unbounded reliable FIFO buffers.

As most verification problems are known to be undecidable for FIFO machines, we focus on various subclasses and approximations of the model. The first model we consider is branch-well structured transition systems (branch-WSTS), a class which strictly includes the well-known class of WSTS. We study the problems of boundedness and termination for such systems, and demonstrate some examples of them. We also define another class of systems where the monotony condition is relaxed and show that a variant of the coverability problem is decidable under effectivity conditions.

We then study the restriction of input-boundedness on FIFO machines, and show that rational reachability and various other properties are decidable for FIFO machines under the input-bounded restriction. In doing so, we answer a long-standing open question regarding the reachability of input-bounded FIFO machines. We also derive some complexity bounds by considering the simplest case, a FIFO machine with a single channel.

Another restriction that we study is synchronizability in communicating systems. In particular, we explore this notion for MSCs (Message Sequence Charts), which is a model to represent executions of a communicating system. We show that if any set of MSCs can satisfy two properties, namely MSO (Monadic Second-order Logic) definability and bounded (special-)tree width, then synchronizability is decidable. Moreover, reachability and model-checking are also decidable within this framework. We also unify some classes from the literature using this framework, and for some other classes, show their undecidability.

Synthèse

Ces dernières années, en raison des progrès technologiques, les systèmes sont devenus de plus en plus complexes. Cela conduit à la question naturelle de déterminer si un système donné est correct. La vérification formelle est l'un des outils utilisés pour vérifier si un système fonctionne selon les propriétés souhaitées. Il s'agit de vérifier si un modèle d'un tel système satisfait certaines spécifications mathématiques.

Dans le cadre de cette thèse, nous modélisons les systèmes en utilisant une formulation mathématique et logique, et essayons de les vérifier algorithmiquement. En particulier, nous nous concentrons sur les automates FIFO (First-In First-Out), qui sont des machines à états finis communiquant via des canaux FIFO fiables pouvant contenir des mots de longueur arbitrairement grande. Comme la plupart des problèmes de vérification sont connus pour être indécidables pour les automates FIFO, nous nous concentrons sur diverses sous-classes et approximations de ce modèle.

Nous proposons et étudions une généralisation des systèmes de transition bien structurés (WSTS) tout en conservant la décidabilité de finitude des états accessibles et de terminaison. Dans notre classe des branch-WSTS, nous généralisons la définition de beau-quasi-ordre (wqo) pour qu'elle soit applicable seulement sur les branches d'états accessibles à partir de l'état initial. De plus, nous généralisons également la condition de monotonie de la même manière. Bien que les branch-WSTS conservent la décidabilité de la terminaison et de la finitude des états accessibles, il apparaît que le problème de couverture est indécidable. Nous définissons une nouvelle notion de monotonie, appelée cover-monotony, qui est strictement plus générale que la monotonie habituelle et permet, malgré tout, de décider une forme restreinte du problème de couverture, que nous appelons le problème de couverture initiale. Nous montrons également quelques exemples de systèmes dans la littérature qui ne sont pas des WSTS mais sont des branch-WSTS.

Ensuite, nous étudions la restriction de la limitation de l'entrée (input-boundedness) sur les canaux FIFO (c'est-à-dire que la séquence de messages envoyés via un canal particulier appartient à un langage borné donné). Nous prouvons, en réduisant ce modèle à une machine à compteurs avec des tests à zéro restreints, que le problème d'accessibilité rationnelle (et par extension, l'accessibilité de l'état de contrôle, la terminaison, les blocages, etc.) est décidable. Cette classe de machines englobe les machines où le langage entrant est borné sur des lettres (letter-bounded), les machines plates (flat FIFO), les réseaux FIFO linéaires et les machines monogènes, pour lesquelles certains de ces problèmes se sont déjà avérés décidables. Ces résultats théoriques peuvent constituer les bases pour construire un outil de vérification des machines FIFO générales basé sur l'analyse des machines à langages bornés en entrée. De plus, nous étudions également le dual naturel de la restriction de la limitation de l'entrée, en considérant des exécutions bornées en sortie. Nous montrons que de nombreuses propriétés entre ces restrictions sont inter-réductibles, tout en soulignant leurs différences. Enfin, nous montrons également des bornes inférieures pour le problème d'accessibilité bornée par les entrées.

Enfin, nous étudions la classe des systèmes synchronisables, qui est une sous-classe stricte des machines à états finis communicantes générales (CFSM). Plusieurs notions de synchronisabilité d'un système distribué communicant ont été introduites dans la littérature. Essentiellement, un système est dit synchronisable si chaque exécution peut être réordonnée afin qu'elle satisfasse à certains critères, par exemple, qu'elle maintient un canal borné. Nous considérons l'ensemble des MSC (Message Sequence Charts), qui représentent graphiquement les exécutions du système. Pour cet ensemble, nous créons un cadre basé sur la logique monadique du second ordre (MSO) et la largeur des graphes, qui unifie la majorité des définitions existantes, explique leurs bonnes propriétés et permet de déduire facilement d'autres définitions plus générales et des résultats de décidabilité pour la synchronisabilité. Nous fournissons également de meilleures bornes inférieures en utilisant la logique dynamique propositionnelle (PDL) pour certains cas. Notre cadre capture également à la fois le modèle de communication peer-to-peer, ainsi que le modèle de communication de boîte aux lettres (mailbox). De plus, pour la notion de synchronisabilité en envoi, qui ne satisfait pas les propriétés de notre framework, nous prouvons également l'indécidabilité des problèmes d'accessibilité (et d'autres problème associés).

Contents

Chapter 1

Introduction

Over the years, due to the advance in technology, systems have been getting increasingly complex. From mobile telephones to automatic doors, we are surrounded by hardware, software and their interactions. This leads to the natural question of investigating the correctness of a given system. Formal verification is one of the tools used to verify if a system works according to its properties.

Let us be more precise. What do we mean by a system? In the context of this thesis, a (distributed) system is a set of autonomous processes which communicate (asynchronously) with one another and accomplish tasks. The problem of formal verification deals with checking if a model of such a system behaves as desired, according to some mathematical specifications.

The objective of this thesis is not to prove that a system performs the function it is intended to do, but rather to verify specific properties for any systems, regardless of their function. This helps detect a large number of common design errors across various systems. For example, one of the problems we consider is the deadlock problem. This occurs when a process reaches a state but cannot progress any further, either by sending any messages to other processes, receiving any messages, or performing any local actions. Deadlocks are common design errors which can be seen in nearly every kind of distributed system.

Let us consider a basic scenario as follows: A user wishes to connect to a server. To this end, they send a request (which we call REQ) to the server, and wait. The server reads the request, provides the service, and sends a message when it is done (which we denote by DONE). The user can read this message and restart the exchange again. When the server is not processing a request, it can self-inspect and might detect a fault in itself. In this case, it sends an alarm to the user (which we denote by ALARM) and moves to a repair state. The user can read this alarm, register it, and direct the server back to the idle state by sending an acknowledgement (ACK). This protocol was introduced in [BZ83].

We model this scenario using a finite-state machine each for the user and the server, and a (potentially unbounded FIFO) buffer of communication for each direction, as shown in Figure 1.1. The notation for expressing sends and receptions is illustrated in the figure. Both the user and the server start at state 0, indicated by an incoming arrow with no source. An exclamation sign ($!m$) identifies the sending of a message m, and a question mark ($?m$) signals the reception of the message m. For example, by taking an arc marked with, say !ACK, the user changes their state and the message ACK is sent to the channel that the server reads from.

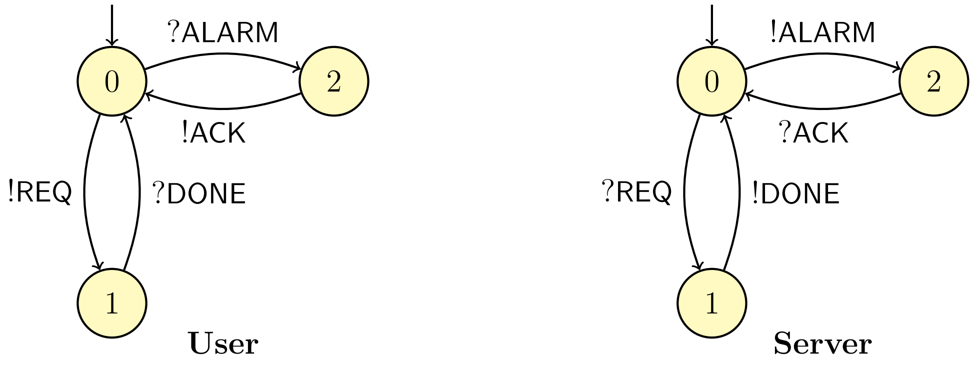


Figure 1.1: The model of the protocol from [BZ83]

Going back to the deadlock problem, we can see that the model above can lead the processes to a deadlock. For example, if the user sends a request REQ to the client, but the client detects a fault before it receives the request and instead sends an alarm ALARM to the user. Then, at states 1 and 2, respectively, the user and the server are both in a state where they cannot send any more messages, or receive the message in their channels.

Some of the central verification problems we study are the non-termination, reachability, and model-checking problems. Non-termination amounts to checking whether or not every run of the system is terminating. Reachability asks if a specific state is reachable in any run of the system. And finally, the model checking problem asks for an algorithm that decides, given a property (expressed as a formula ϕ of a suitable logic), whether every behaviour of the system satisfies it.

We shall now look more closely at the models we study in this thesis, and survey some of the related results from the literature.

1.1 Systems communicating via FIFO channels

The main model being studied in this thesis is where the communication between processes is via (potentially unbounded) First-In First-Out (FIFO) channels, as seen in the example above. Furthermore, we abstract the processes to be finite-state automata, which can send and receive messages from the channels, apart from performing local actions. However, since systems of processes communicating through (at least two) one-directional FIFO channels, or equivalently, a single process with one FIFO channel, simulate Turing machines, most properties, including deadlock detection and the other problems we mentioned earlier, are undecidable for such systems [BZ83].

A natural restriction to retain decidability for such systems is weaking the nature of the channels. In [AJ93, Fin94], the authors showed that if the underlying channels are lossy, then the state reachability problem becomes decidable (but non-elementary). For a survey of the plethora of results in the study of lossy channel systems see [AČJT00, FS01].

If one restricts to runs with B-bounded channels (the number of messages in every channel does not exceed $B \in \mathbb{N}$), then reachability becomes decidable for universally-bounded FIFO systems [GKM07]. In fact, the model-checking problem also becomes decidable for various logics [Pel00, MM01, GMK04]. Moreover, in [LM04], the authors introduced existentially k-bounded systems (see also [LM02, GMK04, GKM07]) where all accepting executions leading to a stable (with empty channels) final states can be re-ordered into a k-bounded execution. For this model as well, the reachability problem becomes decidable. However, checking if a system is existentially k-bounded is undecidable, even if k is part of the input [GKM07]. A more general definition, still called existentially bounded, is given in [KM21], where the considered executions are not supposed to be final or stable. In [HMK$^+$05, LM04], the notion of universally k-bounded is also discussed, and the authors show that the property is undecidable in general.

In [LTMP08], the authors propose a "context-bounded" analysis of systems communicating with message channels. According to this approach, one "context" involves a single process reading from its channel, and posting to the channel of other processes, and the number of contexts per execution is bounded. The context-bounded reachability problem is in 2-EXPTIME, even for recursive FIFO systems [LTMP08, HLMS10]. For non-confluent topology, reachability is in EXPTIME for recursive FIFO systems with 1-bounded channels [HLMS10].

Reachability is in PTIME in half-duplex systems [CF05] with two processes (moreover, the reachability set is recognizable and effectively computable), but the natural extension to three processes leads to undecidability. More recently, in [DGGL21], the authors introduce "greedy systems", as a possible generalization of half-duplex systems for more than two processes.

In [MP11a, AGK14], uniform criteria for decidability of reachability and model-checking questions are established for communicating recursive systems whose restricted architecture or communication mechanism gives rise to behaviours of bounded tree-width.

Input-bounded FIFO machines. Many papers, starting in the 80s until today, have studied FIFO machines in which the input-language of a channel (i.e. the set of words that record the messages entering a channel) is included in the set $Pref(w_1^* w_2^* \dots w_n^*)$ of prefixes of a bounded language $w_1^* w_2^* \dots w_n^*$. We call this class of FIFO machines input-bounded. If all $w_1, \dots, w_n$ are single letters, then the language is called letter-bounded.

If the set of letters that may enter a channel c is reduced to a unique letter m_c, then the input-language of c is included in m_c^*, and this subclass trivially reduces to VASS and Petri nets [YG83]. Also note that, in general, the behaviour of such FIFO machines does not have bounded tree-width.

Monogeneous FIFO nets [Fin82, MF85, FS01] (input-languages of channels c are included in $Pref(u_c \cdot v_c^*)$ where u_c, v_c are two words associated with c) and linear FIFO nets [CF87] (input-languages are included in $Pref(m_1^* m_2^* \dots m_n^*)$ where each m_i is a letter and $m_i \neq m_j$ if $i \neq j$) both generalize Petri nets with still a decidable reachability problem. The deadlock problem, which is in fact, a variant of the reachability problem, is shown decidable for input-letter-bounded FIFO systems in [GGLR87] by reducing to reachability for VASS. However, the extension to general input-bounded machines was left open.

Flat machines are another subclass of input-bounded machines in which the language of their control-graph, considered a finite automaton, is a bounded language. For flat FIFO machines, control-state reachability is NP-complete [EGM12]; this result has recently been extended to reachability, channel unboundedness, and other classical properties [FP19].

Flat, input-letter-bounded, linear (which we also refer to as distinct-letter bounded), and monogeneous FIFO machines (the last three types contain VASS, and are all incomparable) are all included in the more general input-bounded FIFO machines. The unboundedness problem of input-bounded FIFO machines was shown decidable in [JJ93] by using the well-structured concepts but with no extension to decidability of reachability.

Synchronizability. Several notions of synchronizability of a message-passing system have been introduced in the literature. Roughly, a system is called synchronizable if every execution can be rescheduled to meet certain criteria, e.g., a channel bound.

We focus on the boundedness problem, which we know to be undecidable. We could limit our analysis to decide whether, for a given integer $k \geq 0$, known in advance, the FIFO channels are k-bounded, and this property is generally decidable in PSPACE. Unfortunately, the k-boundedness property is too binding since we could want to design an unbounded system that is able, for example, to make unbounded iterations of sending and receiving messages. Hence, to cope with this limitation, one can find variants of the boundedness property that essentially reduce to say that every unbounded execution of a system (i.e., channels are unbounded along the execution) is equivalent (for instance, causally equivalent) to another bounded execution.

In [BB11], the authors introduced synchronizable systems, for which every execution is equivalent (for the projection on sending messages) to an execution of the same system, but communicating by rendezvous. To avoid ambiguity, we call such systems send-synchronizable. However, for peer-to-peer systems, this property was shown to be undecidable in [FL17]. In [SAB20], the authors show that the class of synchronizable peer-to-peer systems are exactly the same as the ones which are ”choreography realisable”, a property which now precisely characterises the class of peer-to-peer systems which are synchronizable.

In [BEJQ18], the authors called a system k-synchronizable (to avoid confusion we call such systems weakly k-synchronizable in the remainder of this thesis) if every message sequence chart (MSC) of the system admits a linearization (which is not necessarily an execution) that can be divided into blocks of at most k messages. After each block, a message is either read, or will never be read. This constraint seems to imply that buffers are bounded to k messages. However, as the linearization need not be an execution, this implies that a weakly k-synchronizable execution, even with the more efficient reschedule, can need unbounded channels to be run by the system. Reachability under this restriction and checking k-synchronizability are both PSPACE-complete [DGLL20].

A short note on communication architectures. A key difference between these works is that they consider different communication architectures. Existentially bounded systems have been studied for peer-to-peer systems (with one channel per pair of processes). On the other hand, k-synchronizability has been studied for mailbox communication, for which each process merges all its incoming messages in a unique channel. The decidability results for k-synchronizability have been extended to peer-to-peer communications [DGLL20], but it is unknown whether the decidability results for existentially bounded systems extend to mailbox communication. Moreover, variants of those definitions can be obtained depending on if we consider messages that are sent but never read, called unmatched messages. Indeed the challenges that arise in [BEJQ18] are due to mailbox communication and unmatched messages blocking a channel so that all messages sent afterwards will never be read.

1.2 Well-structured transition systems

Transition systems, including FIFO systems, broadly refer to any systems that take transitions to move from state to state. Well-structured transition systems (WSTS) (initially called structured transition systems in [Fin90]) are a notable subclass of transition systems. They have decidable termination and boundedness problems. They capture properties common to a wide range of formal models used in model-checking, system verification and concurrent programming [FS01].

A WSTS is an infinite set X (of states) with a transition relation $\rightarrow \subseteq X \times X$. The set X is quasi-ordered by $\leq$, and $\rightarrow$ fulfils one of the various possible monotonies with respect to $\leq$. The quasi-ordering of X is further assumed to be well, i.e. well-founded and with no infinite antichains. These two properties lead to a general framework in which it is possible to algorithmically decide verification problems like coverability, termination and boundedness. Some important classes of infinite-state systems are directly WSTS. Petri Nets (or equivalently, vector addition systems (with states) VASS), and many of their extensions are WSTS. Even Lossy Channel Systems, which we introduced earlier, are WSTS. More recently, the theory of WSTS has been applied to study computational models resulting from a combination of different types of systems like asynchronous systems defined by extending pushdown systems with an external memory [CV09], cryptographic protocols [DS20], and others.

While some systems may not have the required monotonicity, they can be over-approximated in order to be a WSTS. If this means new behaviours of the system are introduced, any negative answer for control state reachability and positive answer for termination or boundedness will still be valid for the original system. This is an approach used, for example, in [ABC+08, ADR09, ACV11].

Many safety properties of real-world systems reduce to the coverability problem, which asks: Given a WSTS $\mathcal{S} = (X, \rightarrow, \leq)$ and two states $x, y \in X$, does there exist a sequence of transitions leading from x to a state $y' \geq y$? For this verification problem, there are two main classes of algorithms on WSTS, both relying heavily on the monotony condition. Firstly, we have the backward algorithms, which start from a final state, then compute progressively an over-approximation of the predecessor states. These algorithms generally allow for deciding safety properties like control state reachability. The properties of WSTS allow for a simple representation of the approximation of predecessor states. On the other hand, forward algorithms, that start from an initial state, progressively compute an over-approximation of the reachable states. These algorithms generally allow for deciding liveness properties like termination or properties related to the full set of reachable states like boundedness. Over the years, a number of strengthenings and weakenings of the notion of monotony (of $\rightarrow$ w.r.t. $\leq$) were introduced, with the goal of allowing WSTS to capture ever more models [AČJT00, FS01]. However, the representation of the approximation of reachable states is not immediate, which makes verification of properties that are more precise than boundedness difficult. Recently, a procedure to decide such properties has been proposed in [BFM17], but its termination was not guaranteed.

1.3 Contributions of the thesis

This thesis aims to study a number of under-approximations of general transition systems. Firstly, we look at branch-well structured transition systems, which we shall henceforth refer to as branch-WSTS. This model tries to relax the aforementioned well-quasi-order condition to elements along an execution. Moreover, we similarly try to relax the monotony condition as well. Secondly, for FIFO machines, we restrict the reachability set to runs where the language that enters a channel is bounded. Finally, we survey various notions of synchronizability from the literature, provide a framework that consolidates many of these classes, and give properties to decide inclusion into the class. We summarize here the main contributions of this thesis.

1.3.1 Branch-WSTS

As mentioned previously, WSTS captures various classes of transition systems, and lets one algorithmically verify properties about them. Our main contribution is to prove that the monotony and well-quasi-order (wqo) assumptions can further be weakened while some problems remain decidable. More precisely, we introduce a notion of WSTS, called branch-well-structured transition systems (branch-WSTS), where the monotony is only applicable to states reachable one from another. Furthermore, we also relax the wqo condition to such states. With this relaxation, retaining the decidability of termination and boundedness is still possible. Moreover, we show some classes of systems that have been studied in the literature that are branch-WSTS.

However, we show that the coverability problem is undecidable for general branch-WSTS. Hence, for the coverability problem, we introduce a notion of monotony, called cover-monotony, which still allows deciding the coverability problem, even in the absence of strong (or strict or transitive or reflexive) monotony. Indeed, while the usual backward algorithm for coverability relies on well-foundedness, the forward algorithm described in [BFM17] does not require that property. Hence, we show that, for a given initial state, we can decide the coverability problem for such systems.

1.3.2 Bounded reachability

Secondly, we solve a problem that was left open in [GGLR87], namely the decidability of the reachability problem for input-bounded FIFO machines. We present a simulation of input-bounded FIFO machines by counter machines with restricted zero tests, for which we show that reachability is decidable. The main idea is to associate a counter with each word in the bounded language, and to ensure that the counters are incremented and decremented in a way that corresponds to the FIFO order. Since we can have repeated letters, and ambiguities in the FIFO machine, we first need to construct a normal form of the FIFO machine. Furthermore, we ensure that for every run in the FIFO machine, we can construct an equivalent run in the counter machine and vice-versa.

As we actually solve the general rational-reachability problem, we can deduce the decidability of other verification properties like control-state reachability, deadlock, unboundedness, and termination.

We study the natural dual of the input-bounded reachability problem, which are systems of output-bounded languages in which the set of words received by each channel is constrained to be bounded, and can deduce the reachability, unboundedness, termination and control-state reachability for the same.

We obtain better upper bounds for the input-bounded reachability of FIFO machines with a single channel (reachability is still undecidable for FIFO machines with a single channel). This is done by reducing it to reachability in unary ordered multi-pushdown systems (a class that was previously analysed in [ABH17]). It is, hence, solvable in EXPTIME.

Following the bounded verification paradigm, applied to FIFO machines (for instance, in [EGM12, FP19]), we open the way to a methodology that would apply existing results on input-bounded FIFO machines to general FIFO machines.

1.3.3 Synchronizability

As we saw earlier, there are a number of different notions of synchronizability in the literature. In order to unify these notions, we introduce a general framework based on monadic second-order (MSO) logic and (special) tree-width that captures most existing definitions of systems that may work with bounded channels. Moreover, reachability and model checking are shown decidable in this framework.

We show that existentially bounded systems can be expressed in our framework and, consequently, the existentially k-bounded property is decidable by using the generic proof. We generalize the existing notion of (weak) k-synchronizability in, and we introduce three new classes of synchronizable systems: weakly synchronizable (which are more general than weakly k-synchronizable [BEJQ18]), strongly synchronizable and strongly k-synchronizable (which are particular cases of weakly synchronizable). We then prove that most of these properties fit in our framework and are all shown decidable using the generic proof. We also show some undecidability results for classes that do not fit in the framework.

We then deduce that reachability and model checking are decidable for these classes. Previously, only control-state reachability was shown to be decidable for weakly k-synchronizable [BEJQ18], and it is clearly also decidable for existentially/universally bounded systems, but reachability properties are generally not studied for these classes of systems.

In order to obtain better complexity results for some classes (strongly and weakly synchronizable systems), we also use the fragment of propositional dynamic logic with loop and converse (LCPDL) instead of MSO logic in our framework.

Finally, we also look closely at the notion of send-synchronizability, a notion that was shown undecidable for peer-to-peer systems in [FL17]. For this model, we go on to show that reachability and control-state reachability are also not decidable, and investigate whether there is a class of properties which are decdidable for send-synchronizable systems.

1.4 Organization of the thesis

Chapter 2 introduces the various types of models we study in the remainder of the thesis. It starts with the most general transition system, and goes on to define the more specific models of FIFO machines, communicating systems, and counter machines. For communicating systems, we also define the graphical representation and give some information on the communication architecture. We also define the notion of finite automata and some conditions on them.

Chapter 3 delves into the first subclass of transition systems that we study in this thesis: branch-well structured transition systems, or branch-WSTS.

We first introduce terminology and some well-known results concerning well-quasi-orderings and well-structured transition systems. Then we introduce branch-WSTS, and show that termination and boundedness are decidable for a class of effective branch-WSTS. Then, we look at some examples of classes of branch-WSTS. Finally, we investigate the coverability problem for WSTS with relaxed conditions.

Chapter 4 investigates the second restriction we study on transition systems. In this chapter, we study the bounded reachability problem for FIFO machines.

We introduce the notion of bounded reachability, and then go on to prove the main result, which is the input-bounded rational-reachability problem. Then, we extend the result to other problems, such as reachability, deadlock, and boundedness. This is followed by the study of the natural dual of input-bounded reachability, which we refer to as output-bounded reachability. We then study the single-channel case, and obtain some lower bounds and elementary upper bounds. Finally, we conclude with a theory to use these results as an under-approximation for general FIFO machines.

Chapter 5 provides a framework, based on MSO (and LCPDL) logic and special tree-width, in order to unify the various definitions of synchronizability in literature.

It starts with providing a logical background for the rest of the chapter, and some of the graphical notions related to executions of communicating systems, such as tree-width and the conflict graph. Then, we present the unifying MSO (and LCPDL) framework and two general theorems on k-synchronizability and model checking. Finally, we apply the framework to various subclasses, including k-synchronous MSCs and existentially-bounded MSCs. We conclude with a brief account of some of the recent developments in the field.

Chapter 6 focuses on the send-synchronizability problem for peer-to-peer systems. We study some of the verification problems for send-synchronizable systems.

We introduce the two notions of send-synchronizable systems, and some of the problems that we investigate for these classes. Then, we show that reachability and some variants are undecidable for both the classes. Then, we study a series of problems for which we can have decidable properties, and conclude with some final remarks.

Most of the results presented in this thesis appear in [BFS20, BDGF$^+$21, BFS22a, BFS22b].

Chapter 2

Transition Systems

Transition systems are an abstraction used in the study of computation, to describe the potential behaviour of discrete and continuous systems. A transition system consists of a set of states, and a collection of transitions between states. We study a variety of transition systems in this work, and this chapter will be their overview.

As a concrete example, consider an elevator. Here, the states are the various positions of the elevator cabin, and the transitions describe the movement of the cabin. It can be viewed as a dynamic process with the users pressing buttons, thereby changing the state of the elevator. Similarly, programs, networks, communication systems, etc., can be seen as transition systems. Moreover, the notion of transition systems not only helps in modelling such behaviours but also helps to develop tools for verifying the modelled system.

Hence, the goal is to develop general mathematical structures which could serve as sufficient conditions for achieving decidability. Over the last few decades, there has been a concerted effort towards defining classes of models and finding appropriate algorithms to automatically verify them. Notable examples include timed automata [ACD90, AH94], hybrid automata [Hen95], Petri nets [Jan90, JM95], well-structured transition systems [AČJT00, FS01], etc.

Preliminaries. Before we start with the formal representation of transition systems, let us recall some basic definitions of words and languages.

Let A be a finite alphabet. As usual, A^* is the set of finite words over A, and A^+ the set of non-empty finite words. We let $|w|$ denote the length of $w \in A^*$. For the empty word ε, we have $|\varepsilon| = 0$. Given $a \in A$, let $|w|_a$ denote the number of occurrences of a in w. With this, we let $\mathit{Alph}(w) = \{a \in A \mid |w|_a \geq 1\}$. The concatenation of two words $u, v \in A^*$ is denoted by $u \cdot v$ or $u.v$ or simply uv.

$u \in A^*$ is a prefix (resp. suffix) of w if $w = u \cdot v$ (resp. $w = v \cdot u$) for $v \in A^*$. Similarly, $u \in A^*$ is an infix of w if $w = v_1 \cdot u \cdot v_2$ for $v_1, v_2 \in A^*$. The sets of prefixes, suffixes, and infixes of $w \in A^*$ are denoted by $\mathit{Pref}(w)$, $\mathit{Suf}(w)$, and $\mathit{Inf}(w)$, respectively. Note that $\{\varepsilon, w\} \subseteq \mathit{Pref}(w) \cap \mathit{Suf}(w) \cap \mathit{Inf}(w)$. For a set X, any mapping $f : A^* \to 2^X$ can be extended to $f : 2^{A^*} \to 2^X$ letting, for $L \subseteq A^*$, $f(L) = \bigcup_{w \in L} f(w)$. In particular, $\mathit{Alph}(L)$, $\mathit{Pref}(L)$, $\mathit{Suf}(L)$, and $\mathit{Inf}(L)$ are extended in that way.

prefix, suffix, infix

Now, we shall begin to define transition systems. At its core, the most general representation of a transition system is the following.

transition system

Definition 2.1. *A transition system is a pair $\mathcal{S} = (X, \to)$ where X is a (potentially infinite) set of states and $\to \subseteq X \times X$ is the transition relation.*

We write $x \to y$ for $(x, y) \in \to$. Transition systems may have additional structures like initial and final states, labels for transitions, causal independence relations, etc.

Example 1. A simplified model of a server is described as follows (see Figure 2.1): The server has three states, a waiting state, a state upon receiving a request, and a state upon completing a request. If the server is in the idle state, it continues to wait until it receives a request. Upon receiving a request, it moves to $\mathrm{x}_{\mathrm{req}}$. It then processes the request and goes to $\mathrm{x}_{\mathrm{done}}$ upon completing the request, and then goes back to the waiting state. Hence,

$$\begin{aligned} X &= \{\mathrm{x}_{\mathrm{wait}}, \mathrm{x}_{\mathrm{req}}, \mathrm{x}_{\mathrm{done}}\} \\ \to &= \{(\mathrm{x}_{\mathrm{wait}}, \mathrm{x}_{\mathrm{wait}}), (\mathrm{x}_{\mathrm{wait}}, \mathrm{x}_{\mathrm{req}}), (\mathrm{x}_{\mathrm{req}}, \mathrm{x}_{\mathrm{done}}), (\mathrm{x}_{\mathrm{done}}, \mathrm{x}_{\mathrm{wait}})\} \end{aligned}$$

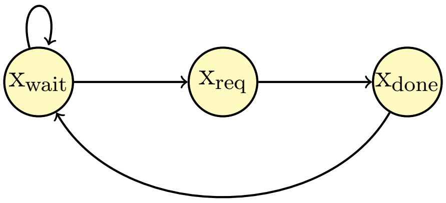


Figure 2.1: Transition system as described in Example 1

We denote by $\xrightarrow{*}$ the reflexive, transitive closure of $\to$, and $\xrightarrow{+}$ denotes the transitive closure of $\to$. For every $x \in X$, the sets of immediate successors and immediate predecessors are defined by:

$$\begin{aligned} \mathit{Pre}_{\mathcal{S}}(x) &= \{y \in X \mid y \to x\} \\ \mathit{Post}_{\mathcal{S}}(x) &= \{y \in X \mid x \to y\} \end{aligned}$$

For every $x \in X$, $\mathit{Post}^*_{\mathcal{S}}(x) = \{y \in X \mid x \xrightarrow{*} y\}$ and $\mathit{Pre}^*_{\mathcal{S}}(x) = \{y \in X \mid y \xrightarrow{*} x\}$ denote respectively the sets of successors and predecessors of x. $\mathit{Post}^*_{\mathcal{S}}(x)$ is also

Pre^*, Post^*

referred to as the reachability set of x . We say that a state y is *reachable* from a state x if $y \in Post^*_{\mathcal{S}}(x)$. We drop the subscript $\mathcal{S}$ when it is clear from context. A transition system $\mathcal{S}$ is finitely branching if for every $x \in X$, $Post(x)$ is finite. It is infinitely branching otherwise. We naturally extend $Post, Pre, Post^*$ and Pre^* to subsets of states, e.g. for $D \subseteq X$ we have $Pre^*(D) = \bigcup_{x \in D} Pre^*(x)$.

reachability set

Often, in order to be more specific while describing transition systems, we label the transitions with actions, which are elements from a finite alphabet. This aids in answering questions about the occurrence of specific transitions. We formally define labelled transition systems (LTS).

Definition 2.2. *A labelled transition system (with an initial state) is a quadruple $\mathcal{S} = (X, A, \rightarrow, init)$ where X is the set of states, A is a finite alphabet, $init \in X$ is the initial state, and $\rightarrow \subseteq X \times A \times X$ is the transition relation.*

labelled transition system

In the case of a labelled transition system, we write $x \xrightarrow{a} x'$ instead of $(x, a, x') \in \rightarrow$. For $x, x' \in X$, let $x \rightarrow x'$ if $x \xrightarrow{a} x'$ for some $a \in A$. For $w \in A^*$, we write $x \xrightarrow{w} x'$ if there is a w-labelled path from x to x'. Formally, $x \xrightarrow{\varepsilon} x'$, where ε is the empty word, if $x = x'$. And $x \xrightarrow{aw} x'$ if there is $y \in X$ such that $x \xrightarrow{a} y$ and $y \xrightarrow{w} x'$. We let $Traces(\mathcal{S}) = \{w \in A^* \mid init \xrightarrow{w} x \text{ for some } x \in X\}$ be the set of traces.

Given $w \in A^*$, we let $Tracereach_{\mathcal{S}}(w) = \{x \in X \mid init \xrightarrow{w} x\}$. Moreover, for $L \subseteq A^*$, $Tracereach_{\mathcal{S}}(L) = \bigcup_{w \in L} Tracereach_{\mathcal{S}}(w)$ is the set of states that are reachable via a word from L. Finally, for a labelled transition system, the *reachability set* of $\mathcal{S}$ is defined as $Reachset_{\mathcal{S}} = Post^*_{\mathcal{S}}(init) = Tracereach_{\mathcal{S}}(A^*)$. We call $\mathcal{S}$ *finite* if $Reachset_{\mathcal{S}}$ is finite (and this is the case if X is finite). Otherwise, $\mathcal{S}$ is called *infinite*.

reachability set for LTS

When we work on purely state problems (hence, labels are meaningless), we will allow ourselves to use an infinite set of actions in order to provide the reader with information about what transitions are used. However, most of this work relies on using labels, hence, we only use a finite set of actions.

2.1 Verification problems for transition systems

We define here formally some of the verification problems that are commonly studied for transition systems. It is important to note that in order to be able to decide these problems for transition systems, at the minimum, we require that the transition relations have some *effectiveness* properties. The most basic condition is that $\rightarrow$ is decidable. This is the classic definition of effectivity for a finite-branching transition system, so if $\rightarrow$ is decidable, we say that the finitely branching transition system $\mathcal{S}$ is effective. However, this is only a necessary condition, not a sufficient one. In this thesis, we refer to this notion as weakly effective to avoid confusion, as we shall see in future chapters stronger effectivity conditions for subclasses of transition systems.

The central problem for the verification of transition systems is reachability, which simply asks whether a state (say, an error state) is reachable from another state (say, an initial state).

Decision Problem: REACHABILITY

Input:	a TS $\mathcal{S} = (X, \rightarrow)$, $x, y \in X$
Question:	is $y \in Post^*_{\mathcal{S}}(x)$?

Reachability is a safety problem, i.e. it only looks at finite transition sequences. We also look at liveness problems. One of these is non-termination, which looks for an infinitely long execution in the system.

Decision Problem: NON-TERMINATION

Input:	a TS $\mathcal{S} = (X, \rightarrow)$, $x_0 \in X$
Question:	is there $x_1, x_2, \ldots \in X$ such that: for all $i \geq 0$, $x_i \rightarrow x_{i+1}$?

Decision Problem: BOUNDEDNESS

Input:	a TS $\mathcal{S} = (X, \rightarrow)$, $x \in X$
Question:	is $Reach_{\mathcal{S}}(x)$ finite?

Note that a finitely branching system can be unbounded only if it does not terminate. Moreover, in many cases, termination reduces to boundedness. However, the converse is not true, as we can find transition systems where termination is decidable but boundedness is not.

2.2 FIFO machines

We now look at some subclasses of transition systems. A FIFO machine $\mathcal{M}$ can be seen as a finite-state automaton equipped with a collection of FIFO (First-In First-Out) channels $c_1, c_2, \ldots$ In this section, we consider FIFO machines having a sequential control graph rather than systems of communicating processes that are distributed systems. We study the latter in Section 2.2.1. It is clear that, given a distributed system, one may compute the Cartesian product of all processes to obtain a FIFO machine (the converse is not always true).

Definition 2.3. *A FIFO machine is a tuple $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$ where Q is a finite set of control-states, $q_0 \in Q$ is an initial control-state, and Ch is a finite set of channels. Moreover, Σ is a finite message alphabet. It is partitioned into $\Sigma = \biguplus_{c \in Ch} \Sigma_c$ where Σ_c contains the messages that can be sent through channel c. Finally, $T \subseteq Q \times Act_{\mathcal{M}} \times Q$ is a transition relation where $Act_{\mathcal{M}} = \{\langle c!m \rangle \mid c \in Ch$ and $m \in \Sigma_c\} \cup \{\langle c?m \rangle \mid c \in Ch$ and $m \in \Sigma_c\}$ is the set of send and receive actions.*

FIFO machine

actions $\langle c!m \rangle, \langle c?m \rangle$

A trace $\tau \in Act_{\mathcal{M}}^*$ is a finite (possibly empty) sequence of actions. We write ε for the empty trace. For a channel c, we let $proj_{c!} : Act_{\mathcal{M}}^* \to \Sigma_c^*$ be the homomorphism defined by $proj_{c!}(\langle c!m \rangle) = m$ for all $m \in \Sigma_c$, and $proj_{c!}(\alpha) = \varepsilon$ if $\alpha \in Act_{\mathcal{M}}$ is not of the form $\langle c!m \rangle$ for some $m \in \Sigma_c$. Moreover, $proj_{c!}(\tau_1 \cdot \tau_2) = proj_{c!}(\tau_1) \cdot proj_{c!}(\tau_2)$.

trace

Furthermore, we let $proj_! : Act_{\mathcal{M}}^* \to \Sigma^*$ be the homomorphism defined by $proj_!(\langle c!m \rangle) = m$ for all $m \in \Sigma$ and $c \in Ch$, and $proj_!(\alpha) = \varepsilon$ if $\alpha \in Act_{\mathcal{M}}$ is not of the form $\langle c!m \rangle$ for some $m \in \Sigma$ and $c \in Ch$. We define $proj_{c?} : Act_{\mathcal{M}}^* \to \Sigma_c^*$ and $proj_? : Act_{\mathcal{M}}^* \to \Sigma^*$ similarly.

Let $\Sigma_{!?} = \{\#m \mid m \in \Sigma$ and $\# \in \{!, ?\}\}$. For $\tau \in Act_{\mathcal{M}}^*$ and channels $c \in Ch$, we write:

- $\tau_c \in Act_{\mathcal{M}}^*$ for the sub-trace of actions $\lambda = \lambda_1 \cdots \lambda_k$ in τ such that $\lambda_i = \langle c\#m \rangle$ for all $1 \leq i \leq k$ and $\# = \{!, ?\}$ and $m \in \Sigma_c$.
- $\mathsf{buffer}_c(\tau)$ for the word $w \in \Sigma^*$, if it exists, such that $proj_{c!}(\tau) = proj_{c?}(\tau) \cdot w$.
- $proj_{!?} : Act_{\mathcal{M}}^* \to \Sigma_{!?}^*$ for the homomorphism defined by $proj_{!?}(\langle c!m \rangle) = !m$ and $proj_{!?}(\langle c?m \rangle) = ?m$ for all $m \in \Sigma$.

A FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$ induces a (potentially infinite) labelled transition system $\mathcal{S}_{\mathcal{M}} = (X_{\mathcal{M}}, Act_{\mathcal{M}}, \to_{\mathcal{M}}, init_{\mathcal{M}})$. Its set of states is $X_{\mathcal{M}} = Q \times \prod_{c \in Ch} \Sigma_c^*$. In $(q, \mathbf{w}) \in X_{\mathcal{M}}$, the first component q denotes the current control-state and $\mathbf{w} = (\mathbf{w}_c)_{c \in Ch}$ determines the contents $\mathbf{w}_c \in \Sigma_c^*$ for every channel $c \in Ch$. The initial state is $init_{\mathcal{M}} = (q_0, \boldsymbol{\varepsilon})$ where $\boldsymbol{\varepsilon} = (\varepsilon, \ldots, \varepsilon)$, i.e., every channel is empty. The transitions are given as follows:

transition system induced by FIFO machine

- $(q,\mathbf{w}) \xrightarrow{\langle c!a\rangle}_{\mathcal{M}} (q',\mathbf{w}')$ if $(q,\langle c!a\rangle,q') \in T$, $\mathbf{w}'_c = \mathbf{w}_c \cdot a$, and $\mathbf{w}'_d = \mathbf{w}_d$ for all $d \in \mathit{Ch} \setminus \{c\}$;
- $(q,\mathbf{w}) \xrightarrow{\langle c?a\rangle}_{\mathcal{M}} (q',\mathbf{w}')$ if $(q,\langle c?a\rangle,q') \in T$, $\mathbf{w}_c = a \cdot \mathbf{w}'_c$, and $\mathbf{w}'_d = \mathbf{w}_d$ for all $d \in \mathit{Ch} \setminus \{c\}$.

stable state

We say a state is *stable* if $\mathbf{w}_c = \varepsilon$ for all $c \in \mathit{Ch}$. The *initial state* $\mathit{init}_{\mathcal{M}}$ of $\mathcal{S}$ is a stable state. The *reachability set* of $\mathcal{M}$ is defined as the reachability set of $\mathcal{S}_{\mathcal{M}}$, i.e., $\mathit{Reachset}_{\mathcal{M}} = \mathit{Reachset}_{\mathcal{S}_{\mathcal{M}}}$. As in the case of transition systems, $\mathit{Traces}(\mathcal{M}) = \mathit{Traces}(\mathcal{S}_{\mathcal{M}}) = \{\sigma \in \mathit{Act}^*_{\mathcal{M}} \mid \mathit{init}_{\mathcal{M}} \xrightarrow{\sigma} x \text{ for some } x \in X_{\mathcal{M}}\}$.

FIFO trace

A trace τ is FIFO (resp. k-bounded FIFO, for $k \geq 1$) if for all $c \in \mathit{Ch}$, for all prefixes τ' of τ, $\mathsf{buffer}_c(\tau')$ is defined (resp. defined and of length at most k); in other words, τ is FIFO if for every prefix τ' of τ, for $c \in \mathit{Ch}$, the sequence of messages received by c in τ' is a prefix of the sequence of messages sent to c. Intuitively, a trace is FIFO if it is an execution of a machine that manipulates FIFO channels. A trace τ is stable if $\mathsf{buffer}_c(\tau) = \varepsilon$ for all $c \in \mathit{Ch}$.

stable trace

synchronous trace

A trace is synchronous if $\mathit{proj}_{!?}(\tau)$ is of the form $!m_1 \cdot ?m_1 \cdot !m_2 \cdot ?m_2 \cdots !m_k \cdot ?m_k$ for some $k \geq 0$ and $m_1,\ldots,m_k \in \Sigma$. In particular, a synchronous trace is a 1-bounded FIFO trace. The converse is not necessarily true. For example, we could have a 1-bounded FIFO trace $\langle c_1!m_1\rangle \cdot \langle c_2!m_2\rangle \cdot \langle c_1?m_1\rangle \cdot \langle c_2?m_2\rangle$ for processes $c_1, c_2 \in \mathit{Ch}$ and $m_1, m_2 \in \Sigma$. Since $\mathit{proj}_{!?}(\tau) = !m_1 \cdot !m_2 \cdot ?m_1 \cdot ?m_2$, the trace is not synchronous.

For $k \geq 1$, we write $\mathit{Traces}_k(\mathcal{S})$ for the set of k-bounded traces of $\mathcal{S}$. Moreover, $\mathit{Traces}_0(\mathcal{S})$ is the set of synchronous traces of $\mathcal{S}$.

2.2.1 Communicating finite-state machines

This section provides preliminaries on communicating finite-state machines (CFMs). In a CFM, a fixed number of finite-state processes communicate by exchanging messages through unbounded FIFO channels. Each process is encoded by an automaton and by abuse of notation, we say that a system is the parallel composition of such processes. In this section, we are looking at peer-to-peer (p2p) systems, where each pair of machines exchange messages via two channels, one per direction of communication. We also look at mailbox, or $*-1$, communication, where each process is limited to a single FIFO channel that it receives from. We will define the model precisely in Section 2.2.3.

set of processes $\mathbb{P}$

Let $\mathbb{P}$ be a finite set of processes. The set of (p2p) channels is $\mathit{Ch} = \{(p,q) \in \mathbb{P} \times \mathbb{P} \mid p \neq q\}$. Let $\Sigma = \biguplus_{c \in \mathit{Ch}} \Sigma_c$ be a set of messages. For $(p,q) \in \mathit{Ch}$, let $\mathit{Send}(p,q) = \{\langle (p,q)!m\rangle \mid m \in \Sigma\}$ and $\mathit{Rec}(p,q) = \{\langle (p,q)?m\rangle \mid m \in \Sigma_{(p,q)}\}$. For $p \in \mathbb{P}$, we set $\mathit{Send}(p, \_) = \{\langle (p,q)!m \rangle \mid q \in \mathbb{P} \setminus \{p\} \text{ and } m \in \Sigma\}$. Similarly, for $m \in \Sigma$, $\mathit{Send}(m) = \{\langle (p,q)!m \rangle \mid (p,q) \in \mathit{Ch}\}$. Moreover, $\mathit{Act}_p = \mathit{Send}(p, \_) \cup \mathit{Rec}(\_, p)$ will denote the set of all actions that are executed by p. Finally, $\mathit{Act} = \bigcup_{p \in \mathbb{P}} \mathit{Act}_p$ is the set of all the actions. For every process $p \in \mathbb{P}$, we let $\Sigma_p = \bigcup_{q \in \mathbb{P}} \Sigma_{(p,q)} \cup \bigcup_{q \in \mathbb{P}} \Sigma_{(q,p)}$. Note that these sets are not disjoint, i.e. $\Sigma_p \cap \Sigma_q = \emptyset$ for $p \neq q$.

Definition 2.4. *A communicating system over $\mathbb{P}$ is a tuple $\mathcal{A} = ((\mathcal{P}_p)_{p \in \mathbb{P}}, \Sigma)$. For each $p \in \mathbb{P}$, $\mathcal{P}_p = (Q_p, \Sigma_p, \delta_p, \ell_{0_p})$ is a finite labelled transition system where Q_p is a finite set of local (control) states, $\delta_p \subseteq Q_p \times \mathit{Act}_p \times Q_p$ is the transition relation, and $\ell_{0_p} \in Q_p$ is the initial state. Moreover, $\Sigma = \bigcup_{p \in \mathbb{P}} \Sigma_p$.*

communicating finite-state machine (CFM)

Given $p \in \mathbb{P}$ and a transition $t = (\ell, \alpha, \ell') \in \delta_p$, we let $\mathit{source}(t) = \ell$, $\mathit{target}(t) = \ell'$, $\mathit{action}(t) = \alpha$, and $\mathit{msg}(t) = m$ if $\alpha \in \mathit{Send}(m) \cup \mathit{Rec}(m)$.

We now define the operational semantics of a communicating system $\mathcal{A}$ as the transition system of the FIFO machine $\mathcal{M}_{\mathcal{A}} = (Q_{\mathcal{A}}, \mathit{Ch}, \Sigma, T_{\mathcal{A}}, \ell_{0_{\mathcal{A}}})$. We define $\mathcal{M}_{\mathcal{A}}$ as follows. The set of control-states is the Cartesian product of the sets of local states of each process, $Q_{\mathcal{A}} = \prod_{p \in \mathbb{P}} Q_p$. Moreover, $\ell_{0_{\mathcal{A}}} = (\ell_{0_p})_{p \in \mathbb{P}}$ and $T_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \mathit{Act} \times Q_{\mathcal{A}}$. The set of transitions can be described as follows: there is a transition $((\ell_p)_{p \in \mathbb{P}}, \alpha, (\ell'_p)_{p \in \mathbb{P}}) \in T_{\mathcal{A}}$ if $(\ell_p, \alpha, \ell'_p) \in \delta_p$ for some $p \in \mathbb{P}$ and for all $q \neq p$, $\ell_q = \ell'_q$.

Example 2 (Connection-Deconnection Protocol)**.** A model for the (simplified) connection-deconnection protocol, CDP, between two processes is described as follows (see Figure 2.2): We model the protocol with two processes and two (unbounded) channels. The first process (on the left) can open a session (this is denoted by sending the message "a" through channel c_1 to the other process). Once a session is open, the first process can close it (by sending message "b" to the other process), or on the demand of the second process (if it receives the message "e"). This protocol has been studied in [JR67, Jer91].

In the example, it is natural to consider a system of two processes p, q. However, we formalize this in terms of the associated FIFO machine. That is, the CDP is modelled as the FIFO machine $\mathcal{M}_{\mathcal{A}} = (Q_{\mathcal{A}}, \mathit{Ch}, \Sigma, T_{\mathcal{A}}, \ell_{0_{\mathcal{A}}})$ where $Q_{\mathcal{A}} = \{0,1\} \times \{0,1\}$ (the Cartesian product of the local states) with initial state $\ell_{0_{\mathcal{A}}} = (0,0)$, $\mathit{Ch} = \{(p,q), (q,p)\}$, $\Sigma = \Sigma_{(p,q)} \uplus \Sigma_{(q,p)}$ with $\Sigma_{(p,q)} = \{a, b\}$ and $\Sigma_{(q,p)} = \{e\}$. Moreover, the transition relation T contains, amongst others, $((0,0), \langle (p,q)!a \rangle, (1,0))$ and $((1,0), \langle (p,q)?a \rangle, (1,1))$.

2.2.2 Message sequence charts

Another (equivalent) way to define the semantics of a communicating system is to consider graphically the interactions between processes. A CFM accepts/generates

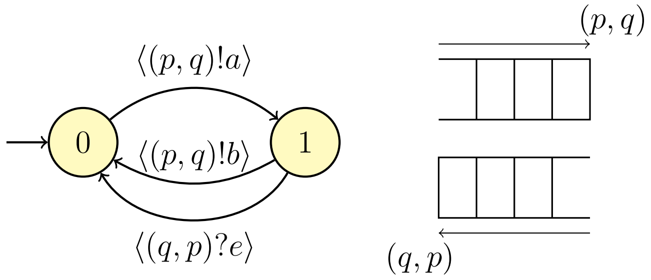

$\langle (p,q)!a \rangle$
$\langle (p,q)!b \rangle$
$\langle (q,p)?e \rangle$
0
1
(p,q)
(q,p)
$\langle (p,q)?a \rangle$
$\langle (p,q)?b \rangle$
$\langle (q,p)!e \rangle$
0
1

Figure 2.2: The model of the connection-deconnection protocol

message-sequence charts (MSCs), which are similar to UML's sequence diagrams [LR11]. MSCs are equipped with Lamport's happened-before relation: we say that an event e happens before an event f if there is a "path" from e to f [Lam78]. Additional binary predicates connect the emission of a message with its reception, and successive events executed by one and the same process. We shall first define the notion of MSCs more formally.

message sequence chart (MSC) A p2p MSC (or simply MSC) over a set of processes $\mathbb{P}$ and message set Σ is a tuple $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ where $\mathcal{E}$ is a finite (possibly empty) set of events and $\lambda : \mathcal{E} \rightarrow Act$ is a labelling function. For $p \in \mathbb{P}$, let $\mathcal{E}_p = \{e \in \mathcal{E} \mid \lambda(e) \in Act_p\}$ be the set of events that are executed by p. We require that $\rightarrow$ (the process relation) *relation $\rightarrow_p$* is the disjoint union $\bigcup_{p \in \mathbb{P}} \rightarrow_p$ of relations $\rightarrow_p \subseteq \mathcal{E}_p \times \mathcal{E}_p$ such that $\rightarrow_p$ is the direct successor relation of a total order on $\mathcal{E}_p$. For an event $e \in \mathcal{E}$, a set of actions $A \in Act$, and a relation $R \subseteq \mathcal{E} \times \mathcal{E}$, let $\#A(R, e) = |\{f \in \mathcal{E} \mid (f, e) \in R \text{ and } \lambda(f) \in A\}|$. *relation $\lhd$* We require that $\lhd \subset \mathcal{E} \times \mathcal{E}$ (the message relation) satisfies the following:

FIFO condition

(1) for every pair $(e, f) \in \lhd$, there is a send action $\langle (p,q)!m \rangle$ such that $\lambda(e) = \langle (p,q)!m \rangle$, $\lambda(f) = \langle (p,q)?m \rangle$, and $\#Send(p,q)(\rightarrow^+, e) = \#Rec(p,q)(\rightarrow^+, f)$,

(2) for all $f \in \mathcal{E}$ such that $\lambda(f)$ is a receive action, there is $e \in \mathcal{E}$ such that $e \lhd f$.

relation $\leq_M$ Finally, letting $\leq_M = (\rightarrow \cup \lhd)^*$, we require that $\leq_M$ is a partial order.

Condition (1) above ensures that every (p2p) channel (p,q) behaves in a FIFO manner. By Condition (2), every receive event has a matching send event. Note that, however, there may be unmatched send events in an MSC. We let $SendEv(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a send action}\}$, $RecEv(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a receive action}\}$, *$Matched(M)$* $Matched(M) = \{e \in \mathcal{E} \mid \text{there is } f \in \mathcal{E} \text{ such that } e \lhd f\}$, and *$Unm(M)$* $Unm(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a send action and there is no } f \in \mathcal{E} \text{ such that } e \lhd f\}$. We do not distinguish isomorphic MSCs and let MSC be the set of all MSCs over the given sets $\mathbb{P}$ and Σ.

Example 3. For a set of processes $\mathbb{P} = \{p, q, r\}$ and a set of messages $\Sigma = \{m_1, m_2, m_3, m_4\}$, $M_1 = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is an MSC where, for example, $e_2 \lhd e'_2$ and

$e_3' \to e_4$ (See Figure 2.3). The dashed arrow means that the send event e_1 does not have a matching receive, so $e_1 \in \mathit{Unm}(M_1)$. Moreover, $e_2 \leq_{M_1} e_4$, but $e_1 \not\leq_{M_1} e_4$. We can find a total order $\rightsquigarrow \supseteq \leq_{M_1}$ such that $e_1 \rightsquigarrow e_2 \rightsquigarrow e_2' \rightsquigarrow e_3 \rightsquigarrow e_3' \rightsquigarrow e_4 \rightsquigarrow e_4'$. We call $\rightsquigarrow$ a linearization, which is formally defined below.

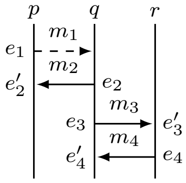


Figure 2.3: MSC M_1

Consider $M = (\mathcal{E}, \to, \lhd, \lambda) \in \mathsf{MSC}$. A *p2p linearization* (or simply *linearization*) of M is a (reflexive) total order $\rightsquigarrow \subseteq \mathcal{E} \times \mathcal{E}$ such that $\leq_M \subseteq \rightsquigarrow$. *linearization*

Let $M = (\mathcal{E}, \to, \lhd, \lambda) \in \mathsf{MSC}$ and consider $E \subseteq \mathcal{E}$ such that E is $\leq_M$-*downward-closed*, i.e, for all $(e, f) \in \leq_M$ such that $f \in E$, we also have $e \in E$. Then, the MSC $(E, \to \cap (E \times E), \lhd \cap (E \times E), \lambda')$, where λ' is the restriction of $\mathcal{E}$ to E, is called a *prefix* of M. In particular, the empty MSC is a prefix of M. We denote the set of prefixes of M by $\mathit{Pref}(M)$. This is extended to sets $L \subseteq \mathsf{MSC}$ as expected, letting $\mathit{Pref}(L) = \bigcup_{M \in L} \mathit{Pref}(M)$.

Let $M_1 = (\mathcal{E}_1, \to_1, \lhd_1, \lambda_1)$ and $M_2 = (\mathcal{E}_2, \to_2, \lhd_2, \lambda_2)$ be two MSCs. Their *concatenation* $M_1 \cdot M_2 = (\mathcal{E}, \to, \lhd, \lambda)$ is defined if, for all $(p, q) \in \mathit{Ch}$, $e_1 \in \mathit{Unm}(M_1)$, and $e_2 \in \mathcal{E}_2$ such that $\lambda(e_1) \in \mathit{Send}(p, q)$ and $\lambda(e_2) \in \mathit{Send}(p, q)$, we have $e_2 \in \mathit{Unm}(M_2)$. As expected, $\mathcal{E}$ is the disjoint union of $\mathcal{E}_1$ and $\mathcal{E}_2$, $\lhd = \lhd_1 \cup \lhd_2$, λ is the "union" of λ_1 and λ_2, and $\to = \to_1 \cup \to_2 \cup R$. Here, R contains, for all $p \in \mathbb{P}$ such that $(\mathcal{E}_1)_p$ and $(\mathcal{E}_2)_p$ are non-empty, the pair (e_1, e_2) where e_1 is the maximal p-event in M_1 and e_2 is the minimal p-event in M_2. Note that $M_1 \cdot M_2$ is indeed an MSC and that concatenation is associative.

Now, given a communicating system $\mathcal{S}$, we will define the language of $\mathcal{S}$ directly as a set of MSCs. This semantic view is essentially equivalent to the transition system view we defined earlier, but they have different advantages depending on the context.

Let $M = (\mathcal{E}, \to, \lhd, \lambda)$ be an MSC. A *run* of $\mathcal{S}$ on M is a mapping $\rho : \mathcal{E} \to \bigcup_{p \in \mathbb{P}} \delta_p$ that assigns to every event e the transition $\rho(e)$ that is executed at e. Thus, we require that *run on MSC*

(i) for all $e \in \mathcal{E}$, we have $\mathit{action}(\rho(e)) = \lambda(e)$,

(ii) for all $(e, f) \in \to$, $\mathit{target}(\rho(e)) = \mathit{source}(\rho(f))$,

(iii) for all $(e, f) \in \lhd$, $msg(\rho(e)) = msg(\rho(f))$, and

(iv) for all $p \in \mathbb{P}$ and $e \in \mathcal{E}_p$ such that there is no $f \in \mathcal{E}$ with $f \rightarrow e$, we have $source(\rho(e)) = \ell_{0_p}$.

$L_{\mathsf{p2p}}(\mathcal{S})$ The *(p2p) language* of $\mathcal{S}$ is $L_{\mathsf{p2p}}(\mathcal{S}) = \{M \in \mathsf{MSC} \mid \text{there is a run of } \mathcal{S} \text{ on } M\}$. Note that, as in [BEJQ18, DGLL20], we do not consider final states, as our purpose is to reason about all possible traces that can be generated by $\mathcal{S}$. Moreover, we see that $L_{\mathsf{p2p}}(\mathcal{S})$ is prefix-closed, i.e. $Pref(L_{\mathsf{p2p}}(\mathcal{S})) \subseteq L_{\mathsf{p2p}}(\mathcal{S})$.

2.2.3 Mailbox semantics

A different model of communication is with the mailbox semantics, also known as $* - 1$ mailbox, where every process receives from a single channel. The notions we had for the peer-to-peer case can be directly extended for the mailbox case as well. However, we need to add constraints to the MSC representation in order to accommodate for this change.

mailbox semantics **Definition 2.5.** *Let $\sqsubset_M \subseteq \mathcal{E} \times \mathcal{E}$ be defined by: $e_1 \sqsubset_M e_2$ if there is $q \in \mathbb{P}$ such that $\lambda(e_1) \in Send(\_, q)$, $\lambda(e_2) \in Send(\_, q)$, and one of the following holds:*

- *$e_1 \in Matched(M)$ and $e_2 \in Unm(M)$, or*
- *$e_1 \lhd f_1$ and $e_2 \lhd f_2$ for some $f_1, f_2 \in \mathcal{E}_q$ such that $f_1 \rightarrow^+ f_2$.*

relation $\preceq_M$ We let $\preceq_M = (\rightarrow \cup \lhd \cup \sqsubset_M)^*$. Note that $\leq_M \subseteq \preceq_M$. We call $M \in \mathsf{MSC}$ a *mailbox MSC* if $\preceq_M$ is a partial order. Intuitively, this means that events can be scheduled in a way that corresponds to the mailbox semantics, i.e., with one incoming channel per process. Following the terminology in [BEJQ18], we also say that a mailbox MSC satisfies *causal delivery.* The set of mailbox MSCs $M \in \mathsf{MSC}$ is denoted by $\mathsf{MSC}_{\mathsf{mb}}$.

Example 4. MSC M_1 is a mailbox MSC. Indeed, even though the order $\rightsquigarrow$ defined in Example 3 does not respect all mailbox constraints, particularly the fact that $e_4 \sqsubset_{M_1} e_1$, there is a total order $\rightsquigarrow \supseteq \preceq_{M_1}$ such that $e_2 \rightsquigarrow e_3 \rightsquigarrow e'_3 \rightsquigarrow e_4 \rightsquigarrow e_1 \rightsquigarrow e'_2 \rightsquigarrow e'_4$. We call $\rightsquigarrow$ a mailbox linearization, which is formally defined below.

Similar to the notion of a *p2p* linearization, a *mailbox linearization* of M is a total order $\rightsquigarrow \subseteq \mathcal{E} \times \mathcal{E}$ such that $\preceq_M \subseteq \rightsquigarrow$. That is, every mailbox linearization is a *p2p* linearization, but the converse is not necessarily true (Example 4). Note that an MSC is a mailbox MSC iff it has at least one mailbox linearization.

Lemma 2.6. *Every prefix of a mailbox MSC is a mailbox MSC.*

Proof. Let $M = (\mathcal{E}, \to, \lhd, \lambda) \in \mathsf{MSC_{mb}}$ and $M_0 = (\mathcal{E}_0, \to_0, \lhd_0, \lambda_0)$ be a prefix of M, i.e., $\mathcal{E}_0 \subseteq \mathcal{E}$. By contradiction, suppose that M_0 is not a mailbox MSC. Then, there are distinct $e, f \in \mathcal{E}_0$ such that $e \preceq_{M_0} f \preceq_{M_0} e$ with $\preceq_{M_0} = (\to_0 \cup \lhd_0 \cup \sqsubset_{M_0})^*$. As $\mathcal{E}_0 \subseteq \mathcal{E}$, we have that $\to_0 \subseteq \to$, $\lhd_0 \subseteq \lhd$, and $\sqsubset_{M_0} \subseteq \sqsubset_M$. Finally, $\preceq_{M_0} \subseteq \preceq_M$ and M is not a mailbox MSC, which is a contradiction. □

We concatenate mailbox MSCs in the same way as their *p*2*p* counterparts. Moreover, concatenation is associative for mailbox MSCs as well.

Similar to the notion of the *p*2*p* language of a communicating system $\mathcal{S}$, we define the mailbox language as follows. The *mailbox language* of $\mathcal{S}$ is $L_{\mathsf{mb}}(\mathcal{S}) = \{M \in \mathsf{MSC_{mb}} \mid \text{there is a run of } \mathcal{S} \text{ on } M\}$.

$L_{\mathsf{mb}}(\mathcal{S})$

And finally, from Lemma 2.6, we obtain prefix closure for mailbox systems as well.

Lemma 2.7. *For all* $\mathrm{com} \in \{\mathsf{p2p}, \mathsf{mb}\}$, $L_{\mathrm{com}}(\mathcal{S})$ *is prefix-closed: Pref*$(L_{\mathrm{com}}(\mathcal{S})) \subseteq L_{\mathrm{com}}(\mathcal{S})$.

Example 5. Fig. 2.4 depicts $\mathcal{A}_1 = (A_p, A_q, A_r)$ such that MSC M_1 in Fig. 2.3 belongs to $L_{\mathsf{p2p}}(\mathcal{A}_1)$ and to $L_{\mathsf{mb}}(\mathcal{A}_1)$. There is a unique run ρ of $\mathcal{A}_1$ on M_1. We can see that $(e'_3, e_4) \in \to$ and $target(\rho(e'_3)) = source(\rho(e_4)) = \ell^1_r$, $(e_2, e'_2) \in \lhd_{M_1}$, and $msg(\rho(e_2)) = msg(\rho(e'_2)) = m_2$.

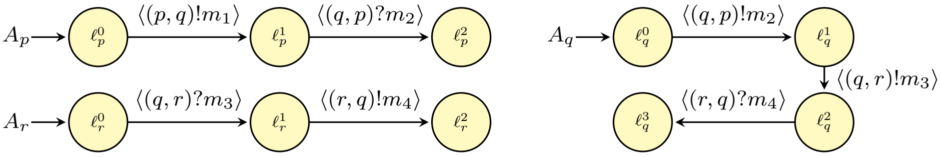

A_p
ℓ^0_p
$\langle(p,q)!m_1\rangle$
ℓ^1_p
$\langle(q,p)?m_2\rangle$
ℓ^2_p
A_q
ℓ^0_q
$\langle(q,p)!m_2\rangle$
ℓ^1_q
$\langle(q,r)!m_3\rangle$
ℓ^2_q
$\langle(r,q)?m_4\rangle$
ℓ^3_q
A_r
ℓ^0_r
$\langle(q,r)?m_3\rangle$
ℓ^1_r
$\langle(r,q)!m_4\rangle$
ℓ^2_r

Figure 2.4: System $\mathcal{A}_1$

2.3 Counter machines

Counter machines, also known as Minsky machines, are finite-state machines that manipulate counters, which are variables that store non-negative integers. Transitions of a counter machine, besides changing control-states, perform a specified operation on a counter: increment by one, decrement by one, along with a set of counters to be tested for zero. We define formally a counter machine below.

Definition 2.8. *A* counter machine *(with zero tests) is a tuple* $\mathcal{C} = (Q, V, T, q_0)$. *Here, Q is the finite set of* control-states *and* $q_0 \in Q$ *is the* initial control-state. *Moreover, V is a finite set of* counters *and* $T \subseteq Q \times Act_{\mathcal{C}} \times Q$ *is the transition*

counter machine

relation where $Act_{\mathcal{C}} = \{\mathsf{inc}(v), \mathsf{dec}(v), \mathsf{noop} \mid v \in V\} \times 2^V$ *(an element of* 2^V *will indicate the set of counters to be tested to* 0*).*

transition system induced by counter machine

The counter machine $\mathcal{C}$ induces an LTS $\mathcal{S}_{\mathcal{C}} = (X_{\mathcal{C}}, Act_{\mathcal{C}}, \rightarrow_{\mathcal{C}}, init)$ with set of states $X_{\mathcal{C}} = Q \times \mathbb{N}^V$. In $(q, \ell) \in X_{\mathcal{C}}$, the first component q is the current control-state and $\ell = (\ell_v)_{v \in V}$ represents the counter values. The initial state is then $init = (q_0, \ell_0)$ with all $\ell_0 = (0, 0, \dots, 0)$.For $op \in \{\mathsf{inc}, \mathsf{dec}\}$, $v \in V$, and $Z \subseteq V$ (the counters tested for zero), there is a transition $(q, \ell) \xrightarrow{op(v),Z}_{\mathcal{C}} (q', m)$ if $(q, (op(v), Z), q') \in T$, $\ell_{v'} = 0$ for all $v' \in Z$ (applies the zero tests), $m_v = \ell_v + 1$ if $op = \mathsf{inc}$ and $m_v = \ell_v - 1$ if $op = \mathsf{dec}$, and $m_{v'} = \ell_{v'}$ for all $v' \in V \setminus \{v\}$.

For $op = \mathsf{noop}$, and $Z \subseteq V$, there is a transition $(q, \ell) \xrightarrow{op,Z}_{\mathcal{C}} (q', m)$ if $(q, (op, Z), q') \in T$, $\ell_{v'} = 0$ for all $v' \in Z$ (applies the zero tests), and $m_v = \ell_v$ for all $v \in V$. We sometimes omit writing noop and label the transition with only the set of counters to be tested to zero, or we write $\mathsf{zero}(Z)$. Similarly, we omit Z if $Z = \emptyset$.

The model of (deterministic) counter machines is Turing-complete, and therefore all the verification problem we looked at in Section 2.1 is undecidable (even for machines with 2 counters). Hence, in this thesis, we look at a weakening of this model, which we refer to as counter machines with restricted zero tests.

counter machine with restricted zero tests (CMRZ)

Counter machines with restricted zero tests. We define *counter machines with restricted zero tests (CMRZ)* imposing the following requirement: Once a counter has been tested for zero, it cannot be incremented or decremented any more. Formally, we say that $\mathcal{C}$ is a counter machine with restricted zero tests if for all transition sequences of the form $q_0 \xrightarrow{op(v_1),Z_1} q_1 \xrightarrow{op(v_2),Z_2} \dots \xrightarrow{op(v_{n-1},Z_{n-1})} q_{n-1} \xrightarrow{op(v_n),Z_n} q_n$, for every two positions $1 \leq i \leq j \leq n$, we have $v_j \notin Z_i$.

2.4 Finite-state automata

finite-state automaton

We now formally recall the notion of a finite-state automaton (also known as finite-state machine or FSM in the literature). A (deterministic) finite-state automaton (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$, where Q is the set of control-states, Σ is the input alphabet, $q_0 \in Q$ is the initial state, and $\mathcal{F} \subseteq Q$ is the set of final (or accepting) states. We define the transition relation as $\delta : Q \times \Sigma \rightarrow Q$.

A non-deterministic finite-state automaton (NFA) is the same as a DFA, except that the transition relation is $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q. We say that a word w is accepted by a DFA $\mathcal{A}$ if and only if starting from q_0, it can read w (according to the transition function) and reach a state $q \in \mathcal{F}$. By abuse of notation, we denote this as $\delta^*(q_0, w) = q$. The language of an automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of all accepted words.

complete DFA
trimmed DFA

We call a DFA $\mathcal{A}$ complete if for every $q \in Q$ and for every $m \in \Sigma$, $\delta(q, m)$ is defined. Moreover, we call a DFA trimmed if for every state $q \in Q$, there exists a word $w \in \Sigma^*$ such that $\delta^*(q, w) = q_f$ where $q_f \in \mathcal{F}$, i.e. from every state of the DFA, we can reach a final state. Note that given an untrimmed DFA $\mathcal{A}$, we can construct a trimmed DFA $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. We do this by removing all the states (and associated transitions) that do not reach a final state, and doing this does not alter the language of the automaton.

Chapter 3

Branch-Well Structured Transition Systems

Let us recall that a well-structured transition system (WSTS) is an ordered transition system $\mathcal{S} = (X, \rightarrow, \leq)$ such that the set X is quasi-ordered by $\leq$, and $\rightarrow$ fulfils one of various possible monotonies with respect to $\leq$. The quasi-ordering of X is further assumed to be well, i.e. well-founded and with no infinite antichains.

In this chapter, we introduce branch-well structured transition systems (branch-WSTS). Branch-WSTS is a relaxation of general WSTSs. In the case of branch-WSTS, the set of states in the same branch of the reachability tree forms a well-quasi-ordered set, and monotony is enforced only between states reachable one from another.

As boundedness and termination are decidable for WSTS with some effectivity conditions, it is a model that has been studied since its inception in [Fin90]. We show that the branch relaxation still preserves the decidability of boundedness and termination if the underlying system is effective.

We show that for branch-WSTS, the coverability problem is undecidable, which leads to another definition of monotony, the so-called cover-monotony, still a relaxation of the monotony condition. We call the class of ordered transition systems cover-WSTS if the ordering is a wqo and the monotony enforced is the cover-monotony. For effective cover-WSTS $\mathcal{S} = (X, \rightarrow, \leq)$, i.e. systems such that X is a well-quasi-ordered set over $\leq$, and $\rightarrow$ is cover-monotone with respect to $\leq$, the decidability of coverability from the initial state is still decidable, using the forward algorithm from [BFM17].

We also explore the systems which fall into these two WSTS classes, including (normalized) input-bounded FIFO systems and counter machines with restricted zero tests [BFS20], which are branch-WSTS, and a class of counter systems which are cover-WSTS.

3.1 Orderings

quasi, partial, and total ordering

antichain

Let X be a set and $\leq \subseteq X \times X$ a binary relation over X, which we also write as $(X, \leq)$. We call $\leq$ a *quasi-ordering* if it is reflexive and transitive. When a set X is equipped with a quasi-ordering $\leq$, we refer to it as a qo $(X, \leq)$. A *partial ordering* is a quasi-ordering that is also anti-symmetric (if $x \leq y$ and $y \leq x$, then $x = y$.) Furthermore, it is a total ordering (sometimes also known as a linear ordering) if for any $x, y \in X$, we either have $x \leq y$ or $y \leq x$. We write $x < y$ if $x \leq y$ and $y \not\leq x$. If $\leq$ is a partial ordering, $x < y$ is then equivalent to $x \leq y$ and $x \neq y$. An *antichain* of $(X, \leq)$ is a subset $Y \subseteq X$ such that elements of Y are pairwise incomparable (i.e. for all $y, y' \in Y$, $y \neq y' \implies y \not\leq y'$).

upward, downward closure

basis

Given $(X, \leq)$ a (quasi-)ordered set, the *upward closure* (in X) of a set $E \subseteq X$ is $\uparrow E = \{y \in X \mid \exists x \in E, x \leq y\}$ and conversely the *downward closure* of E is $\downarrow E = \{y \in X \mid \exists x \in E, y \leq x\}$. E is *upward-closed* (resp. *downward-closed*) if $E = \uparrow E$ (resp. $E = \downarrow E$). A downward-closed (resp. upward-closed) set E has a *basis* B if $E = \downarrow B$ (resp. $E = \uparrow B$). E has a *finite basis* if B can be chosen finite. For example, let $X \subseteq \mathbb{N}^2$ such that $X = \{(x, y) \mid x + y \geq 1\}$. X can be expressed as the upward closure of the finite basis $B = \{(0, 1), (1, 0)\}$, and

$$X = \uparrow\{(0, 1), (1, 0)\}.$$

An *upper bound* of $E \subseteq X$, if it exists, is an element $x \in X$ such that $y \leq x$ for every $y \in E$. The *least upper bound* of a set E, if it exists, is written $lub(E)$.

3.1.1 Well-orderings

well-quasi-ordered set (wqo)

A quasi-ordered set $(X, \leq)$ is *well founded* if there is no infinite strictly decreasing sequence $x_1 > x_2 > \ldots$ of elements of X. It is *well* if there is also no infinite antichain. There are equivalent formulations for the definition of well-quasi-ordered set (wqo for short), as given by the following proposition:

Proposition 3.1. *[Kru72] Given a quasi-ordered set $(X, \leq)$, the following properties are equivalent:*

- *$(X, \leq)$ is well-quasi-ordered.*
- *$(X, \leq)$ is well founded and there is no infinite antichain in $(X, \leq)$.*
- *From any infinite sequence $x_0, x_1, x_2 \ldots$ one can find $i < j$ such that $x_i \leq x_j$.*
- *From any infinite sequence $x_0, x_1, x_2 \ldots$ one can extract an infinite increasing subsequence.*
- *Any non-empty upward closed subset of X admits a finite basis.*

3.1.2 Default orderings

Some common sets and the associated orderings that we study are listed below.

- The set of natural numbers $\mathbb{N}$ and integers $\mathbb{Z}$ are ordered by their canonical ordering $\leq$. Moreover, $(\mathbb{N}, \leq)$ is a wqo. On the other hand, $(\mathbb{Z}, \leq)$ is not a wqo as we can have an infinite decreasing sequence, e.g. $0 > -1 > -2 > \dots$

- $(\mathbb{N}^k, \leq)$, i.e. the set of vectors of $k \geq 1$ natural numbers with component-wise ordering, is a wqo [Dic13]. More generally, if $(X, \leq)$ is wqo, then the set of vectors of k elements of X with the component-wise ordering, $(X^k, \leq)$, is also a wqo.

- The set of finite words over a finite alphabet Σ, i.e. Σ^*, along with the prefix ordering, which we denote by $\preceq$, where $u \preceq w$ iff $w = u \cdot v$ for $u, v, w \in \Sigma^*$. $(\Sigma^*, \preceq)$ is not a wqo if $|\Sigma| \geq 2$ as we can find an infinite antichain. For example, let $\Sigma = \{a, b\}$. Then, $a \not\preceq ba \not\preceq bba \not\preceq \dots$ is an infinite sequence of incomparable elements.

  prefix ordering $\preceq$

- We can also define the subword ordering on the set of finite words over a finite alphabet Σ, which we will denote by $\leq_{\mathsf{sw}}$. $(\Sigma^*, \leq_{\mathsf{sw}})$ is a wqo, as in [Hig52].

  subword ordering $\leq_{\mathsf{sw}}$

- We sometimes use the *extended ordering*, when talking about states which are in fact, tuples, with the first component being the control-state and the other component(s) being integers, words, etc. In this case, the ordering is equality on the control-states, and the (component-wise) canonical ordering on the other components. For example, the extended prefix ordering $\leq_p$ is defined on $Q \times (\Sigma^*)^k$ by $(q, \mathbf{w}) \leq_p (q', \mathbf{w}')$ if $q = q'$ and for all $1 \leq i \leq k$, one have $\mathbf{w}_i \preceq \mathbf{w}'_i$, where $\mathbf{w} = (\mathbf{w}_i)_{1 \leq i \leq k}$, $\mathbf{w}' = (\mathbf{w}'_i)_{1 \leq i \leq k}$.

  extended prefix ordering $\leq_p$

3.2 Well-structured transition systems

An important class of transition systems that we will consider are well-structured transition systems, or WSTS for short, for which many properties are known to be decidable.

A (well-)ordered transition system is a tuple $\mathcal{S} = (X, \rightarrow, \leq)$ consisting of a transition system $\mathcal{S} = (X, \rightarrow)$, equipped with a (well) qo $(X, \leq)$. We extend these definitions for labelled transition systems: An ordered labelled transition system (OLTS) is a tuple $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$ consisting of a labelled transition system $\mathcal{S} = (X, A, \rightarrow, \mathit{init})$, equipped with a qo $(X, \leq)$.

ordered labelled transition system (OLTS)

There are different monotony properties that the transition relation can fulfil. We define here the most important ones.

well-structured transition system (WSTS)

Definition 3.2. *[FS01] A well-structured transition system (WSTS) is a well-ordered transition system $\mathcal{S} = (X, \rightarrow, \leq)$ that satisfies monotony: for all $x, x', y \in X$, we have: $x \leq y \wedge x \rightarrow x' \implies \exists y' \in X\colon x' \leq y' \wedge y \xrightarrow{*} y'$.*

We define a labelled WSTS as an OLTS such that the ordering is well and it satisfies the monotony condition: for all $x, x', y \in X$ and $a \in A$, we have: $x \leq y \wedge x \xrightarrow{a} x' \implies \exists y' \in X$ and $w \in A^*$: $x' \leq y' \wedge y \xrightarrow{w} y'$.

other monotonies

Let us define the other types of monotony. We say that a well-ordered transition system $\mathcal{S} = (X, \rightarrow, \leq)$ satisfies strong monotony (resp., transitive monotony) if, for all $x, y, x' \in X$ such that $x \leq y$ and $x \rightarrow x'$, there is $y' \in X$ such that $x' \leq y'$ and $y \rightarrow y'$ (resp., $y \xrightarrow{+} y'$). The transition system $\mathcal{S}$ satisfies strict monotony if, for all $x, y, x' \in X$ such that $x < y$ and $x \rightarrow x'$, there is $y' \in X$ such that $x' < y'$ and $y \xrightarrow{*} y'$. We can similarly define these monotonies for ordered labelled transition systems.

Decidability of verification problems for WSTS. In order to be able to decide the problems defined in Section 2.1 for WSTS, we need to require that the transition systems have some effectiveness properties. Throughout this work, WSTS will be assumed *effective* in the following sense:

effectivity conditions for WSTS

1. the set of states X is recursively enumerable (which suffices to compute $Post_{\mathcal{S}}(x)$ when $|Post_{\mathcal{S}}(x)|$ is known and finite),
2. the transition relation is decidable, i.e., the WSTS comes equipped with an algorithm that can decide, given $x, y \in X$, whether $x \rightarrow y$ or equivalently whether $y \in Post_{\mathcal{S}}(x)$,
3. the quasi-ordering $\leq$ is decidable, i.e., the WSTS also comes equipped with an algorithm that can decide, given $x, y \in X$, whether $x \leq y$.

3.3 Branch-WSTS

As we saw in Section 2.1, boundedness and termination are important problems in the study of transition systems, and they are decidable for a large class of systems. In this section, we propose a relaxation to the definition of well-structured transition systems (WSTS), and still retain the decidability of boundedness and termination. Let us define the model formally.

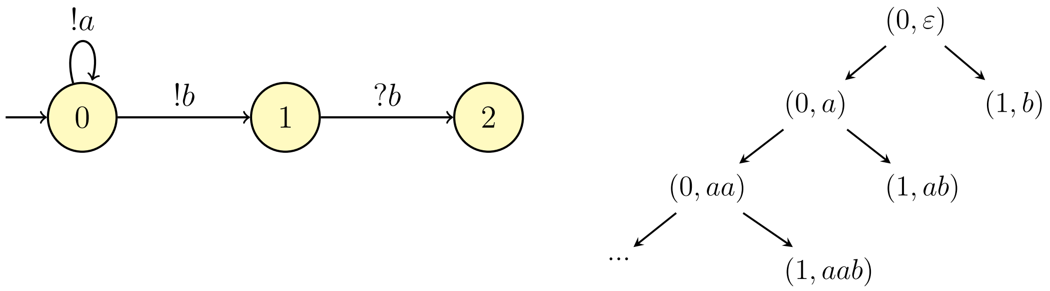


Figure 3.1: The FIFO machine $\mathcal{M}_1$ (left), and its corresponding (incomplete) infinite reachability tree (right).

Let $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$ be an ordered labelled transition system. A *run*, or a *branch*, of $\mathcal{S}$ is a finite or infinite sequence $\rho = (x_0 \rightarrow x_1)(x_1 \rightarrow x_2)\ldots$, simply written $x_0 \rightarrow x_1 \rightarrow x_2 \ldots$. *branch*

Branch-wqo

Consider an ordered labelled transition system, $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$. We say that ρ is *branch-wqo* if the set of states $\{x_0, x_1, x_2, \ldots\}$ visited along ρ is wqo with respect to $\leq$.

Definition 3.3. *An ordered labelled transition system $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$ is branch-wqo if every run of $\mathcal{S}$ starting from init is branch-wqo.* *branch-wqo*

Example 6. Consider the FIFO machine $\mathcal{M}_1$ in Figure 3.1 with one FIFO channel c. In control-state 0, it makes a loop by sending letter a to the channel. Then, we may go, non-deterministically, to control-state 1 by sending letter b once, and then we either stop or consume a b and go to 2 if possible. Let us consider the set of states $X_1 = \{0, 1\} \times \{a, b\}^*$ together with the extended prefix ordering $\leq_p$. The reachability set of $\mathcal{M}_1$ from $(0, \varepsilon)$ is equal to $\{(0, w), (1, w') \mid w \in a^*, w' \in a^*b\}$. Note that $\leq_p$ is not a wqo since elements of the set $\{(1, w) \mid w \in a^*b\}$ forms an infinite antichain for $\leq_p$. However, every branch in the reachability tree of $\mathcal{M}_1$ is branch-wqo for the initial state $(0, \varepsilon)$. Hence, there exist branch-wqo ordered labelled transition systems $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$ such that $(X, \leq)$ is not a wqo.

Note that the property of branch-wqo for a given system depends on the initial state. There could be a system $\mathcal{S} = (X, A, \rightarrow, \mathit{init}, \leq)$ and $\mathit{init}' \in X$ such that $\mathcal{S}$ is branch-wqo but $\mathcal{S} = (X, A, \rightarrow, \mathit{init}', \leq)$ is not branch-wqo (cf. Figure 3.2).

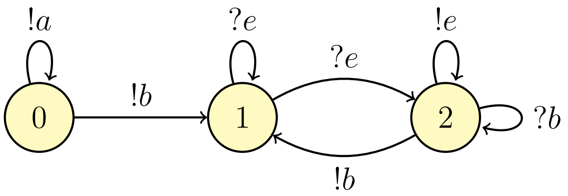


Figure 3.2: The transition system associated to the FIFO machine $\mathcal{M}_2$ (with a single channel) is branch-wqo if the initial control-state is 0. If the initial control-state is 2, then it is not branch-wqo as the states in the set $\{(1, w) \mid w \in e^+b\}$, which form an infinite antichain, are reachable from $(2, \varepsilon)$.

Branch-monotony

We now define a generalization of strong monotony, which we refer to as *branch-monotony*.

Definition 3.4. *An ordered labelled transition system $\mathcal{S} = (X, A, \rightarrow, init, \leq)$ is* branch-monotone *if, for all $x, x' \in X$, $\sigma \in A^*$ such that $init \xrightarrow{*} x \xrightarrow{\sigma} x'$ and $x \leq x'$, there exists a state y such that $x' \xrightarrow{\sigma} y$ and $x' \leq y$.*

branch-monotony

Let us remark that branch-monotone systems have strong (hence, transitive) monotony. As in the case of general monotony, strict branch-monotony is defined using strict inequalities in both cases: An OLTS $\mathcal{S} = (X, A, \rightarrow, init, \leq)$ is strictly branch-monotone if, for all $x, x' \in X$, $\sigma \in A^*$ such that $init \xrightarrow{*} x \xrightarrow{\sigma} x'$ and $x < x'$, there exists a state y such that $x' \xrightarrow{\sigma} y$ and $x' < y$.

Example 7. Consider $\mathcal{M}_1$ from Figure 3.1 once again. $\mathcal{M}_1$ along with the extended prefix ordering $\leq_p$ induces an ordered labelled transition system by considering the actions on the edges to be the labels. Let us show that it is branch-monotone. Let us consider two states $x, x' \in X$ such that $init \xrightarrow{*} x \xrightarrow{\sigma} x'$ and $x \leq x'$. By the definition of the extended prefix ordering, we know that the control-state of x and x' need to be the same. Moreover, when we look at the FIFO machine, we see that the only possibility for x' to be reachable from x and $x \leq x'$ is if the control-state for both x and x' is 0. Furthermore, the only option is $x = (0, a^n)$ and $x' = (0, a^{n+k})$ for $n, k \in \mathbb{N}$. Once again, from the figure, we see that there exists a transition sequence σ such that $(0, a^n) \xrightarrow{\sigma} (0, a^{n+k})$, where $\sigma = (\langle c!a \rangle)^k$. Finally, we can see that $x' \xrightarrow{\sigma} y$ such that $y = (0, a^{n+k+k})$ and hence, $x' \leq y$. We deduce that $\mathcal{M}_1$ is branch-monotone.

Note that if we consider the extended subword ordering for the FIFO system in Figure 3.1, the resulting OLTS is not monotone as $(1, b) \leq_{\mathsf{sw}} (1, ab)$ but $(1, b) \xrightarrow{\langle c?b \rangle} (2, \varepsilon)$ but this transition is not possible from $(1, ab)$. However, by the same argument as above, $\mathcal{M}_1$ is branch-monotone for the extended subword ordering.

Branch-WSTS

We are now ready to extend the definition of WSTS.

Definition 3.5. *A* branch-WSTS *is an OLTS* $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ *that is finitely branching, branch-monotone, and branch-wqo.*

branch-WSTS

When we say, without ambiguity, that a machine $\mathcal{M}$ is branch-wqo, WSTS, or branch-WSTS, we mean that the ordered transition system $\mathcal{S}_{\mathcal{M}}$, associated with machine $\mathcal{M}$, is branch-wqo, WSTS, or branch-WSTS, respectively.

Branch-WSTS is a strict superclass of labelled WSTS. For example, machine $\mathcal{M}_1$ in Figure 3.1 is branch-WSTS for the ordering $\preceq$ but it is not WSTS for $\preceq$ since $\preceq$ is not a wqo on $\{q_0, q_1\} \times \{a, b\}^*$ or on the subset $\{(q_1, w) \mid w \in a^*b\}$.

3.4 Termination and Boundedness for Branch-WSTS

In this section, we will look at the two verification problems of termination and boundedness for branch-WSTS.

Let us recall the *Reduced Reachability Tree (RRT)*, which was defined as Finite Reachability Tree in [FS01]. Suppose that $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ is an OLTS. Then, the *Reduced Reachability Tree* from *init*, denoted by $RRT(\mathcal{S}, init)$, is a tree where nodes are labelled by states of X, and $n(x)$ denotes that node n is labelled by state x. Nodes are either *dead* or *live*. The root node $n_0(init)$ is live. A dead node has no child node. A live node $n(y)$ has one child $n'(y')$ for each successor $y' \in Post_{\mathcal{S}}(y)$. If there is a path in the tree $n_0(init) \xrightarrow{*} n'(y') \xrightarrow{+} n(y)$ such that $n' \neq n$ and $y' \leq y$, we say that n' *subsumes* n, and then n is dead. Otherwise, n is live.

Reduced Reachability Tree (RRT)

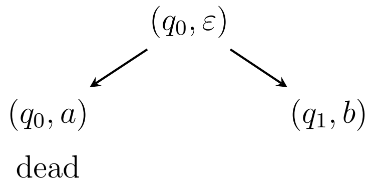


Figure 3.3: The Reduced Reachability Tree of $\mathcal{M}_1$ (from Figure 3.1) starting from (q_0, ε). Note that (q_0, a) is dead because it is subsumed by state (q_0, ε). As a matter of fact, we have $(q_0, \varepsilon) \xrightarrow{*} (q_0, a)$ and $(q_0, \varepsilon) \leq_p (q_0, a)$. State (q_1, b) is also dead but it is not subsumed.

Proposition 3.6. *Let $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ be an OLTS that is finitely branching and branch-wqo. Then, $RRT(\mathcal{S}, init)$ is finite.*

Proof. Let us assume that the $RRT(\mathcal{S}, init)$ is infinite. Since $(\mathcal{S}, init)$ is finitely branching and branch-wqo, there is an infinite branch, in the reachability tree, beginning with the finite prefix $n_0(init) \xrightarrow{*} n_1(x_1) \xrightarrow{+} n_2(x_2) \xrightarrow{+} n_3(x_3)$, in $RRT(\mathcal{S}, init)$, such that nodes n_0, n_1, n_2, n_3 are all different and $x_1 \leq x_2 \leq x_3$. Hence, the node $n_2(x_2)$ has been marked as dead, and the tree has not been explored any further. Thus, there is a contradiction. Hence, $RRT(\mathcal{S}, init)$ is finite. □

Proposition 3.7. *Let $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ be a branch-WSTS, equipped with strict branch-monotony and such that $\leq$ is a partial ordering. The reachability set $Post^*_{\mathcal{S}}(init)$ is infinite iff there exists a branch $n_0(init) \xrightarrow{*} n_1(x_1) \xrightarrow{+} n_2(x_2)$ in $RRT(\mathcal{S}, init)$ such that $x_1 < x_2$.*

Proof. The following proof is an adaptation of the proof of boundedness for WSTS in [FS01] to branch-WSTS. Let us assume $(\mathcal{S}, init)$ is unbounded, i.e. $Post^*_{\mathcal{S}}(init)$ is infinite. Then, there are an infinite number of distinct states which are reachable from $init$. We first show that there exists a computation starting from $init$ without any loop, where all states are distinct. We consider the finitely branching tree of all prefixes of computations, and prune this tree by removing prefixes which contain a loop. Because any reachable state can be reached without a loop, the pruned tree still contains an infinite number of prefixes. By König's lemma, there exists an infinite computation with no loop. Any computation starting from $init$ has a finite prefix labelling a maximal path in $RRT(\mathcal{S}, init)$. To simplify the notations, we confuse nodes and states without ambiguity. Hence, there must be a node x_2 which is subsumed by a node x_1 such that $x_1 \neq x_2$. Since we assumed $\leq$ to be a partial ordering, we deduce from $x_1 \neq x_2$, and $x_1 \leq x_2$ that $x_1 < x_2$.

Conversely, let us assume that there exist two states x_1, x_2 in $RRT(\mathcal{S}, init)$ such that $x_1 \xrightarrow{\sigma} x_2$ and $x_1 < x_2$.

Since the system is strictly branch-monotone, there exists a state x_3 such that $x_2 \xrightarrow{\sigma} x_3$ and $x_2 < x_3$. By iterating this process, we construct an infinite sequence of states $(x_k)_{k \geq 0}$ such that for all $k \geq 1$, one has $x_k \xrightarrow{\sigma} x_{k+1}$ and $x_k < x_{k+1}$. Since $\leq$ is a partial ordering, we deduce that all x_k are different. Hence, $Post^*_{\mathcal{S}}(init)$ is infinite, and $\mathcal{S}$ is unbounded. □

We now need a notion of effectivity adapted to branch-WSTS.

branch-effective

Definition 3.8. *A branch-WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ is* branch-effective *if $\mathcal{S}$ is effective and $Post_{\mathcal{S}}(x)$ is a (finite) computable set, for all $x \in X$.*

Theorem 3.9. *Boundedness is decidable for branch-effective branch-WSTS* $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ *with strict branch-monotony such that* $\leq$ *is a partial ordering.*

Proof. Suppose $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ satisfies the above conditions. From Proposition 3.6, we obtain that $RRT(\mathcal{S}, init)$ is finite. By hypothesis, $\mathcal{S}$ is finitely branching and branch-effective. In particular, for all x, $Post_{\mathcal{S}}(x)$ is a finite computable set. As $\leq$ is decidable, we deduce that $RRT(\mathcal{S}, init)$ is effectively computable. From Proposition 3.7, we know that $Post^*_{\mathcal{S}}(init)$ is infinite iff there exists a finite branch $n_0(init) \xrightarrow{*} n_1(x_1) \xrightarrow{+} n_2(x_2)$ such that $x_1 < x_2$. This last property can be decided on $RRT(\mathcal{S}, init)$, and so the boundedness property can be decided, too. □

We also generalize the decidability of termination for WSTS [FS01] to branch-WSTS.

Proposition 3.10. *A branch-WSTS* $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ *does not terminate from state init iff there exists a subsumed node in* $RRT(\mathcal{S}, init)$*.*

Proof. Let us assume that $\mathcal{S}$ is non-terminating. Then there exists an infinite branch $b : n_0(init) \rightarrow n_1(x_1) \rightarrow n_2(x_2) \rightarrow n_3(x_3) \rightarrow ...$ in the reachability tree of $(\mathcal{S}, init)$. Since $\mathcal{S}$ is branch-wqo, consider the first index i such that there is an $j > i$ such that $x_i \leq x_j$. Now, we say that the prefix of the branch b containing the first $j + 1$ nodes, $n_0(init) \xrightarrow{*} n_i(x_i) \xrightarrow{*} n_j(x_j)$, necessarily appears in $RRT(\mathcal{S}, init)$. Node n_i subsumes node n_j.

Conversely, let us assume there is a subsumed node in $RRT(\mathcal{S}, init)$. Then there exists a branch in $RRT(\mathcal{S}, init)$ of the form

$$n_0(init) \xrightarrow{*} n_1(x_1) \xrightarrow{a_1} n_2(x_2) \ldots \xrightarrow{a_k} n_{k+1}(x_{k+1})$$

such that $x_1 \leq x_{k+1}$.

From the remark following the definition of branch-monotony, we deduce that one may repeat the sequence $\sigma = a_1 a_2 \ldots a_k$ with $\sigma \neq \varepsilon$ and then there exists an infinite branch in the reachability tree and the system does not terminate. □

Theorem 3.11. *Termination is decidable for branch-effective branch-WSTS.*

Proof. Given a branch-WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$, we apply Proposition 3.10 so that it is sufficient to build $RRT(\mathcal{S}, init)$ and check if there exists a subsumed node. Since $\mathcal{S}$ is branch-effective, we can effectively construct $RRT(\mathcal{S}, init)$ and verify the existence of a subsumed node. □

Note that we can thus solve the termination and boundedness problems for the example machine $\mathcal{M}_1$, and since there exists nodes $n_0(\mathit{init})$ and $n_1(x_1)$ in the RRT such that $\mathit{init} = (q_0, \varepsilon)$ and $x_1 = (q_0, a)$ such that $\mathit{init} < x_1$ and $\mathit{init} \xrightarrow{+} x_1$, the machine $\mathcal{M}_1$ is unbounded. Moreover, since $n_1(x_1)$ is also a subsumed node, it is non-terminating.

On the other hand, boundedness becomes undecidable if we relax the strict monotony condition to general monotony (even when we strengthen the order to be wqo). This is because boundedness is undecidable for Reset Petri nets [DFS98]. Reset Petri nets are effective WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$, hence branch-effective WSTS, where $\leq$ is the wqo on vectors of integers. Hence, we deduce:

Proposition 3.12. *Boundedness is undecidable for branch-effective branch-WSTS.*

3.5 Classes of branch-WSTS

We now look at some examples of systems which are branch-WSTS (but not WSTS).

3.5.1 Counter machine with restricted zero tests

In [BFS20], it was shown that termination and boundedness (and moreover, reachability) are decidable for this class of systems. However, using the alternative approach of branch-WSTS, we can verify that termination and boundedness are decidable for this class without reducing these problems to reachability.

Given a CMRZ $\mathcal{C} = (Q, V, T, q_0)$, we consider the associated transition system $\mathcal{S}_\mathcal{C} = (X_\mathcal{C}, \mathit{Act}_\mathcal{C}, \rightarrow_\mathcal{C}, \mathit{init})$. From this system, we construct an OLTS over the extended ordering $\leq$ such that $(q, \mathbf{v}) \leq (q', \mathbf{v}')$ iff $q = q'$ and $\mathbf{v} \leq \mathbf{v}'$ (component-wise). Note that $(X_\mathcal{C}, \leq)$ is a wqo. Moreover, this ordering is a partial ordering.

We now show that CMRZ are branch-monotone for $\leq$. We drop the subscript while denoting $X_\mathcal{C}$ for the remainder of this section, as it is clear from context.

Proposition 3.13. *CMRZ are branch-monotone and strictly branch-monotone for the wqo $\leq$.*

Proof. Consider the OLTS $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$ associated to a CMRZ with states $x, x' \in X$ such that $x \leq x'$ (resp. $x < x'$) and $\mathit{init} \xrightarrow{*} x \xrightarrow{\sigma} x'$. We need to show that there exists a state y such that $x' \xrightarrow{\sigma} y$ and $x' \leq y$ (resp. $x' < y$).

We first prove the following claim:

Claim: For states $x, x' \in X$ such that $init \xrightarrow{*} x \xrightarrow{\sigma} x'$ where $x \leq x'$ (resp. $x < x'$) and $|\sigma| = n$, the following property holds: For all $\nu \preceq \sigma$, we have $z, z' \in X$ such that $x \xrightarrow{\nu} z$ and $x' \xrightarrow{\nu} z'$ and $z \leq z'$ (resp. $z < z'$). We prove this claim by induction on the length of ν.

For the base case, $|\nu| = 0$. We have from the hypothesis, $x \leq x'$, hence the claim is trivially true.

Let us assume that the claim holds for $|\nu| = k$. We show that it holds for $|\nu| = k + 1$. From the induction hypothesis, we know that for $\nu = \nu' \cdot a$ where $a \in A$, there exists z_1, z_1' such that $x \xrightarrow{\nu'} z_1$ and $x' \xrightarrow{\nu'} z_1'$ and $z_1 \leq z_1'$. Since $x \xrightarrow{\sigma} x'$, we know that there exists $z \in X$ such that $x \xrightarrow{\nu'} z_1 \xrightarrow{a} z$. We can now be in one of the following cases:

Case i: If a is of the form noop and $Z = \emptyset$, then we can trivially execute a from z_1' and reach z' such that $z \leq z'$ (resp. $z < z'$).

Case ii: The action a is of the form inc(v) or dec(v), and the set of counters to be tested for zero $Z = \emptyset$, i.e. $z_1 \xrightarrow{a} z$ only increments one counter and leaves the others unchanged (and no counters are tested to zero). Since $z_1 \leq z_1'$ (resp. $z_1 < z_1'$), we know that z_1, z_1' have the same control state. Hence, this action is enabled in z_1'. Moreover, because of the CMRZ property, we know that v is not tested to zero even once until the state z_1' is reached in this run. Therefore, we can execute the increment/decrement operation on v. Furthermore, since $z_1 \leq z_1'$ (resp. $z_1 < z_1'$), the value of v in z_1' is greater than or equal to (resp. strictly greater than) the value of v in z_1. Hence, we can execute a from z_1' and reach a state z' such that $z \leq z'$ (resp. $z < z'$).

Case iii: $Z \neq \emptyset$ in the transition $z_1 \xrightarrow{a} z$. Hence, there are a set of counters Z which are tested to zero. By the CMRZ property, we know that all counters $v \in Z$, are never incremented or decremented further. Hence, during the execution $z_1 \xrightarrow{a} z \xrightarrow{w} z_1'$ where $w \in A^*$, we know that none of these counters are incremented or decremented. Hence, the value of the counters in z_1' is also equal to zero. Therefore, we can execute a from z_1' to reach z'. Moreover, since $z_1 \leq z_1'$ (resp. $z_1 < z_1'$), and none of these counters change value, we can conclude that $z \leq z'$ (resp. $z < z'$).

Hence, as a special case of the claim where $\nu = \sigma$, we prove that CMRZ are branch-monotone (resp. strictly branch-monotone).

□

Proposition 3.14. *CMRZs are branch-effective branch-WSTS.*

Proof. Given an OLTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ associated to a CMRZ, for any two states $x, x' \in X$, we can decide if $x \leq x'$. Furthermore, since we have a finite automaton, $\rightarrow$ is decidable, and $Post_{\mathcal{S}}(x)$ is computable for all $x \in X$. Hence, it is branch-effective. □

Hence, we deduce:

Theorem 3.15. *Termination and boundedness are decidable for counter machines with restricted zero tests.*

3.5.2 Input-bounded FIFO machines

We now study a subclass of FIFO machines, and investigate if they are branch-WSTS.

Consider a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$. We define the input-language of a FIFO channel c as the set of all words that are sent into the channel, i.e. $proj_{c!}(Traces(\mathcal{M}))$. We say that the machine is input-bounded if there is a tuple $(L_c)_{c \in Ch}$ of regular bounded languages L_c(i.e. languages of the form $w_1^* \dots w_n^*$) such that, for all $c \in Ch$, $proj_{c!}(Traces(\mathcal{M})) \subseteq L_c$, i.e. every run of the FIFO machine is input-bounded language. We say that L is distinct-letter if $|w_1 \dots w_n|_a = 1$ for all $a \in \Sigma$.

This class is studied in more depth in Chapter 4. For more precise definitions of these notions, we refer the reader to Section 4.1.

Proposition 3.16. *Input-bounded FIFO machines are branch-wqo for the prefix-ordering $\leq_p$.*

Proof. Let us consider the transition system $\mathcal{S}_{\mathcal{M}} = (X_{\mathcal{M}}, Act_{\mathcal{M}}, \rightarrow_{\mathcal{M}}, init_{\mathcal{M}})$ associated an input-bounded FIFO machine $\mathcal{M}$ with a single channel, and an infinite run $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow ...x_i...$ with $x_i = (q_i, w_i) \in X_{\mathcal{M}}$ and $x_0 = init_{\mathcal{M}}$.

The infinite run is of the form $x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} x_2 ... x_{i-1} \xrightarrow{\sigma_i} x_i \xrightarrow{\sigma_{i+1}} \dots$ and we denote $\sigma[i] = proj_!(\sigma_1 \sigma_2 \dots \sigma_i)$. It can be observed that $\sigma[i]$ is a prefix of $\sigma[i+1]$ for all $i \in \mathbb{N}$. Since $\sigma[i]$ is of the form $v_1^{n_{1,i}} ... v_m^{n_{m,i}} u_m$ for $u_m \preceq v_m$ and $n_{1,i} ..., n_{m,i} \geq 0$ and $1 \leq m \leq k$, the infinite sequence $(\sigma[i])_{i \in \mathbb{N}}$ satisfies two possible exclusive cases:

Case i: There exists an i_0 such that $\forall i \geq i_0, proj_!(\sigma_i) = \epsilon$ so there exists $i_1 \geq i_0$ such that for all $i \geq i_1, w_i = w_{i+1}$. Hence, because there are finitely many control states, we deduce that there exist $i_2, i_3 \geq i_1$ such that $x_{i_2} \xrightarrow{+} x_{i_3}$ and $x_{i_2} = x_{i_3}$ hence also in particular $x_{i_2} \leq_p x_{i_3}$.

Case ii: There are infinitely many indices i such that $proj_!(\sigma_i) \neq \epsilon$, which means that the infinite sequence $(\sigma[i])_{i\in\mathbb{N}}$ is not stationary. This implies that the set $S_\sigma = \{(n_{1,i}, ..., n_{k,i}) \mid i \in \mathbb{N}\}$, associated with σ, is infinite. Hence, there exists a least index p such that the set $\{n_{p,i}\}_{i\in\mathbb{N}}$ is infinite. Then the set $F = \{(n_{1,i}, ..., n_{p-1,i}) \mid i \in \mathbb{N}\}$ is finite.

We claim that for all indices $\ell \geq p+1$, $n_{\ell,i} = 0$ for all i. Let us assume to the contrary that there is some index $\ell \geq p+1$ and i_0 such that $n_{\ell,i_0} \neq 0$. This means that the word v_ℓ is in the channel in state x_{i_0}, which means that the word v_ℓ was sent to the channel before (or at) the step i_0, i.e, $\sigma[i_0] = v_1^{n_{1,i_0}}...v_p^{n_{p,i_0}}...v_m^{n_{m,i_0}}u_m$ for some $u_m \preceq v_m$ and $n_{l,i_0} > 0$ and $1 \leq m \leq k$. So, in particular, word v_p cannot be sent after i_0, hence, $n_{p,i} = n_{p,i_0}$ $\forall i > i_0$. Hence, $\{n_{p,i}\}_{i\in\mathbb{N}}$ is finite which is a contradiction to our assumption that $\{n_{p,i}\}_{i\in\mathbb{N}}$ is infinite.

This means that after some state x_t, we only write word v_p to the channel. Since, the set $F = \{(n_{1,j}, ..., n_{p-1,j}) \mid j \in \mathbb{N}\}$ is finite, we can extract an infinite subsequence $(q, w_i)_{i\in K\subseteq\mathbb{N}}$ where $w_i = uv_p^{n_{p,i}}$ with $u \in F$ and $(n_{p,i})_{i\in K}$ is non-decreasing. Hence, there exist two indices $a, b > 0$ such that $w_a = u.v_p^{n_{p,a}}$ and $w_{a+b} = u.v_p^{n_{p,a+b}}$ and $n_{p,a} \leq n_{p,a+b}$ hence $w_{a+b} = w_a.v_p^{n_{p,a+b}-n_{p,a}}$ hence $w_a \preceq w_{a+b}$. So we have found two states x_a, x_{a+b} such that $x_a \leq_p x_{a+b}$. Hence, the machine is branch-wqo for the prefix ordering.

We show that it is branch-wqo for a single channel, but using the same argument for each channel, we can conclude that input-bounded FIFO machines with multiple channels are branch-wqo for the prefix-ordering $\leq_p$. □

It is clear that $\mathcal{M}_1$ belongs to this class of FIFO systems. But, we see below that the class of input-bounded FIFO machines is not branch-WSTS.

Example 8. Consider the FIFO machine $\mathcal{M}_2$ in Figure 3.4 that is input-bounded for $L = (ab)^*$. We have $(q_0, \epsilon) \xrightarrow{\sigma} (q_0, b)$, where $\sigma = !a?a!b$. Moreover, $(q_0, \epsilon) \leq_p (q_0, b)$. However, we cannot repeat σ from (q_0, b), as it is not possible to execute $(q_1, ba) \xrightarrow{?a}$. Hence, the machine is not branch-monotone for the prefix-ordering.

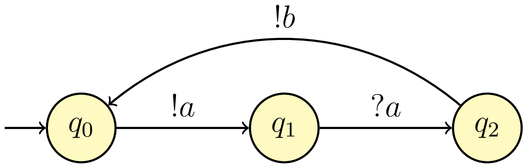


Figure 3.4: The FIFO machine $\mathcal{M}_2$

From the above counter-example, we see that the class of distinct-letter input-bounded FIFO machines are not branch-WSTS. Hence, we impose another restriction on such systems.

Consider an input-bounded FIFO machine $\hat{\mathcal{M}} = (\hat{Q}, Ch, \Sigma, \hat{T}, \hat{q_0})$ (with a single channel) with a distinct-letter bounded input-language L. We first consider a deterministic, trimmed, (not necessarily complete) finite automaton $\mathcal{A} = (Q_{\mathcal{A}}, Act_{\mathcal{M}}, T_{\mathcal{A}}, q^0_{\mathcal{A}}, F_{\mathcal{A}})$, with set of final states $F_{\mathcal{A}} \subseteq Q_{\mathcal{A}}$, whose language is $L(\mathcal{A}) = L_! \cap Pref(L_?)$, where $L_! = \{\sigma \mid proj_!(\sigma) \in L\}$ and $L_? = \{\sigma \mid proj_?(\sigma) \in L\}$.

With this, we define $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$ as the product of the FIFO machine $\hat{\mathcal{M}}$ and $\mathcal{A}$ in the expected manner. In particular, the set of control-states of $\mathcal{M}$ is $Q = \hat{Q} \times Q_{\mathcal{A}}$, and its initial state is the pair $q_0 = (\hat{q_0}, q^0_{\mathcal{A}})$.

Note that since $\hat{\mathcal{M}}$ is input-bounded, every run in $\hat{\mathcal{M}}$ belongs to $L_! \cap Pref(L_?)$. Therefore, the language of the new FIFO machine $\mathcal{M}$ is the same as the language of the old FIFO machine $\hat{\mathcal{M}}$, and a subset of the trimmed automata.

Proposition 3.17. *The machine $\mathcal{M}$ constructed as a product of a distinct-letter input-bounded FIFO machine $\hat{\mathcal{M}}$ and the deterministic finite automaton of its input language is branch-compatible.*

Proof. Let L $= w_1^* \dots w_n^*$. Let $(q_0, \varepsilon) \xrightarrow{\tau} (q, w) \xrightarrow{\sigma} (q, w')$ such that $w \preceq w'$. To prove branch-compatibility, we need to show that there exists w'' such that $(q, w') \xrightarrow{\sigma} (q, w'')$ and $w' \preceq w''$.

Firstly, we know that $proj_!(\tau) = w_1^{n_1} \dots w_i^{n_i}.u_i$ where $u_i \preceq w_i$ and $1 \leq i \leq n$ and $n_p \in \mathbb{N}$ for all $1 \leq p \leq i$. Moreover, $proj_!(\sigma) \in Pref(u_i' \cdot w_i^{n_i'} \dots w_j^{n_j})$ where $u_i.u_i' = w_i$ and $1 \leq i \leq j \leq n$. Let us consider the channel content w now. From the characterization of τ above, we can express $w = v_\ell \cdot w_\ell^{n_\ell} \dots w_i^{n_i}.u_i$, where $1 \leq \ell \leq i$ and $v_\ell \in Suf(w_\ell)$. Now, let us analyse the cases based on the value of $proj_?(\sigma)$:

- $proj_?(\sigma) = \varepsilon$. In other words, this means that there are only send actions in σ. Hence, it is possible to take the same sequence of moves once again as we are in the same control-state q. Therefore, $(q, w') \xrightarrow{\sigma} (q, w'')$ for some value of w''. Furthermore, since σ has only send actions, $w' = w.v$ for some $v \in \Sigma^*$. Therefore, after we repeat σ once again from (q, w'), we reach (q, w'') such that $w'' = w'.v = w.v.v$. Therefore, $(q, w') \leq_p (q, w'')$ and we are done with this case.
- For $proj_?(\sigma) \neq \varepsilon$, we can be in one of three cases:
  - $w \neq \varepsilon$ and $\exists p_1, p_2$ such that $1 \leq p_1 < p_2 \leq i$ such that $w = v_{p_1}.v.u_{p_2}$ where $v_{p_1} \in Suf(w_{p_1})$, $u_{p_2} \in Pref(w_{p_2})$, and $v_{p_1}, v_{p_2} \neq \varepsilon$ and $v \in \Sigma^*$. In other words, in this case, there is a non-empty part of at least two distinct words in the channel contents w. Since the FIFO machine is input-bounded, we can conclude that $proj_!(\sigma)$ does not contain any occurrences of alphabets from the word w_{p_1}. Therefore, in order for

the condition $w \preceq w'$ to be satisfied, it is necessary that $\mathit{proj}_?(\sigma) = \varepsilon$, which is a contradiction to our assumption. Hence, this case is not a valid case.

- Therefore, if $w \neq \varepsilon$ and $\mathit{proj}_?(\sigma) \neq \varepsilon$, then the only possibility is that $w = v_i.w_i^{n_i}.u_i$ such that $v_i \in \mathit{Suf}(w_i)$. Therefore, $\mathit{proj}_?(\sigma)$ only consists of letters from words w_j such that $j \geq i$. However, since $w \preceq w'$, we can be certain that it only consists of letters from the word w_i (if we start receiving a word w_j where $j > i$ then there can be no occurrence of the word w_i in the channel). Therefore, $\mathit{proj}_?(\sigma)$ consists of only letters belonging to w_i. Moreover, since $\mathit{proj}_!(\sigma)$ is non-empty, there is at least one letter that is read from w. Therefore, the first letter that is sent in the sequence σ most certainly belongs to the word w_i (to ensure $w \preceq w'$).

  Let us consider this subsequence σ' from (q, w) to the first send action. Let us say we have $(q, w) \xrightarrow{\sigma'} (q', v')$. Now, since the subsequence σ' only consists of receptions from (q, w), along with the first send action, this subsequence is also possible from $(q, w.v)$ for all $v \in \Sigma^*$. Therefore, we can execute the same sequence from (q, w'). Hence, $\mathit{proj}_!(\tau.\sigma.\sigma') \in L$. Therefore, since $\mathit{Alph}(\mathit{proj}_!(\sigma')) \in \mathit{Alph}(w_i)$, we can be sure that $\mathit{Alph}(\mathit{proj}_!(\sigma)) \in \mathit{Alph}(w_i)$. Therefore, σ only sends and receives letters from a single word w_i.

  Moreover, since the system is input-bounded, and the first send action in σ' matches the first send action in σ, we see that $w' = v_i.w_i^{n_i}.u_i.(v_i'.w_i^{n_i'}.u_i)$ $= w.(v_i'.w_i^{n_i'}.u_i)$ such that $u_i.v_i' = w_i$. Therefore, we can repeat this sequence from (q, w') and reach a state (q, w'') such that $w' \preceq w''$, and hence, it is branch-compatible for this case.

- The final case we need to consider is $w = \varepsilon$. In this case, it is clear that σ consists of at least one send action before the first reception. Therefore, because of the input-bounded property and the fact that this action can be executed at (q, w'), we can once again see that $\mathit{proj}_!(\sigma)$ consists only sending only letters from a single word. Moreover, since the same action can be executed, once again we see that $\mathit{proj}_!(\sigma) = v_j.w_j^{n_j}.u_j$ such that $u_j.v_j = w_j$. Therefore, $\mathit{proj}_?(\sigma) \in \mathit{Pref}(v_j.w_j^{n_j}.u_j)$.

  Now let us consider the run $\tau.\sigma$ in the automaton $\mathcal{A}$ that we constructed. Since $\tau.\sigma$ is a run in $\mathcal{M}$, there is also a run in $\mathcal{A}$ such that $q^0_{\mathcal{A}} \xrightarrow{\tau} q_s \xrightarrow{\sigma} q_s$. Moreover, we can also repeat σ to obtain $q^0_{\mathcal{A}} \xrightarrow{\tau} q_s \xrightarrow{\sigma} q_s \xrightarrow{\sigma} q_s$. Therefore, $\tau.\sigma.\sigma \in \mathit{Pref}(L_?)$. Moreover, since $\mathit{proj}_?(\sigma) \neq \varepsilon$, $\mathit{proj}_?(\sigma) = u_j'.w_j^{n_j'}.v_j'$ such that $v_j'.u_j' = w_j$. Therefore, we can repeat σ from (q, w') in $\mathcal{M}$, and we reach a state (q, w'') such that $w' \preceq w''$.

Hence, we see that for all cases, if $(q_0, \varepsilon) \xrightarrow{\tau} (q, w) \xrightarrow{\sigma} (q, w')$ such that $w \preceq w'$. then there exists w'' such that $(q, w') \xrightarrow{\sigma} (q, w'')$ and $w' \preceq w''$. Hence, $\mathcal{M}$ is branch-compatible.

We can extend the same argument for each channel in case of multiple channels, and we obtain branch-compatibility for the case of distinct-letter FIFO machines with multiple channels (intersected with the corresponding finite automata). □

Example 9. Consider the FIFO machine $\mathcal{M}_2$ as in Figure 3.5 that is input-bounded for $L = (ab)^*$. As we saw in Example 8, it is not branch-monotone. However, let us consider the product $\mathcal{M}_2'$ of $\mathcal{M}_2$ along with the finite automaton $\mathcal{A}$ that recognizes $L_! \cap \mathit{Pref}(L_?)$. Here, we see that the counter-example we had previously is no longer there. In fact, the loop that could not be realized has now been opened up and we can see that the FIFO machine $\mathcal{M}_2'$ has only a finite run. Moreover, we see that it is branch-monotone for the prefix-ordering.

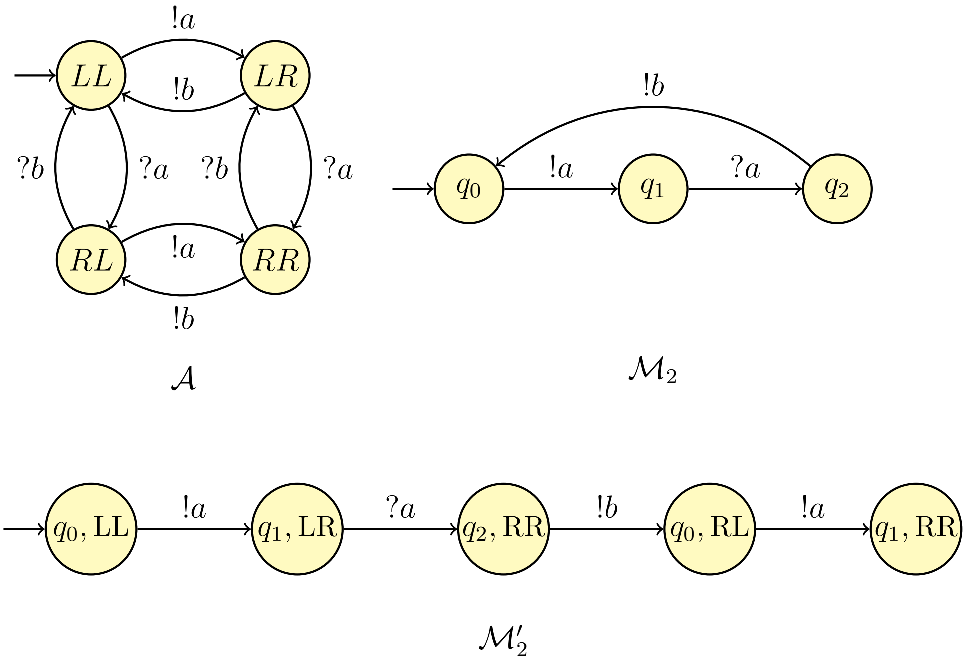


Figure 3.5: The FIFO machine $\mathcal{M}_2$ (right) and the automaton $\mathcal{A}$ that recognizes $L_! \cap \mathit{Pref}(L_?)$ (left) intersect to give $\mathcal{M}_2'$(below) which is branch-monotone.

We remark here that in Chapter 4, when we show that input-bounded rational-reachability is decidable, we construct a "normalized" FIFO machine, from the given FIFO machine and bounded language. Using the same principles, we can modify every input-bounded FIFO machine into one that is distinct-letter, and using the product construction from above, we have the following proposition:

Proposition 3.18. *Normalized input-bounded FIFO machines are branch-effective branch-WSTS (for the prefix ordering).*

Proof. Given an input-bounded FIFO machine $\mathcal{S}_{\mathcal{M}}$, for any two states $x, x' \in X$, we can decide if $x \leq_p x'$. Furthermore, since we have a finite automaton, → is decidable, and $\mathit{Post}_{\mathcal{S}}(x)$ is computable for all $x \in X$. Hence, it is branch-effective. □

Moreover, the extended prefix ordering is a partial ordering. Hence, we deduce:

Theorem 3.19. *Termination and boundedness are decidable for input-bounded FIFO machines.*

3.6 Coverability

We look at the notion of coverability now. Let us formally define the problem we are trying to solve.

Decision Problem: COVERABILITY

Input: a TS $\mathcal{S} = (X, \rightarrow)$,
$x, y \in X$
Question: does there exist a $y' \in Post^*_{\mathcal{S}}(x)$ such that $y \leq y'$?

Coverability is decidable for a large class of WSTS:

Theorem 3.20 ([FS01, AČJT00])**.** *The coverability problem is decidable for effective WSTS $\mathcal{S} = (X, \rightarrow, \leq)$ equipped with an algorithm that, for all finite sets $I \subseteq X$, computes a finite basis $pb(I)$ of $\uparrow Pre(\uparrow I)$.*

Assume $\mathcal{S} = (X, \rightarrow, \leq)$ is a WSTS and $x \in X$ is a state. The *backward coverability algorithm* involves computing (a finite basis of) $Pre^*(\uparrow x)$ as the limit of the infinite increasing sequence $\uparrow I_0 \subseteq \uparrow I_1 \subseteq \ldots$ where $I_0 = \{x\}$ and $I_{n+1} \stackrel{\text{def}}{=} I_n \cup pb(I_n)$. Since there exists an integer k such that $\uparrow I_{k+1} = \uparrow I_k$, the finite set I_k is computable (one may test, for all n, whether $\uparrow I_{n+1} = \uparrow I_n$) and I_k is then a finite basis of $Pre^*(\uparrow x)$ so one deduces that coverability is decidable.

Coverability can be also decided by using the *forward coverability algorithm* that relies on two semi-decision procedures (as described below). It applies to the class of well-behaved transition systems, which are more general than WSTS. A *well-behaved transition system (WBTS)* is an ordered transition system $\mathcal{S} = (X, \rightarrow, \leq)$ with monotony such that $(X, \leq)$ contains no infinite antichain. We describe effectiveness hypotheses that allow manipulating downward-closed sets in WBTS. But first, we introduce some notions.

Procedure 1 : Checks for a coverability certificate of y from x

```
input: $\mathcal{S} = (X, \rightarrow, \leq)$ and $x, y$
  $D := \downarrow x$
  while $y \notin D$ do
    $D := \downarrow(D \cup Post_{\mathcal{S}}(D))$
  end while
  return "$y$ is coverable from $x$"
```

Ideals. An *ideal* is a downward-closed set $I \subseteq X$ that is also *directed*, i.e., it is non-empty and, for every $x, y \in I$, there exists $z \in I$ such that $x \leq z$ and $y \leq z$. The set of ideals is denoted by $\mathsf{Ideals}(X) = \{\emptyset \subset I \subseteq X \mid I = \downarrow I$ and I is directed$\}$. A directed complete partial ordering (dcpo) is an ordered set $(X, \leq)$ such that every directed set $D \subseteq X$ has a least upper bound (lub) in X: for instance, $(\mathbb{N}, \leq)$, with the usual notations, is not a dcpo since the directed set $\mathbb{N}$ has no lub in $\mathbb{N}$; if we add the lub ω to $\mathbb{N}$, then $(\mathbb{N}_\omega, \leq)$ is a dcpo. There is a way to add all lubs to any ordered set $(X, \leq)$, that is called the ideal completion. The elements of the ideal completion of X are its ideals, i.e., its downward-closed directed families, ordered by inclusion.

ideally effective

Definition 3.21 ([BFM17, Definition 3.4]). *A class C of WBTS is* ideally effective *if, given $\mathcal{S} = (X, \rightarrow, \leq) \in C$,*

- *the set of encodings of $\mathsf{Ideals}(X)$ is recursive,*
- *the function mapping the encoding of a state $x \in X$ to the encoding of the ideal $\downarrow x \in \mathsf{Ideals}(X)$ is computable;*
- *inclusion of ideals of X is decidable;*
- *the downward closure $\downarrow Post(I)$ expressed as a finite union of ideals is computable from the ideal $I \in \mathsf{Ideals}(X)$.*

Theorem 3.22 ([BFM17]). *The coverability problem is decidable for ideally effective WBTS.*

The proof is done by two semi-decision procedures where downward-closed sets are represented by their finite decomposition in ideals and this is effective. Procedure 1 checks for coverability of y from x, by recursively computing $\downarrow x$, $\downarrow(\downarrow x \cup Post(\downarrow x))$ and so on. This procedure terminates only if y belongs to one of these sets, hence it terminates if y is coverable. Hence, we deduce:

Proposition 3.23 ([BFM17]). *For an ideally effective WBTS $\mathcal{S} = (X, \rightarrow, \leq)$, an initial state x, and a state y, Procedure 1 terminates iff y is coverable from x.*

Procedure 2 : Checks for non-coverability

```
input: $\mathcal{S} = (X, \rightarrow, \leq)$ and $x, y$
  enumerate $D_1, D_2, \ldots$
  $i := 1$
  while $\neg(\downarrow Post(D_i) \subseteq D_i$ and $x \in D_i$ and $y \notin D_i)$ do
    $i := i + 1$
  end while
  return false
```

Procedure 2 enumerates all downward-closed subsets (by means of their finite decomposition in ideals) in some fixed order $D_1, D_2, \ldots$ such that for all i, $D_i \subseteq X$ and $\downarrow Post(D_i) \subseteq D_i$. We note that this enumeration is effective since $\mathcal{S}$ is ideally effective. If such a set D_i contains x, it is an over-approximation of $Post^*(x)$. Hence, if there is such a set D_i such that $x \in D_i$ but $y \notin D_i$, it is a certificate of non-coverability. Moreover, this procedure terminates if y is non-coverable because $\downarrow Post^*(x)$ is such a set, and hence, will eventually be found.

Proposition 3.24 ([BFM17]). *For a WBTS $\mathcal{S} = (X, \rightarrow, \leq)$, states x and y, Procedure 2 terminates iff y is not coverable from x.*

Coverability for branch-WSTS

We show that the two existing coverability algorithms for WSTS do not allow one to decide coverability for branch-WSTS. We note that, contrary to WSTS, $Pre^*(\uparrow x)$ is not necessarily upward-closed. In fact, even for a CMRZ with a single zero-test, this property is not satisfied.

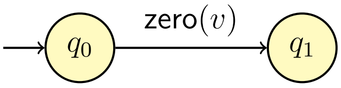


Figure 3.6: System $\mathcal{M}_4$ is branch-WSTS.

Example 10. In Figure 3.6, let us consider the counter machine $\mathcal{M}_4$ with a single counter v. Let $x = (q_1, 0)$. We see that $Pre^*(\uparrow x) = \{(q_1, n) \mid n \geq 0\} \cup \{(q_0, 0)\}$. However, $\uparrow Pre^*(\uparrow x) = Pre^*(\uparrow x) \cup \{(q_0, n) \mid n \geq 1\}$.

Thus, we get:

Proposition 3.25. *Given a branch-effective branch-WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ and a state $x \in X$, the set $Pre^*(\uparrow x)$ is not necessarily upward-closed.*

So we deduce that we cannot use the backward algorithm.

Let us consider using the forward algorithm instead. The second procedure computes all sets X which satisfy the property $\downarrow Post^*(X) \subseteq X$. This is because for WSTS, the set $\downarrow Post^*(x)$ satisfies this property. However, we now show a counter-example of a branch-WSTS which does not satisfy this property.

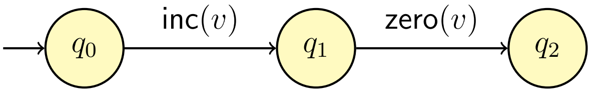


Figure 3.7: System $\mathcal{M}_5$ is branch-WSTS.

Example 11. Consider the counter machine $\mathcal{M}_5$ from Figure 3.7, with $init = (q_0, 0)$. We compute $\downarrow Post^*(init)$. We see that $Post^*(init) = \{(q_0, 0), (q_1, 1)\}$, hence, $Y = \downarrow Post^*(init) = \{(q_0, 0), (q_1, 1), (q_1, 0)\}$. However, $\downarrow Post^*(Y) \not\subseteq Y$, as $\downarrow Post^*(Y) = \{(q_0, 0), (q_1, 1), (q_1, 0), (q_2, 0)\}$, which is strictly larger than Y.

This gives us:

Proposition 3.26. *For branch-effective, branch-WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ such that $\downarrow Post(\downarrow x)$ is computable for all $x \in X$, the set $Y = \downarrow Post^*(init)$ does not necessarily satisfy the property $\downarrow Post^*(Y) \subseteq Y$.*

Not only do the two coverability algorithms not terminate but we also show that coverability is undecidable. To this end, let us define well-structured nets, which was introduced in [FMP04].

well-structured net

Definition 3.27. *A well-structured net is a triple $\mathcal{N} = (\mathbb{N}^p, F, \leq)$ for some dimension p where F is a finite set of non-decreasing functions defined on upward-closed subsets of $\mathbb{N}^p$.*

Let us denote by TM_j the j^{th} Turing Machine in some enumeration. Consider the family of functions $f_j : \mathbb{N}^2 \rightarrow \mathbb{N}^2$ defined by $f_j(n, k) = (n, 0)$ if $k = 0$ and TM_j runs for more than n steps, else $f_j(n, k) = (n, n + k)$. Let $g : \mathbb{N}^2 \rightarrow \mathbb{N}^2$ be the function defined by $g(n, k) = (n+1, k)$. Now, consider the strongly monotone WSN $\mathcal{N}_j = (\mathbb{N}^2, \{f_j, g\}, \leq)$. Since $\mathcal{N}_j$ is strongly monotone, it is also branch-monotone. Moreover, system $\mathcal{N}_j$ is branch-effective and we observe that $Post$ is computable and $\leq$ is wqo. Hence, $\mathcal{N}_j$ belongs to a class of branch-effective branch-WSTS.

From [FMP04, Theorem 4.3], we deduce that the state $(1, 1)$ is coverable from $(0, 0)$ in $\mathcal{N}_j$ iff TM_j halts.

Hence, we obtain the following corollary:

Corollary 3.28. *The coverability problem is undecidable for branch-effective branch-WSTS $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ (even if $\mathcal{S}$ is strongly monotone and $\leq$ is wqo).*

3.7 Cover-WSTS

We show that coverability is decidable for a class of WSTS-like systems with a wqo but with a new notion of monotony, which we refer to as cover-monotony.

We recall the classical definition of the coverability set of $x \in X$ as $Cover_{\mathcal{S}}(x) = {\downarrow}Post^*_{\mathcal{S}}(x)$. Let us consider the following monotony condition. *coverability set*

Definition 3.29. *Let $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ be an ordered labelled transition system. We say that $\mathcal{S}$ is* cover-monotone *(resp. strongly cover-monotone) if, for all $y_1 \in Cover_{\mathcal{S}}(init)$ and for all $x_1, x_2 \in X$ such that $x_1 \leq y_1$ and $x_1 \rightarrow x_2$, there exists a state $y_2 \in X$ such that $y_1 \xrightarrow{*} y_2$ (resp. $y_1 \rightarrow y_2$) and $x_2 \leq y_2$.* *cover-monotone*

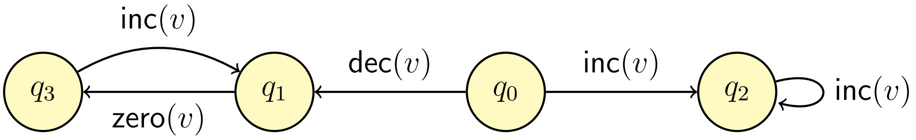


Figure 3.8: Machine $\mathcal{M}_6$ is cover-monotone if we consider the associated transition system to have initial state $(q_0, 0)$. However, if we consider the initial state as $(q_0, 1)$, then it is not cover-monotone.

Cover-monotony is more general than the usual monotony, i.e. a system can be cover-monotone but not monotone (as we shall see in this section). However, let us emphasize that cover-monotony of a system $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ is a property that depends on the initial state *init* while the usual monotony does not depend on any initial state (see Figure 3.8). Hence, the two notions are incomparable.

The strong cover-monotony property is not trivially decidable for general models while (usual) strong-monotony is decidable for many powerful models like FIFO machines and counter machines. However, this notion is still of theoretical interest, as it shows that we can relax the general monotony condition.

However, there is a link between general monotony and cover-monotony.

Proposition 3.30. *A system $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ is monotone iff for all $x \in X$, $(X, A, \rightarrow, \leq, x)$ is cover-monotone.*

Proof. Every monotone system is trivially cover-monotone for all $x \in X$. Conversely, consider a system $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ such that $(X, A, \rightarrow, \leq, x)$ is cover-monotone for all $x \in X$. Let us consider $x_1, y_1, x_2 \in X$ such that $x_1 \leq y_1$ and $x_1 \rightarrow x_2$. In order to show that $\mathcal{S}$ is monotone, we need to prove that there exists y_2 such that $y_1 \xrightarrow{*} y_2$ and $x_2 \leq y_2$. Since $x_1 \leq y_1$ (by hypothesis), $x_1 \in Cover_{\mathcal{S}}(y_1)$. By the hypothesis, $(X, A, \rightarrow, \leq, y_1)$ is cover-monotone, hence there exists y_2 such that $y_1 \xrightarrow{*} y_2$ with $x_2 \leq y_2$. Hence, $\mathcal{S}$ is monotone. □

We may now define cover-WSTS as follows.

cover-WSTS **Definition 3.31.** *A* cover-WSTS *is a finitely branching cover-monotone system* $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$ *such that* $(X, \leq)$ *is wqo.*

For cover-WSTS, the backward algorithm fails. This is once again because the presence of a single zero test removes the property of the set $\mathit{Pre}^*(\uparrow y)$ being upward-closed. But we will now show that the forward coverability approach is possible.

Proposition 3.32. *Given a system* $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$ *and a downward-closed set* $D \subseteq X$ *such that* $\downarrow \mathit{Post}(D) \subseteq D$*, then we have the inclusion* $\downarrow \mathit{Post}^*(D) \subseteq D$*.*

Proof. We first prove the following claim by induction.

Claim: Given a system $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$, for every downward-closed set $D \subseteq X$ such that $\downarrow \mathit{Post}(D) \subseteq D$, the following inclusion holds: $\downarrow \mathit{Post}^k(D) \subseteq D$, for all $k \geq 1$.

Base case: For $k = 1$, this is the hypothesis.

The inductive hypothesis asserts that: Given a system $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$, for every downward-closed set $D \subseteq X$ such that $\downarrow \mathit{Post}(D) \subseteq D$, the following inclusion holds: $\downarrow \mathit{Post}^k(D) \subseteq D$. We now prove that it is also true for $k + 1$.

Let $Z = \mathit{Post}^k(D)$. Since $Z \subseteq \downarrow \mathit{Post}^k(D)$, we know that $Z \subseteq D$ (by hypothesis). Furthermore, for any subset $Y \subseteq D$, $\downarrow \mathit{Post}(Y) \subseteq Y$, hence, $\downarrow \mathit{Post}(Y)$ is also a subset of D. Therefore, $\downarrow \mathit{Post}(Z) \subseteq D$. Hence, we deduce $\downarrow \mathit{Post}(\mathit{Post}^k(D)) \subseteq D$, i.e. $\downarrow \mathit{Post}^{k+1}(D) \subseteq D$. Hence, we have proved the claim by induction.

From the above claim, we know that $\downarrow \mathit{Post}^k(D) \subseteq D$ for all $D \subseteq X$.

Note also that $\mathit{Post}^*(D) = D \cup \bigcup\limits_{k \geq 1} \mathit{Post}^k(D)$. Therefore, $\mathit{Post}^*(D) \subseteq D$, and finally since D is downward-closed, $\downarrow \mathit{Post}^*(D) \subseteq D$. □

Let us define a particular instance of the coverability problem in which we verify if a state is coverable from the initial state.

Decision Problem: *init*-COVERABILITY

Input:	an ordered transition system $\mathcal{S} = (X, A, \rightarrow, \leq, \mathit{init})$, a state $y \in X$
Question:	is $y \in \downarrow \mathit{Post}^*_{\mathcal{S}}(\mathit{init})$?

We show that *init*-coverability is decidable for cover-WSTS:

Theorem 3.33. *Let $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ be an ideally effective cover-WSTS such that Post is computable. Then, the init-coverability problem is decidable.*

Proof. Consider a system $\mathcal{S} = (X, A, \rightarrow, \leq, init)$ that is cover-WSTS, and let us consider a state $y \in X$. To find a certificate of coverability (if it exists), we cannot use Procedure 1 since general monotony is not satisfied and then, in general, $\downarrow Post^*(init) \neq \downarrow Post^*(\downarrow init)$ but we can use a variation of Procedure 1, where we iteratively compute $init$, $Post(init)$, $Post(Post(init))$, and so on, and at each step check if $y \leq x$ for some x in the computed set. This can be done because $\mathcal{S}$ is finitely branching and the sets $Post^k(init)$ are computable for all $k \geq 0$. Hence, if there exists a state that can cover y reachable from $init$, it will eventually be found.

Now, let us prove that Procedure 2 terminates for input y iff y is not coverable from $init$.

If Procedure 2 terminates, then at some point, the `while` condition is not satisfied and there exists a set D such that $y \notin D$ and $init \in D$ and $\downarrow Post(D) \subseteq D$. Moreover, $\downarrow Post^*(I) \subseteq I$ for every inductive invariant I (see Proposition 3.32). Hence, $Cover_{\mathcal{S}}(init) \subseteq D$, therefore, since $y \notin D$, we deduce that $y \notin Cover_{\mathcal{S}}(init)$ and then y is not coverable from $init$.

Note that every downward-closed subset of X decomposes into finitely many ideals since $(X, \leq)$ is wqo [FGL09]. Moreover, since $init$ is ideally effective, ideals of X may be effectively enumerated. By [BFM18] and [BFM17], for ideally effective systems, testing of inclusion of downward-closed sets, and checking the membership of a state in a downward-closed set, are both decidable.

To show the opposite direction, let us prove that if y is not coverable from $init$, the procedure terminates. It suffices to prove that $Cover_{\mathcal{S}}(init)$ is an inductive invariant. Indeed, this implies that $Cover_{\mathcal{S}}(init)$ is eventually found by Procedure 2 when y is not coverable from $init$.

Let us show $\downarrow Post(Cover_{\mathcal{S}}(init)) \subseteq Cover_{\mathcal{S}}(init)$. Let $y \in \downarrow Post(Cover_{\mathcal{S}}(init))$. Then, there exists x, x', y' such that $init \xrightarrow{*} x'$, $x \leq x'$, $x \rightarrow y'$ and $y \leq y'$. Hence, $x, x' \in Cover(init)$. And by cover-monotony, there exists $y'' \geq y'$ such that $x' \xrightarrow{*} y''$. Therefore, $init \xrightarrow{*} y''$ and $y \leq y' \leq y''$, hence, $y \in Cover_{\mathcal{S}}(init)$. Hence, the $init$-coverability problem is decidable. □

However, we see that the general coverability problem is undecidable for cover-WSTS. We see that as for branch-WSTS, for the class of cover-WSTS the property of $\downarrow Post^*(Y) \subseteq Y$ does not hold for sets excluding the coverability set from the initial state.

Theorem 3.34. *The coverability problem is undecidable for cover-WSTS.*

Proof. Given any counter machine $\mathcal{C} = (Q, V, T, q_0)$, let $\mathcal{S}_\mathcal{C} = (X_\mathcal{C}, Act_\mathcal{C}, \rightarrow_\mathcal{C}, \leq, init)$ be its transition system equipped with the natural order on counters. We can construct a system $\mathcal{S}'_\mathcal{C} = (X'_\mathcal{C}, Act_\mathcal{C}, \rightarrow'_\mathcal{C}, \leq, init')$ such that $\mathcal{S}'_\mathcal{C}$ is cover-monotone, and any state $x \in X_\mathcal{C}$ is coverable iff it is also coverable in $X'_\mathcal{C}$. The construction is as follows. We add a new control-state q from the initial state in the counter machine (q_0) reachable via a noop transition, therefore, $X'_\mathcal{C} = X_\mathcal{C} \cup \{(q, 0) \mid n \in \mathbb{N}\}$. This new control-state is a sink state, i.e. there are no transitions from q to any other control-state (except itself). Moreover, we let $init' = (q, 0)$. Note that $\mathcal{S}'_\mathcal{C}$ is cover-monotone, because there is no state reachable from $init'$, hence, the property is vacuously satisfied. However, for all other states, as we leave the system unchanged, we see that a state x is coverable in $\mathcal{S}_\mathcal{C}$ by a state y iff it is coverable in $\mathcal{S}'_\mathcal{C}$. Hence, coverability for counter machines reduces to the coverability problem for cover-WSTS, and coverability is therefore, undecidable for cover-WSTS. □

Chapter 4

Input-Boundedness

As we saw in the previous chapters, for general FIFO machines, boundedness, termination and reachability are all undecidable. Moreover, as FIFO systems are neither WSTS, nor branch-WSTS, the problem of analysing an interesting subclass with decidable properties is still left open.

There have been many works since the 1980s which have studied FIFO machines in which the input-language of a channel (i.e. the set of words that record the messages entering a channel) is included in the set $\mathit{Pref}(w_1^* w_2^* \ldots w_n^*)$ for words $w_1, w_2, \ldots, w_n$. We call this class of FIFO machines input-bounded.

In this chapter, we solve a problem that was left open in [GGLR87], the decidability of the reachability problem for input-bounded FIFO machines. We present a simulation of input-bounded FIFO machines by counter machines with restricted zero tests.

In fact, we actually solve the more general problem of rational-reachability, hence, we can deduce the decidability of other verification properties like control-state reachability, deadlock, unboundedness, and termination. We also study the natural dual of the input-bounded reachability problem, which are machines in which the output-language, or set of words received by each channel, is bounded. For this class, we are able to deduce the decidability of reachability, unboundedness, termination and control-state reachability.

We obtain EXPTIME upper bounds and NP lower bounds for the input-bounded reachability of FIFO machines with a single channel. And finally, following the bounded verification paradigm like in [EGM12, FP20], we suggest a methodology that would apply existing results on input-bounded FIFO machines to general FIFO machines.

4.1 Bounded reachability

We formally define a bounded language first. Let A be an alphabet.

bounded language

Definition 4.1 ([GS64])**.** *Let $w_1, \dots, w_n \in A^+$ be non-empty words where $n \geq 1$. A* bounded language *over $(w_1, \dots, w_n)$ is a language $L \subseteq w_1^* \dots w_n^*$.*

We always assume that a bounded language L is given together with its tuple $(w_1, \dots, w_n)$ and that $\mathit{Alph}(L) = \mathit{Alph}(w_1 \dots w_n)$. We say L is *distinct-letter* if $|w_1 \dots w_n|_a \leq 1$ for all $a \in A$. If $|w_1| = \dots = |w_n| = 1$, i.e. $w_1, \dots, w_n \in A$, then L is a *letter-bounded language.* Note that the set of bounded languages is closed under *Pref* and *Suf*.

bounded reachability set

input-bounded FIFO machine

We say that a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$ has a *bounded reachability set* if there is a tuple $(L_c)_{c \in \mathit{Ch}}$ of regular bounded languages $L_c \subseteq \Sigma_c^*$ such that, for all $(q, \mathbf{w}) \in \mathit{Reachset}_{\mathcal{M}}$, we have $\mathbf{w} \in \prod_{c \in \mathit{Ch}} L_c$. We define the input-language of a FIFO channel c as the set of all words that are sent into the channel, i.e. $\mathit{proj}_{c!}(\mathit{Traces}(\mathcal{M}))$. Moreover, we say that the machine is input-bounded if there is a tuple $(L_c)_{c \in \mathit{Ch}}$ of regular bounded languages $L_c \subseteq \Sigma_c^*$ such that, for all $c \in \mathit{Ch}$, $\mathit{proj}_{c!}(\mathit{Traces}(\mathcal{M})) \subseteq L_c$.

Let us also briefly recall a few notions that have been studied in the literature. Monogeneous FIFO nets [Fin82, MF85, FS01] are machines where input-languages of channels c are included in $\mathit{Pref}(u_c \cdot v_c^*)$ where u_c, v_c are two words associated with c. Linear FIFO nets (or as we will refer to them, distinct-letter-bounded FIFO machines) [CF87] are machines where the input-language is distinct-letter. Finally, flat machines are a class of FIFO machines such that every control-state is part of at most one non-elementary loop[FP19]. All these classes are subclasses of input-bounded FIFO machines.

We show that restricting the reachability set to be bounded is not sufficient to obtain a decidable reachability problem, by simulating any two counter (Minsky) machine by a FIFO machine with fixed languages L_c.

Theorem 4.2. *The reachability problem is undecidable for FIFO machines (with a single channel) with a (given) bounded reachability set.*

Proof. We prove this by reducing reachability in a two counter machine to reachability in a FIFO machine that has a single channel and a bounded reachability set. Intuitively, for every state (q, v_1, v_2) reachable in the counter machine, where q is the control-state and v_1, v_2 are the counter contents, we are able to reach a state (q, w) in the FIFO machine, where $w = \#a^{v_1}\#b^{v_2}\#$.

Consider a counter machine $\mathcal{C} = (Q_{\mathcal{C}}, V, T_{\mathcal{C}}, q_{init})$, where $Q_{\mathcal{C}}$ is the set of control-states, $V = \{x_1, x_2\}$ is the set of counters, q_{init} the initial control-state, and

$T = Q \times Act \times Q$ such that $Act = \{\mathsf{inc}(x), \mathsf{dec}(x), \mathsf{zero}(x) \mid x \in V\}$ to denote the increment, decrement and testing for zero of the counters.

We construct a FIFO machine $\mathcal{M} = (Q_\mathcal{M}, Ch, \Sigma, T_\mathcal{M}, q_0)$ such that $Q_\mathcal{M} = Q_\mathcal{C} \cup Q_{\mathsf{mid}}$, where Q_{mid} is a set of intermediate control-states, $Ch = \{c\}$, $\Sigma = \{a, b, \#\}$, and $q_0 = q_{init}$.

For every transition $\delta_\mathcal{C} \in T_\mathcal{C}$:

- If $\delta_\mathcal{C} = (q, \mathsf{inc}(x_1), q')$: We add the sequence of transitions as shown in Figure 4.1 to $T_\mathcal{M}$. Put simply, if counter x_1 is incremented, we increment the number of a's in the channel by 1. By rotating the tape contents, we ensure that the channel contents at q' are of the form $\#a^{v_1}\#b^{v_2}\#$.

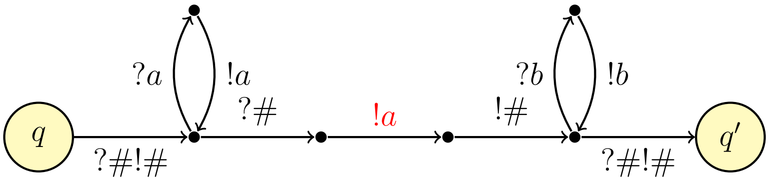


Figure 4.1: Incrementing x_1, corresponding to rotating the channel contents and the addition of the letter a as shown in the red transition.

- If $\delta_\mathcal{C} = (q, \mathsf{dec}(x_1), q')$: We add the sequence of transitions as shown in Figure 4.2 to $T_\mathcal{M}$. As before, if counter x_1 is decremented, we decrement the number of a's in the channel by 1. By rotating the tape contents, we ensure that the channel contents at q' are of the form $\#a^{v_1}\#b^{v_2}\#$. Note that, for the FIFO machine, if there is no a in the channel, then the sequence will not be completed and q' cannot be reached.

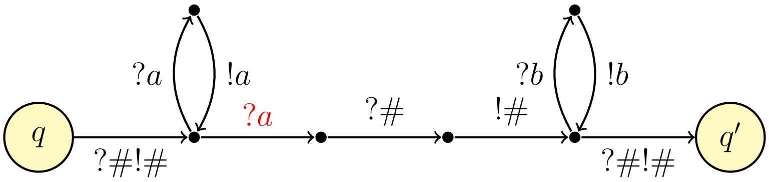


Figure 4.2: Decrementing x_1, corresponding to rotating the channel contents and the removal of the letter a as shown in the red transition.

- If $\delta_\mathcal{C} = (q, \mathsf{zero}(x_1), q')$: In this case, we have the sequence as shown in Figure 4.3 added to $T_\mathcal{M}$. This transition sequence can only be executed if there is no occurrence of a in the channel.

Similar constructions are made for the transitions involving counter x_2. Consider the transition system associated to $\mathcal{M}$, i.e. $\mathcal{S}_\mathcal{M}$. Observe that the channel contents

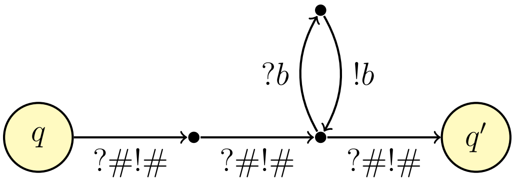


Figure 4.3: Testing if $x_1 = 0?$, which corresponds to ensuring there are no occurrences of a.

always belong to the bounded language $\#^*a^*\#^*b^*\#^*a^*\#^*b^*\#^*$. Hence, $\mathcal{S}_{\mathcal{M}}$ has a bounded reachability set.

Consider the associated transition systems, i.e. $\mathcal{S}_{\mathcal{C}}$ and $\mathcal{S}_{\mathcal{M}}$, we prove the following claim: There is a transition t of the form $(q, v_1, v_2) \xrightarrow{t} (q', v_1', v_2')$ in $\mathcal{S}_{\mathcal{C}}$ iff there is a transition sequence of the form $(q, w) \xrightarrow{\alpha_1} (q_1, w_1) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (q', w')$ in $\mathcal{S}_{\mathcal{M}}$ such that $w = \#a^{v_1}\#b^{v_2}\#$, states $q_1, \ldots, q_{n-1} \in Q_{\mathsf{mid}}$ and $w_1, \ldots, w_{n-1} \in \Sigma^*$ and $w' = \#a^{v_1'}\#b^{v_2'}\#$.

Let us first show that for every transition t of the form $(q, v_1, v_2) \xrightarrow{t} (q', v_1', v_2')$ in $\mathcal{S}_{\mathcal{C}}$ there is a corresponding sequence in $\mathcal{S}_{\mathcal{M}}$ as defined above. Without loss of generality, let us assume that the transition t modifies counter x_2. (The same arguments can be extended in the case of counter x_1). By construction, for an increment, we know that there is a transition sequence that can be executed from state (q, w) in $\mathcal{S}_{\mathcal{M}}$, which leads to (q', w'). Similarly, if t involves decrement of the counter x_2, then we know that $v_2 \geq 1$. Hence, there is at least one occurrence of the letter b in w. Hence, we can successfully execute the transition sequence from (q, w) to (q', w') as per the construction. If t tests x_2 to zero, then we know that there is no occurrence of b in w. Hence, in $\mathcal{S}_{\mathcal{M}}$, the transition sequence corresponding to the zero test can be executed to reach from (q, w) to (q', w').

To show the opposite direction, we assume there is a transition sequence of the form $(q, w) \xrightarrow{\alpha_1} (q_1, w_1) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (q', w')$ in $\mathcal{S}_{\mathcal{M}}$ such that $w = \#a^{v_1}\#b^{v_2}\#$, control-states $q_1, \ldots, q_{n-1} \in Q_{\mathsf{mid}}$ and $w_1, \ldots, w_{n-1} \in \Sigma^*$ and $w' = \#a^{v_1'}\#b^{v_2'}\#$. Hence, there is a corresponding transition in $\mathcal{S}_{\mathcal{C}}$ as all the transition sequences between states in $Q_{\mathcal{C}}$ are constructed based on transitions in the counter machine. Moreover, if the transition system corresponding to a zero test of say, counter x_2, is executed, we can be sure that there is no b in w. Hence, $v_2 = 0$. Hence, from the state (q, v_1, v_2) the zero test of x_2 can be executed and (q', v_1', v_2') can be reached. The same arguments can be extended for all transition sequences, and we prove our claim.

We can extend this claim by induction to consider any transition sequences in $\mathcal{S}_{\mathcal{C}}$. Hence, we can derive the following claim: (q, v_1, v_2) is reachable from $(q_{init}, 0, 0)$ iff (q, w) is reachable from $(q, \#\#\#)$, where $w = \#a^{v_1}\#b^{v_2}\#$. And since reachability is undecidable for 2-counter machines, we can say it is also undecidable for FIFO machines with a bounded reachability set.

□

Note that the above result only uses a single channel, and still the reachability problem is undecidable. Moreover, in the proof above, $|\Sigma| = 3$, but we can modify the construction to use only 2 letters (we represent both counters by the same letter a). Hence, for a single channel and $|\Sigma| = 2$, the reachability problem is still undecidable for FIFO machines with a bounded reachability set. Furthermore, if we modify the restriction to consider non-repeating letters (i.e. distinct-letter bounded), we still obtain undecidability of the reachability problem. We know that if $|\Sigma| = 1$, we can represent the channel by a single counter, for which reachability is known to be decidable.

We see in the previous proof that if there is a loop in the counter machine, the input-language of the channel is not bounded: If we have a transition from q in the original machine of the kind $(q, \mathsf{inc}(x_i), q)$, there exists a corresponding $\sigma \in \mathit{Act}_{\mathcal{M}}^*$ in the FIFO machine such that $\mathit{proj}_{c!}(\sigma) = (\#a^*\#a^*\#)^*$, which is not bounded. Hence, the input-language need not be a bounded language even if the reachability set is. Furthermore, if the original counter machine is not flat, the FIFO machine would not be flat either, since there can be control-states that are in more than one elementary loop.

We therefore consider a different restriction to obtain decidability of reachability for FIFO systems. The reachability problem for FIFO systems can be seen as follows:

Given a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$, and a state $(q, \mathbf{w})$ in the set of states $X_{\mathcal{M}}$ of the associated transition system, do we have $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L)$ where $L = \mathit{Act}_{\mathcal{M}}^*$, i.e. is there any path from the initial state to $(q, \mathbf{w})$? Since this problem is undecidable, we are looking for language classes $\mathfrak{C}$ that render the problem decidable under the restriction that $L \in \mathfrak{C}$.

For a given FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$, we are interested in computing $\mathit{Tracereach}_{\mathcal{M}}(L)$ where $L \subseteq \mathit{Act}_{\mathcal{M}}^*$ is *input-bounded* i.e. for every channel c, the sequence of messages that are sent through channel c is from a given regular bounded language $L_c \subseteq \Sigma_c^*$.

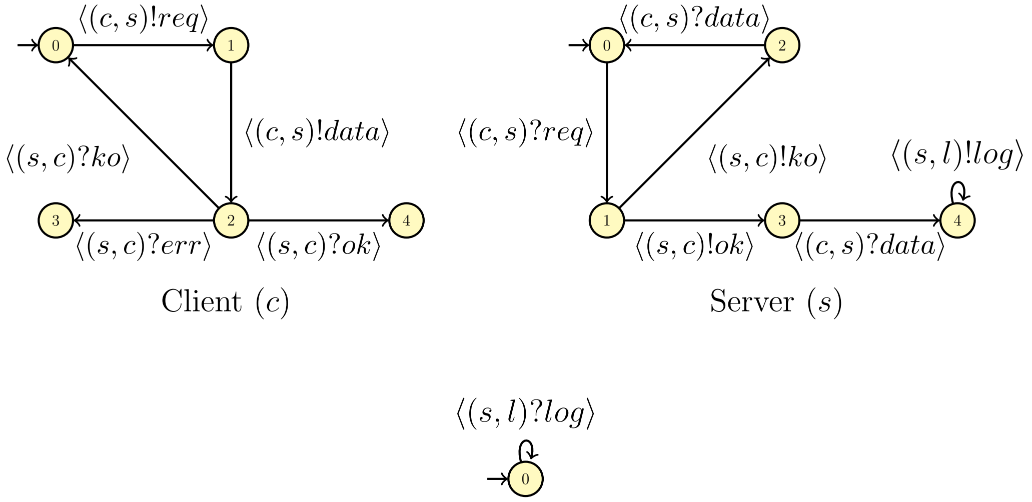
With this, given a tuple $L = (L_c)_{c \in \mathit{Ch}}$ of bounded languages $L_c \subseteq \Sigma_c^*$, we set $L_! = \{\sigma \in \mathit{Act}_{\mathcal{M}}^* \mid \mathit{proj}_{c!}(\sigma) \in L_c \text{ for all } c \in \mathit{Ch}\}$ and $L_? = \{\sigma \in \mathit{Act}_{\mathcal{M}}^* \mid \mathit{proj}_{c?}(\sigma) \in L_c \text{ for all } c \in \mathit{Ch}\}$. We observe that, if all L_c are regular, then so are $L_!$ and $L_?$.

We now define the input-bounded (IB) reachability problem as follows:

Decision Problem: INPUT-BOUNDED (IB) REACHABILITY

Input:	a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$
	a tuple of non-empty regular bounded languages $L = (L_c)_{c \in Ch}$
	a state $(q, \mathbf{w}) \in X_{\mathcal{M}}$
Question:	is $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$?

Note that, if $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_{\mathcal{M}} (q, \mathbf{w})$ and $\sigma \in L_!$, then because we are considering FIFO channel, we also have $\sigma \in \mathit{Pref}(L_?)$. Thus, $\mathit{Tracereach}_{\mathcal{M}}(L_!) = \mathit{Tracereach}_{\mathcal{M}}(L_! \cap \mathit{Pref}(L_?))$ so that we can restrict to action sequences from $L_! \cap \mathit{Pref}(L_?)$. We will call $L_! \cap \mathit{Pref}(L_?)$ the set of *valid* words.


$\langle (c,s)!req \rangle$
$\langle (c,s)!data \rangle$
$\langle (s,c)?ko \rangle$
$\langle (s,c)?err \rangle$
$\langle (s,c)?ok \rangle$
0
1
2
3
4
$\langle (c,s)?data \rangle$
$\langle (c,s)?req \rangle$
$\langle (s,c)!ko \rangle$
$\langle (s,l)!log \rangle$
$\langle (s,c)!ok \rangle$
$\langle (c,s)?data \rangle$
0
1
2
3
4
$\langle (s,l)?log \rangle$
0

Client (c)

Server (s)

Logger (l)

Figure 4.4: The Client-Server-Logger example from [LY19]

Example 12. Let us consider a simple model of a Client-Server-Logger system from [LY19], which consists of three processes, a client (denoted by c), server (s) and logger (l), and channels between them. The client sends to the server a request, followed by some data in a fire-and-forget fashion, i.e. before receiving any response from the server. The server either grants the requests and sends a log message to the logger, or sends a message *ko* to the client, after which the client tries again. This system is neither flat nor monogeneous nor linear. However, the input-languages of the channels are $L_{cs} = (\mathit{req} \cdot \mathit{data})^*$, $L_{sc} = (\mathit{ko})^* \cdot (\mathit{ok})^*$ and $L_{sl} = (\mathit{log})^*$. Hence, they are all bounded languages. Therefore, computing the reachability set for the entire system is equivalent to computing reachability over a set of bounded languages.

4.2 The IB rational-reachability problem.

Actually, instead of reachability of a single state as stated above, we study a more general problem, called the *input-bounded rational-reachability problem.* It asks whether a state $(q, \mathbf{w})$ is reachable for some channel contents $\mathbf{w}$ from a given *rational* relation. Let us formally define this notion.

Semi-Linear Sets. Let $d \geq 1$ be a positive integer. A subset Y of $\mathbb{N}^d$ is a linear set of dimension d if there exists a basis $\mathbf{b} \in \mathbb{N}^d$ and a finite set of periods $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\} \subseteq \mathbb{N}^d$ such that

linear set

$$Y = \{\mathbf{b} + i_1\mathbf{p}_1 + \ldots + i_m\mathbf{p}_m \mid i_1, \ldots, i_m \in \mathbb{N}\}.$$

The basis along with the periods are referred to as the generators of the linear set Y. A finite union of linear sets is called a semi-linear set. Every finite subset of $\mathbb{N}^d$ is semi-linear (including the empty set $\emptyset$) - it can be seen as a finite union of linear sets with no period.

semi-linear set

Example 13. The set $Y_1 = \{(x, y) \mid x \geq 1\} \subseteq \mathbb{N}^2$ is a linear (hence, semi-linear) set, with $\mathbf{b} = (1, 0)$, $\mathbf{p}_1 = (0, 1)$ and $\mathbf{p}_2 = (1, 0)$. On the other hand, $Y_2 = \{(x, y) \mid y \leq x^2\}$ is not a semi-linear set.

It is well known that the language of a non-deterministic finite-state automaton (NFA) over a d-letter alphabet $\Sigma = \{a_1, \ldots, a_d\}$ can be seen as a semi-linear set [Par66]. Given a word $w \in \Sigma^*$, we define the Parikh map of w by $\phi(w) = (|w|_{a_1}, \ldots, |w|_{a_d})$. By extension, the Parikh map of a language L, is denoted by $\phi(L) = \{\phi(w) \mid w \in L\}$. Hence, if we are given a semilinear set Y, we can construct an NFA which accepts a language L such that $\phi(L) = Y$.

Parikh map

Rational and Recognizable Relations. We say that a subset $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$ is *rational* if there is a regular word language $R \subseteq \Theta^*$ over the alphabet $\Theta = \prod_{c \in Ch}(\Sigma_c \cup \{\varepsilon\})$ such that $Rel = \{(\mathbf{a}_c^1 \cdot \ldots \cdot \mathbf{a}_c^n)_{c \in Ch} \mid \mathbf{a}^1 \ldots \mathbf{a}^n \in R$ with $n \in \mathbb{N}$ and $\mathbf{a}^i = (\mathbf{a}_c^i)_{c \in Ch} \in \Theta$ for $i \in \{1, \ldots, n\}\}$. Here, $\mathbf{a}_c^1 \cdot \ldots \cdot \mathbf{a}_c^n \in \Sigma_c^*$ is the concatenation of all $\mathbf{a}_c^i \in \Sigma_c \cup \{\varepsilon\}$ while ignoring the neutral element ε. For example, in the presence of two channels, $Rel = \{(a^m, b^n) \mid m \geq n\}$ is a rational relation, witnessed by $R = ((a, b) + (a, \varepsilon))^*$. In the following, we will always assume that a rational relation is given in terms of a finite automaton for the underlying regular language R.

rational set

A relation $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$ is called *recognizable* if it is the finite union of relations of the form $\prod_{c \in Ch} R_c$ where all $R_c \subseteq \Sigma_c^*$ are regular languages. Note that every recognizable relation is rational while the converse is, in general, false.

recognizable relation

Parikh image of a relation

We define the *Parikh image* of a relation $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$ as

$$\text{Parikh}(Rel) = \{(\pi_a)_{a \in \Sigma} \in \mathbb{N}^\Sigma \mid \exists \mathbf{w} = (\mathbf{w}_c)_{c \in Ch} \in Rel \text{ such that } \pi_a = |\mathbf{w}_c|_a \text{ for all } c \in Ch \text{ and } a \in \Sigma_c\}$$

It is well known that, if Rel is rational, then Parikh(Rel) is semi-linear.

We are now prepared to define the input-bounded (IB) rational-reachability problem.

Decision Problem: INPUT-BOUNDED (IB) RATIONAL-REACHABILITY

Input:	a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$
	a tuple $L = (L_c)_{c \in Ch}$ of non-empty regular bounded languages
	a control-state $q \in Q$, and
	a rational relation $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$
Question:	is $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$ for some $\mathbf{w} \in Rel$?

We shall now try to prove that the above problem is decidable for FIFO machines.

Let $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$, and let $L = (L_c)_{c \in Ch}$ be a tuple of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$ over $(w_{c,1}, \ldots, w_{c,n_c})$. We proceed by reduction to counter machines. Intuitively, we represent the contents of channel c in terms of a set of counters, one for every component $w_{c,i}$.

However, in order to simulate FIFO machines by counter machines, we require that the FIFO machine (along with the bounded languages associated to it) be in a normal form. This can be achieved at the expense of an exponential blow-up of the FIFO machine.

normal form

Definition 4.3. *We say that $\mathcal{M}$ and L are in* normal form *if the following hold:*

1. *For all $c \in Ch$, $\Sigma_c \subseteq Alph(L_c)$ and L_c is distinct-letter.*
2. *We have $Traces((Q, Act_{\mathcal{M}}, T, q_0)) \subseteq Pref(\mathcal{V})$ where $\mathcal{V} = L_! \cap Pref(L_?)$. Note that $(Q, Act_{\mathcal{M}}, T, q_0)$ is the finite transition system induced by the control graph of $\mathcal{M}$.*

set of valid words $\mathcal{V}$

Distinct-Letter Property. Let $\hat{\Sigma}$ be an alphabet. Consider the bounded language $\hat{L} \subseteq \hat{\Sigma}^*$ over $(\hat{w}_1, \ldots, \hat{w}_n)$. For $i \in \{1, \ldots, n\}$, let $m_i = |\hat{w}_i|$ be the length of the i-th word. Furthermore, let $m = m_1 + \ldots + m_n$. Let Σ denote the alphabet $\{\ell_1, \ldots, \ell_m\}$. We construct a distinct-letter bounded language $L \subseteq \Sigma^*$ over

$(w_1, \dots, w_n)$ such that $w_i = \ell_{j+1} \dots \ell_{j+m_i}$, where $j = \sum_{p<i} m_p$, for all $1 \leq i \leq n$. In order to obtain the distinct-letter language L, we first construct the homomorphism $h : \Sigma^* \to \hat{\Sigma}^*$ where $h(\ell_i)$ is the i-th letter of the word $\hat{w}_1 \dots \hat{w}_n$. We obtain L as $h^{-1}(\hat{L}) \cap w_1^* \dots w_n^*$, hence, preserving regularity and boundedness. We then remove all the words from the bounded language $(w_1, \dots, w_n)$ (and their letters from Σ) whose letters do not occur in L.

Example 14. For example, suppose we have $\hat{L} = (ab)^*bb^*$ over (ab, b). We determine the language L over (a_1a_2, a_3). The homomorphism $h : \{a_1, a_2, a_3\}^* \to \{a, b\}^*$ is given by $h(a_1) = a$ and $h(a_2) = h(a_3) = b$. We have $h^{-1}(\hat{L}) = (a_1(a_2 + a_3))^*(a_2 + a_3)(a_2 + a_3)^*$, which we intersect with $(a_1a_2)^*a_3^*$. We thus get the regular bounded language $L = (a_1a_2)^*a_3a_3^*$ over (a_1a_2, a_3). All letters from $\{a_1, a_2, a_3\}$ occur in L so we are done.

Trace Property. Given a FIFO machine $\hat{\mathcal{M}} = (\hat{Q}, Ch, \hat{\Sigma}, \hat{T}, \hat{q}_0)$ and the tuple $\hat{L} = (\hat{L}_c)_{c \in Ch}$ of non-empty regular bounded languages $\hat{L}_c \subseteq \hat{\Sigma}_c^*$, we now construct the FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$ such that $\mathit{Traces}((Q, \mathit{Act}_{\mathcal{M}}, T, q_0)) \subseteq \mathit{Pref}(\mathcal{V})$ with $\mathcal{V} = L_! \cap \mathit{Pref}(L_?)$. Let $h_c : \Sigma_c^* \to \hat{\Sigma}_c^*$ be the homomorphism defined earlier for channel $c \in Ch$. First, to take care of the homomorphisms h_c, we define the transition relation $h^{-1}(\hat{T}) = \{(q, \langle c!e \rangle, q') \mid (q, \langle c!a \rangle, q') \in \hat{T}$ and $e \in h_c^{-1}(a)\} \cup \{(q, \langle c?e \rangle, q') \mid (q, \langle c?a \rangle, q') \in \hat{T}$ and $e \in h_c^{-1}(a)\}$. Thus, the set of actions of $\mathcal{M}$ will be $\mathit{Act}_{\mathcal{M}} = \{\langle c!e \rangle \mid c \in Ch$ and $e \in \Sigma_c\} \cup \{\langle c?e \rangle \mid c \in Ch$ and $e \in \Sigma_c\}$.

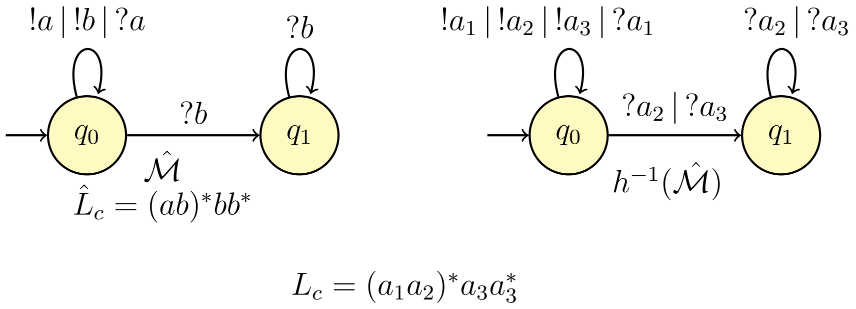


Figure 4.5: The FIFO machine $\hat{\mathcal{M}}$ (left) and the modified FIFO machine $h^{-1}(\hat{\mathcal{M}})$ with distinct letters

Example 15. Consider a FIFO machine $\hat{\mathcal{M}}$ with one single channel c (which is therefore omitted) as shown in Figure 4.5 (left) and its bounded language $\hat{L}_c = (ab)^*bb^*$ over (ab, b). Recall from Example 14 that the corresponding homomorphism h_c maps a_1 to a and both a_2 and a_3 to b, and that we obtain $L_c = (a_1a_2)^*a_3a_3^*$. To continue the example, a transition $t = (q, \langle c!b \rangle, q')$ would be replaced by $h^{-1}(t)$ with the two transitions $(q, \langle c!a_2 \rangle, q')$ and $(q, \langle c!a_3 \rangle, q')$. We construct $h^{-1}(\hat{\mathcal{M}}) = (\hat{Q}, Ch, \Sigma, h^{-1}(\hat{T}), \hat{q}_0)$ as shown in the Figure 4.5 (right).

To guarantee trace inclusion in $\mathit{Pref}(\mathcal{V})$, we follow the same idea as in Section 3.5.2. To recall, we will consider a deterministic (not necessarily complete) finite automaton $\mathcal{A} = (Q_{\mathcal{A}}, \mathit{Act}_{\mathcal{M}}, T_{\mathcal{A}}, q^0_{\mathcal{A}}, F_{\mathcal{A}})$, with set of final states $F_{\mathcal{A}} \subseteq Q_{\mathcal{A}}$, whose language is $L(\mathcal{A}) = \mathcal{V}$ and where, from every state, a final state is reachable in the finite graph $(Q_{\mathcal{A}}, T_{\mathcal{A}})$. With this, we define $\mathcal{M}$ as the product of the FIFO machine $h^{-1}(\hat{\mathcal{M}}) = (\hat{Q}, \mathit{Ch}, \Sigma, h^{-1}(\hat{T}), \hat{q}_0)$ and $\mathcal{A}$ in the expected manner. In particular, the set of control-states of $\mathcal{M}$ is $\hat{Q} \times Q_{\mathcal{A}}$, and its initial state is the pair $(\hat{q}_0, q^0_{\mathcal{A}})$.

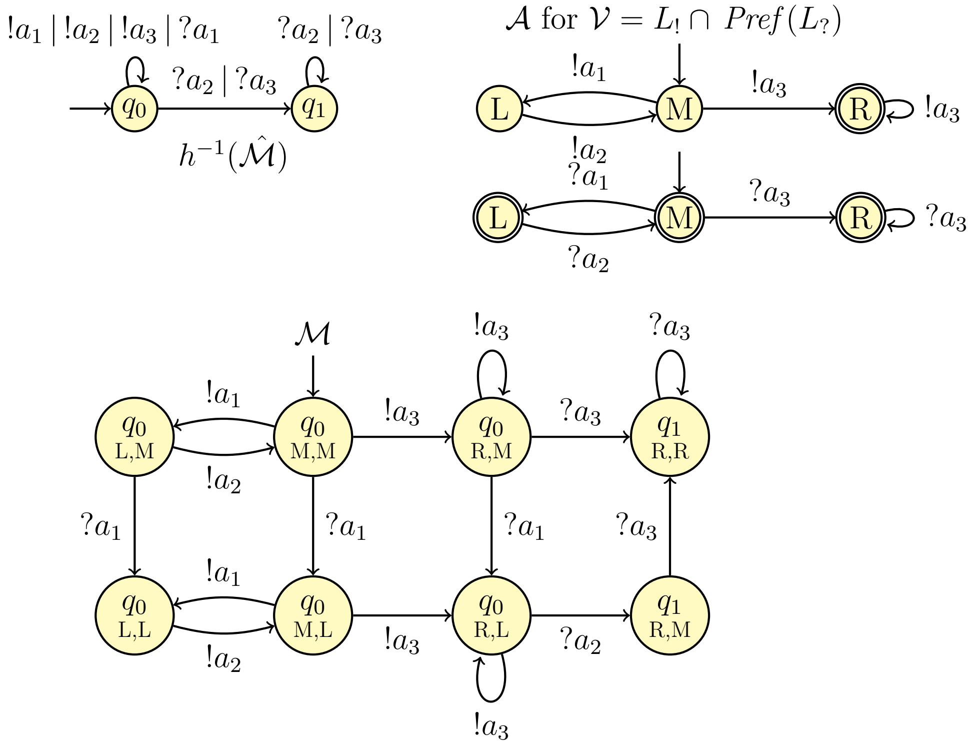


Figure 4.6: We construct a FIFO machine $\mathcal{M}$ (bottom), together with $L_c = (a_1 a_2)^* a_3 a_3^*$, in normal form as the product of $h^{-1}(\hat{\mathcal{M}})$ (top left) and an automaton for $\mathcal{V}$ (top right).

Example 16. Figure 4.6 illustrates the completion of the normalization procedure for the FIFO machine $\hat{\mathcal{M}}$. We see that $\mathcal{M}$ (see Figure 4.6 bottom) is the product of $h^{-1}(\hat{\mathcal{M}})$ (top left) and a finite automaton $\mathcal{A}$ for $\mathcal{V} = L_! \cap \mathit{Pref}(L_?)$ (obtained as the shuffle of the two finite automata on the top right). The state names in $\mathcal{M}$ reflect the states of $\hat{\mathcal{M}}$ and $\mathcal{A}$ they originate from. We depict only accessible states of $\mathcal{M}$ from which we can still complete the word read so far to a word in $\mathcal{V}$. For example, $(q_1, \mathrm{M}, \mathrm{L})$ and $(q_1, \mathrm{L}, \mathrm{R})$ would no longer allow us to reach the final state R of the $L_!$-component, and hence, are not visible in the figure.

Now suppose we are given a reachability query for $\hat{\mathcal{M}}$ in terms of $\hat{q} \in \hat{Q}$ and a rational relation $\hat{\mathit{Rel}} \subseteq \prod_{c \in \mathit{Ch}} \hat{\Sigma}_c^*$. The lemma below shows how to reduce it to a reachability query in $\mathcal{M}$. Here, for $\mathbf{w} = (\mathbf{w}_c)_{c \in Ch} \in \prod_{c \in Ch} \Sigma_c^*$, we define $h(\mathbf{w}) = (h_c(\mathbf{w}_c))_{c \in Ch} \in \prod_{c \in Ch} \hat{\Sigma}_c^*$. Note that $h^{-1}(\widehat{Rel})$ is rational. Furthermore, $h : \Sigma^* \to \hat{\Sigma}^*$ is defined by $h(a) = h_c(a)$ for all $c \in Ch$ and $a \in \Sigma_c$, and we extend this to $h : Act_{\mathcal{M}}^* \to Act_{\hat{\mathcal{M}}}^*$ in the expected manner.

Lemma 4.4. *We have $(\hat{q}, \hat{\mathbf{w}}) \in Tracereach_{\hat{\mathcal{M}}}(\hat{L}_!)$ for some $\hat{\mathbf{w}} \in \widehat{Rel}$ iff $((\hat{q}, q_{\mathcal{A}}), \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_!)$ for some $q_{\mathcal{A}} \in Q_{\mathcal{A}}$ and $\mathbf{w} \in h^{-1}(\widehat{Rel})$.*

Proof. Let us assume we have $(\hat{q}, \hat{\mathbf{w}}) \in Tracereach_{\hat{\mathcal{M}}}(\hat{L}_!)$ for some $\hat{\mathbf{w}} \in \widehat{Rel}$. Hence, there exists $\hat{\sigma} \in \hat{L}_!$ such that $(\hat{q}_0, \boldsymbol{\varepsilon}) \xrightarrow{\hat{\sigma}} (\hat{q}, \hat{\mathbf{w}})$. For channel c, let $\hat{w}_c = proj_{c!}(\hat{\sigma})$. Since $\hat{\sigma} \in \hat{L}_!$, we have $\hat{w}_c \in \hat{L}_c$ for all $c \in Ch$. Let $w_c \in L_c = h_c^{-1}(\hat{L}_c) \cap (w_{c,1})^* \dots (w_{c,n_c})^*$ such that $h_c(w_c) = \hat{w}_c$. There is a unique $\sigma \in Act_{\mathcal{M}}^*$ such that $h(\sigma) = \hat{\sigma}$ and $proj_{c!}(\sigma) = w_c$ and $proj_{c?}(\sigma) \in Pref(w_c)$ for all $c \in Ch$. Here, $h : \Sigma^* \to \hat{\Sigma}^*$ is defined by $h(a) = h_c(a)$ for all $c \in Ch$ and $a \in \Sigma_c$, and we extend this to $h : Act_{\mathcal{M}}^* \to Act_{\hat{\mathcal{M}}}^*$ in the expected manner. Note that $\sigma \in L_! \cap Pref(L_?)$. Hence, we know that in the FIFO machine $h^{-1}(\hat{\mathcal{M}})$, one has $(\hat{q}_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma} (\hat{q}, \mathbf{w})$ for some $\mathbf{w}$ (by construction of $h^{-1}(\hat{T})$), and that $\sigma \in L(\mathcal{A})$. Therefore, since $\mathcal{M}$ is a product of the two machines, we can deduce that there is a run in $\mathcal{M}$ of the kind $((\hat{q}_0, q_{\mathcal{A}}^0), \boldsymbol{\varepsilon}) \xrightarrow{\sigma} (\hat{q}, q_{\mathcal{A}}), \mathbf{w})$, for some value of $q_{\mathcal{A}}$. Furthermore, by $h(\sigma) = \hat{\sigma}$, we have $h(\mathbf{w}) = \hat{\mathbf{w}}$. Hence, $\mathbf{w} \in h^{-1}(\widehat{Rel})$.

Conversely, let us assume that $((\hat{q}, q_{\mathcal{A}}), \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_!)$ for some $q_{\mathcal{A}} \in Q_{\mathcal{A}}$ and channel contents $\mathbf{w} \in h^{-1}(\widehat{Rel})$. Then, we know that there exists $\sigma \in L_!$ such that $((\hat{q}_0, q_{\mathcal{A}}^0), \boldsymbol{\varepsilon}) \xrightarrow{\sigma} (\hat{q}, q_{\mathcal{A}}), \mathbf{w})$. Let $\hat{\sigma} = h(\sigma)$. Since $\sigma \in L_!$, we have $proj_{c!}(\sigma) \in L_c$ for all $c \in Ch$. In particular, $proj_{c!}(\sigma) \in h_c^{-1}(\hat{L}_c)$ and, therefore, $h_c(proj_{c!}(\sigma)) = proj_{c!}(h(\sigma)) \in \hat{L}_c$. We deduce $\hat{\sigma} \in \hat{L}_!$. Furthermore, we can execute $\hat{\sigma}$ in $\hat{\mathcal{M}}$ (by construction) to reach states $(\hat{q}, \hat{\mathbf{w}})$ for some $\hat{\mathbf{w}}$. By $\hat{\sigma} = h(\sigma)$, we have $\hat{\mathbf{w}} = h(\mathbf{w})$. Therefore, $\hat{\mathbf{w}} \in \widehat{Rel}$. □

Reduction of Normal Form to Counter Machine The next step in our proof is to construct a counter machine $\mathcal{C}$ such that the IB rational-reachability problem for $\mathcal{M}$ can be solved by answering a reachability query in $\mathcal{C}$, However, as we already know, for general counter machines, reachability is undecidable. Hence, in order to obtain decidability, we impose a restriction on counter machines.

We modify the syntactic restriction in Section 2.3, where we imposed that once a counter has been tested for zero, it cannot be incremented any further. We now impose this notion not syntactically but on the semantics of the counter machine. To define this, let $L_{\mathcal{C}}^{\mathsf{zero}}$ be the set of words $(op_1(v_1), Z_1) \dots (op_n(v_n), Z_n) \in A_{\mathcal{C}}^*$ $L_{\mathcal{C}}^{\mathsf{zero}}$ such that, for every two positions $1 \leq i \leq j \leq n$, we have $v_j \notin Z_i$. As discussed before, this restriction leads us to an extension of VASS. Hence, the reachability problem is decidable, which we demonstrate below.

Theorem 4.5. *The following problem is decidable (though inherently non-elementary): Given a counter machine $\mathcal{C} = (Q, V, T, q_0)$, a regular language $L \subseteq A_{\mathcal{C}}^*$, a control state $q \in Q$, and a semi-linear set $\mathcal{V} \subseteq \mathbb{N}^V$, do we have $(q, \mathbf{v}) \in \mathit{Tracereach}_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap L)$ for some $\mathbf{v} \in \mathcal{V}$?*

Proof sketch. We reduce reachability in counter systems with restricted zero tests in the presence of a semi-linear target set to state reachability in VASS. The idea is to have $2^{|V|}$ many copies of the counter machine, such that instead of testing for zero, we branch into a copy corresponding to storing the information of the counter to be tested to zero (and we do not modify the counter further in this copy). Finally, to check whether a counter valuation belongs to $\mathcal{V}$, we branch when we are in a control-state q into a component that decrements counters accordingly and eventually checks whether they are all zero.

Hence, reachability for counter machines with restricted zero tests straightforwardly reduces to state-reachability in VASS, which is decidable [May84], though inherently non-elementary [CLL$^+$19]. □

Henceforth, we suppose that $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$ and $L = (L_c)_{c \in \mathit{Ch}}$ are in normal form, where L_c is a bounded language over $(w_{c,1}, \ldots, w_{c,n_c})$. In particular, for every letter $a \in \Sigma_c$, there is a unique index $i \in \{1, \ldots, n_c\}$ such that $a \in \mathit{Alph}(w_{c,i})$. We denote this index i by i_a.

We build a counter machine $\mathcal{C}$ such that the IB rational-reachability problem for $\mathcal{M}$ can be solved by answering a reachability query in $\mathcal{C}$, using Theorem 4.5. Each run in $\mathcal{C}$ will simulate a run in $\mathcal{M}$. However, counter values are just natural numbers and hence, store less information than channel contents, which could have a non-unary alphabet. To tackle this, the idea is to represent each word $w_{c,i}$ of a tuple $(w_{c,1}, \ldots, w_{c,n_c})$ as a counter $x_{(c,i)}$. Then, we can replace the send actions by increments of the corresponding counter, and similarly receive actions can be replaced by decrements. More precisely, $\langle c!a \rangle$ becomes $(\mathsf{inc}(x_{(c,i_a)}), \emptyset)$, thus incrementing the counter associated with the unique word $w_{c,i}$ in which a occurs. Similarly, $\langle c?a \rangle$ translates to $(\mathsf{dec}(x_{(c,i_a)}), Z)$ (for suitable Z).

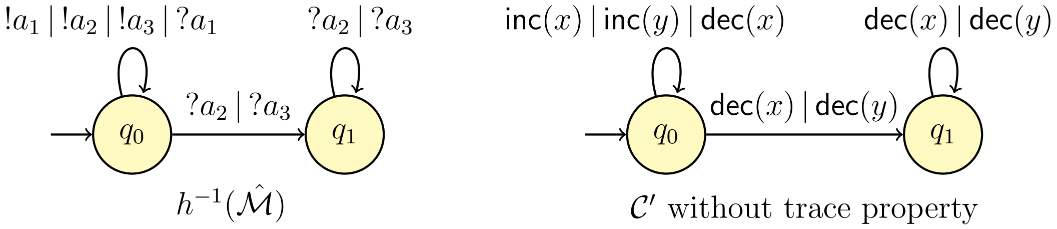
This alone does not put us in a position yet where, from a counter valuation, we can infer unique channel contents, as we only currently have information about the number of letters of the word in the channel. To overcome this, we additionally store the last messages that are sent in each channel. Note that since the language is bounded, if we have the additional information of one terminal letter, we can reconstruct unique channel contents.

The last detail to note is that in general, a set of counters do not immediately behave like a channel. More specifically, in case of a channel, the decrements happen in FIFO order, and hence, there should be an order in decrementing the sets of counters. There is one more thing to consider here. Translated to the counter setting, this means that performing $\mathsf{dec}(x_{(c,j)})$ should require all counters $x_{(c,i)}$ with $i < j$ to be 0, so this is where the zero tests come into play. As the L_c are bounded languages and thanks to the normal form, however, a counter that has been tested for zero does not need to be modified any more.

We can directly implement these ideas formally and define $\mathcal{C} = (Q, V, T', q_0)$ as follows (note that Q and q_0 remain unchanged):

- The set of counters is $V = \{x_{(c,i)} \mid c \in \mathit{Ch} \text{ and } i \in \{1, \dots, n_c\}\}$.
- For every $(q, \langle c!a \rangle, q') \in T$, we have $(q, (\mathsf{inc}(x_{(c,i_a)}), \emptyset), q') \in T'$.
- For every $(q, \langle c?a \rangle, q') \in T$, we have $(q, (\mathsf{dec}(x_{(c,i_a)}), Z), q') \in T'$ where the set of counters to be tested for zero is $Z = \{x_{(c,j)} \mid j < i_a\}$.

Let us first observe that it is important for the FIFO machine to satisfy the trace property. We suppose in the following example that we construct the counter machine directly from $h^{-1}(\hat{\mathcal{M}})$ (instead of $\mathcal{M}$ in Example 16).



$!a_1 \mid !a_2 \mid !a_3 \mid ?a_1$
$?a_2 \mid ?a_3$
$?a_2 \mid ?a_3$
q_0
q_1
$h^{-1}(\hat{\mathcal{M}})$
$\mathsf{inc}(x) \mid \mathsf{inc}(y) \mid \mathsf{dec}(x)$
$\mathsf{dec}(x) \mid \mathsf{dec}(y)$
$\mathsf{dec}(x) \mid \mathsf{dec}(y)$
q_0
q_1
$\mathcal{C}'$ without trace property

Figure 4.7: We see that the counter machine $\mathcal{C}'$ constructed directly from $h^{-1}(\hat{\mathcal{M}})$ does not have a tight correspondence with the FIFO machine.

Example 17. Let us look at the counter machine $\mathcal{C}'$ constructed from $h^{-1}(\hat{\mathcal{M}})$ (cf. Figure 4.7) according to the above construction. Then, state $(q_1, (1, 0))$ would be reachable in the counter machine via $\tau = \mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)$. If we look at the only possible transition sequence corresponding to τ in $h^{-1}(\hat{\mathcal{M}})$, we obtain $\langle c!a_2 \rangle \langle c!a_2 \rangle \langle c?a_2 \rangle$. However the only trace corresponding to τ from $\mathit{Pref}(\mathcal{V})$ is $\langle c!a_1 \rangle \langle c!a_2 \rangle \langle c?a_1 \rangle$, which in fact, leads to (q_0, a_2) in $h^{-1}(\hat{\mathcal{M}})$. Hence, we do not have a tight enough correspondence between the transitions in $h^{-1}(\hat{\mathcal{M}})$ and $\mathcal{C}'$.

Due to the non-determinism still present in $h^{-1}(\hat{\mathcal{M}})$, we modify it so as to obey the trace property and obtain the following construction.

Example 18. Figure 4.8 illustrates the construction of $\mathcal{C}$ from a FIFO machine $\mathcal{M}$ in normal form (cf. Example 16). Recall that we have one channel c and the bounded language $L_c = (a_1 a_2)^* a_3 a_3^*$ over $(a_1 a_2\,,\, a_3)$. Thus, $\mathcal{C}$ will have two counters,

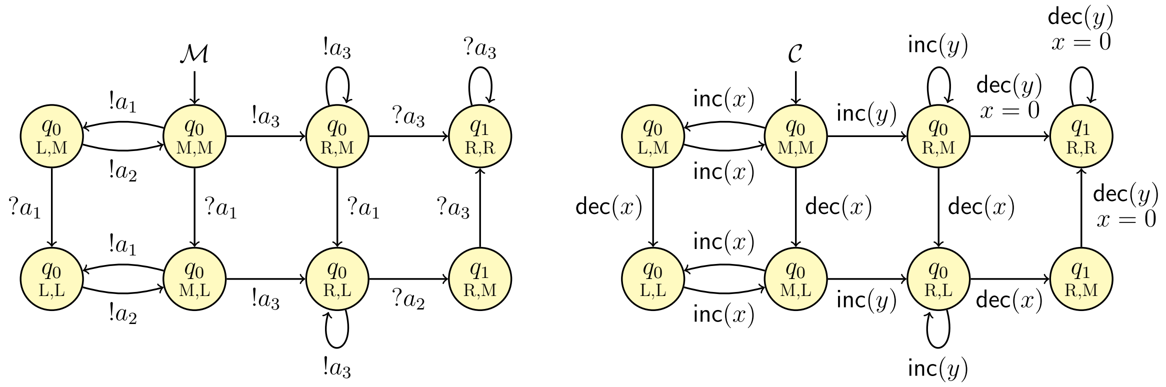


Figure 4.8: Following Example 16, from the normalized FIFO machine $\mathcal{M}$ (left), we obtain the counter machine $\mathcal{C}$ (right) which has a tight correspondence with $\mathcal{M}$.

say x for a_1a_2 and y for a_3. Note that performing $\mathsf{dec}(y)$ indeed comes with a test of x for zero.

So consider $\mathcal{M}$ and its counter machine $\mathcal{C}$. A channel contents $\mathbf{w} \in \Sigma_c^*$ (here, we have one channel) has a natural counter analogue $\langle\!\langle \mathbf{w} \rangle\!\rangle = (|\mathbf{w}|_{a_1} + |\mathbf{w}|_{a_2}, |\mathbf{w}|_{a_3})$. In fact, if $(\overline{q}, \mathbf{w})$ is reachable in $\mathcal{M}$, then following the corresponding transitions in $\mathcal{C}$ will lead us to $(\overline{q}, \langle\!\langle \mathbf{w} \rangle\!\rangle)$. For example, $((q_0, \mathrm{R}, \mathrm{L}), a_2a_3a_3)$ is reachable in $\mathcal{M}$ along the trace $\langle c!a_1\rangle\langle c!a_2\rangle\langle c?a_1\rangle\langle c!a_3\rangle\langle c!a_3\rangle$, and so is $((q_0, \mathrm{R}, \mathrm{L}), (1, 2))$ in $\mathcal{C}$ along the trace $\mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)\mathsf{inc}(y)\mathsf{inc}(y)$ (all zero tests are empty).

But how about the converse? In general, one may associate with a counter valuation such as $(4, 0)$ several channel contents. Actually, both $a_1a_2a_1a_2$ and $a_2a_1a_2a_1$ seem suitable. However, if we know the most recent message that has been sent, say a_1, then this leaves only one option, namely $a_2a_1a_2a_1$. In this way, we can associate with each counter valuation $\mathbf{v}$ and message $a_i \in \Sigma_c$ a unique (if it exists at all) possible channel contents $[\![\mathbf{v}]\!]_{a_i}$. Suppose that τ is a trace in $\mathcal{C}$ arising from a trace σ in $\mathcal{M}$ whose last sent message is a_i. If $(\overline{q}, \mathbf{v})$ is reachable in $\mathcal{C}$ via τ, then $(\overline{q}, [\![\mathbf{v}]\!]_{a_i})$ is reachable in $\mathcal{M}$ via σ. For example, $\tau = \mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)$ allows us to go to state $((q_0, \mathrm{M}, \mathrm{L}), (1, 0))$. It arises from $\sigma = \langle c!a_1\rangle\langle c!a_2\rangle\langle c?a_1\rangle \in \mathit{Pref}(\mathcal{V})$, whose last sent message is a_2. We have $[\![(1, 0)]\!]_{a_2} = a_2$. Indeed, σ leads to $((q_0, \mathrm{M}, \mathrm{L}), a_2)$.

Relation between FIFO Machine and Counter Machine Recall that the FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$ and $L = (L_c)_{c \in \mathit{Ch}}$ are in normal form, where L_c is a bounded language over $(w_{c,1}, \ldots, w_{c,n_c})$. Let $\mathcal{C} = (Q, V, T', q_0)$ be the associated counter machine. We will now formalize the tight back-and-forth correspondence that allows us to solve reachability queries in $\mathcal{M}$ in terms of reachability queries in $\mathcal{C}$.

We start with a simple observation concerning the traces of $\mathcal{M}$ and $\mathcal{C}$.

Lemma 4.6. *We have* $\mathit{Traces}(\mathcal{M}) \subseteq \mathit{Pref}(\mathcal{V})$ *and* $\mathit{Traces}(\mathcal{C}) \subseteq L_{\mathcal{C}}^{\mathsf{zero}}$.

Proof. Observe that $\mathit{Traces}(\mathcal{M}) \subseteq \mathit{Traces}((Q, \mathit{Act}_{\mathcal{M}}, T, q_0)) \subseteq \mathit{Pref}(\mathcal{V})$. Thus, the first property holds.

For the second statement, consider

$$(q_0, \mathbf{0}) = (q_0, \mathbf{v}_0) \xrightarrow{\alpha_1}_{\mathcal{C}} (q_1, \mathbf{v}_1) \xrightarrow{\alpha_2}_{\mathcal{C}} \ldots \xrightarrow{\alpha_n}_{\mathcal{C}} (q_n, \mathbf{v}_n)$$

and let $\tau = \alpha_1 \ldots \alpha_n$. Thus, $\tau \in \mathit{Traces}(\mathcal{C})$. Suppose that we apply a zero test at position $\ell \in \{1, \ldots, n\}$, i.e., $\alpha_\ell = (\mathsf{dec}(x_{(c,j)}), Z)$ for some (c, j), where Z contains the counters $x_{(c,i)}$ with $i < j$. Then, there is $k < \ell$ such that $\alpha_k = (\mathsf{inc}(x_{(c,j)}), \emptyset)$. By the construction of $\mathcal{C}$, we have transitions $(q_{k-1}, \langle c!a \rangle, q_k)$ and $(q_{\ell-1}, \langle c?b \rangle, q_\ell)$ in $\mathcal{M}$ for some $a, b \in \Sigma_{c,j}$. By the trace property of $\mathcal{M}$, none of the actions "reachable" from q_ℓ in $\mathcal{M}$ employs a message from $\Sigma_{c,i}$, for all $i < j$. Thus, none of the actions α_m with $\ell \leq m$ modifies a counter from Z. We deduce that $\tau \in L_{\mathcal{C}}^{\mathsf{zero}}$. □

With every channel contents $\mathbf{w} \in \prod_{c \in \mathit{Ch}} \Sigma_c^*$ of the FIFO machine $\mathcal{M}$, we associate a counter valuation $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v} \in \mathbb{N}^V$ where, for each counter $x_{(c,i)}$, we let $\mathbf{v}_{x_{(c,i)}} = \sum_{a \in \Sigma_{c,i}} |\mathbf{w'}_c|_a$. Furthermore, abusing notation, we define a homomorphism $\langle\!\langle\, . \,\rangle\!\rangle : \mathit{Act}_{\mathcal{M}}^* \to A_{\mathcal{C}}^*$ which maps a sequence of actions of $\mathcal{M}$ to a sequence of actions of $\mathcal{C}$. It is defined by $\langle\!\langle \langle c!a \rangle \rangle\!\rangle = (\mathsf{inc}(x_{(c,i_a)}), \emptyset)$ and $\langle\!\langle \langle c?a \rangle \rangle\!\rangle = (\mathsf{dec}(x_{(c,i_a)}), Z)$ where $Z = \{x_{(c,j)} \mid j < i_a\}$.

$\langle\!\langle \mathbf{w} \rangle\!\rangle$

Conversely, we will associate, with counter values and traces of $\mathcal{C}$ the corresponding objects in the FIFO machine. Because of the inherent ambiguity, this is, however, less straightforward. First, we define a partial mapping $[\![\, . \,]\!] : A_{\mathcal{C}}^* \to \mathit{Act}_{\mathcal{M}}^*$ (that is not a homomorphism). For $\tau \in A_{\mathcal{C}}^*$, we let $[\![\tau]\!]$ be the unique (if it exists) word $\sigma \in \mathit{Pref}(\mathcal{V})$ such that $\langle\!\langle \sigma \rangle\!\rangle = \tau$.

$[\![\mathbf{v}]\!]$

Next, we associate with a counter valuation a corresponding channel contents. As explained above, there is no unique choice unless we make an assumption on the last messages that have been sent. For $c \in \mathit{Ch}$, we set $\Sigma_c^{\perp} = \Sigma_c \uplus \{\perp\}$. Let $a \in \Sigma_c^{\perp}$ and $w \in \Sigma_c^*$. We say that a is *good* for w if $w \in \mathit{Inf}(L_c)$ and either $w = \varepsilon$ or $w = u.a$ for some $u \in \Sigma_c^*$. Intuitively, it may be possible to obtain contents w in channel c when a is the last message sent (no message was sent yet through c if $a = \perp$). Note that the set of words $w \in \Sigma_c^*$ such that a is good for w is a regular language. Moreover, with $\mathbf{w} \in \prod_{c \in \mathit{Ch}} \Sigma_c^*$, we associate the finite set $G(\mathbf{w}) \subseteq \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$ of tuples $\mathbf{a} = (\mathbf{a}_c)_{c \in \mathit{Ch}}$ such that, for all $c \in \mathit{Ch}$, $\mathbf{a}_c$ is good for $\mathbf{w}_c$.

good

$G(\mathbf{w})$

Let $\mathbf{v} \in \mathbb{N}^V$ and $\mathbf{a} \in \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$. Abusing notation, we will associate with $\mathbf{v}$ and $\mathbf{a}$ the channel contents $[\![\mathbf{v}]\!]_{\mathbf{a}} \in \prod_{c \in \mathit{Ch}} \Sigma_c^*$ (if it exists). We let $[\![\mathbf{v}]\!]_{\mathbf{a}} = \mathbf{w}$ if $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$ and $\mathbf{a} \in G(\mathbf{w})$. There is at most one such $\mathbf{w}$ so that this is well-defined. Note that $\langle\!\langle [\![\mathbf{v}]\!]_{\mathbf{a}} \rangle\!\rangle = \mathbf{v}$.

$[\![\mathbf{v}]\!]_{\mathbf{a}}$

Example 19. If we have one channel c and our bounded language is $L_c = (a_1a_2a_3)^*(a_4)^*$, then $[\![(4,0)]\!]_{a_2} = a_2a_3a_1a_2$ and $[\![(2,1)]\!]_{a_4} = a_2a_3a_4$, whereas $[\![(3,1)]\!]_{a_3}$ is undefined. Moreover, $G(a_2a_3) = \{a_3\}$ and $G(\varepsilon) = \{a_1, a_2, a_3, a_4, \perp\}$.

Given $\mathbf{v}$ and $\mathbf{a}$, we can easily compute $[\![\mathbf{v}]\!]_{\mathbf{a}}$ since there are only finitely many words $\mathbf{w}$ for a given $\mathbf{v}$ such that $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$. Furthermore, we can also compute $G(\mathbf{w})$ for a given $\mathbf{w}$ as we have finitely many possibilities of $\mathbf{a}$.

$L^{\mathsf{last}}_{\mathbf{a}}$ Finally, for $\mathbf{a} \in \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$, we let $L^{\mathsf{last}}_{\mathbf{a}} \subseteq \mathit{Act}^*_{\mathcal{M}}$ be the set of words σ such that, for all $c \in \mathit{Ch}$, $\mathbf{a}_c$ is the last message sent to c in σ (no message was sent if $\mathbf{a}_c = \perp$). We are now ready to state that runs in the FIFO machine are faithfully simulated by runs in the counter machine (the proof is by induction on the length of the trace):

Proposition 4.7. *Let $\sigma \in \mathit{Act}^*_{\mathcal{M}}$. For all $(q, \mathbf{w}) \in X_{\mathcal{M}}$ and $\mathbf{a} \in \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$ such that $\sigma \in L^{\mathsf{last}}_{\mathbf{a}}$, we have: $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_{\mathcal{M}} (q, \mathbf{w}) \implies \big((q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle}_{\mathcal{C}} (q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$ and $\mathbf{a} \in G(\mathbf{w})\big)$.*

Proof. We will prove the statement by induction on the length of σ. In the base case, $|\sigma| = 0$. The only value of σ such that $|\sigma| = 0$ is $\sigma = \varepsilon$. Furthermore, $\varepsilon \in \mathit{Pref}(\mathcal{V}) \cap L^{\mathsf{last}}_{\mathbf{a}}$ where $\mathbf{a} = (\perp)^{\mathit{Ch}}$. The initial state $(q_0, \boldsymbol{\varepsilon}) \in X_{\mathcal{M}}$ is the only state reachable by ε. In $\mathcal{S}_{\mathcal{C}}$, the only state reachable via $\langle\!\langle \varepsilon \rangle\!\rangle = \varepsilon$ is $(q_0, \mathbf{0}) = (q, \langle\!\langle \boldsymbol{\varepsilon} \rangle\!\rangle)$, and $\langle\!\langle \varepsilon \rangle\!\rangle \in L^{\mathsf{zero}}_{\mathcal{C}}$. Finally, we also see that $\mathbf{a} \in G(\boldsymbol{\varepsilon})$. Therefore, the base case is valid.

We suppose that the statement is true for all $\sigma \in \mathit{Act}^*_{\mathcal{M}}$ such that $|\sigma| = n$. We will now show that it is true for $\sigma' \in \mathit{Act}^*_{\mathcal{M}}$ where $|\sigma'| = n + 1$. Let $\mathbf{a}' \in \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$ and suppose $\sigma' \in \mathit{Pref}(\mathcal{V}) \cap L^{\mathsf{last}}_{\mathbf{a}'}$.

We write $\sigma' = \sigma.\beta$ such that $\sigma \in \mathit{Pref}(\mathcal{V}) \cap L^{\mathsf{last}}_{\mathbf{a}}$ for some $\mathbf{a} \in \prod_{c \in \mathit{Ch}} \Sigma_c^{\perp}$, $|\sigma| = n$, and $\beta \in \mathit{Act}_{\mathcal{M}}$. There exists such a σ since the set $\mathit{Pref}(\mathcal{V})$ is prefix-closed.

Let $(q', \mathbf{w}') \in X_{\mathcal{M}}$ such that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma'}_{\mathcal{M}} (q', \mathbf{w}')$. Then, there is $(q, \mathbf{w}) \in X_{\mathcal{M}}$ such that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_{\mathcal{M}} (q, \mathbf{w}) \xrightarrow{\beta}_{\mathcal{M}} (q', \mathbf{w}')$. Hence, there exists $t = (q, \beta, q') \in T$ in the FIFO machine.

Case (1):

Suppose $\beta = \langle c!a \rangle$, for some $c \in \mathit{Ch}$ and $a \in \Sigma_c$. Hence, we have $\mathbf{w''}_c = \mathbf{w'}_c.a$, and $\mathbf{w''}_d = \mathbf{w'}_d$ for all $d \in \mathit{Ch} \setminus \{c\}$. Moreover, since $\sigma' = \sigma.\beta$ and $\beta = \langle c!a \rangle$, we can deduce that $\mathbf{a}'_c = a$ and $\mathbf{a}'_d = \mathbf{a}_d$ for all $d \neq c$. This is because the only change in the last sent letters between σ and σ' is in the channel c.

By construction of $\mathcal{C}$, we know that there is a transition $t' = (q, \mathsf{inc}(x_{(c,i_a)}), \emptyset, q') \in T'$ in $\mathcal{C}$, and by the definition of $\langle\!\langle\,.\,\rangle\!\rangle$, we also have $\langle\!\langle \beta \rangle\!\rangle = (\mathsf{inc}(x_{(c,i_a)}), \emptyset)$. By induction hypothesis, we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle} (q, \mathbf{v})$ where $\mathbf{v} = \langle\!\langle \mathbf{w} \rangle\!\rangle$. Since $\langle\!\langle \beta \rangle\!\rangle$ increases a counter, we have $(q, \mathbf{v}) \xrightarrow{\langle\!\langle \beta \rangle\!\rangle} (q', \mathbf{v}')$ for the counter valuation $\mathbf{v}'$ such that $\mathbf{v}'_p = \mathbf{v}_p + 1$ for $p = x_{(c,i_a)}$ and $\mathbf{v}'_{p'} = \mathbf{v}_{p'}$ for all $p' \in V \setminus \{p\}$. Hence, $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$ and we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma.\beta \rangle\!\rangle} (q', \langle\!\langle \mathbf{w}' \rangle\!\rangle)$. Note that, by Lemma 4.6, we have $\langle\!\langle \sigma.\beta \rangle\!\rangle \in L^{\mathsf{zero}}_{\mathcal{C}}$.

From the induction hypothesis, we know that $\mathbf{a} \in G(\mathbf{w})$. In order to show that $\mathbf{a}' \in G(\mathbf{w}')$, we only need to address the case of the channel c, since the values of $\mathbf{a}_d, \mathbf{w}_d$ remain unchanged for all $d \neq c$. We know that $\mathbf{w}'_c = \mathbf{w}_c.\mathbf{a}'_c$. By induction hypothesis, $\mathbf{w}_c \in \mathit{Inf}(L_c)$. Since $\sigma' \in \mathit{Pref}(L_!)$, we also have $\mathbf{w}_c.\mathbf{a}'_c \in \mathit{Inf}(L_c)$. Hence, $\mathbf{a}' \in G(\mathbf{w}')$.

Case (2):

Suppose $\beta = \langle c?a \rangle$, for some $c \in \mathit{Ch}$ and $a \in \Sigma_c$. We have $\mathbf{w}'{}_c = a.\mathbf{w}''{}_c$, and $\mathbf{w}''{}_d = \mathbf{w}'{}_d$ for all $d \in \mathit{Ch} \setminus \{c\}$. Since $\sigma' = \sigma.\beta$ and $\beta = \langle c?a \rangle$, we can deduce that $\mathbf{a}' = \mathbf{a}$. This is because there is no change in the last letter sent between σ and σ'.

By construction of $\mathcal{C}$, we know that there is a transition $t' = (q, \mathsf{dec}(x_{(c,i_a)}), Z, q') \in T'$ in $\mathcal{C}$ where $Z = \{(x_{(c,j)}) \mid j < i_a\}$. By the definition of $\langle\!\langle\,.\,\rangle\!\rangle$, we also have $\langle\!\langle \beta \rangle\!\rangle = (\mathsf{dec}(x_{(c,i_a)}), Z)$. By the induction hypothesis, we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle} (q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$. Furthermore, recall that $\mathbf{w}'{}_c = a.\mathbf{w}''{}_c$. This implies that a is at the head of the channel c in the state $(q, \mathbf{w})$. Hence, all the letters b such that $i_b < i_a$ are not present in the channel. Therefore, in the state $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$, all the counters $x_{(c,i_b)}$ are equal to zero for $i_b < i_a$. Hence, we can execute the transition t' from $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$, and we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle} (q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \xrightarrow{\langle\!\langle \beta \rangle\!\rangle} (q', \mathbf{v}')$ for some counter valuation $\mathbf{v}'$. By Lemma 4.6, we have $\langle\!\langle \sigma.\beta \rangle\!\rangle \in L^{\mathsf{zero}}_{\mathcal{C}}$. We write $\mathbf{v} = \langle\!\langle \mathbf{w} \rangle\!\rangle$, and from the counter machine transition relation, we know that $\mathbf{v}'_p = \mathbf{v}_p - 1$ for $p = x_{(c,i_a)}$ and $\mathbf{v}'_{p'} = \mathbf{v}_{p'}$ for all $p' \in V \setminus \{p\}$. Hence, $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$.

From the induction hypothesis, we know that $\mathbf{a} \in G(\mathbf{w})$. In order to show that $\mathbf{a}' \in G(\mathbf{w}')$, we only need to address the case of the channel c, since the values of $\mathbf{a}_d, \mathbf{w}_d$ remain unchanged for all $d \neq c$. We know from the induction hypothesis that $\mathbf{w}_c \in \mathit{Inf}(L_c)$. Hence, we can immediately deduce that $\mathbf{w}'_c \in \mathit{Suf}(\mathbf{w}_c) \subseteq \mathit{Inf}(L_c)$.

Furthermore, we know that $\mathbf{w}_c = u.\mathbf{a}'_c$ for some $u \in \Sigma_c^*$. If $u = \varepsilon$, then we can deduce that $\mathbf{w}'_c = \varepsilon$. If $u \neq \varepsilon$, then we have $\mathbf{w}'_c = u'.\mathbf{a}'_c$ such that $a.u' = u$. Hence, $\mathbf{a}' \in G(\mathbf{w}')$. □

Conversely, we can show that runs of the counter machine can be retrieved in the FIFO machine (again, the proof proceeds by induction on the length of the trace):

Proposition 4.8. *Let $\tau \in A_{\mathcal{C}}^*$. For all $(q, \mathbf{v}) \in X_{\mathcal{C}}$ and $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^{\perp}$ such that $\tau \in \langle\!\langle Pref(\mathcal{V}) \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle$, we have: $(q_0, \mathbf{0}) \xrightarrow{\tau}_{\mathcal{C}} (q, \mathbf{v}) \implies (q_0, \boldsymbol{\varepsilon}) \xrightarrow{[\![\tau]\!]}_{\mathcal{M}} (q, [\![\mathbf{v}]\!]_{\mathbf{a}})$.*

Proof. We proceed by induction on the length of τ. In the base case, $|\tau| = 0$. The only value of τ such that $|\tau| = 0$ is $\tau = \varepsilon$. Furthermore, $\varepsilon \in \langle\!\langle Pref(\mathcal{V}) \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle$ where $\mathbf{a}_c = \perp$ for all $c \in Ch$. The only state reachable via ε is $(q_0, \mathbf{0})$. In the FIFO machine, the state $(q_0, \boldsymbol{\varepsilon})$ is the only state reachable via $[\![\varepsilon]\!] = \varepsilon$. We know that the initial contents is $\boldsymbol{\varepsilon} = [\![\mathbf{0}]\!]_{\mathbf{a}}$. Hence, the base case is valid.

Let us suppose that the statement holds for $\tau \in A_{\mathcal{C}}^*$ where $|\tau| = n$. We show that it is true for τ' with $|\tau'| = n+1$. Let $\mathbf{a}'$ such that $\tau' \in \langle\!\langle Pref(\mathcal{V}) \cap L_{\mathbf{a}'}^{\mathsf{last}} \rangle\!\rangle$. Then we can write $\tau' = \tau.\alpha$ for some $\tau \in Act_{\mathcal{C}}^*$ and $\alpha \in Act_{\mathcal{C}}$. There exists $\sigma' \in Pref(\mathcal{V}) \cap L_{\mathbf{a}'}^{\mathsf{last}}$ such that $\langle\!\langle \sigma' \rangle\!\rangle = \tau'$. Furthermore, since $\langle\!\langle \,.\, \rangle\!\rangle$ is a homomorphism, we can express $\sigma' = \sigma.\beta$ for some $\sigma \in Act_{\mathcal{M}}^*$ and $\beta \in Act_{\mathcal{M}}$ where $\langle\!\langle \beta \rangle\!\rangle = \alpha$ and $\langle\!\langle \sigma \rangle\!\rangle = \tau$. Since $Pref(\mathcal{V})$ is prefix-closed, we have $\sigma \in Pref(\mathcal{V}) \cap L_{\mathbf{a}}^{\mathsf{last}}$ for some $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^{\perp}$. Therefore, $\tau \in \langle\!\langle Pref(\mathcal{V}) \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle$.

Let $(q_0, \mathbf{0}) \xrightarrow{\tau'}_{\mathcal{C}} (q', \mathbf{v}')$. Note that, by Lemma 4.6, we have $\tau' \in L_{\mathcal{C}}^{\mathsf{zero}}$. We will prove that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{[\![\tau']\!]}_{\mathcal{M}} (q, \mathbf{w}')$ where $\mathbf{w}' = [\![\mathbf{v}']\!]_{\mathbf{a}'}$. Since $\tau' = \tau.\alpha$, there is $(q, \mathbf{v}) \in X_{\mathcal{C}}$ such that $(q_0, \mathbf{0}) \xrightarrow{\tau} (q, \mathbf{v}) \xrightarrow{\alpha} (q', \mathbf{v}')$. Hence, there exists a transition $t = (q, \alpha, q') \in T'$.

By the induction hypothesis, we know that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{[\![\tau]\!]}_{\mathcal{M}} (q, \mathbf{w})$ where $\mathbf{w} = [\![\mathbf{v}]\!]_{\mathbf{a}}$.

Case (1):

Suppose $\alpha = (\mathsf{inc}(x_{(c,i)}), \emptyset)$ for $c \in Ch$ and $i \in \{1, \ldots, n_c\}$. We have $\mathbf{v}'_p = \mathbf{v}_p + 1$ for $p = x_{(c,i)}$ and $\mathbf{v}'_{p'} = \mathbf{v}_{p'}$ and for all $p' \in V \setminus \{p\}$.

By construction of $\mathcal{C}$, we know that there is a transition $t' = (q, \gamma, q') \in T$ in $\mathcal{M}$ with $\gamma = \langle c!a \rangle$ for some $a \in \Sigma_c$ such that $i_a = i$. Thanks to the trace property (Definition 4.3 (2.)), we have $\beta = \gamma$. Let $\mathbf{w}'$ be given by $\mathbf{w}''_c = \mathbf{w}'_c.a$ and $\mathbf{w}''_d = \mathbf{w}'_d$ for all $d \in Ch \setminus \{c\}$. Then, $(q, \mathbf{w}) \xrightarrow{\beta}_{\mathcal{M}} (q', \mathbf{w}')$. Furthermore, since $i_a = i$ and $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$, we can deduce that $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$. Moreover, recall that $\langle\!\langle \sigma.\beta \rangle\!\rangle = \tau.\alpha$, hence, $\sigma' = \sigma.\beta = [\![\tau']\!]$.

Since $\sigma' = \sigma.\beta$ and $\beta = \langle c!a \rangle$, we can deduce that $\mathbf{a}'_c = a$ and $\mathbf{a}'_d = \mathbf{a}_d$ for all $d \neq c$. This is because the only change in the last sent letters between σ and σ' is in the channel c.

We recall that $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$. From the induction hypothesis, we know that $[\![\mathbf{v}]\!]_{\mathbf{a}} = \mathbf{w}$. Hence, in order to show that $[\![\mathbf{v}']\!]_{\mathbf{a}'} = \mathbf{w}'$, we only need to address the case of the channel c, since the values of $\mathbf{a}_d, \mathbf{w}_d$ remain unchanged for all $d \neq c$. We know that $\mathbf{w}'_c = \mathbf{w}_c.\mathbf{a}'_c$. Since $\sigma' \in \mathit{Pref}(L_!)$, we have $\mathbf{w}_c.\mathbf{a}'_c \in \mathit{Inf}(L_c)$. Therefore, $\mathbf{w}' = [\![\mathbf{v}']\!]_{\mathbf{a}'}$.

Case (2):

Suppose $\alpha = (\mathsf{dec}(x_{(c,i)}), Z)$, for $c \in \mathit{Ch}$, $i \in \{1, \ldots, n_c\}$ and $Z = \{x_{(c,j)} \mid j < i\}$. We have $\mathbf{v}'_p = \mathbf{v}_p - 1$ for $p = x_{(c,i)}$ and $\mathbf{v}'_{p'} = \mathbf{v}_{p'}$ for all $p' \in V \setminus \{p\}$.

By construction of $\mathcal{C}$, we know that there is a transition $t' = (q, \gamma, q') \in T$ in $\mathcal{M}$ with $\gamma = \langle c?a \rangle$ for some $a \in \Sigma_c$ such that $i_a = i$. Again, by the trace property (Definition 4.3 (2.)), we get $\beta = \gamma$. In order to execute t' from $(q, \mathbf{w})$, it is necessary that we have $\mathbf{w}_c = a.u$ for some word u.

Since $\sigma.\beta \in \mathit{Pref}(L_?)$ and $\mathit{proj}_{c?}(\sigma).\mathbf{w}'c = \mathit{proj}_{c!}(\sigma)$, we can deduce that $\mathbf{w}'c = a.u$ for some word u. Hence, the transition t' can be executed to reach a state $(q', \mathbf{w}')$ such that $\mathbf{w}'_c = a \cdot \mathbf{w}''_c$, and $\mathbf{w}''_d = \mathbf{w}'_d$ for all $d \in \mathit{Ch} \setminus \{c\}$. Furthermore, since $i_a = i$ and $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$, we can deduce that $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$. Moreover, recall that $\langle\!\langle \sigma.\beta \rangle\!\rangle = \tau.\alpha$, hence, $\sigma' = \sigma.\beta = [\![\tau']\!]$.

Since $\sigma' = \sigma.\beta$ and $\beta = \langle c?a \rangle$, we can deduce that $\mathbf{a}' = \mathbf{a}$. This is because no letters are sent between σ and σ'.

We also know from the induction hypothesis that $\mathbf{w} = [\![\mathbf{v}]\!]_{\mathbf{a}}$. Also recall that $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$. In order to show that $\mathbf{w}' = [\![\mathbf{v}']\!]_{\mathbf{a}'}$, we only need to address the case of the channel c, since the values of $\mathbf{a}_d, \mathbf{w}_d$ remain unchanged for all $d \neq c$.

We know from the induction hypothesis that $\mathbf{w}_c$ is contained in $\mathit{Inf}(()L_c)$ and, thus, so is $\mathbf{w}'_c$. Furthermore, we know that $\mathbf{w}_c = u.\mathbf{a}'_c$ for some $u \in \Sigma_c^*$. If $u = \varepsilon$, then we can deduce that $\mathbf{w}'_c = \varepsilon$. On the other hand, if $u \neq \varepsilon$, then we know that $\mathbf{w}'_c = u'.\mathbf{a}'_c$ such that $a.u' = u$. We also recall that $\langle\!\langle \mathbf{w}' \rangle\!\rangle = \mathbf{v}'$. Hence, $\mathbf{w}' = [\![\mathbf{v}']\!]_{\mathbf{a}'}$. □

From Propositions 4.7 and 4.8 and Lemma 4.6, we obtain the following corollary.

Corollary 4.9. *For all $(q, \mathbf{w}) \in X_{\mathcal{M}}$, we have: $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!) \iff (q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in \mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle \mathcal{V} \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L^{\mathsf{last}}_{\mathbf{a}} \rangle\!\rangle)$.*

From Theorem 4.5, we know that verifying whether $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in \mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap L)$ where $L = \langle\!\langle \mathcal{V} \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L^{\mathsf{last}}_{\mathbf{a}} \rangle\!\rangle$ is decidable. Hence, we can already deduce decidability of the (state-)reachability problem. In fact, using Propositions 4.7 and

4.8, we can solve the more general IB rational-reachability problem. For this, it is actually enough to check, in the counter machine, the reachability of a counter value that belongs to a semi-linear set. For $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^{\perp}$ and a rational relation $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$, let $V_{\mathbf{a}}(R) = \{\mathbf{v} \in \mathbb{N}^V \mid [\![\mathbf{v}]\!]_{\mathbf{a}} \in Rel\}$.

Lemma 4.10. *The set $V_{\mathbf{a}}(R)$ is effectively semi-linear.*

Proof. For $c \in Ch$, let $\mathcal{G}_c$ be the set of words $w \in \Sigma_c^*$ such that $\mathbf{a}_c$ is good for w. Moreover, let $\mathcal{G} = \prod_{c \in Ch} \mathcal{G}_c$.

As $\mathcal{G}_c$ is regular for every $c \in Ch$, the relation $\mathcal{G}$ is recognizable. As the intersection of a rational and a recognizable relation is rational, we have that $Rel \cap \mathcal{G}$ is rational. It follows that $\text{Parikh}(Rel \cap \mathcal{G})$ is semi-linear. From the definitions, we obtain

$$\begin{aligned} [\![\mathbf{v}]\!]_{\mathbf{a}} \in Rel \iff & \exists \mathbf{w} \in Rel \cap \mathcal{G} : & \forall x_{(c,i)} \in V : \mathbf{v}_{x_{(c,i)}} = \textstyle\sum_{a \in \Sigma_{c,i}} |\mathbf{w}_c|_a \\ \iff & \exists \pi \in \text{Parikh}(Rel \cap \mathcal{G}) : & \forall x_{(c,i)} \in V : \mathbf{v}_{x_{(c,i)}} = \textstyle\sum_{a \in \Sigma_{c,i}} \pi_a \end{aligned}$$

Thus,

$$V_{\mathbf{a}}(R) = \{\mathbf{v} \in \mathbb{N}^V \mid [\![\mathbf{v}]\!]_{\mathbf{a}} \in Rel\} = \left\{ \left(\textstyle\sum_{a \in \Sigma_{c,i}} \pi_a\right)_{x_{(c,i)} \in V} \mid \pi \in \text{Parikh}(Rel \cap \mathcal{G}) \right\}.$$

Let $\varphi((X_a)_{a \in \Sigma})$ be a Presburger formula defining $\text{Parikh}(Rel \cap \mathcal{G})$. Then, by the above equivalence, the following Presburger formula defines $V_{\mathbf{a}}(R)$:

$$\psi_c((Z_y)_{y \in V}) = \exists (X_a)_{a \in \Sigma} : \Big(\varphi((X_a)_{a \in \Sigma}) \wedge \bigwedge_{y \in V} Z_y = \sum_{a \in \Sigma_y} X_a \Big)$$

where, for $y = x_{(c,i)} \in V$, we let $\Sigma_y = \Sigma_{c,i}$. It follows that $V_{\mathbf{a}}(R)$ is effectively semi-linear. □

Using this property, we finally reduce the IB rational-reachability problem to a reachability problem in counter machines:

Corollary 4.11. *For every $q \in Q$, we have: $(q, \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_!)$ for some $\mathbf{w} \in Rel \iff (q, \mathbf{v}) \in Tracereach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle)$ for some $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^{\perp}$ and $\mathbf{v} \in V_{\mathbf{a}}(R)$.*

Proof. Suppose $(q, \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_!)$ with $\mathbf{w} \in Rel$. There are $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^{\perp}$ and $\sigma \in \mathcal{V} \cap L_{\mathbf{a}}^{\mathsf{last}}$ such that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_{\mathcal{M}} (q, \mathbf{w})$. By Proposition 4.7 and Lemma 4.6, we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle}_{\mathcal{C}} (q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$ and $\langle\!\langle \sigma \rangle\!\rangle \in L_{\mathcal{C}}^{\mathsf{zero}}$ and $\mathbf{a} \in G(\mathbf{w})$. By definition, the latter implies $\mathbf{w} = [\![\langle\!\langle \mathbf{w} \rangle\!\rangle]\!]_{\mathbf{a}}$ and hence $\langle\!\langle \mathbf{w} \rangle\!\rangle \in V_{\mathbf{a}}(R)$.

Conversely, suppose we have $(q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle}_{\mathcal{C}} (q, \mathbf{v})$ where $\sigma \in \mathcal{V} \cap L_{\mathbf{a}}^{\mathsf{last}}$, $\langle\!\langle \sigma \rangle\!\rangle \in L_{\mathcal{C}}^{\mathsf{zero}}$, and $\mathbf{v} \in V_{\mathbf{a}}(R)$. By Proposition 4.8, we get $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{[\![\langle\!\langle \sigma \rangle\!\rangle]\!]}_{\mathcal{M}} (q, [\![\mathbf{v}]\!]_{\mathbf{a}})$. Note that $[\![\langle\!\langle \sigma \rangle\!\rangle]\!] = \sigma \in L_!$. Moreover, $\mathbf{v} \in V_{\mathbf{a}}(R)$ implies $[\![\mathbf{v}]\!]_{\mathbf{a}} \in \mathit{Rel}$, which concludes the proof. □

By Theorem 4.5, we can now deduce the following theorem:

Theorem 4.12. *IB rational-reachability is decidable for FIFO machines.*

4.3 Reachability and deadlock

We now address some other commonly studied reachability problems, which, as it turns out, can be reduced to the IB rational-reachability problem studied in the previous section. We first look at the control-state reachability problem.

Decision Problem: IB CONTROL-STATE REACHABILITY

Input:	a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$
	a tuple $L = (L_c)_{c \in \mathit{Ch}}$ of non-empty regular bounded languages
	a control-state $q \in Q$
Question:	is $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$ for some $\mathbf{w}$?

In [FP20], it was shown that reachability reduces to control-state reachability for flat FIFO machines but the converse is not true. However, using the same reductions as in [FP20], we obtain the following result:

Proposition 4.13. *IB reachability is recursively equivalent to IB control-state reachability for FIFO machines.*

Proof. Let us consider a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$ with $\mathit{Ch} = \{1, \ldots, m\}$, a control-state q, a state $(q, \mathbf{w})$, and a tuple $L = (L_c)_{c \in \mathit{Ch}}$ of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$. Suppose $\mathbf{w} = (\mathbf{w}_c)_{c \in \mathit{Ch}}$ with $\mathbf{w}_c = \mathbf{w}_c^1 \ldots \mathbf{w}_c^{n_c}$.

We first reduce IB reachability to IB control-state reachability. We construct another machine $\mathcal{M}_{(q,\mathbf{w})} = (Q', \mathit{Ch}, \Sigma', T', q_0)$ as follows. We set $Q' = Q \cup \{q_{\text{end}}\}$ such that $q_{\text{end}} \notin Q$, and $\Sigma' = \Sigma \cup \{\$\}$ such that $\$ \notin \Sigma$. Moreover, a "path" $q \xrightarrow{\sigma} q_{\text{end}}$

with $\sigma = \sigma_1 \dots \sigma_m$ is added from the control-state q as follows. For $c \in \mathit{Ch}$, we have

$$\sigma_c = \begin{cases} \langle c!\$\rangle\langle c?\mathbf{w}_c^1\rangle \dots \langle c?\mathbf{w}_c^{n_c}\rangle\langle c?\$\rangle & \text{if } |\mathbf{w}_c| > 0, \\ \langle c!\$\rangle\langle c?\$\rangle & \text{otherwise.} \end{cases}$$

Then, $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$ iff q_{end} is reachable, i.e. $(q_{\text{end}}, \mathbf{w}') \in \mathit{Tracereach}_{\mathcal{M}_{(q,\mathbf{w})}}(L'_!)$ for *some* $\mathbf{w}'$, where $L' = (L_c.\mathbf{w}_c.\$)_{c \in \mathit{Ch}}$. Furthermore, L' is bounded if L is bounded, since concatenation of a finite word with a bounded language results in a bounded language. Therefore, IB reachability in $\mathcal{M}$ reduces to IB control-state reachability in $\mathcal{M}_{(q,\mathbf{w})}$.

Conversely, in order to show that IB control-state reachability is reducible to IB reachability, we construct $\mathcal{M}_q$ as follows. To $\mathcal{M}$, we add $|\Sigma| \times m$ self-loops around the control-state q as follows: $q \xrightarrow{c?a} q$ for all $c \in \mathit{Ch}$ and $a \in \Sigma_c$.

The control-state q is reachable in $\mathcal{M}$ iff there exists $\mathbf{w}$ such that $(q, \mathbf{w})$ is reachable in $\mathcal{M}$ iff $(q, \boldsymbol{\varepsilon})$ is reachable in $\mathcal{M}_q$ (as we can read all the channel contents in q due to the self-loops). Moreover, consider $\sigma \in \mathit{Pref}(L_!)$ such that $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_{\mathcal{M}} (q, \mathbf{w})$ for some channel contents $\mathbf{w}$. Let us append to σ a sequence of actions $\sigma' = \sigma_1 \dots \sigma_m$ such that $\sigma_c = \langle c?\mathbf{w}_c\rangle$ for all $c \in \mathit{Ch}$, where $\langle c?\mathbf{w}_c\rangle$ is to be understood as a sequence of transitions whose effect is to consume the string $\mathbf{w}_c$ from the channel c. By construction, we have, $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma\cdot\sigma'}_{\mathcal{M}_q} (q, \boldsymbol{\varepsilon})$. Furthermore, $\mathit{proj}_{c!}(\sigma) = \mathit{proj}_{c!}(\sigma \cdot \sigma')$ for all $c \in \mathit{Ch}$. Hence, $\sigma \cdot \sigma' \in \mathit{Pref}(L_!)$ and we can conclude that the control state q is reachable in $\mathcal{M}$ iff $(q, \boldsymbol{\varepsilon}) \in \mathit{Tracereach}_{\mathcal{M}_q}(L_!)$.

Therefore, IB control-state reachability reduces to IB reachability for FIFO machines. □

deadlock state **Definition 4.14.** *We say that a state $(q, \mathbf{w})$ of a FIFO machine $\mathcal{M}$ is a deadlock if there is no state reachable from it, i.e. there exists no $(q', \mathbf{w}')$ such that $(q, \mathbf{w}) \rightarrow_{\mathcal{M}} (q', \mathbf{w}')$.*

Decision Problem: IB DEADLOCK

Input:	a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$
	a tuple $L = (L_c)_{c \in \mathit{Ch}}$ of non-empty regular bounded languages
	a control-state $q \in Q$
Question:	does $\mathit{Tracereach}_{\mathcal{M}}(L_!)$ contain a deadlock?

Proposition 4.15. *IB reachability is recursively reducible to IB deadlock for FIFO machines.*

Proof. Given a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$, a state $(q, \mathbf{w})$, and a tuple $L = (L_c)_{c \in Ch}$ of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$, we construct $\mathcal{M}_{(q,\mathbf{w})}$ as in the case of reducing reachability to control-state reachability (see proof of Proposition 4.13). We then modify $\mathcal{M}_{(q,w)}$ to $\mathcal{M}'_{(q,w)}$ as follows. We add a new channel d to the existing set of channels Ch (the set of channels is now Ch'). For all $q \neq q_{\mathsf{end}}$, we add the following transition: $(q, \langle d!\$\rangle, q)$. Hence, except for q_{end}, every control-state has at least one send action. Finally, we also construct a new tuple $L' = (L'_c)_{c \in Ch'}$ such that $L'_c = L_c.\$$ for all $c \in Ch$ and $L'_d = \$^*$.

We claim that $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$ iff $\mathit{Tracereach}_{\mathcal{M}'_{(q,w)}}(L'_!)$ contains a deadlock. To see this, first, we observe that, if there is a deadlock in $\mathit{Tracereach}_{\mathcal{M}'_{(q,w)}}(L'_!)$, then the associated control-state would be q_{end} since from every other state s' such that the associated control-state q' is not q_{end}, the transition $(q', \langle d!\$\rangle, q')$ can always be taken, and hence, there will never be a deadlock. Moreover, as in the construction in Proposition 4.13, the control-state q_{end} is reachable iff the state $(q, \mathbf{w})$ is in $\mathit{Tracereach}_{\mathcal{M}}(L_!)$.

Let us now suppose that the state $(q, \mathbf{w})$ is in $\mathit{Tracereach}_{\mathcal{M}}(L_!)$. We can execute the same set of transitions as in $\mathcal{M}$ and reach the control-state q with the channel contents $\mathbf{w}$ via an execution σ' in $\mathcal{M}'_{(q,w)}$. Having done that, we can then execute the path $q \xrightarrow{\sigma} q_{\mathsf{end}}$ as described in T' in order to reach q_{end}. Also observe that $\sigma' \cdot \sigma \in L'_!$. Since there are no transitions from this control-state, we reach a deadlock.

Suppose now that $(q, \mathbf{w})$ is not in $\mathit{Tracereach}_{\mathcal{M}}(L_!)$. Hence, we can never reach $(q, \mathbf{w})$ in $\mathcal{M}'$ and thus, can never execute $q \xrightarrow{\sigma} q_{\mathsf{end}}$ and reach q_{end}. Furthermore, as we saw previously, we can never be in a deadlock as we can always send \$ to the channel d. □

Corollary 4.16. *The problems IB reachability, IB control-state reachability, and IB deadlock are decidable for FIFO machines.*

Since input-bounded FIFO machines subsume VASS (a VASS can be seen as an input-bounded FIFO machine with an alphabet reduced to a unique letter), the complexity of IB reachability is not elementary, which is inherited from the lower bound for VASS [CLL+19].

4.4 Unboundedness and termination

We now introduce two new decision problems, firstly unboundedness, defined as follows:

Decision Problem: IB UNBOUNDEDNESS

Input:	a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$
	a tuple $L = (L_c)_{c \in Ch}$ of non-empty regular bounded languages
Question:	is $\mathit{Tracereach}_{\mathcal{M}}(\mathit{Pref}(L_!))$ infinite?

IB unboundedness in FIFO machines reduces to an equivalent problem in counter machines. Given a FIFO machine $\hat{\mathcal{M}}$ and $\hat{L}$, the associated FIFO machine $\mathcal{M}$ in normal form (with the corresponding tuple L of distinct-letter languages), as well as the associated counter machine $\mathcal{C}$, the following result can be derived.

Proposition 4.17. *$\mathit{Tracereach}_{\hat{\mathcal{M}}}(\hat{L}_!)$ is infinite iff $\mathit{Tracereach}_{\mathcal{M}}(L_!)$ is infinite iff $\mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle \mathcal{V} \rangle\!\rangle)$ is infinite.*

Proof. We first show that unboundedness is preserved by the normal-form construction. This essentially follows from Lemma 4.4 and the fact that h is length-preserving.

If $(q, \hat{\mathbf{w}}) \in \mathit{Tracereach}_{\hat{\mathcal{M}}}(\hat{L}_!)$, then $((q, q_{\mathcal{A}}), \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$ for some $q_{\mathcal{A}}$ and $\mathbf{w}$ such that $h(\mathbf{w}) = \hat{\mathbf{w}}$. Thus, if $\mathit{Tracereach}_{\hat{\mathcal{M}}}(\hat{L}_!)$ is infinite, then so is $\mathit{Tracereach}_{\mathcal{M}}(L_!)$.

Conversely, if $((q, q_{\mathcal{A}}), \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$, then $(q, h(\mathbf{w})) \in \mathit{Tracereach}_{\hat{\mathcal{M}}}(\hat{L}_!)$. Thus, if $\mathit{Tracereach}_{\mathcal{M}}(L_!)$ is infinite, then so is $\mathit{Tracereach}_{\hat{\mathcal{M}}}(\hat{L}_!)$.

Now, let us assume that $\mathit{Tracereach}_{\mathcal{M}}(L_!)$ is infinite. Hence, there are infinitely many states $(q, \mathbf{w}) \in X_{\mathcal{M}}$ which are reachable from $(q_0, \boldsymbol{\varepsilon})$. From Corollary 4.9, for each of these states, $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in \mathit{Tracereach}_{\mathcal{C}}(L)$, where $L = L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L^{\mathsf{last}}_{\mathbf{a}} \rangle\!\rangle$. Since $L \subseteq L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \rangle\!\rangle$, we have $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in \mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \rangle\!\rangle)$. Furthermore, there are only finitely many states $(q, \mathbf{w}) \in X_{\mathcal{M}}$ that correspond to a state $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in X_{\mathcal{C}}$. Hence, if $\mathit{Tracereach}_{\mathcal{M}}(L_!)$ is infinite, $\mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \rangle\!\rangle)$ is infinite.

For the converse direction, let us assume that $\mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \rangle\!\rangle)$ is infinite. Hence, there are infinitely many states $(q, \mathbf{v})$ reachable from $(q_0, \mathbf{0})$ via some τ such that $\tau \in L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \rangle\!\rangle$. Let us choose one such $(q, \mathbf{v})$ and τ. There exists a unique σ such that $\langle\!\langle \sigma \rangle\!\rangle = \tau$. Consider the vector $\mathbf{a}$ such that $\sigma \in L^{\mathsf{last}}_{\mathbf{a}}$ and let $\mathbf{w} = [\![\mathbf{v}]\!]_{\mathbf{a}}$. Thus, $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$. Hence, $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in \mathit{Tracereach}_{\mathcal{C}}(L^{\mathsf{zero}}_{\mathcal{C}} \cap \langle\!\langle L_! \cap \mathit{Pref}(L_?) \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L^{\mathsf{last}}_{\mathbf{a}} \rangle\!\rangle)$. From Corollary 4.9, we can deduce that $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$. Therefore, for every $\mathbf{v}$ such that $(q, \mathbf{v})$ is reachable, there is a corresponding state $(q, \mathbf{w})$ reachable in $\mathit{Tracereach}_{\mathcal{M}}(L_!)$. □

In particular, this statement applies to prefix-closed languages, i.e., we can reduce checking whether $\mathit{Tracereach}_{\hat{\mathcal{M}}}(\mathit{Pref}(\hat{L}_!))$ is infinite to checking whether

$\mathit{Tracereach}_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \rangle\!\rangle)$ is infinite for some $\mathcal{C}$ and prefix-closed language $\mathcal{V}$. The latter is decidable as we establish in the following. Recall that, by the construction of $\mathcal{C}$, we then have $\mathit{Reachset}_{\mathcal{C}} = \mathit{Tracereach}_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathit{Pref}(\mathcal{V}) \rangle\!\rangle) = \mathit{Tracereach}_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \rangle\!\rangle)$.

The main idea that follows is the reduction of unboundedness of the counter machine to reachability in a modified counter machine. It is not immediate that we can use the results in the literature (for example [DF97]) which reduce boundedness to reachability in Petri nets/VASS. This is because the property of monotonicity does not extend to zero tests; if one can execute a zero test at $(q, \mathbf{v})$, it is not necessarily the case that it can be executed at $(q, \mathbf{v}')$ with $\mathbf{v} \leq \mathbf{v}'$, where we let $\mathbf{v} \leq \mathbf{v}'$ if $\mathbf{v}_x \leq \mathbf{v}'_x$ for all $x \in V$. However, we show that, for the counter machine that we construct from the FIFO machine, this property does hold. If we are able to show this, the constructions used in the case of VASS can be adapted.

Lemma 4.18. *For every execution in $\mathcal{C}$ of the form $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ such that $\mathbf{v} \leq \mathbf{v}'$, the following holds: The counters that are tested to zero during σ' already evaluate to zero at $(q, \mathbf{v})$, and do not change their value throughout the execution of σ'.*

Proof. Let us assume to the contrary that there is at least one counter which is incremented or decremented during σ' and also tested to zero during the execution. Without loss of generality, let us consider $x_{(c,i)}$ to be the first counter along the execution σ' that is tested to zero during σ' and also incremented/decremented before it was tested to zero.

- Case (1): It has a non-zero value at $(q, \mathbf{v})$, and is then either decremented, or first incremented and then decremented, and finally tested to zero. Since we know that $\sigma.\sigma' \in L_{\mathcal{C}}^{\mathsf{zero}}$, no counter tested to zero can then be incremented. Hence, its value will remain zero. But this is a contradiction to our assumption that $\mathbf{v} \leq \mathbf{v}'$. Hence, all the counters with non-zero values at $(q, \mathbf{v})$ cannot be tested to zero during σ'.

- Case (2): It has value zero in $(q, \mathbf{v})$, and is incremented, then decremented, then tested to zero during σ'. This implies that it first has to be incremented. Consider now some sub-execution $\sigma'' \in \mathit{Pref}(\sigma')$ where $(q, \mathbf{v}) \xrightarrow{\sigma''} (q_1, \mathbf{v}_1)$ such that the value of $x_{(c,i)}$ in the state $(q_1, \mathbf{v}_1)$ is non-zero. Since there are no "new" zero-tests along the execution σ'' (by our assumption), we can execute σ'' from $(q, \mathbf{v}')$ (by the monotonicity and trace property). However, we cannot increment the counter $x_{(c,i)}$ along σ'', because it was tested to zero during the run $(q_0, \mathbf{0}) \xrightarrow{\sigma.\sigma'} (q, \mathbf{v}')$. Hence, we once again have a contradiction. □

Now, we can use results from [FS01] to show the following:

Proposition 4.19. *The set $Reachset_{\mathcal{C}}$ is infinite iff there exist $\sigma, \sigma' \in Act_{\mathcal{C}}^*$, $q \in Q$, and $\mathbf{v}, \mathbf{v}' \in \mathbb{N}^V$ such that $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $\mathbf{v} < \mathbf{v}'$ (i.e., $\mathbf{v} \leq \mathbf{v}'$ and $\mathbf{v} \neq \mathbf{v}'$).*

Proof. Let us assume that $Reachset_{\mathcal{C}}$ is infinite. As in [FS01], we consider the tree of all prefixes of computations. We prune this tree by removing all prefixes where there is at least a loop, i.e., containing two nodes that are labelled by the same state. Since every reachable state can be reached without a loop, we still have an infinite number of prefixes in the pruned tree. For every state $(q, \mathbf{v})$ in the tree, there are finitely many successors (as the transition system is finitely branching). Hence, in order for the tree to be infinite, there is at least one infinitely long execution (by König's lemma). This execution has no loop. Therefore, by Dickson's lemma, there is an infinite subsequence of states $(q_1, \mathbf{v}_1)$, $(q_2, \mathbf{v}_2), \ldots$ such that $\mathbf{v}_1 < \mathbf{v}_2 < \ldots$. Once we extract this sequence, since there are only finitely many control-states in Q, we know that there is at least one pair $(q, \mathbf{v}), (q, \mathbf{v}')$ such that $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $\mathbf{v} < \mathbf{v}'$.

Conversely, let us assume that there is an execution $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ such that $\mathbf{v} < \mathbf{v}'$. We know from Lemma 4.18 that the only counters which may be tested for zero during σ' already evaluate to zero at $(q, \mathbf{v})$, and do not change their value throughout the execution of σ'. Therefore, all the transitions of the counter system in σ' can be considered as VASS operations. Hence, the property of monotonicity holds, i.e., if $(q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $(q, \mathbf{v}) < (q, \mathbf{v}')$, then we know there exists an infinite sequence $\mathbf{v}_1 < \mathbf{v}_2 < \ldots$ reachable from state $(q, \mathbf{v}')$. □

Construction of modified counter system. We modify the counter system $\mathcal{C}$ and construct a new counter system $\mathcal{C}'$ such that $Reachset_{\mathcal{C}}$ is infinite iff a specific state is reachable in $\mathcal{C}'$. The construction is loosely based on the reduction of boundedness to reachability for Petri Nets in [DF97].

Proposition 4.20. *We can effectively construct a counter machine $\mathcal{C}'$ (with bounded zero tests) and a finite set of states $R \subseteq S_{\mathcal{C}'}$ such that $Reachset_{\mathcal{C}}$ is infinite iff some state from R is reachable in $\mathcal{C}'$.*

Proof. We modify the counter machine $\mathcal{C}$ and construct a new counter machine $\mathcal{C}'$ such that $Reachset_{\mathcal{C}}$ is infinite iff a state belonging to a finite set is reachable in $\mathcal{C}'$. The construction is loosely based on the reduction of boundedness to reachability for Petri Nets in [DF97]. Since we do not know the values of $\mathbf{v}$ and $\mathbf{v}'$ a priori, we will try to characterize the general condition. The difference $\mathbf{v}' - \mathbf{v}$ is a non negative vector, with at least one strictly positive component. We add a duplicate set of counters for every counter in the system. The intuition is that the counter machine non-deterministically moves from operating on both sets to a state from where it only operates on this second set. The first set will remain unchanged (with the value $\mathbf{v}$), and the second set will keep track of the values (until it reaches $\mathbf{v}'$). From this state (which represents $(q, \mathbf{v}')$), we move to a new control-state, q_{reach}. Here, we check for the condition $\mathbf{v}' - \mathbf{v} > \mathbf{0}$ by first decrementing each counter in the first set which has a non-zero value in tandem with the corresponding counter in the second set. We do this until all the counters in the first set are equal to zero. If $\mathbf{v}' - \mathbf{v} > \mathbf{0}$, then there is at least one counter in the second set with a non-zero counter value. We non-deterministically decrement all the counters in the second set until we reach a state that has some counter c in the second set with a value of 1, and all other counters evaluate to zero. Since there are finitely many such states, we can just check every case. □

From the above results, we have the following theorem.

Theorem 4.21. *The IB unboundedness problem is decidable for FIFO machines.*

Gouda et al, stated that unboundedness is in EXPSPACE for letter-bounded systems [GGLR87]. However, they only give an idea of the proof, stating that it can be done in a similar fashion as for the deadlock problem. In the construction for solving the deadlock problem, they reduce the input language to *tally* letter-bounded languages (tally means that the input-language is included in a^* where a is a letter). They add as many channels as letters in the original letter-bounded-language. Furthermore, in order to ensure that no channel is non-empty before the next channel is read, they ensure that in all control-states where a later channel is being read, there are reception transitions of previous channel contents which lead to a sink state (where there is never a deadlock). Notice that it is still possible to leave a channel non-empty before the next channel is read. But one never reaches a deadlock in such an "incorrect" run, since there is always the option of reading the unread channel contents of the previous channels and reach the sink state.

However, when we consider this model for unboundedness, there may exist unbounded "incorrect" runs since we can leave a channel non-empty and proceed to the next and may have an unbounded run there. Hence, it seems that one still needs some reachability test to check if the runs are correct because we cannot ensure that some channels are zero in an unbounded run.

We now look at the IB termination problem, which we will define below:

Decision Problem: IB TERMINATION

Input:	a FIFO machine $\mathcal{M} = (Q, \mathit{Ch}, \Sigma, T, q_0)$ a tuple $L = (L_c)_{c \in \mathit{Ch}}$ of non-empty regular bounded languages a state $s \in X_{\mathcal{M}}$
Question:	is there no infinite execution $\mathit{init}_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \ldots$ such that, for all $i \in \mathbb{N}$, we have $s_i \in X_{\mathcal{M}}, \beta_i \in \mathit{Act}_{\mathcal{M}}$, and $\beta_1 \ldots \beta_i \in \mathit{Pref}(L_!)$?

For termination, we take a similar approach as for unboundedness. Suppose again that we are given $\hat{\mathcal{M}}$ and $\hat{L}$, the associated FIFO machine $\mathcal{M}$ in normal form (with the corresponding tuple L of distinct-letter languages), and the associated counter machine $\mathcal{C}$. We first show that the normal form preserves the (non-)termination property. In the following, β_i will denote a send or receive action and α_i will denote an increment or decrement action.

We obtain the following equivalence.

Proposition 4.22. *The following statements are equivalent:*

- *There is an infinite execution of the form $\mathit{init}_{\hat{\mathcal{M}}} \xrightarrow{\beta_1}_{\hat{\mathcal{M}}} s_1 \xrightarrow{\beta_2}_{\hat{\mathcal{M}}} s_2 \xrightarrow{\beta_3}_{\hat{\mathcal{M}}} \cdots$ such that, for all $i \in \mathbb{N}$, we have $\beta_1 \ldots \beta_i \in \mathit{Pref}(\hat{L}_!)$.*
- *There is an infinite execution of the form $\mathit{init}_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \cdots$ such that, for all $i \in \mathbb{N}$, we have $\beta_1 \ldots \beta_i \in \mathit{Pref}(L_!)$.*

Proof. Assume that there is an infinite execution of the form $\mathit{init}_{\hat{\mathcal{M}}} \xrightarrow{\hat{\beta}_1}_{\hat{\mathcal{M}}} \hat{s}_1 \xrightarrow{\hat{\beta}_2}_{\hat{\mathcal{M}}} \hat{s}_2 \xrightarrow{\hat{\beta}_3}_{\hat{\mathcal{M}}} \ldots$ in $\hat{\mathcal{M}}$ such that, for all $i \in \mathbb{N}$, we have $\hat{\beta}_1 \ldots \hat{\beta}_i \in \mathit{Pref}(\hat{L}_!)$. Hence, there are $\beta_1, \beta_2, \beta_3, \ldots \in \mathit{Act}_{\mathcal{M}}$ such that, for all $i \in \mathbb{N}$, letting $\sigma_i = \beta_1 \ldots \beta_i$, we have $h(\sigma_i) = \hat{\beta}_1 \ldots \hat{\beta}_i$ and $\sigma_i \in \mathit{Pref}(\mathcal{V})$. Hence, we know that, in the FIFO machine $h^{-1}(\hat{\mathcal{M}})$, one has $(\hat{q}_0, \boldsymbol{\varepsilon}) \xrightarrow{\beta_1} (\hat{q}_1, \mathbf{w}_1) \xrightarrow{\beta_2} (\hat{q}_2, \mathbf{w}_2) \xrightarrow{\beta_3} \ldots$ for suitable $(\hat{q}_i, \mathbf{w}_i)$ (by construction of $h^{-1}(\hat{T})$), and that $\sigma_i \in \mathit{Pref}(L(\mathcal{A}))$. Therefore, since $\mathcal{M}$ is a product of the two machines, we can deduce that there is an infinite execution $\mathit{init}_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \ldots$ as required.

Conversely, assume that there is an infinite execution of the form $\mathit{init}_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \ldots$ such that, for all $i \in \mathbb{N}$, we have $\beta_1 \ldots \beta_i \in \mathit{Pref}(L_!)$. Let $\hat{\sigma_i} = h(\beta_1 \ldots \beta_i)$ for all $i \in \mathbb{N}$. Since $\beta_1 \ldots \beta_i \in \mathit{Pref}(L_!)$, we have $\mathit{proj}_{c!}(\beta_1 \ldots \beta_i) \in \mathit{Pref}(L_c)$ for all $c \in \mathit{Ch}$. In particular, $\mathit{proj}_{c!}(\beta_1 \ldots \beta_i) \in h_c^{-1}(\mathit{Pref}(\hat{L}_c))$ and, therefore, $h_c(proj_{c!}(\beta_1 \dots \beta_i)) = proj_{c!}(h(\beta_1 \dots \beta_i)) \in Pref(\hat{L}_c)$. We deduce $\hat{\sigma}_i \in Pref(\hat{L}_!)$. Furthermore, we can execute $\hat{\sigma}$ in $\hat{\mathcal{M}}$ (by construction) for all $i \in \mathbb{N}$. Hence, we can build an infinite execution in $\hat{M}$, such that $\hat{\sigma}_i \in Pref(\hat{L}_!)$ for all $i \in \mathbb{N}$. □

The latter property can be reduced to checking a decidable property in the counter machine as follows:

Proposition 4.23. *The following statements are equivalent:*

- *There is an infinite execution of the form* $init_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \dots$ *such that, for all* $i \in \mathbb{N}$*, we have* $\beta_1 \dots \beta_i \in Pref(L_!)$*.*
- *There is an infinite execution of the form* $init_{\mathcal{C}} \xrightarrow{\alpha_1}_{\mathcal{C}} s_1 \xrightarrow{\alpha_2}_{\mathcal{C}} s_2 \xrightarrow{\alpha_3}_{\mathcal{C}} \dots$ *such that, for all* $i \in \mathbb{N}$*, we have* $\alpha_1 \dots \alpha_i \in L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle Pref(\mathcal{V}) \rangle\!\rangle$*.*
- *There exist* $\sigma \in Act_{\mathcal{C}}^*$*,* $\sigma' \in Act_{\mathcal{C}}^+$*,* $(q, \mathbf{v}) \in X_{\mathcal{C}}$*, and* $\mathbf{v}'$ *such that* $(q_0, \mathbf{0}) \xrightarrow{\sigma}_{\mathcal{C}} (q, \mathbf{v}) \xrightarrow{\sigma'}_{\mathcal{C}} (q, \mathbf{v}')$ *and* $\mathbf{v} \leq \mathbf{v}'$*.*

Proof. The equivalence of the first two items is obtained as a corollary of Propositions 4.7 and 4.8. So let us show that the latter two are equivalent.

Consider a non-terminating execution as described above. By Dickson's lemma, there is an infinite subsequence of states $(q_1, \mathbf{v}_1), (q_2, \mathbf{v}_2), \dots$ such that $\mathbf{v}_1 \leq \mathbf{v}_2 \leq \dots$. Once we extract this sequence, since there are only finitely many control-states in Q, we know that there is at least one pair $(q, \mathbf{v}), (q, \mathbf{v}')$ such that $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $\mathbf{v} \leq \mathbf{v}'$.

Conversely, assume $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $\mathbf{v} \leq \mathbf{v}'$. From the same argument as for boundedness, we can deduce that no counter is being tested to zero for the first time in σ' and all the counters previously tested stay unchanged. Hence, all the transitions of the counter system in σ' can be considered as VASS operations. Therefore, we know there exists $\mathbf{v}''$ such that $(q, \mathbf{v}') \xrightarrow{\sigma'} (q, \mathbf{v}'')$ and $\mathbf{v}' \leq \mathbf{v}''$. Repeating this reasoning, we can build an infinite sequence starting from $(q, \mathbf{v}')$. □

Finally, we construct a modified counter machine as in the case for boundedness (Proposition 4.20) and get the following:

Proposition 4.24. *We can effectively construct a counter machine* $\mathcal{C}''$ *(with bounded zero tests) and a state* $s \in S_{\mathcal{C}''}$ *such that the following statements are equivalent:*

- *There exist* $\sigma \in Act^*_{\mathcal{C}}$, $\sigma' \in Act^+_{\mathcal{C}}$, $(q, \mathbf{v}) \in X_{\mathcal{C}}$, *and* $\mathbf{v}'$ *such that* $(q_0, \mathbf{0}) \xrightarrow{\sigma}_{\mathcal{C}} (q, \mathbf{v}) \xrightarrow{\sigma'}_{\mathcal{C}} (q, \mathbf{v}')$ *and* $\mathbf{v} \leq \mathbf{v}'$.

- *State* s *is reachable in* $\mathcal{C}''$.

Proof. We can adapt the construction of $\mathcal{C}'$ for unboundedness. The difference is that we now allow for $\mathbf{v} = \mathbf{v}'$. Hence, there is no need any more to check that, after decrementing both sets of counters in tandem, there is still a positive counter left in the second set. We can therefore also empty the second set of counters in a new control-state q and check whether $(q, \mathbf{0})$ is reachable. □

Thus, we have the following theorem.

Theorem 4.25. *The IB termination problem is decidable for FIFO machines.*

4.5 Output-bounded problems

We consider the dual case of input-bounded languages in which the set of words that may be received by each channel (the output-language) is constrained to be bounded. The *OB problems* are defined as follows:

OB decision problems

Definition 4.26. *Given a FIFO machine* $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$, *a tuple* $L = (L_c)_{c \in Ch}$ *of non-empty regular bounded languages* $L_c \subseteq \Sigma_c^*$ *(each given in terms of a finite automaton), a control-state* $q \in Q$, *a state* $s \in X_{\mathcal{M}}$, *and a rational relation* $Rel \subseteq \prod_{c \in Ch} \Sigma_c^*$.

- OB rational reachability*: Do we have* $(q, \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_?)$ *for some* $\mathbf{w} \in Rel$*?*

- OB reachability*: Do we have* $s \in Tracereach_{\mathcal{M}}(L_?)$ *?*

- OB control-state reachability*: Do we have* $(q, \mathbf{w}) \in Tracereach_{\mathcal{M}}(L_?)$ *for some* $\mathbf{w}$ *?*

- OB deadlock*: Does* $Tracereach_{\mathcal{M}}(L_?)$ *contain a deadlock?*

- OB unboundedness*: Is* $Tracereach_{\mathcal{M}}(Pref(L_?))$ *infinite?*

- OB termination*: Is there no infinite execution of the form* $init_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \ldots$ *such that, for all* $i \in \mathbb{N}$, *we have* $s_i \in X_{\mathcal{M}}$, $\beta_i \in Act_{\mathcal{M}}$, *and* $\beta_1 \ldots \beta_i \in Pref(L_?)$*?*

Theorem 4.27. *OB reachability is decidable for FIFO machines.*

Proof. Given a state $(q, \mathbf{w})$ in $\mathcal{M}$, and a tuple of bounded languages L, the output-bounded reachability problem asks if $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_?)$. Since the output language is in $L_?$, we know that the contents of the channels which have already been read is in the corresponding input-language, i.e., $L_!$. Therefore, $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L_?)$ iff $(q, \mathbf{w}) \in \mathit{Tracereach}_{\mathcal{M}}(L'_!)$ where $L' = (L'_c)_{c \in Ch}$ and $L'_c = L_c.\mathbf{w}_c$. Hence, the OB reachability problem is decidable for FIFO machines. □

Theorem 4.28. *OB control-state reachability is decidable for FIFO machines.*

Proof. In order to show that OB control-state reachability is decidable for FIFO machines, we first convert it to the normal form. This construction is similar to that as specified in Section 4.1, however we make a few changes. Firstly, when we change the alphabet to distinct letter, we add an additional letter, say \$, to represent all the letters which are not present in the bounded language, but are present in the transitions of the FIFO system. Then, in addition to the transitions we already add to the new FIFO system as in Section 4.1, for every send action in the original FIFO system, we add a send action with this new letter \$. However, for the reception, we leave it as before. This new FIFO system can now only read letters in the output language, however, it can potentially send any letter. Hence, we have only restricted the output language.

Next, to the automata that is constructed to accept the bounded language for send actions, we add to all the states a transition that enables the automata to send \$ and go to a sink state which loops with send actions sending \$. Hence, once again, we have ensured that the input language is not restricted. To the reception automata, we make no such changes. Hence, the reception automata only accept the bounded language.

In this new machine, we can solve for control-state reachability, over the input-language $L' = (L'_c)_{c \in Ch}$ such that $L'_c = L_c.\$^*$ (but we let the output-language remain L when we construct the trimmed automata). If a state $(q, \mathbf{w})$ is reachable in this new machine, then there exists a path which can be taken in the original machine to reach control-state q via $\sigma \in L_?$. Furthermore, if there is a path which can be taken in the original machine, it can also be taken in the new machine. Hence, we can decide if there exists a $\mathbf{w}$ such that $(q, \mathbf{w})$ is reachable. □

Theorem 4.29. *OB unboundedness and OB termination problems are decidable for FIFO machines.*

Proof. We can decide the OB unboundedness and termination problems as well. We can construct a counter system for the normal form described in the proof of Theorem 4.28. Note that the counter corresponding to \$ will only have increments and no decrements, which is in line with the fact that the contents corresponding to this counter do not belong to the output-language. The FIFO machine has an infinite run iff this newly constructed counter machine has also an infinite run. We then construct the modified counter machine, as is the case for boundedness (see Section 4.4), and test if there is a run $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ such that $\mathbf{v} < \mathbf{v}'$. Since this modified counter system has bounded zero tests (the added counter has no decrements or zero tests associated to it), we can decide the reachability of the state, and hence, decide if the FIFO machine is unbounded. A similar explanation can be made for termination. □

The rational reachability problem cannot be directly reduced to the input-bounded case. For the output bounded case, we do not know precisely the reachability set, since we only restrict the output-language. Hence, in order to check if some $\mathbf{w} \in Rel$ is reachable, we need to be able to compute the reachability set. Similarly, unlike the input-bounded case, we cannot determine the deadlock problem a priori, since the deadlock problem is reduced to rational reachability for the input-bounded case. Hence, these two problems have been left open.

4.6 FIFO machines with a single channel

When we restrict the communication to a single channel, we obtain better upper bound for reachability.

Upper bound for reachability: EXPTIME We consider the model of ordered multi-pushdown systems, studied in [ABH17]. We define it using our counter systems. A counter system can be seen as a multi-pushdown system with a unary alphabet. Unary ordered multi-pushdown systems (UOMPDSs) are multi-pushdown systems which impose a total order on the counters and limit decrements to the lowest non-empty counter. We use the following result of reachability of UOMPDS.

UOMPDS

Theorem 4.30 ([ANKS13], Theorem 13)**.** *The reachability problem in UOMPDS is solvable in EXPTIME.*

Consider the counter machine $\mathcal{C}$ with the semantic restriction $L_{\mathcal{C}}^{\mathsf{zero}}$ which can be built from a single channel FIFO machine $\mathcal{M}$. We see that a counter x_i can only be decremented if the previous counters x_j such that $j < i$ are equal to zero. Therefore, the counter system $\mathcal{C}$ is a UOMPDS, where the order of the counters is defined as $x_i < x_j$ iff $i < j$. Hence, we have the following proposition.

Proposition 4.31. *IB reachability in single channel FIFO machines is polynomially reducible to reachability in UOMPDS.*

In [ANKS13], it was also shown that repeated reachability for UOMPDS is solvable in EXPTIME. We know that a system is non-terminating if and only if we can reach any control-state infinitely often. Therefore, we can guess a control-state and verify if it is reachable repeatedly in order to verify if the system is non-terminating. Furthermore, we see that even for FIFO machines with a single channel, using the same construction as in Proposition 4.13, reachability and control-state reachability are recursively equivalent.

We then have the following corollary.

Corollary 4.32. *The IB reachability, termination and control-state reachability problems are in EXPTIME for FIFO machines with a single channel.*

However, for the unboundedness result in Section 4.4, we use the reachability for counter systems with bounded zero tests. Furthermore, the set of all counters in the modified counter system do not have a total order any more. Hence, following our constructions, we may only say that the unboundedness problem in FIFO machines with a single channel (over a bounded language) is solvable by using the Petri net reachability (which is tower-hard).

In the case of a single channel, we have a total order on the counters, and therefore we are able to use results from Ordered Multipushdown Systems. However, even when we consider two channels, we cannot immediately extend these results since there is no longer a total order in the decrement of counters.

Lower bound for reachability: NP-hard We consider a sub-problem of the IB reachability, unboundedness, and termination problems as follows. Recall that, given a bounded language $L = w_1^* \dots w_n^*$, if $|w_1| = \dots = |w_n| = 1$, i.e., $w_1, \dots, w_n \in A$, then L is called a letter-bounded language.

Definition 4.33. *Given a FIFO machine $\mathcal{M} = (Q, Ch, \Sigma, T, q_0)$, a control-state $q \in Q$, a state $s \in X_{\mathcal{M}}$, and a tuple $L = (L_c)_{c \in Ch}$ of non-empty regular letter-bounded languages $L_c \subseteq \Sigma_c^*$.*

ILB decision problems

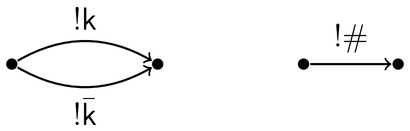
- ILB-reachability*: Do we have $s \in \mathit{Tracereach}_{\mathcal{M}}(L_!)$?*
- ILB-unboundedness*: Is $\mathit{Tracereach}_{\mathcal{M}}(\mathit{Pref}(L_!))$ infinite?*
- ILB-termination*: Is there no infinite execution of the form $\mathit{init}_{\mathcal{M}} \xrightarrow{\beta_1}_{\mathcal{M}} s_1 \xrightarrow{\beta_2}_{\mathcal{M}} s_2 \xrightarrow{\beta_3}_{\mathcal{M}} \dots$ such that, for all $i \in \mathbb{N}$, we have $s_i \in X_{\mathcal{M}}$, $\beta_i \in \mathit{Act}_{\mathcal{M}}$, and $\beta_1 \dots \beta_i \in \mathit{Pref}(L_!)$?*

We see that most of the properties are *NP*-hard for FIFO machines with a single channel over a letter-bounded language. This is proved by simulating 3-CNF formula with such machines. Our simulation follows the same ideas as the proof of *NP*-hardness for flat FIFO machines with *multiple* channels [EGM12, FP20], except that we use a unique channel.

Theorem 4.34. *ILB-reachability, ILB-unboundedness, and ILB-non-termination are NP-hard for machines with a single channel, even when the input language is letter bounded.*

Proof. We reduce from 3SAT. Given a 3-CNF formula $C_1 \wedge \ldots \wedge C_m$ over variables $x_1, \ldots, x_n$, we construct a FIFO machine with one channel. The message alphabet has $2n+1$ letters and is as follows $\Sigma_\# = \Sigma \uplus \{\#\}$, where $\Sigma = \{\mathsf{1}, \ldots, \mathsf{n}\} \uplus \{\bar{\mathsf{1}}, \ldots, \bar{\mathsf{n}}\}$. The FIFO machine consists of the gadgets shown below. The gadget for variable x_k adds either k (in the top transition) or $\bar{\mathsf{k}}$ (in the bottom edge) to the channel. At the end of this gadget, the channel will have either k or $\bar{\mathsf{k}}$. We will sequentially compose the gadgets for all variables. Starting from the initial control-state of the gadget for variable x_1, we reach the final control-state of the gadget for variable x_n, such that for every variable x_k we either have k or $\bar{\mathsf{k}}$ in the channel - and this determines the truth valuation.

We then add the gadget that adds the stop symbol to the channel, as shown in Figure 4.9.



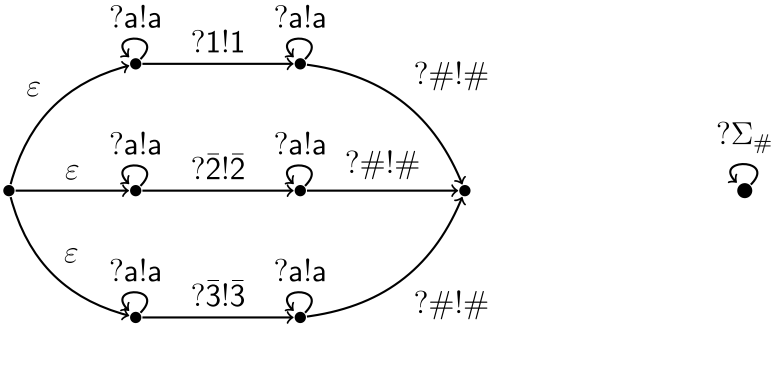
(a) Gadget for variable x_k (b) Gadget for stop marker

Figure 4.9: Gadget for variables

Next, we add gadgets for the clauses. The gadget for the example clause $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$ (gadgets for other clauses follow similar pattern) is shown in Figure 4.10. The gadget checks that the channel has either $\mathsf{1}$ (in the top path) or has $\bar{\mathsf{2}}$ (in the middle path) or has $\bar{\mathsf{3}}$ (in the bottom path). We append the clause gadgets to the end of the variable gadgets one after the other. All clauses are satisfied by the truth valuation determined by the contents of channels $x_1, \ldots, x_n$ iff we can reach the last control-state of the last clause.

The gadget for cleaning up all the variables is shown below (it receives all the letters from the channel). We append the clean-up gadget to the end of the clause gadget for C_m.

Note that in the FIFO machine given above, every gadget can only be visited once, and the input language of each gadget for a variable x_k is equal to $\{\mathsf{k}, \bar{\mathsf{k}}\}$ which is included in the letter-bounded language $\mathsf{k}^*\bar{\mathsf{k}}^*$. Hence, for every execution σ



(a) Gadget for clause $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$

(b) Gadget for clean-up

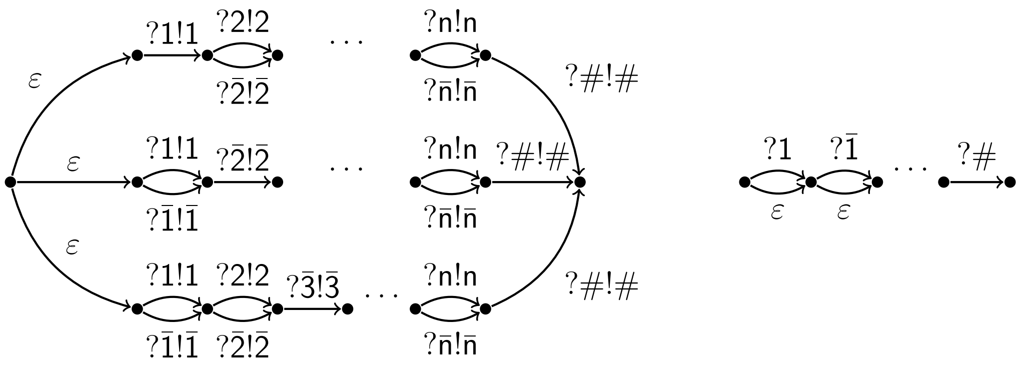
Figure 4.10: Gadget for clauses (the loop labelled ?a!a represents loops for all $\mathsf{a} \in \Sigma$)

along the sequence of variable gadgets, we have $\sigma \in L_!$ where $L = \mathsf{1}^*\bar{\mathsf{1}}^* \ldots \mathsf{n}^*\bar{\mathsf{n}}^*\#^*$. For every clause gadget, we have, once again, an execution $\sigma' \in L_!$. Hence, we see that the input-language of every run can be restricted to the bounded language $(\mathsf{1}^*\bar{\mathsf{1}}^* \ldots \mathsf{n}^*\bar{\mathsf{n}}^*\#^*)^{m+1}$. Furthermore, while there are loops in the FIFO machine, it can be seen that no loop can be executed infinitely often. We can also see that along every run, the channel is bounded and the size of the channel does not exceed $n + 1$.

The given 3-CNF formula is satisfiable iff the control-state of the clean-up gadget can be reached with the channel being empty. Hence, this constitutes a reduction to the reachability problem. Furthermore, if we add a self loop to the state of the clean-up gadget, such that it sends the letter # to the channel, then this loop can be iterated infinitely often to add unboundedly many occurrences of the letter # to the channel. Now, the given 3-CNF formula is satisfiable iff the constructed FIFO machine is unbounded iff channel is unbounded iff there is a non-terminating run. Hence reachability, unboundedness and non-termination are all NP-hard. □

We can remove the ε-transitions in the above construction by instead non-deterministically choosing one of the three variables per clause. This would eliminate all ε-transitions, and corresponds to the model we define, which does not have them.

We can adapt the proof above to the more restrictive case of FIFO machines whose input language is restricted to a distinct-letter-bounded language, by modifying the transitions in the gadget for clause C_i as follows: For every transition sequence of the form ?a!a, we replace it by ?a_{i-1}!a_i, thereby ensuring that we write different letters to the channel in every gadget. The transitions for the gadgets for each variable x_k (when it is set initially) would be modified by $!\mathsf{k}_0$ and $?\bar{\mathsf{k}}_0$.



(a) Gadget for clause $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$

(b) Gadget for clean-up

Figure 4.11: Showing NP-hardness for flat systems with a single channel

Furthermore, we can modify gadgets for the clauses (see Figure 4.11) in order to have the following corollary, which improves a similar result for flat FIFO machines with *multiple* channels in [FP20].

Corollary 4.35. *For flat FIFO machines with a single channel, reachability, unboundedness, and non-termination are NP-hard, hence, they are also NP-complete.*

4.7 Towards a theory of boundable FIFO machines.

We conclude by briefly summarizing the results in this chapter in Table 4.1. As shown, we were able to extend a lot of the decidability results from flat and letter-bounded systems to the bounded case.

Table 4.1: Summary of key results in this chapter; results for all other extensions are subsumed by these results (D stands for decidable).

	Flat	Letter-bounded	Bounded
UNBOUND	NP-C ([FP20])	D ([GGLR87])	D ([JJ93])
TERM	NP-C ([FP20])	D	**D**
REACH	NP-C ([FP20])	D	**D, not ELEM**
CS-REACH	NP-C ([EGM12, FP20])	D	D
DEADLOCK	D	D ([GGLR87])	**D**

Let us now look at the implications of bounded verification problems in the general case.

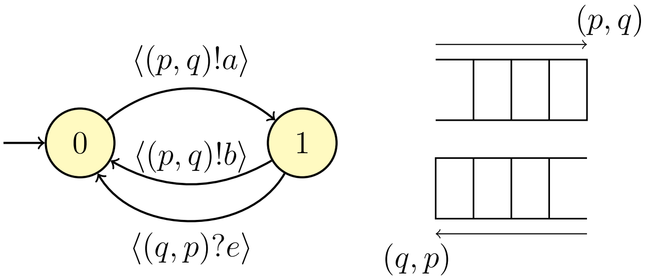


Figure 2.2: The model of the connection-deconnection protocol from Example 2

Example 20. Coming back to the protocol CDP $\mathcal{M}$ from Example 2 and Figure 2.2, we see that it is neither monogeneous nor linear nor flat. Since the input-languages of the two channels contain $\{a, ab\}^*$ and e^* resp., and since $\{a, ab\}^*$ is not a bounded language, we have $\mathit{proj}_{c!}(\mathit{Traces}(\mathcal{M})) \not\subseteq L_{c!}$ for every pair of bounded languages $(L_c)_{c \in \mathit{Ch}}$. In other words, $\mathcal{M}$ is not input-bounded. However, when we look at the reachability set obtained by considering the tuple of bounded languages $L = (L_{(p,q)}, L_{(q,p)})$ where $L_{(p,q)} = (ab)^*(a + \varepsilon)(ab)^*$ is a bounded language over (ab, a, ab), and $L_{(q,p)} = e^*$ is a bounded language over (e), we still obtain the entire reachability set. That is, we have $\mathit{Reachset}_{\mathcal{M}} = \mathit{Tracereach}_{\mathcal{M}}(L_!)$. Hence, even though the input-languages of the system are not all bounded, we can still compute the reachability set by restricting our exploration to a tuple of (regular) bounded languages L.

From the above example, we have seen that all states that are reachable in the CDP protocol are already reachable in presence of a suitable collection L of bounded input-languages. Analogous to the well-established theory of *flattable* machines [BFLP08, DFGD10, CFS11], we propose the following definition.

Definition 4.36. *Let $\mathcal{M}$ be a FIFO machine and let L be a tuple of regular bounded languages. We say that $\mathcal{M}$ is L-*boundable *if $\mathit{Reachset}_{\mathcal{M}} = \mathit{Tracereach}_{\mathcal{M}}(L_!)$. We say that $\mathcal{M}$ is* boundable *if there exists a tuple L of regular bounded languages such that $\mathcal{M}$ is L-*boundable*.*

boundable language

Hence, we deduce that reachability is decidable for L-boundable FIFO machines, which is a *strictly larger* class than input-bounded machines. CDP is not input-bounded but it is L_{CDP}-boundable with $L_{CDP} = ((ab)^*(a+\varepsilon)(ab)^*, e^*)$. Let us also remark that CDP is flattable by using the bounded set of runs $(!a!b)^*!a!e?e(!a!b)^* + (!a!b)^*$ (where we omit channel information for readability), because it covers the reachability set which is equal to $(ab)^*(a + \varepsilon)(ab)^*$ on control-state $(0, 0)$. It is not clear whether reachability is decidable for boundable machines. A strategy that would fairly enumerate *all* regular bounded families $L_1, L_2, \ldots, L_n, \ldots$ will necessarily find a good one, if $\mathcal{M}$ is boundable, but this is not sufficient because we must be able to *recognize* $\mathit{Reachset}_{\mathcal{M}}$. Observe that boundable machines are more robust than flat machines. Consider a system $\mathcal{S} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ of n flat finite automata $\mathcal{A}_i$ communicating peer to peer (P2P) through one-directional FIFO channels. Let $M_{\mathcal{S}}$ denote FIFO the machine obtained as the Cartesian product of all automata $\mathcal{A}_i$ of $\mathcal{S}$; there is no reason to assume that $M_{\mathcal{S}}$ is flattable but it is input-bounded and thus $M_{\mathcal{S}}$ is L-boundable where L is easily computable from $\mathcal{S}$.

Chapter 5

Framework for Synchronizability

In this chapter, we are interested in the notion of bounded FIFO machines. If we limit our analysis to decide whether for a given integer $k \geq 0$, the FIFO channels are k-bounded, we could verify this property in PSPACE. However, such a boundedness property is too restricting as we would not be able to design any possibility of unbounded runs. Hence, in order to overcome this limitation, we try to relax the boundedness property to instead only require that every unbounded execution of a system (i.e., channels are unbounded along the execution) is equivalent (for instance, causally equivalent) to another bounded execution.

This leads us study the notion of synchronizability, which roughly translates to verifying if every run can be rescheduled to such a "bounded" run. The definition of such a bounded run varies in the literature - it could mean a channel bound of size k, a rendezvous run, etc. Moreover, the criteria for rescheduling the run also varies in the literature.

We study a framework based on monadic second-order (MSO) logic and (special) tree-width that aims to unify these different notions of synchronizability. Moreover, reachability and model checking are shown decidable in this framework. We also study the various classes of systems which belong to this class, and briefly compare them to each other. For a detailed comparison, one can refer to [Lav21]. It should be noted that we study not only the peer-to-peer model, but also the mailbox model, described in Section 2.2.1.

5.1 Logical background

We first introduce the notions of MSO Logic and PDL.

Monadic Second-Order Logic

We first recall the syntax and semantics of Monadic Second-Order Logic (MSO).

monadic second-order logic (MSO)

Let $x, y, \ldots$ be an infinite set of first-order variables ranging over events in an MSC, and $X, Y, \ldots$ second-order variables ranging over sets of events. Then, the set of MSO formulas over MSCs (over $\mathbb{P}$ and Σ) is given by the grammar:

$$\phi ::= x \rightarrow y \mid x \lhd y \mid \lambda(x) = a \mid x = y \mid x \in X \mid \exists x.\phi \mid \exists X.\phi \mid \phi \vee \phi \mid \neg\phi$$

where $a \in \mathit{Act}$, x and y are first-order variables, and X is a second-order variable. We assume that we have an infinite supply of variables, and we use common abbreviations such as $\wedge$, $\forall$, etc.

Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC. Let Var be a set of variables. An M-interpretation is a function $\mathcal{I}$ that maps every first-order variable $x \in \mathit{Var}$ to some element of $\mathcal{E}$ and every second-order variable $X \in \mathit{Var}$ to some subset of $\mathcal{E}$. The satisfaction relation is defined inductively as follows:

$$\begin{array}{ll}
M \models_{\mathcal{I}} x \rightarrow y & \text{if } \mathcal{I}(x) \rightarrow \mathcal{I}(y) \\
M \models_{\mathcal{I}} x \lhd y & \text{if } \mathcal{I}(x) \lhd \mathcal{I}(y) \\
M \models_{\mathcal{I}} \lambda(x) = a & \text{if } \lambda(\mathcal{I}(x)) = a \\
M \models_{\mathcal{I}} x = y & \text{if } \mathcal{I}(x) = \mathcal{I}(y) \\
M \models_{\mathcal{I}} x \in X & \text{if } \mathcal{I}(x) \in \mathcal{I}(X) \\
M \models_{\mathcal{I}} \exists x.\phi & \text{if there is } e \in \mathcal{E} \text{ such that } M \models_{\mathcal{I}[x \mapsto e]} \phi \\
M \models_{\mathcal{I}} \exists X.\phi & \text{if there is } E \subseteq \mathcal{E} \text{ such that } M \models_{\mathcal{I}[X \mapsto E]} \phi \\
M \models_{\mathcal{I}} \phi \vee \psi & \text{if } M \models_{\mathcal{I}} \phi \text{ or } M \models_{\mathcal{I}} \psi \\
M \models_{\mathcal{I}} \neg\phi & \text{if } M \not\models_{\mathcal{I}} \phi
\end{array}$$

Here, $\mathcal{I}[x \mapsto e]$ maps x to e and coincides with $\mathcal{I}$ on $\mathit{Var} \setminus \{x\}$. When ϕ is a sentence (i.e. a formula without free variables), we omit the (redundant) subscript $\mathcal{I}$, and write $M \models \phi$ instead.

Example 21. We use the abbreviation $\mathit{matched}(x) = \exists y.x \lhd y$ to say that the event associated to x is a matched send event (i.e. there exists a matching reception). Then, the formula $\neg\exists x.(\bigvee_{a \in \mathit{Send}(\_,\_)} \lambda(x) = a \wedge \neg\mathit{matched}(x))$ says that there are no unmatched send events. It is not satisfied by MSC M_1 of Fig. 5.1, as message m_1 is not received, but by M_4 from Fig. 5.7.

Given a sentence ϕ, we let $L(\phi)$ denote the set of p2p MSCs that satisfy ϕ. Note that the (reflexive) transitive closure of a binary relation defined by an MSO formula with free variables x and y, such as $x \rightarrow y$, is MSO-definable. The logic can, therefore, freely use formulas of the form $x \rightarrow^+ y$ or $x \leq y$ (where $\leq$ is interpreted as $\leq_M$ for the given MSC M).

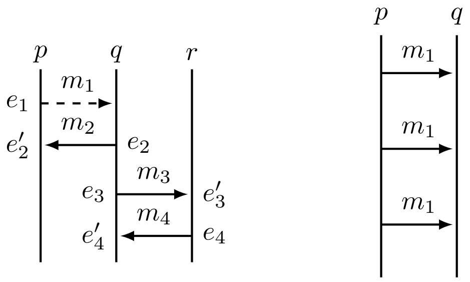


Figure 5.1: MSC M_1 (left) does not satisfy the MSO formula in Example 21 while MSC M_2 does.

Hence, the definition of a mailbox MSC can be readily translated into the formula

MSO mailbox formula

$$\varphi_{\mathsf{mb}} = \neg\exists x.\exists y.(\neg(x = y) \wedge x \preceq y \wedge y \preceq x)$$

so that we have $L(\varphi_{\mathsf{mb}}) = \mathsf{MSC}_{\mathsf{mb}}$. Here, $x \preceq y$ is obtained as the MSO-definable reflexive transitive closure of the union of the MSO-definable relations $\rightarrow$, $\lhd$, and $\sqsubset$. In particular, we may define $x \sqsubset y$ by :

$$x \sqsubset y = \bigvee_{\substack{q\in\mathbb{P}\\ a,b\in Send(\_,q)}} \lambda(x) = a \wedge \lambda(y) = b \wedge \begin{pmatrix} & \mathit{matched}(x) \wedge \neg\mathit{matched}(y) \\ \vee & \exists x'.\exists y'.(x \lhd x' \;\wedge\; y \lhd y' \;\wedge\; x' \rightarrow^{+} y') \end{pmatrix}$$

Propositional Dynamic Logic

While MSO is a very natural and expressive logic, we also study Propositional Dynamic Logic (PDL), which is another classical logic (albeit less expressive than MSO) which has better algorithmic properties. It was introduced to reason about programs originally in [FL79]. However, since then, PDL and its extensions have found applications in verification of transition systems, artificial intelligence, etc. We study an extension of PDL with Loop and Converse, which we call LCPDL (cf. [BG21, BFG21, Str82] for more details).

PDL consists of two types of formulas: event (or state) formulas which are evaluated at events in a structure, and path formulas or programs which are evaluated at pairs of events and allow us to traverse inside the structure. In addition, we also define sentences to reason about global properties of the model. Its syntax is:

propositional dynamic logic (PDL)

$$\Phi ::= \mathsf{E}\sigma \mid \Phi \vee \Phi \mid \neg\Phi \qquad \text{(sentence)}$$

$$\sigma ::= a \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma \mid \mathsf{Loop}\langle\pi\rangle \qquad \text{(event formula)}$$

$$\pi ::= \rightarrow \mid \lhd \mid \mathsf{test}(\sigma) \mid \mathsf{jump} \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^{*} \mid \pi^{-1} \qquad \text{(path formula)}$$

where $a \in Act$. We use the symbol $\top$ to denote a tautology event formula (such as $a \vee \neg a$). We describe the semantics for the logic below:

The semantics of an event formula σ with respect to M is a set $[\![\sigma]\!]_M \subseteq \mathcal{E}$, inductively defined below:

$$\begin{array}{lll}
[\![a]\!]_M & := & \{e \in \mathcal{E} \mid \lambda(e) = a\} \\
[\![\sigma_1 \vee \sigma_2]\!]_M & := & [\![\sigma_1]\!]_M \cup [\![\sigma_2]\!]_M \\
[\![\neg\sigma]\!]_M & := & \mathcal{E} \setminus [\![\sigma]\!]_M \\
[\![\langle\pi\rangle\sigma]\!]_M & := & \{e \in \mathcal{E} \mid \exists f \in [\![\sigma]\!]_M : (e,f) \in [\![\pi]\!]_M\} \\
[\![\mathsf{Loop}\langle\pi\rangle]\!]_M & := & \{e \in \mathcal{E} \mid (e,e) \in [\![\pi]\!]_M\}
\end{array}$$

Intuitively, $[\![\langle\pi\rangle\sigma]\!]_M$ holds at e if there exists a path π which starts at e and ends in some f such that σ holds at f.

The semantics of a path formula π with respect to M is a set $[\![\pi]\!]_M \subseteq \mathcal{E} \times \mathcal{E}$, inductively defined below:

$$\begin{array}{ll}
[\![\rightarrow]\!]_M & := \rightarrow \\
[\![\lhd]\!]_M & := \lhd \\
[\![\mathsf{test}(\sigma)]\!]_M & := \{(e,e) \mid e \in [\![\sigma]\!]_M\} \\
[\![\mathsf{jump}]\!]_M & := \mathcal{E} \times \mathcal{E} \\
[\![\pi_1 + \pi_2]\!]_M & := [\![\pi_1]\!]_M \cup [\![\pi_2]\!]_M \\
[\![\pi_1 \cdot \pi_2]\!]_M & := \{(e,f) \in \mathcal{E} \times \mathcal{E} \mid \exists g \in \mathcal{E} : (e,g) \in [\![\pi_1]\!]_M \text{ and } (g,f) \in [\![\pi_2]\!]_M\} \\
[\![\pi^*]\!]_M & := \bigcup_{n \in \mathbb{N}} [\![\pi]\!]_M^n \\
[\![\pi^{-1}]\!]_M & := \{(e,f) \in \mathcal{E} \times \mathcal{E} \mid (f,e) \in [\![\pi]\!]_M\}
\end{array}$$

A sentence Φ is evaluated wrt. an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$. For sentences, we write

$$M \models \mathsf{E}\sigma \ \text{ if } \ [\![\sigma]\!]_M \neq \emptyset$$

Finally, we let $L(\Phi) = \{M \in \mathsf{MSC} \mid M \models \Phi\}$. Note that every LCPDL-definable property is MSO-definable. However, the converse is not true. For example, expressing that a graph is connected is possible in MSO, but not possible in LCPDL.

LCPDL mailbox formula

It can be seen below that the mailbox semantics can be readily translated into the LCPDL formula: $\Phi_{\mathsf{mb}} = \neg\mathsf{E}\,(\mathsf{Loop}\langle(\lhd + \rightarrow + \sqsubset)^+\rangle)$ such that $L(\Phi_{\mathsf{mb}}) = \mathsf{MSC}_{\mathsf{mb}}$. Hereby, we let

$$\sqsubset \; = \; \lhd \cdot \rightarrow^+ \cdot \lhd^{-1} \; + \sum_{\substack{q \in \mathbb{P} \\ a,b \in Send(\_,q)}} \mathsf{test}(a) \cdot \lhd \cdot \mathsf{jump} \cdot \mathsf{test}(b \wedge \neg\langle\lhd\rangle\top)\,.$$

The first part of the formula checks for the condition that there exists receive messages in the same order as the sends, and the second part checks the condition for the unmatched send.

5.2 Tree-width and conflict graph

We now introduce two notions associated to MSCs.

Special Tree-Width

Special tree-width [Cou10], is a graph measure that indicates how close a graph is to a tree (we may also use classical *tree-width* instead). This or similar measures are commonly employed in verification. For instance, tree-width and split-width have been used in [MP11b] and, respectively, [CGK12, AGK14] to reason about graph behaviour generated by pushdown and channel systems. There are several ways to define the special tree-width of an MSC. We adopt the following game-based definition from [BG21]. We refer the reader to [Cou10] for more details on special tree-width and tree-width.

Definition 5.1. *Given an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$, we define an MSC fragment $M' = (\mathcal{E}', \rightarrow', \lhd', \lambda)$ such that $\mathcal{E}' \subseteq \mathcal{E}$, $\rightarrow' \subseteq \rightarrow$ and $\lhd' \subseteq \lhd$. Moreover, we define a marked MSC fragment as a pair (M, U) such that $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is an MSC fragment, and $U \subseteq \mathcal{E}$ is the subset of "marked" events.*

MSC fragment

Adam and Eve play a two-player turn based "decomposition game" on an arena $\mathsf{Arena}(\mathbb{P}, \Sigma) = (\mathsf{Pos}_\exists \uplus \mathsf{Pos}_\forall, \mathsf{Moves})$, whose positions are MSCs with some pebbles placed on some events. More precisely, Eve's positions $\mathsf{Pos}_\exists$ are *marked MSC fragments* (M, U) and Adam's set of positions $\mathsf{Pos}_\forall$ consists of pairs of marked MSC fragments. A move by Eve consists in the following steps:

decomposition game

1. marking some events of the MSC resulting in (M, U') with $U \subseteq U' \subseteq \mathcal{E}$,
2. removing (process and/or message) edges whose endpoints are marked, resulting in (M', U) with M' being a fragment of M
3. dividing (M, U) in (M_1, U_1) and (M_2, U_2) such that M is the disjoint (unconnected) union of M_1 and M_2 and marked nodes are inherited.

When it is Adam's turn, he simply chooses one of the two marked MSC fragments. The initial position is $(M, \emptyset)$ where M is the (complete) MSC at hand. A terminal position is any position belonging to Eve such that all events are marked. Neither player can move from terminal positions which are winning for Eve. For $k \in \mathbb{N}$, we say that the game is k-winning for Eve if she has a (positional) strategy that allows her, starting in the initial position and independently of Adam's moves, to reach a terminal position such that, in every single position visited along the play, there are at most $k+1$ marked events.

special tree-width

Theorem 5.2 ([BG21]). *The special tree-width of an MSC is the least k such that the associated game is k-winning for Eve.*

The set of MSCs whose special tree-width is at most k is denoted by $\mathsf{MSC}^{k\text{-}\mathsf{stw}}$.

Conflict Graph

Before we delve into verification problems that we are interested in, we first recall the notion of a conflict graph associated to an MSC, as defined in [BEJQ18], which is also referred to as a dependency graph in the literature. This graph is used to depict the causal dependencies between message exchanges. Each vertex of the graph represents a message and each edge between two vertices represents a temporal dependence between two actions of the two messages concerned. A dependency exists if both messages have a process in common. A dependency linked to a sending will be represented by an S while a dependency linked to a reception will be represented by an R.

For instance, an $\xrightarrow{SS}$ dependency between message exchanges v and v' expresses the fact that v' has been sent after v, by the same process. This model is of interest because it was seen in [BEJQ18] that the notion of synchronizability in MSCs (which is studied in this paper) can be graphically characterized by the nature of the associated conflict graph. It is defined in terms of linearization in [DGLL20], but we equivalently express it directly in terms of MSCs.

For an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ and $e \in \mathcal{E}$, we define the type $\tau(e) \in \{S, R\}$ of e by $\tau(e) = S$ if $e \in \mathit{SendEv}(M)$ and $\tau(e) = R$ if $e \in \mathit{RecEv}(M)$. Moreover, for $e \in \mathit{Unm}(M)$, we let $\mu(e) = e$, and for $(e, e') \in \lhd$, we let $\mu(e) = \mu(e') = (e, e')$.

conflict graph

Definition 5.3. *The* conflict graph $\mathsf{CG}(M)$ *of an MSC* $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ *is the labelled graph* $(\mathit{Nodes}, \mathit{Edges})$*, with* $\mathit{Edges} \subseteq \mathit{Nodes} \times \{S, R\}^2 \times \mathit{Nodes}$*, defined by* $\mathit{Nodes} = \lhd \cup \mathit{Unm}(M)$ *and* $\mathit{Edges} = \{(\mu(e), \tau(e)\tau(f), \mu(f)) \mid (e, f) \in \rightarrow^+\}$*. In particular, a node of* $\mathsf{CG}(M)$ *is either a single unmatched send event or a message pair* $(e, e') \in \lhd$*.*

Example 22. We see in Figure 5.2 the MSC M_1 and its associated conflict graph. Note that since m_1 is not received by process q, we do not have any dependencies between e_1 and nodes associated to events in q.

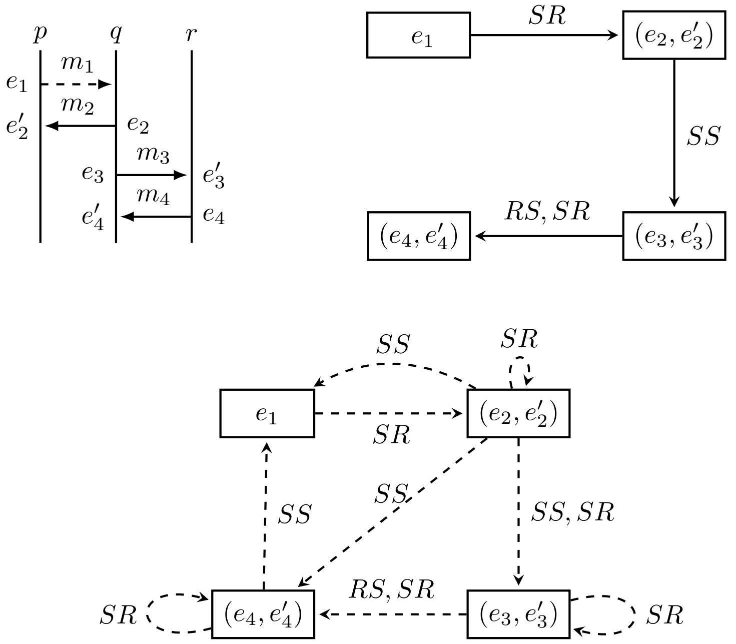


Figure 5.2: MSC M_1 (top left) and its associated conflict graph (top right) and the extended conflict graph (below)

Extended Conflict Graph

As was shown in [DGLL20], in order to capture the mailbox semantics, we need extended edges. We recall from [DGLL20] the extended edge relation $\overset{XY}{\dashrightarrow}$ with $X, Y \in \{S, R\}$ in Figure 5.3. We call the conflict graph along with the new extended edges the *extended conflict graph* (ECG).

extended conflict graph

$$\frac{v_1 \xrightarrow{XY} v_2}{v_1 \overset{XY}{\dashrightarrow} v_2}\ \text{(Rule 1)} \qquad \frac{v \in \lhd}{v \overset{SR}{\dashrightarrow} v}\ \text{(Rule 2)}$$

$$\frac{v_1 \xrightarrow{RR} v_2}{v_1 \overset{SS}{\dashrightarrow} v_2}\ \text{(Rule 3)} \qquad \frac{v_1 \overset{XY}{\dashrightarrow} \overset{YZ}{\dashrightarrow} v_2}{v_1 \overset{XZ}{\dashrightarrow} v_2}\ \text{(Rule 4)}$$

$$\frac{\begin{array}{c} e_1 \in \mathit{Matched}(M) \quad e_2 \in \mathit{Unm}(M) \\ e_1 \in \mathit{Send}(\_, q),\ e_2 \in \mathit{Send}(\_, q),\ q \in \mathbb{P} \end{array}}{\mu(e_1) \overset{SS}{\dashrightarrow} e_2}\ \text{(Rule 5)}$$

Figure 5.3: Additional rules for extended conflict graph; $\xrightarrow{XY}$ refers to an edge in the conflict graph

Example 23. We see in Figure 5.2 the MSC M_1 and its associated extended conflict graph. Note that there is now a dependence between e_1 and (e_4, e_4') by Rule 5, since the message m_1 is not received by q, it should causally be after m_4 which is received by q.

5.3 Model checking and synchronizability

In this section, we introduce two decision problems for communicating systems, which we have not encountered in the previous chapters.

The first problem is the model-checking problem, in which one checks whether a given system satisfies a given specification. A canonical specification language for MSCs is monadic second-order (MSO) logic. However, model checking in full generality is undecidable. A common approach is, therefore, to restrict the behaviour of the given system to MSCs of bounded (special) tree-width.

Model Checking

We have already seen that in the general case, verification problems such as control-state reachability, are undecidable for communicating systems. However, they are decidable when we restrict to behaviour of bounded special tree-width, which motivates the following definition of a generic **bounded model-checking problem** for com $\in \{\mathsf{p2p}, \mathsf{mb}\}$:

Decision Problem:	BOUNDED MODEL-CHECKING
Input:	Two finite sets $\mathbb{P}$ and Σ, a communicating system $\mathcal{S}$, an MSO sentence ϕ, and $k \in \mathbb{N}$
Question:	do we have $L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{k\text{-}\mathsf{stw}} \subseteq L(\phi)$?

Theorem 5.4 ([BG21])**.** *The bounded model-checking problem for* com $=$ $\mathsf{p2p}$ *is decidable. When the formulas ϕ are from LCPDL, then the problem is solvable in exponential time.*

Note that [BG21] does not employ the LCPDL modality jump, but it can be integrated easily. Using φ_{mb} or Φ_{mb}, we obtain the corresponding result for mailbox systems as a corollary:

Theorem 5.5. *The bounded model-checking problem for* com $=$ mb *is decidable. When the formulas ϕ are from LCPDL, then the problem is solvable in exponential time.*

Proof. Using the mailbox semantics and the MSO formula φ_{mb}, we get

$$\begin{aligned} & L_{\mathsf{mb}}(\mathcal{S}) \cap \mathsf{MSC}^{k\text{-}\mathsf{stw}} \subseteq L(\phi) \\ \iff\; & L_{\mathsf{p2p}}(\mathcal{S}) \cap \mathsf{MSC}^{k\text{-}\mathsf{stw}} \cap L(\varphi_{\mathsf{mb}}) \subseteq L(\phi) \\ \iff\; & L_{\mathsf{p2p}}(\mathcal{S}) \cap \mathsf{MSC}^{k\text{-}\mathsf{stw}} \subseteq L(\phi \vee \neg\varphi_{\mathsf{mb}})\,. \end{aligned}$$

The latter is decidable due to Theorem 5.4. Similarly, we can use the LCPDL formula Φ_{mb}, whose size is polynomial in the number of processes and messages. □

Synchronizability

The above model-checking approach is incomplete in the sense that a positive answer does not imply correctness of the whole system. The system may still produce behaviour of special tree-width greater than k that violate the given property. However, if we know that a system only generates behaviour from a class whose special tree-width is bounded by k, we can still conclude that the system is correct.

This motivates the *synchronizability problem*. Several notions of synchronizability have been introduced in the literature. However, they all amount to asking whether all behaviours generated by a given communicating system have a particular shape, i.e., whether they are all included in a fixed (or given) set of MSCs $\mathcal{C}$.

Hence, we study the synchronizability problem as an inclusion problem, i.e. we check if $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}$ or $L_{\mathsf{mb}}(\mathcal{S}) \subseteq \mathcal{C}$. We show that if $\mathcal{C}$ is MSO-definable and special-tree-width-bounded (STW-bounded), then this inclusion is decidable.

We call $\mathcal{C} \subseteq \mathsf{MSC}$

(i) *MSO-definable* if there is an MSO-formula ϕ such that $L(\phi) = \mathcal{C}$,

(ii) *LCPDL-definable* if there is an an LCPDL-formula Φ such that $L(\Phi) = \mathcal{C}$, and

(iii) *STW-bounded* if there is $k \in \mathbb{N}$ such that $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-}\mathsf{stw}}$.

In order to prove decidability of inclusion, we first define and prove the following lemma, which essentially claims that every MSC which does not belong to $\mathcal{C} \subseteq$

$\mathsf{MSC}^{k\text{-stw}}$ has a minimal violation, i.e. a prefix of special tree-width $k+2$ which also does not belong to $\mathcal{C}$. Note that a similar property was shown in [GKM07, Proposition 5.4] for the specific class of existentially k-bounded MSCs.

Lemma 5.6. *Let $k \in \mathbb{N}$ and $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$. For all $M \in \mathsf{MSC} \setminus \mathcal{C}$, we have $(\mathit{Pref}(M) \cap \mathsf{MSC}^{(k+2)\text{-stw}}) \setminus \mathcal{C} \neq \emptyset$.*

Proof. Let k and $\mathcal{C}$ be fixed, and let $M \in \mathsf{MSC} \setminus \mathcal{C}$ be an MSC that does not belong to $\mathcal{C}$. Let $M' \in \mathit{Pref}(M) \setminus \mathcal{C}$ be a prefix of M such that, for all $\leq_{M'}$-maximal events e (all events e such that there exists no event f such that $e \leq_{M'} f$) of M', removing e (and its adjacent edge(s)) creates an MSC in $\mathcal{C}$.

We can obtain such an MSC by successively removing maximal events. If M' is the empty MSC, we are done, since then $M' \in (\mathit{Pref}(M) \cap \mathsf{MSC}^{k+2\text{-stw}}) \setminus \mathcal{C}$. Otherwise, let e be $\leq_{M'}$-maximal and let $M'' = M' \setminus \{e\}$.

From our construction, we now know that $M'' \in \mathcal{C}$. So Eve has a winning strategy with $k+1$ pebbles for M''. Let us design a winning strategy with $k+3$ pebbles for Eve for M', which will show the claim.

Observe that the event e occurs at the end of the timeline of a process (say p) since it is $\leq_{M'}$-maximal, and it is part of at most two edges:

- one with the previous p-event (if any)
- one with the corresponding send event (if e is a receive event)

Note that e cannot be a send event with a matching receive in M' because then it would not be $\leq_{M'}$-maximal.

Let e_1, e_2 be the two Neighbors of e (if they exist). The strategy of Eve is the following: in the first round, mark e, e_1, e_2, then erase the edges (e_1, e) and (e_2, e), then split the remaining graph in two parts: M'' on the one side, and the single node graph $\{e\}$ on the other side. Then Eve applies its winning strategy for M'', except that initially the two events e_1, e_2 are marked (so she may need up to $k+3$ pebbles).

Therefore, the MSC M' has special tree-width $k+2$. □

An important component of the decidability of inclusion is the following lemma, which shows that we can reduce synchronizability to bounded model-checking for an STW-bounded class, which uses the above lemma.

Lemma 5.7. *Let $\mathcal{S}$ be a communicating system,* com $\in \{\mathsf{p2p}, \mathsf{mb}\}$*, $k \in \mathbb{N}$, and $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$. Then, $L_{\mathrm{com}}(\mathcal{S}) \subseteq \mathcal{C}$ iff $L_{\mathrm{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-stw}} \subseteq \mathcal{C}$.*

Proof. It is obvious that if $L_{\text{com}}(\mathcal{S}) \subseteq \mathcal{C}$, then every MSC $M \in L_{\text{com}}(\mathcal{S})$ has special tree-width k, and therefore $L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-}\mathsf{stw}} = L_{\text{com}}(\mathcal{S}) \subseteq \mathcal{C}$.

For the case when $L_{\text{com}}(\mathcal{S}) \not\subseteq \mathcal{C}$, let us assume there exists an MSC $M \in L_{\text{com}}(\mathcal{S}) \setminus \mathcal{C}$. Then, by Lemma 5.6, we know that $(\mathit{Pref}(M) \cap \mathsf{MSC}^{(k+2)\text{-}\mathsf{stw}}) \setminus \mathcal{C} \neq \emptyset$, and since, $L_{\text{com}}(\mathcal{S})$ is prefix-closed, $L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-}\mathsf{stw}} \not\subseteq \mathcal{C}$. □

We now have all ingredients to state a generic decidability result for the synchronizability problem for com $\in \{\mathsf{p2p}, \mathsf{mb}\}$:

Decision Problem:	SYNCHRONIZABILITY
Input:	Two finite sets $\mathbb{P}$ and Σ, an MSO-definable and STW-bounded class $\mathcal{C} \subseteq \mathsf{MSC}$ a communicating system $\mathcal{S}$,
Question:	do we have $L_{\text{com}}(\mathcal{S}) \subseteq \mathcal{C}$?

Theorem 5.8. *The synchronizability problem is decidable.*

Proof. Consider the MSO-formula ϕ such that $L(\phi) = \mathcal{C}$, and let $k \in \mathbb{N}$ such that $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-}\mathsf{stw}}$. We have $L_{\text{com}}(\mathcal{S}) \subseteq \mathcal{C} \overset{\text{Lemma 5.7}}{\iff} L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-}\mathsf{stw}} \subseteq \mathcal{C} \iff L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-}\mathsf{stw}} \subseteq L(\phi)$. The latter can be solved thanks to Theorem 5.4 and Theorem 5.5. □

Note that, in some cases (cf. Section 5.4), $\mathbb{P}$ and Σ are part of the input and the concrete class $\mathcal{C}$ may be parametrized by a natural number so that it is part of the input, too. Then, we need to be able to compute the MSO formula characterizing the class as well as the bound on the special tree-width.

5.4 Application to concrete classes of synchronizability

In this section, we instantiate our general framework by specific classes.

5.4.1 A new general class: Weakly synchronous MSCs

We first introduce the class of weakly synchronous MSCs. This is a generalization of synchronous MSCs studied earlier, in [BEJQ18, DGLL20], which we shall discuss in Section 5.4.2. We say an MSC is weakly synchronous if it is breakable into *exchanges* where an exchange is an MSC that allows one to schedule all sends before all receives. Let us define this formally:

exchange

Definition 5.9. *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC. We say that M is an* exchange *if $\mathit{SendEv}(M)$ is a $\leq_M$-downward-closed set.*

weakly synchronous

Definition 5.10. *We say that $M \in \mathsf{MSC}$ is* weakly synchronous *if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is an exchange.*

We use the term *weakly* to distinguish from variants introduced later.

Example 24. Consider the MSC M_2 in Fig. 5.4. It is is weakly synchronous. Indeed, m_1, m_2, and m_5 are independent and can be put alone in an exchange. Repetitions of m_3 and m_4 are interlaced, but they constitute an exchange, as we can do all sends and then all receptions.

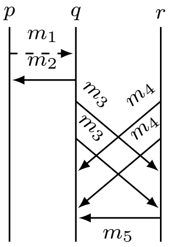


Figure 5.4: MSC M_2

An easy adaptation of a characterization from [DGLL20] yields the following result for weakly synchronous MSCs:

Proposition 5.11. *Let M be an MSC. Then, M is weakly synchronous iff no RS edge occurs on any cyclic path in the conflict graph $\mathsf{CG}(M)$.*

Proof. Let us first show that the conflict graph of every weakly synchronous MSC has no RS edge on any cyclic path. Let M be an MSC. If M is weakly synchronous, then $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is an exchange. Hence, for every vertex $v \in \mathit{Nodes}$ of the conflict graph $\mathsf{CG}(M)$, the associated send action belongs to exactly one k-exchange, which we denote by$M_{\iota(v)}$. In other words, if $v \in \lhd$ (resp. $v \in \mathit{Unm}(M)$), then v can be represented by (e_ℓ, e'_ℓ) (resp. (e_ℓ)), and $\lambda^{-1}(e_\ell) \in M_{\iota(v)}$. Note that if there is an edge from v to v' in the conflict graph, some action of v must happen before some action of v', and hence, $\iota(v) \leq \iota(v')$. Furthermore, note that if $v \xrightarrow{RS} v'$, then $\iota(v) < \iota(v')$, since $SendEv(M_{\iota(v')})$ is downward-closed, i.e. all the send events in an exchange precede all the receive events. Hence, there can be no edge $v' \to v$ in the conflict graph, so there is no cycle in the conflict graph with an RS edge.

Let M be an MSC. Conversely, we assume now that the conflict graph of M does not contain a cyclic path with an RS edge. Let $V_1, \ldots, V_n$ be the set of maximal SCCs (strongly connected components) of the conflict graph, listed in some topological order. For a fixed i, let $M_i = s_1 \ldots s_m r_1 \ldots r_{m'}$ be the enumeration of the actions of the message exchanges of V_i defined by first taking all send actions of V_i obeying the relation $\leq_M$, and then all the receive actions of V_i in the same order as in $\leq_M$. Let $M' = M_1 \ldots M_n$. Then the conflict graph of M' is the same as that of M, as the permutation of actions we defined could only postpone a receive after a send of a same SCC, therefore it could only replace some $v \xrightarrow{RS} v'$ edge with an $v \xrightarrow{SR} v'$ edge between two vertices v, v' of a same SCC. However, since we assumed that the cycles (hence by extensions SCCs) do not contain RS edges, this cannot happen. Therefore M and M' have the same conflict graph, and correspond to the same MSC. Furthermore, since the sends precede the receives in each M_i, $SendEv(M_i)$ is a downward-closed set, hence, M' is weakly synchronous, and so is M. □

We now show that the characterization from Proposition 5.11 is LCPDL-definable:

Corollary 5.12. *The sets of weakly synchronous MSCs and weakly synchronous* mailbox *MSCs are LCPDL-definable. Both formulas have polynomial size.*

Proof. LCPDL can be used to express the graphical characterization of weakly synchronous MSCs. This follows from the formulas below. Here, we let $S = \bigvee_{a \in Send(\_,\_,\_)} a$ and $R = \bigvee_{a \in Rec(\_,\_,\_)} a$.

$$\begin{aligned}
\xrightarrow{SS} &= \mathsf{test}(\neg R) \cdot \to^+ \cdot \mathsf{test}(\neg R) \\
\xrightarrow{RR} &= \mathsf{test}(\neg R) \cdot \lhd \cdot \to^+ \cdot \lhd^{-1} \cdot \mathsf{test}(\neg R) \\
\xrightarrow{RS} &= \mathsf{test}(\neg R) \cdot \lhd \cdot \to^+ \cdot \mathsf{test}(\neg R) \\
\xrightarrow{SR} &= \mathsf{test}(\neg R) \cdot \to^+ \cdot \lhd^{-1} \cdot \mathsf{test}(\neg R) \\
\xrightarrow{CG} &= (\xrightarrow{SS} + \xrightarrow{RR} + \xrightarrow{RS} + \xrightarrow{SR})
\end{aligned}$$

The absence of RS edges in any cycle in the conflict graph can be expressed by the following formula:

$$\Phi_{wsync} = \neg \mathsf{Loop}\langle (\xrightarrow{CG})^* \cdot \xrightarrow{RS} \cdot (\xrightarrow{CG})^* \rangle$$ □

Moreover, under the mailbox semantics, we can show:

Proposition 5.13. *The set of weakly synchronous mailbox MSCs is STW-bounded (in fact, it is included in* $\mathsf{MSC}^{4|\mathbb{P}|\text{-}\mathsf{stw}}$*).*

Proof. Let M be fixed, and let us sketch Eve's winning strategy. Let $n = |\mathbb{P}|$.

The first step for Eve is to split M in exchanges. She first disconnects the first exchange from the rest of the graph ($2n$ pebbles are needed as she can mark the maximal event of each process of the first exchange and the minimal event of each process of the following exchange), then she disconnects the second exchange from the rest of the graph ($2n$ pebbles are needed again, plus the n pebbles we used from the first round), and so on for each exchange.

Now, given an exchange M_i, we design a winning strategy for Eve with $4n+1$ pebbles, where initially there are (at most) n pebbles placed on the first event of each process and also (at most) n pebbles placed on the last event of each process. Eve also places (at most) n pebbles on the last send event of each process and also (at most) n pebbles on the first receive event of each process. Eve erases the (at most) n $\rightarrow$-edges between the last send event and the first receive event (as sends precede the receives in an exchange).

We are now in a configuration that will be our invariant.

Let us fix a mailbox linearization of M_0 and let e be the first send event in this linearization.

- if e is an unmatched send of process p, Eve places her last pebble on the next send event of p (if it exists), let us call it e'. Then Eve erases the $\rightarrow$-edge (e, e'), and now e is completely disconnected, so it can be removed and the pebble can be taken back.
- if $e \lhd e'$, with e' a receive event of process q, then due to the mailbox semantics e' is the first receive event of q, so it has a pebble placed on it. Eve removes the $\lhd$-edge between e and e', then using the extra pebble she disconnects e and places a pebble on the $\rightarrow$-successor of e, then she also disconnects e' and places a pebble on the $\rightarrow$-successor of e'.

After that, we are back to our invariant, so we can repeat the same strategy with the second send event of the linearization, and so on until all edges have been erased. □

Example 25. Consider the MSC M_2 given in Figure 5.5(a). We shall play the decomposition game on it. First Eve splits M into exchanges. As explained in Example 24, m_1, m_2 and m_5 can be put alone into an exchange. First, Eve separates m_1 from the rest. To this end, she puts the pebbles at the maximal event of the first exchange. In this case, that is just the send event of m_1, and the minimal events on all processes after this event. She also removes the associate edges (see Fig 5.5 (b)).

Now m_1 is separated from the rest. For Adam, the best option is to choose the rest of the MSC. Let us assume he does that. Eve then proceeds to remove the second exchange. In this case, that is the sole message m_2. Eve repeats the process of separating it from the rest (see Fig 5.5 (c)).

Once again Adam best choice is the larger graph. Eve proceeds to separate the exchange with m_3 and m_4 messages from the rest. (see Fig 5.5 (d)).

Now Adam can only choose the exchange with m_3 and m_4 messages. Eve places pebbles on the last send and first receive of each process (see Fig 5.5 (e)). We now see that the sends are separated from the receives along a process. Moreover, since the MSC is weakly synchronous, we have (at least) one event which we can separate from the rest. We show the graph after removing the edges in Fig 5.5 (f). Now, Eve separates the first process from the rest, as shown in Fig 5.5 (g).

In the next step, all events will marked when Eve chooses the next node to separate from the rest. Hence, Eve wins with less that 13 pebbles. In fact, since there are no longer any p-events from step (b), we even see that Eve does it with less than 9 pebbles.

We obtain the following result as a corollary. Note that it assumes the mailbox semantics.

Theorem 5.14. *The following problem is decidable in exponential time: Given $\mathbb{P}$, Σ, and a communicating system $\mathcal{S}$ (over $\mathbb{P}$ and Σ), is every MSC in $L_{\mathsf{mb}}(\mathcal{S})$ weakly synchronous?*

Proof. According to Corollary 5.12, we determine the LCPDL formula Φ_{wsmb} such that $L(\Phi_{\mathsf{wsmb}})$ is the set of weakly synchronous mailbox MSCs. Moreover, recall from Proposition 5.13 that the special tree-width of all weakly synchronous mailbox MSCs is bounded by $4|\mathbb{P}|$. By Lemma 5.7, $L_{\mathsf{mb}}(\mathcal{S}) \subseteq L(\Phi_{\mathsf{wsmb}})$ iff $L_{\mathsf{mb}}(\mathcal{S}) \cap \mathsf{MSC}^{(4|\mathbb{P}|+2)\text{-}\mathsf{stw}} \subseteq L(\Phi_{\mathsf{wsmb}})$. The latter is an instance of the bounded model-checking problem. As the length of Φ_{wsmb} is polynomial in $|\mathbb{P}|$, we obtain that the original problem is decidable in exponential time by Theorem 5.5. □

For the same reasons, the model-checking problem for "weakly synchronous" systems is decidable. However, a reduction from the Post correspondence problem (PCP) [Pos46] shows that decidability fails when adopting the p2p semantics:

Theorem 5.15. *The following problem is undecidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$ as well as a communicating system $\mathcal{S}$, is every MSC in $L_{\mathsf{p2p}}(\mathcal{S})$ weakly synchronous?*

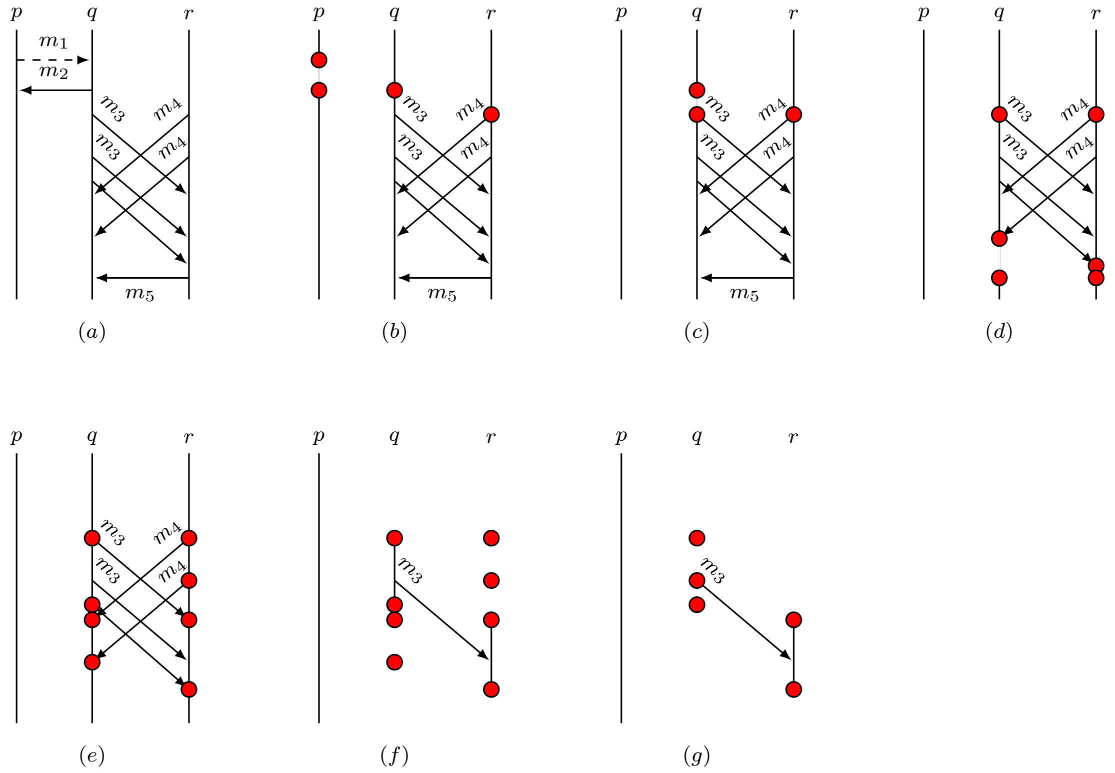


Figure 5.5: Playing the decomposition game on M_2.

Proof. We show that the control state reachability problem for p2p weakly synchronizable systems is not decidable. This immediately shows that the model-checking problem for p2p weak synchronizable systems is not decidable. We can also argue that the membership problem (decide whether a given system is p2p weakly synchronizable) is undecidable: indeed, it is enough to add behaviour that is not weakly-synchronizable after the control states for which reachability is undecidable: the system will be not weakly synchronizable iff the control states are reached.

We reduce from Post correspondence problem (PCP). Let us recall the Post correspondence problem.

Decision Problem: POST CORRESPONDENCE PROBLEM [Pos46]

Input:	An alphabet A and, two finite lists $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$ of words over A
Question:	do we have a sequence $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq N$ such that $\alpha_{i_1} \ldots \alpha_{i_K} = \beta_{i_1} \ldots \beta_{i_K}$?

It is well known that this problem is already undecidable for $N = 7$ and $|A| = 2$.

We let the set of messages be $\Sigma = \{1, \ldots, N\} \uplus A \uplus \{\sharp\}$, and we consider a system with four machines: Prover$_1$, Prover$_2$, Verifier$_1$, and Verifier$_2$. We can have a partitioned alphabet to model the definition by adding an indicator of the source of each message, but for simplicity we omit it and assume a shared alphabet.

Informally, the system works as follows:

- Prover$_1$ guesses a solution $i_1 \ldots i_K$ of the PCP instance, and Prover$_2$ also guesses a solution $i_1 \ldots i_K$.
- Prover$_1$ sends $\alpha_{i_1} \ldots \alpha_{i_K}$ to Verifier$_1$ and sends simultaneously $i_1 \ldots i_K$ to Verifier$_2$
- Prover$_2$ sends $\beta_{i_1} \ldots \beta_{i_K}$ to Verifier$_1$ and sends simultaneously $i_1 \ldots i_K$ to Verifier$_2$
- Verifier$_1$ checks that the two words are equal and Verifier$_2$ checks that the sequences of indices are equal.

Let us now formally define these machines. We will describe them with regular expressions (that lead them to a specific control-state). For $w = a_1 \cdots a_n$, we write $\langle (p,q)!w \rangle$ (respectively $\langle (p,q)?w \rangle$) for $\langle (p,q)!a_1 \rangle \ldots \langle (p,q)!a_n \rangle$ (respectively $\langle (p,q)?a_1 \rangle \ldots \langle (p,q)?a_n \rangle$).

Furthermore, we denote Prover$_i$ as P_i and Verifier$_i$ as V_i. The channels between, say, Prover$_1$ and Verifier$_1$ will be denoted by (P_1, V_1) (and similarly for the other channels).

The language of

- Prover$_1$ is
$$\Big(\sum_{i=1}^{K} \langle (P_1,V_1)!\alpha_i \rangle \cdot \langle (P_1,V_2)!i \rangle \Big)^+ \cdot \langle (P_1,V_1)!\# \rangle \cdot \langle (P_1,V_2)!\# \rangle$$
- Prover$_2$ is
$$\Big(\sum_{i=1}^{K} \langle (P_2,V_1)!\beta_i \rangle \cdot \langle (P_2,V_2)!i \rangle \Big)^+ \cdot \langle (P_2,V_1)!\# \rangle \cdot \langle (P_2,V_2)!\# \rangle$$
- Verifier$_1$ is
$$\Big(\sum_{a \in \Sigma} \langle (P_1,V_1)?a \rangle \cdot \langle (P_2,V_1)?a \rangle \Big)^* \cdot \langle (P_1,V_1)?\# \rangle \cdot \langle (P_2,V_1)?\# \rangle$$

- Verifier$_2$ is

$$\Big(\sum_{i=1}^{K}\langle(P_1,V_2)?i\rangle\cdot\langle(P_2,V_2)?i\rangle\Big)^*\cdot\langle(P_1,V_2)?\#\rangle\cdot\langle(P_2,V_2)?\#\rangle$$

As the channels are unidirectional, the system is weakly synchronous by default. Moreover, it can be seen that all machines reach the specified control-state if and only if the PCP instance has a solution. □

5.4.2 Weakly k-synchronous MSCs

The negative result for the p2p semantics of weakly synchronous MSCs motivates the study of other classes. We now look at a class from the literature.

k-exchange

Definition 5.16. *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC and $k \in \mathbb{N}$. We call M a k-*exchange *if M is an exchange and $|SendEv(M)| \leq k$.*

Let us now recall the definition from [BEJQ18, DGLL20], but (equivalently) expressed directly in terms of MSCs rather than via *executions*. It differs from the weakly synchronous MSCs in that here, we insist on constraining the number of messages sent per exchange to be at most k.

weakly k-synchronous

Definition 5.17. *Let $k \in \mathbb{N}$. We say that $M \in$ MSC is weakly k-synchronous if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is a k-exchange.*

Example 26. MSC M_3 in Fig. 5.6 is weakly 1-synchronous, as it can be decomposed into three 1-exchanges (the decomposition is depicted by the horizontal dashed lines). We remark that $M_3 \in \mathsf{MSC}_{\mathsf{mb}}$. Note that there is a p2p linearization that respects the decomposition. On the other hand, a mailbox linearization needs to reorganize actions from different MSCs: the sending of m_3 needs to be done before the sending of m_1. Note that M_1 in Fig. 2.3 is also weakly 1-synchronous.

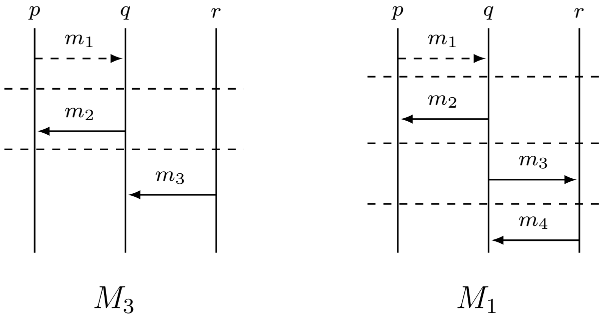


Figure 5.6: MSC M_3 and MSC M_1 (right) are weakly 1-synchronous.

For weakly k-synchronous MSCs, MSO-definability essentially follows from the following known theorem:

Theorem 5.18 ([DGLL20]). *Let M be an MSC. Then, M is weakly k-synchronous iff every SCC in its conflict graph $\mathsf{CG}(M)$ is of size at most k and no RS edge occurs on any cyclic path.*

Proposition 5.19. *Let $k \in \mathbb{N}$. The set of weakly k-synchronous p2p (mailbox, respectively) MSCs is effectively MSO-definable.*

Proof. We first denote some relations between events.

$$\begin{array}{rcl}
rec(x) & = & \exists y.(y \lhd x) \\
SS(e_1, e_2) & = & e_1 \neq e_2 \wedge \neg rec(e_1) \wedge \neg rec(e_2) \wedge e_1 \to^* e_2 \\
RR(e_1, e_2) & = & e_1 \neq e_2 \wedge \neg rec(e_1) \wedge \neg rec(e_2) \wedge \exists f_1, f_2.[e_1 \lhd f_1 \wedge e_2 \lhd f_2 \wedge f_1 \to^* f_2] \\
RS(e_1, e_2) & = & e_1 \neq e_2 \wedge \neg rec(e_1) \wedge \neg rec(e_2) \wedge \exists f_1.[e_1 \lhd f_1 \wedge f_1 \to^* e_2] \\
SR(e_1, e_2) & = & e_1 \neq e_2 \wedge \neg rec(e_1) \wedge \neg rec(e_2) \wedge \exists f_2.[e_2 \lhd f_2 \wedge e_1 \to^* f_2] \\
CG(e_1, e_2) & = & \bigvee_{X,Y \in \{R,S\}} XY(e_1, e_2)
\end{array}$$

The formula for the property that there is no strongly connected component of size greater than k in the conflict graph can be expressed as follows:

$$\nexists e_1, \ldots, e_{k+1}.\ [\bigwedge_{i \neq j} CG^*(e_i, e_j)]$$

And finally, we express the property that there is no RS edge in any cycle of the conflict graph

$$\nexists e_1, e_2.[CG^*(e_1, e_2) \wedge RS(e_2, e_1)]$$

□

This property is similar to the graphical characterization of weakly synchronous MSCs, except for the condition that every SCC in the conflict graph is of size at most k. However, note that the set of weakly k-synchronous MSCs is not directly expressible in LCPDL (the reason is that LCPDL does not have a built-in counting mechanism). However, its *complement* is expressible in the extension of LCPDL with existentially quantified propositions (we need $k + 1$ of them). The model-checking problem for this kind of property is still in EXPTIME and, therefore, so is the problem from Theorem 5.21 when k is given in unary. It is very likely that our approach can also be used to infer the PSPACE upper bound from [BEJQ18] by showing bounded *path width* and using finite word automata instead of tree automata. Finally, note that the problem to decide whether there exists an integer $k \in \mathbb{N}$ such that all MSCs in $L_{\text{com}}(\mathcal{S})$ are weakly k-synchronous has recently been studied in [GLL21] and requires different techniques.

Next, we establish a bound on the special tree-width:

Proposition 5.20. *Let $k \in \mathbb{N}$. The set of MSCs that are weakly k-synchronous have special tree-width bounded by $2k + |\mathbb{P}|$.*

Proof. Let M be a k-synchronous MSC. By definition, we know that $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is a k-exchange.

Eve's strategy is to mark the vertices belonging to the set M_1. Hence, she marks at most $2k$ vertices. We can remove the edges between these vertices. Let the new marked MSC fragment be (G, U), where G is the new MSC fragment (with the edges between marked vertices removed), and U the set of marked vertices.

Notice that $|U| \leq 2k$. Furthermore, since every vertex corresponding to a send message in U is either unmatched or matched with a reception in U (by definition), we can be sure that there are no message edges between vertices of U and any other vertex. Moreover, we can also be sure that there are at most $|\mathbb{P}|$ process edges between vertices in U and vertices outside this set. Let us mark these $|\mathbb{P}|$ vertices. We call these vertices U'. Let the new marked MSC fragment be $(G', U \cup U')$, where G' is the new MSC fragment (with the edges between all vertices in $U \cup U'$ removed). Now, we see that there are no edges between any of the vertices in U and any other vertex, i.e. all the vertices in U are isolated. We can divide the MSC fragment to consist of the vertices U and $V \setminus U$ and the corresponding edges.

Let the MSC fragment with vertices in U be (G_1, U_1). It consists of at most $2k$ isolated coloured vertices. Let the MSC fragment with vertices in $V \setminus U$ be (G_2, U_2). We observe that $|U_2| = n$. Adam trivially loses if he chooses (G_1, U_1), hence, he has to choose (G_2, U_2). Now, we mark the vertices corresponding to M_2, which are again, at most $2k$. We have two possibilities for each vertex in U_2, either they belong to the set M_2 or belong to another set M_p where $p > 2$. However, if they belong to M_p, we can be sure that there is no other event on the same process that belongs to M_2 - this is because it was the successor of some event in M_1. Hence, we see once again, that marking all the vertices in M_2 and the immediate successors along each process will result in marked vertices of size at most $2k + |\mathbb{P}|$. And once again, we see that we can separate into MSC fragments (G_1', U_1') and (G_2', U_2') such that every vertex in U_1' is isolated, and $|U_1'| \leq 2k$. We do this for all $i \in [n]$, and hence, we can effectively use $2k + |\mathbb{P}|$ colours. Therefore, set of MSCs over $|\mathbb{P}|$ processes which are k-synchronous have bounded special tree-width. □

Hence, we can conclude that the class of weakly k-synchronous MSCs is MSO-definable and STW-bounded. As a corollary, we get the following (known) decidability result, but via an alternative proof:

Theorem 5.21 ([BEJQ18, DGLL20]). *For* com $\in \{\mathsf{p2p}, \mathsf{mb}\}$*, the following problem is decidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, and $k \in \mathbb{N}$, is every MSC in $L_{\text{com}}(\mathcal{S})$ weakly k-synchronous?*

Proof. We proceed similarly to the proof of Theorem 5.14. For the given $\mathbb{P}$, $\mathbb{M}$, and k, we first determine the MSO formula ϕ_k such that $L(\phi_k)$ is the set of weakly k-synchronous p2p/mailbox MSCs, from Proposition 5.19. From Proposition 5.20, we know that the special tree-width of all weakly k-synchronous MSCs is bounded by $2k + |\mathbb{P}|$. By Lemma 5.7, we have $L_{\text{com}}(\mathcal{S}) \subseteq L(\phi_k)$ iff $L_{\text{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(2k+|\mathbb{P}|+2)\text{-}\mathsf{stw}} \subseteq L(\phi_k)$. The latter is an instance of the bounded model-checking problem. By Theorem 5.4 and Theorem 5.5, we obtain decidability. □

Observe also that we can remove the constraint of all the sends preceding all the receives in a k-exchange, and still have decidability. We then have the following definition.

Definition 5.22. *Let $M = (\mathcal{E}, \to, \lhd, \lambda)$ be an MSC and $k \in \mathbb{N}$. We call M a* modified k-exchange *if $|SendEv(M)| \leq k$.*

modified k-exchange

We extend this notion to consider modified weakly k-synchronous executions as before, and the graphical characterization of this property is that there are at most k nodes in every SCC of the conflict graph. Hence, this class is also MSO-definable, and since each modified k-exchange has at most $2k$ events, it also has bounded special tree-width.

5.4.3 Strongly k-synchronous MSCs and other classes

Our framework can be applied to a variety of other classes. Here we show how the decidability results can be shown for a variant of the class of weakly k-synchronous MSCs.

Definition 5.23. *Let $M = (\mathcal{E}, \to, \lhd, \lambda) \in \mathsf{MSC}_{\mathsf{mb}}$. We call M* strongly k-synchronous *if it can be written as $M = M_1 \cdot \ldots \cdot M_n$ such that every MSC $M_i = (\mathcal{E}_i, \to_i, \lhd_i, \lambda_i)$ is a k-exchange and, for all $(e, f) \in \sqsubset_M$, there are $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$.*

strongly k-synchronous

Example 27. MSC $M_4 \in \mathsf{MSC}_{\mathsf{mb}}$ in Fig. 5.7 is strongly 1-synchronous. Indeed, we can decompose it into 1-exchanges and this decomposition allows for a total order compatible with $\sqsubset_{M_4}$. Moreover, MSC M_3 in Fig. 5.6, which is weakly 1-synchronous, is strongly 3-synchronous. Indeed, we need to put the three messages in the same k-exchange to regain our total order. Finally, for all k, MSC M_1 in Fig. 2.3 is not strongly k-synchronous, as we cannot put all messages in the same k-exchange, where all sends are followed by all receptions. Here, this is not possible as the reception of m_3 has to take place before the sending of m_4.

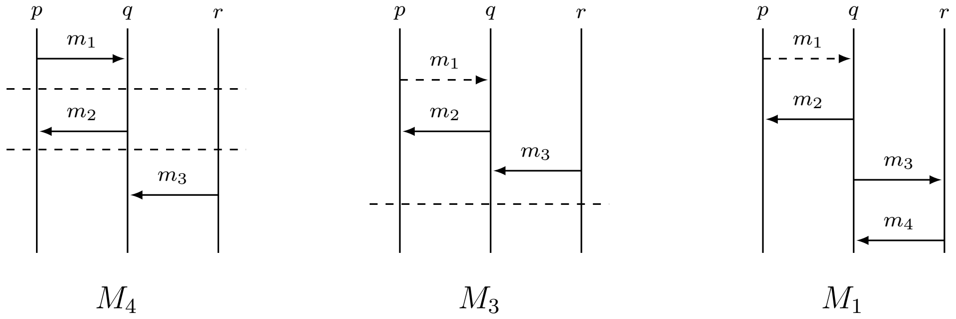


Figure 5.7: MSC M_4 is strongly 1-synchronizable, M_3 is strongly 3-synchronizable and M_1 is not strongly k-synchronizable for any k.

Similar to Theorem 5.18, we now show the graphical characterization of strong synchronizability.

Theorem 5.24. *Let $M \in \mathsf{MSC_{mb}}$ be an MSC. M is strongly k-synchronous iff every SCC in the extended conflict graph $\mathsf{ECG}(M)$ is of size at most k and no RS edge occurs on any cycle in $\mathsf{ECG}(M)$.*

Proof. Let us assume that we have an MSC M which is strongly k-synchronous. Hence, $M = M_1 \dots M_n$, where each M_i is a k-exchange. We denote the edges of the ECG by the dashed arrow ($\dashrightarrow$).

We prove, by contradiction, that the size of the largest SCC in $\mathsf{ECG}(M)$ is at most k. Let us assume, to the contrary, that there exists an SCC which is of size $k' > k$. As there at most k messages in each k-exchange, there exists vertices v, v' which belong to the SCC such that $v \in M_i$ and $v' \in M_j$, where $1 \leq i < j \leq n$. Since v, v' belong to the same SCC, we have $v \dashrightarrow^* v' \dashrightarrow^* v$. We prove by induction on the length of the path between v' and v that $v' \dashrightarrow v$ implies that $j \leq i$.

For the base case, we have $v' \overset{XY}{\dashrightarrow} v$.

- If $XY \neq SS$, then there is a path from v' to v in the conflict graph (because all the rules in Figure 5.3 except Rule 5 add edges between vertices which are already connected in the conflict graph). Therefore, some action of v' precedes an action of v, and hence, $j \leq i$.

- On the other hand, if $XY = SS$ and it is built by Rule 5 (all the other SS dependencies fall into the above case) then v' is matched whereas v is unmatched, and both v, v' involve sending messages to the same process. Hence, to obey mailbox semantics, the send of v' has to precede the send of v therefore $j \leq i$.

By induction hypothesis, we assume that if there is a path of length h from v' to v in the SCC, then $j \leq i$. Let us now show that it holds true if the length is $h+1$. Assume that we have $v' \dashrightarrow^{*} v'' \dashrightarrow v$, where there is a path of length h between v' and v''. Moreover, by the induction hypothesis, $v'' \in M_\ell$ where $j \leq \ell \leq n$. Once again, we can be in one of two cases.

- $v'' \overset{XY}{\dashrightarrow} v$, $XY \neq SS$. By the same argument as for the base case, $\ell \leq i$ and therefore, $j \leq i$.
- $v'' \overset{SS}{\dashrightarrow} v$. Then, v'' has to be sent before v and once again $j \leq \ell \leq i$.

Therefore, if the size of the SCC is larger than k, then all the vertices need to belong to the same k-exchange, which is a contradiction.

Next, we show that there is no RS edge in any SCC. If we assume to the contrary that there is an RS edge, i.e. $v \overset{RS}{\dashrightarrow} v' \dashrightarrow^{*} v$ in $\mathsf{ECG}(M)$, then using the same argument as above, v and v' have to belong to the same k-exchange. Moreover, $v \overset{RS}{\dashrightarrow} v'$ implies that the reception associated to v, say $f \in \mathcal{E}$ has to happen before the send action associated to v', say $e' \in \mathcal{E}$. However, in a strongly synchronous k-exchange, all the sends precede the receives and hence, we have a contradiction.

Conversely, assume that every SCC in $\mathsf{ECG}(M)$ is of size at most k and no RS edge occurs on any cyclic path. Then, we first show that every SCC in the extended conflict graph is k-synchronous. Let C be an SCC formed from a set of nodes $v_1, \ldots, v_m$, for some $1 \leq m \leq k$. Let us denote by e_i the send actions associated to vertex v_i for all $1 \leq i \leq m$. Because the MSC is mailbox, it satisfies causal delivery, which means that there is no SS cycle in $\mathsf{ECG}(M)$ [DGLL20]. Hence, let us assume an indexing of the vertices consistent with the edges labelled by SS, i.e. for every $1 \leq i_1 \leq i_2 \leq m$, C does not contain an edge $v_{i_2} \overset{SS}{\dashrightarrow} v_{i_1}$. We further index the nodes in such a way that the sends associated to each process are all consecutive, i.e. for every $1 \leq i < j < l \leq m$, if $e_i, e_l \in \mathit{Send}(p, \_)$, then $e_j \in \mathit{Send}(p, \_)$.

Let $i_1, \ldots, i_r$ be the maximal subsequence of $1, \ldots, m$ such that f_ℓ is the receive action associated to the send e_ℓ for all $\ell \in \{i_1, \ldots, i_r\}$. C is therefore the graph of the execution $E = e_1 \ldots e_m f_{i_1} \ldots f_{i_r}$. The fact that all sends can be executed before the receives is a consequence of the fact that C does not contain edges labelled by RS. Then, the order between receives is consistent with the one between sends because C satisfies causal delivery. Therefore, C is strongly k-synchronous.

Now, let $C_1, \dots, C_n$ be the set of maximal SCCs in $\mathsf{ECG}(M)$, listed in topological order. For each i, let E_i refer to the send and receive actions ordered as above. Let us consider $M' = E_1 \dots E_n$ to be the MSC associated with this ordering. We can see that $\mathsf{ECG}(M) = \mathsf{ECG}(M')$. Now, let us show that M' satisfies the mailbox property. Let us assume to the contrary that M' does not satisfy the mailbox property. Then, there exists indices $i < j$ such that there is an unmatched send e_i in E_i and a matched send e_j in E_j such that $e_i, e_j \in \mathit{Send}(p, \_)$ for some $p \in \mathbb{P}$. Then, there is an edge $v_j \overset{SS}{\dashrightarrow} v_i$, where v_i, v_j are the vertices associated to e_i, e_j respectively. But since $i < j$, there is already a path $v \dashrightarrow^* v'$ for some vertices $v \in C_i$ and $v' \in C_j$. Hence, C_i and C_j are the same SCC, which is a contradiction.

□

Therefore, M' satisfies the mailbox property.

Proposition 5.25. *For all $k \in \mathbb{N}$, the set of strongly k-synchronous mailbox MSCs is MSO-definable and STW-bounded.*

Proof. **MSO definability.** We first express the extended relations between events in MSO:

$$\begin{aligned}
ERR(e_1, e_2) &= RR(e_1, e_2)\\
ERS(e_1, e_2) &= RS(e_1, e_2)\\
ESR(e_1, e_2) &= SR(e_1, e_2) \ \vee \ [\mathit{matched}(e_1) \wedge (e_1 = e_2)]\\
ESS(e_1, e_2) &= SS(e_1, e_2) \ \vee \ RR(e_1, e_2) \ \vee\\
&\left[\mathit{matched}(e_1) \wedge \neg \mathit{matched}(e_2) \wedge \bigvee_{\substack{q \in \mathbb{P}\\ a,b \in \mathit{Send}(\_,q)}} \lambda(e_1) = a \ \wedge \ \lambda(e_2) = b\right]
\end{aligned}$$

And we define Rule 4 in Figure 5.3 as follows:

$$EXZ_{\text{closed}}(e_1, e_2) \ = \ EXZ(e_1, e_2) \vee \left[\exists e_3. \bigvee_{Y \in \{R,S\}} (EXY_{\text{closed}}(e_1, e_3) \wedge EYZ_{\text{closed}}(e_3, e_2))\right]$$

for all $X, Z \in \{R, S\}$. The MSO formula proceeds similarly to what has been shown in the previous case, but now relies on the extended conflict graph.

$$ECG(e_1, e_2) \ = \bigvee_{X,Y \in \{R,S\}} EXY_{\text{closed}}(e_1, e_2)$$

The formula for the property that there is no strongly connected component of size greater than k in the extended conflict graph can be expressed as follows:

$$\nexists e_1, \ldots, e_{k+1}.\ [\bigwedge_{i \neq j} ECG^*(e_i, e_j)]$$

And finally, we express the property that there is no RS edge in any cycle of the extended conflict graph:

$$\nexists e_1, e_2.[ECG^*(e_1, e_2) \wedge ERS_{\text{closed}}(e_2, e_1)]$$

STW-bound. For the condition of bounded STW, it is sufficient to see that the set of strongly k-synchronizable MSCs is included in the set of weakly k-synchronizable MSCs. Hence, Eve can use the same decomposition strategy as in Proposition 5.13 to obtain a bound on the special tree-width. □

As a corollary, we thus obtain:

Theorem 5.26. *The following problem is decidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, and $k \in \mathbb{N}$, is every MSC in $L_{\mathsf{mb}}(\mathcal{S})$ strongly k-synchronous?*

Only mailbox MSCs are considered for the definition of strongly k-synchronous MSCs for the following reason: A natural p2p analogue of Definition 5.23 would require from the decomposition that, for all $(e, f) \in \leq_M$, there are indices $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$. But this is always satisfied. So the natural definition of "strongly k-synchronous MSCs" would coincide with weakly k-synchronous MSCs.

Proposition 5.27. *Consider a p2p MSC $M = M_1 \ldots M_n$ such that every MSC $M_i = (\mathcal{E}_i, \rightarrow_i, \lhd_i, \lambda_i)$ is a k-exchange. Then, for all $(e, f) \in \leq_M$, there are $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$.*

Proof. If $e \leq_M f$, then there is a sequence of events $e_0 \bowtie_1 e_1 \bowtie_2 \ldots \bowtie_m e_m$ where $e_0 = e$ and $e_m = f$ and $\bowtie_i \in \{\rightarrow, \lhd\}$ for all $1 \leq i \leq m$. For every $0 \leq \ell < m$, there are indices $1 \leq i \leq j \leq n$ such that $e_\ell \in \mathcal{E}_i$ and $e_{\ell+1} \in \mathcal{E}_j$. Hence, by transitivity, the proposition is proved. □

Like the variant for the case of weakly synchronous MSCs, we can also generalize strongly k-synchronous MSCs by removing the restriction on the number of messages per exchange:

strongly synchronous

Definition 5.28. *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC_{mb}}$. We call M* strongly synchronous *if it can be written as $M = M_1 \cdot \ldots \cdot M_n$ such that every MSC $M_i = (\mathcal{E}_i, \rightarrow_i, \lhd_i, \lambda_i)$ is an exchange and, for all $(e, f) \in \sqsubset_M$, there are indices $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$.*

Similarly to the constructions for strongly k-synchronous MSCs, we can obtain a graphical characterization where we only look for the absence of RS-edges in a cycle. Hence, this class is also MSO-definable (in fact, even LCPDL-definable) and STW-bounded.

5.4.4 Existentially k-p2p-bounded MSCs

Definition 5.29. *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC}$ and $k \in \mathbb{N}$. A linearization $\rightsquigarrow$ of M is called* k-p2p-bounded *if, for all $e \in Matched(M)$, with $\lambda(e) = \langle (p, q)!m \rangle$, $\#_{Send(p,q)}(\rightsquigarrow, e) - \#_{Rec(p,q)}(\rightsquigarrow, e) \leq k$,*

existentially k-p2p-bounded

We call M *existentially k-p2p-bounded* if it has some *p2p* linearization that is k-p2p-bounded.

Proposition 5.30. *For all $k \in \mathbb{N}$, the set of existentially k-p2p-bounded MSCs is MSO-definable and STW-bounded.*

Proof. The set of existentially k-p2p-bounded MSCs was shown to be MSO-definable (in fact, even FO-definable) in [LM04]. Note that there are minor differences in the definitions (in particular, the fact that we deal with unmatched messages), which, however, do not affect FO-definability. In [BG21], it was shown that their special tree-width is bounded. □

Theorem 5.31. *For $\mathrm{com} \in \{\mathsf{p2p}, \mathsf{mb}\}$, the following problem is decidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, and $k \in \mathbb{N}$, is every MSC in $L_{\mathrm{com}}(\mathcal{S})$ existentially k-p2p-bounded?*

Proof. Again, the proof follows exactly the same lines as that or Theorem 5.21, now using Proposition 5.30. □

Note that this is similar to the problem considered in [GKM07, KM21], though there is a subtle difference: in [GKM07, KM21], there are a notion of deadlock and distinguished final states.

5.4.5 Existentially k-bounded MSCs

Now, we turn to existentially k-bounded MSCs [LM02, GMK04, GKM07]. A linearization $\rightsquigarrow$ of an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC}$ is called *k-mailbox-bounded* if, for all $e \in \mathit{Matched}(M)$, say with $\lambda(e) = \langle (p,q)!m \rangle$, we have $\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e) \leq k$.

Definition 5.32. *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC}$ and $k \in \mathbb{N}$. We call M* existentially k-mailbox-bounded *if it has some mailbox linearization that is k-mailbox-bounded.*

existentially k-mailbox-bounded

Note that every existentially k-mailbox-bounded MSC is a mailbox MSC.

Example 28. MSC M_5 in Fig. 5.8 is existentially 1-mailbox-bounded, as witnessed by the (informally given) linearization $\langle (q,p)!m_2 \rangle \rightsquigarrow \langle (p,q)!m_1 \rangle \rightsquigarrow \langle (q,r)!m_3 \rangle \rightsquigarrow \langle (q,r)?m_3 \rangle \rightsquigarrow \langle (p,q)?m_1 \rangle \rightsquigarrow \langle (p,q)!m_1 \rangle \rightsquigarrow \langle (q,p)?m_2 \rangle \rightsquigarrow \langle (q,r)!m_3 \rangle \ldots$ Note that M_5 is neither weakly nor strongly synchronous as we cannot divide it into exchanges.

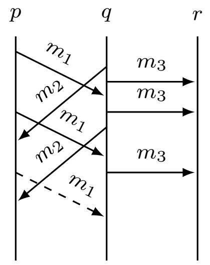


Figure 5.8: MSC M_5

Let $k \geq 1$, and let M be a fixed mailbox MSC. Let $\xrightarrow{\text{rev}}_k$ be the binary relation among events of M defined as follows: $f \xrightarrow{\text{rev}}_k e$ if

$\xrightarrow{\mathit{rev}}_k$

1. f is a receive event of a process p;
2. let f' be the k-th receive event of process p after f; then $e \lhd f'$.

Lemma 5.33. *M is existentially k-mailbox-bounded if and only if $\preceq_M \cup \xrightarrow{\mathit{rev}}_k$ is acyclic.*

Proof. Assume that M is existentially k-mailbox-bounded. Let $\rightsquigarrow$ be a mailbox linearization of M such that for all $e \in \mathit{Matched}(M)$, with $\lambda(e) = \langle (p,q)!m \rangle$,

$$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e) \leq k \ .$$

Then $\rightsquigarrow$ is also a linearization of $(\preceq_M \cup \xrightarrow{\text{rev}}_k)^*$. Indeed, if it was not the case, there would be a pair of events e', f' such that $f' \xrightarrow{\text{rev}}_k e'$ and $e' \rightsquigarrow f'$. But then we would have

$$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e') - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e') > k \ ,$$

which is a contradiction. So $\rightsquigarrow$ is a linearization of $(\preceq_M \cup \xrightarrow{\text{rev}}_k)^*$ and $\preceq_M \cup \xrightarrow{\text{rev}}_k$ is acyclic.

Conversely, assume that $\preceq_M \cup \xrightarrow{\text{rev}}_k$ is acyclic. Let $\rightsquigarrow$ be a linearization of $(\preceq_M \cup \xrightarrow{\text{rev}}_k)^*$. In particular, $\rightsquigarrow$ is a mailbox linearization of M. Let us show that for all $e \in \mathit{Matched}(M)$, with $\lambda(e) = \langle (p,q)!m \rangle$,

$$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e) \leq k \ .$$

Let $e \in \mathit{Matched}(M)$ be fixed, and let f' be such that $e \lhd f'$. There are two cases:

- $\#_{\mathit{Rec}(\_,q)}(\rightarrow, f') \leq k$. Then

  $$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) \leq k \, ,$$

  because all sends to the channel of process q before e are matched. So

  $$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e) \leq k \, ,$$

- $\#_{\mathit{Rec}(\_,q)}(\rightarrow, f') > k$. Then there is f on process q such that $f \xrightarrow{\text{rev}}_k e$. Since $\preceq_M \cup \xrightarrow{\text{rev}}_k$ is acyclic, we have $f \rightsquigarrow e$, and there are at most k messages in the channel of q at the time of event e, or in other words,

  $$\#_{\mathit{Send}(\_,q)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q)}(\rightsquigarrow, e) \leq k \ .$$

So $\rightsquigarrow$ is a mailbox linearization with k bounded buffers, and M is existential k-mailbox-bounded. □

Proposition 5.34. *For all $k \in \mathbb{N}$, the set of existentially k-mailbox-bounded MSCs is MSO-definable and STW-bounded.*

Proof. Let $k \geq 1$ be fixed. Since every existentially k-mailbox-bounded MSCs is also existentially k-p2p-bounded, and since the class of existentially k-p2p-bounded MSCs is STW bounded (see Proposition 5.30), the class of existentially k-mailbox-bounded MSCs is also STW bounded.

Let us show that it is moreover MSO definable.

By Lemma 5.33, it is enough to show that the acyclicity of $\preceq_M \cup \xrightarrow{\text{rev}}_k$ is MSO definable, and since $\preceq_M$ was already shown MSO definable and acyclicity is easily

MSO definable, it is enough to show that $\xrightarrow{\text{rev}}_k$ is MSO definable. It is indeed the case, as demonstrated by this formula

$$\phi(r,s) = \exists r_1, r_2, \ldots, r_n . r \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n \wedge s \lhd r_n .$$

Finally, let us show that existentially k-mailbox-bounded is also LCPDL definable. This follows from the following formulas:

$$\begin{aligned}
\prec_M = &\ (\lhd + \rightarrow)^+ \\
R = &\ \langle \lhd^{-1} \rangle \top \\
\xrightarrow{\text{next R}} = &\ (\rightarrow \wedge \mathsf{test}(\neg R))^* \cdot (\rightarrow \wedge \mathsf{test}(R)) \\
\xrightarrow{\text{rev}}_k = &\ (\xrightarrow{\text{next R}})^k . (\lhd)^{-1} . \\
\Phi_{\exists k \text{ mb-bounded}} = &\ \neg \mathsf{ELoop}\langle (\prec_M + \xrightarrow{\text{rev}}_k)^+ \rangle
\end{aligned}$$

□

5.5 Discussion

This chapter presents a unifying framework in order to decide synchronizability among other interesting properties. It is based on MSO (and LCPDL) logic, and the notion of bounded (special) tree-width.

Table 5.1 gives a summary of the results. For a detailed comparison between these classes, refer to [BDGF+21, Lav21].

Table 5.1: Summary of the decidability of the synchronizability problem in various classes

	Peer-to-Peer	Mailbox
Weakly synchronous	Undecidable [Thm. 5.15]	EXPTIME [Thm. 5.14]
Weakly k-synchronous	Decidable [BEJQ18, DGLL20] and [Thm. 5.21]	
Strongly k-synchronous	—	Decidable [Thm. 5.26]
Existentially k-p2p-bounded	Decidable [GKM07, BG21]	
Existentially k-mailbox-bounded	—	Decidable [Prop. 5.34]

Let us look at some of the other related problems in this field. This extension is also valid for the *p2p* definition of existentially k-bounded MSCs, which were addressed in [GKM07]. Moreover, our framework can also be adapted to treat universally bounded systems [HMK$^+$05, LM02]. However, the send-synchronizability problem from [BB11] does not fit in our framework. This is clear, because the inclusion question $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_0$ would be decidable, where $\mathcal{C}_0$ is the set of send-synchronizable ($p2p$) MSCs, but this property is undecidable (as checking whether a given system $\mathcal{S}$ is send-synchronizable is undecidable, from [FL17]).

Note that there is a subtle difference between the notion of existential boundedness as studied in this work, compared to some other works that have been studied, for example in [GKM07] as we do not take into account the unmatched messages.

Some directions we could explore further would be regarding the decidability of the question whether there exists a $k \geq 0$ such that $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_k$ allows us to build a bounded model checking strategy by first deciding whether there exists such a $k \geq 0$ and then by testing if $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_k$ for $k = 0, 1, 2, \ldots$. One may use this strategy for weakly/strongly synchronizable systems, but not for existentially bounded systems (except for deadlock-free systems) or for deterministic deadlock-free universally bounded systems. In [LY19], the authors introduced an asynchronous compatibility property and it would also be interesting to verify whether this property could be expressed into our framework. Finally, we could also explore the notion of synchronizability over other communication architectures, apart from $p2p$ and mailbox, and see whether the resulting classes could fit into our framework.

Chapter 6

Send-Synchronizability

The results in this chapter are joint work with Benedikt Bollig, Alain Finkel and S. Krishna, unpublished.

This chapter further investigates the notion of send-synchronizability for peer-to-peer systems. As we saw earlier, the decidability of send-synchronizability was shown to be undecidable for peer-to-peer systems in [FL17]. However, apart from the trivial properties obtained from the definition, the decidability of global properties such as reachability, boundedness, etc. are not known. With regards to choreography realizability, it has been shown in [SAB20] that synchronizability is decidable for choreography-defined peer-to-peer systems.

We recall the two notions of send-synchronizability defined in the literature, which we shall refer to as $\mathcal{I}$-synchronizability and $\mathcal{J}$-synchronizability (which has also been referred to as language-synchronizability in the literature). We go on to investigate the reachability problem, which we show is undecidable for both classes.

Finally, we propose some variants of problems which are decidable for send-synchronizable systems, and investigate some subclasses of these systems. We conclude with some directions of future work.

Note that as we only consider peer-to-peer systems in this chapter, we will assume that the mode of communication is *p2p* unless otherwise specified.

6.1 $\mathcal{I}$ and $\mathcal{J}$-synchronizability

Let us recall from Section 2.2 that for a communicating system, a trace $\tau \in \mathit{Act}^*$ is a finite sequence of actions. Furthermore, we let $\mathit{proj}_! : \mathit{Act}^* \to \Sigma^*$ be the homomorphism defined by $\mathit{proj}_!(\langle c!m \rangle) = m$ for all $m \in \Sigma$ and $c \in \mathit{Ch}$, and $\mathit{proj}_!(\alpha) = \varepsilon$ if $\alpha \neq \langle c!m \rangle$ for some $m \in \Sigma$ and $c \in \mathit{Ch}$.

Let $\mathcal{S} = (X, Act_{,} \to, init)$ be the transition system associated with a CFM. Then, for any $k \geq 0$, we define:

$$\mathcal{J}_k(\mathcal{S}) = \{proj_!(\tau) \mid \tau \in Traces_k(\mathcal{S})\}$$
$$\mathcal{I}_k(\mathcal{S}) = \mathcal{J}_k(\mathcal{S}) \cup \{(proj_!(\tau), \gamma) \mid init \xrightarrow{\tau} \gamma,\ \gamma \in X \text{ and } \gamma \text{ is stable},\ \tau \in Traces_k(\mathcal{S})\}$$

$\mathcal{J}(\mathcal{S})$, $\mathcal{I}(\mathcal{S})$

Furthermore,

$$\mathcal{J}(\mathcal{S}) = \bigcup_{k \geq 0} \mathcal{J}_k(\mathcal{S})$$
$$\mathcal{I}(\mathcal{S}) = \bigcup_{k \geq 0} \mathcal{I}_k(\mathcal{S})$$

We remark that $\mathcal{J}_k \subseteq \mathcal{I}_k$, $\mathcal{I}_k \subseteq \mathcal{I}_{k+1}$, and $\mathcal{J}_k \subseteq \mathcal{J}_{k+1}$.

Following [BBO12b], we define the observable behaviour of a system as its set of send traces enriched with their final states when they are stable. In [BBO12b], the authors consider the slack elasticity of a system: the property that the asynchronous distributed behaviour behaves “equivalently” regardless of the slack of the channels. Hence, synchronizability is defined as the slack elasticity of these observable behaviours.

send-synchronizability ($\mathcal{I}$ and $\mathcal{J}$-synchronizability)

Definition 6.1 (Synchronizability [FL17, BBO12b]). *We say that a system $\mathcal{S}$ is*

- $\mathcal{J}$-synchronizable *if $\mathcal{J}(\mathcal{S}) = \mathcal{J}_0(\mathcal{S})$,*
- $\mathcal{I}$-synchronizable *if $\mathcal{I}(\mathcal{S}) = \mathcal{I}_0(\mathcal{S})$.*

Let us remark that $\mathcal{S}$ is $\mathcal{I}$-synchronizable (resp. $\mathcal{J}$-synchronizable) if and only if $\mathcal{I}_k = \mathcal{I}_0$ (resp. $\mathcal{J}_k = \mathcal{J}_0$) for all $k \geq 0$. Moreover, for all $k, \ell \geq 0$, we have that $\mathcal{I}_k = \mathcal{I}_\ell$ implies $\mathcal{J}_k = \mathcal{J}_\ell$. Let us prove this:

Proposition 6.2. *If a system is $\mathcal{I}$-synchronizable, then it is $\mathcal{J}$-synchronizable.*

Proof. If $\mathcal{S}$ is $\mathcal{I}$-synchronizable, then $\mathcal{I}(\mathcal{S}) = \mathcal{I}_0(\mathcal{S})$, i.e. $\mathcal{I}(\mathcal{S}) \setminus \mathcal{I}_0(\mathcal{S}) = \emptyset$. Since $\mathcal{J}(\mathcal{S}) \subseteq \mathcal{I}(\mathcal{S})$, we have $\mathcal{J}(\mathcal{S}) \setminus \mathcal{I}_0(\mathcal{S}) = \emptyset$. Moreover, $\mathcal{I}_0(\mathcal{S}) = \mathcal{J}_0(\mathcal{S}) \cup \{(proj_!(\tau), \gamma) \mid init \xrightarrow{\tau} \gamma, \tau \in Traces_0(\mathcal{S})\}$. Hence, $\mathcal{J}(\mathcal{S}) \setminus \mathcal{I}_0(\mathcal{S}) = \mathcal{J}(\mathcal{S}) \setminus \mathcal{J}_0(\mathcal{S}) = \emptyset$. Therefore, $\mathcal{J}(\mathcal{S}) = \mathcal{J}_0(\mathcal{S})$, and $\mathcal{S}$ is $\mathcal{J}$-synchronizable. □

The converse, however, is false in general and there exist $\mathcal{J}$-synchronizable systems that are not $\mathcal{I}$-synchronizable (see Example 29).

Example 29. Consider the CFM $\mathcal{A}$ with two processes as shown in Figure 6.1. We see that $\mathcal{S}_{\mathcal{A}}$ is $\mathcal{J}$-synchronizable, because for every trace, there is the equivalent synchronous trace where both processes loop in the initial control-state. However,

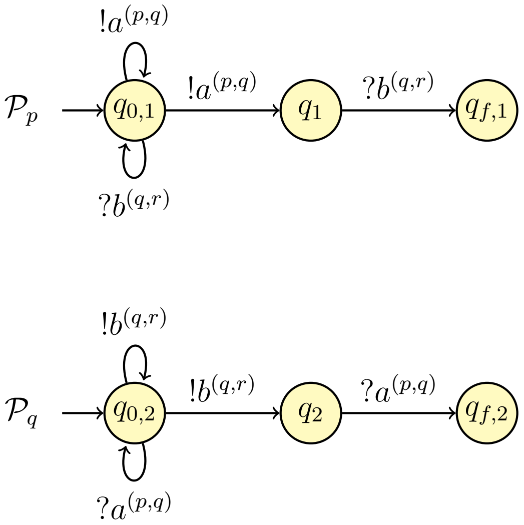


Figure 6.1: CFM $\mathcal{A} = (\mathcal{P}_p, \mathcal{P}_q, \Sigma)$ which is $\mathcal{J}$-synchronizable but not $\mathcal{I}$-synchronizable.

the state $(q_{f,1}, q_{f,2}, \varepsilon, \varepsilon)$, which is reachable via $(q_{0,1}, q_{0,2}, \varepsilon, \varepsilon) \xrightarrow{\sigma} (q_{f,1}, q_{f,2}, \varepsilon, \varepsilon)$, where $\sigma = !a^{(p,q)}.!b^{(q,r)}.?a^{(p,q)}.?b^{(q,r)}$. However, as we can see, $(q_{f,1}, q_{f,2}, \varepsilon, \varepsilon)$ is not reachable via any synchronous trace. Hence, the system $\mathcal{S}_\mathcal{A}$ is $\mathcal{J}$-synchronizable but not $\mathcal{I}$-synchronizable.

Also note that for $k \in \mathbb{N}$, $\mathcal{I}_k(\mathcal{S}) = \mathcal{I}_{k+1}(\mathcal{S})$ does not imply $\mathcal{I}(\mathcal{S}) = \mathcal{I}_k(\mathcal{S})$ (respectively for $\mathcal{J}(\mathcal{S})$).

Theorem 6.3 ([FL17]). *We have the following decidability results:*

- *$\mathcal{I}$-synchronizability and $\mathcal{J}$-synchronizability are undecidable for systems of three p2p.*
- *$\mathcal{I}$-synchronizability and $\mathcal{J}$-synchronizability are decidable for systems of two p2p.*

However, for such systems, there has been no study on the decidability of reachability, boundedness, or the other verification problems that we study in this thesis. This leads us to first study the problem of reachability.

6.2 Reachability for send-synchronizable systems

We first analyse the reachability problem for $\mathcal{J}$-synchronizable systems.

Theorem 6.4. *The reachability problem is undecidable for $\mathcal{J}$-synchronizable systems of (two) CFMs.*

Proof. We reduce reachability in systems of two CFMs (that is well-known to be undecidable [BZ83]) to reachability in $\mathcal{J}$-synchronizable systems of two CFMs.

Let us consider any CFM $\mathcal{A} = (\mathcal{P}_p, \mathcal{P}_q, \Sigma)$ with two peers, where $\mathcal{P}_i = (Q_i, \Sigma_i, \delta_i, \ell_{0_i})$ for $i \in \{p, q\}$. We add a new control-state ℓ_i to each $\mathcal{P}_i$ with self loops to send and receive every letter of the message alphabet, such that ℓ_i is only reachable from the initial control-state ℓ_{0_i} via every possible transition. Let us call this modified system $\mathcal{A}' = (\mathcal{P}'_p, \mathcal{P}'_q, \Sigma)$, and we can see it in Figure 6.2.

The transition system $\mathcal{S}'$ associated to $\mathcal{A}'$ is $\mathcal{J}$-synchronizable. This is because, for any trace $\tau \in \mathit{Traces}(\mathcal{S}')$, there exists $\nu \in \mathit{Traces}(\mathcal{S}')$ such that $(\ell_{0_p}, \ell_{0_q}, \epsilon, \epsilon) \xrightarrow{\epsilon} (\ell_p, \ell_q, \epsilon, \epsilon) \xrightarrow{\nu} (\ell_p, \ell_q, \epsilon, \epsilon)$ where $\mathit{proj}_!(\tau) = \mathit{proj}_!(\nu)$ and ν is of the form $!?a_1!?a_2\ldots$ for messages $m_1, m_2, \ldots \in \Sigma$. Therefore, we can deduce $\nu \in \mathcal{J}_0(\mathcal{S}')$ and hence, $\tau \in \mathcal{J}_0(\mathcal{S}')$. Note that $\mathcal{J}(\mathcal{S}') = \mathit{Act}^*$, hence $\mathcal{J}(\mathcal{S}) \subseteq \mathcal{J}(\mathcal{S}')$ but it is not necessary that $\mathcal{J}(\mathcal{S}) = \mathcal{J}(\mathcal{S}')$.

Furthermore, we see that if a state γ is reachable in $\mathcal{S}$, then γ is reachable in $\mathcal{S}'$. And conversely, any state involving a control-state of $\mathcal{S}$ (in $Q_p \times Q_q$), that is reachable in $\mathcal{S}'$, is also reachable in $\mathcal{S}$ (note that if we reach, in $\mathcal{S}'$, the control-state ℓ_p or ℓ_q, we cannot return to the original state space of $\mathcal{S}$ any more).

Thus, if reachability would be decidable for $\mathcal{S}'$, we could decide reachability for $\mathcal{S}$, i.e. for any system of communicating machines, and this is not the case. Hence the reachability problem is undecidable for $\mathcal{J}$-synchronizable systems of (at least two) CFMs.

□

Using a similar idea, we can also prove that the control-state reachability problem is undecidable for $\mathcal{J}$- synchronizable systems. Moreover, reachability is also undecidable for $\mathcal{J}$- synchronizable counter systems by a similar construction. However, for boundedness and termination we cannot use the same argument, as addition of the extra control-state in each peer results in a potentially unbounded (resp. non-terminating) system.

Reachability for $\mathcal{I}$-synchronizable systems

In order to show that reachability is also undecidable for $\mathcal{I}$-synchronizable systems, we use the construction from [FL17]. However, since the underlying problem

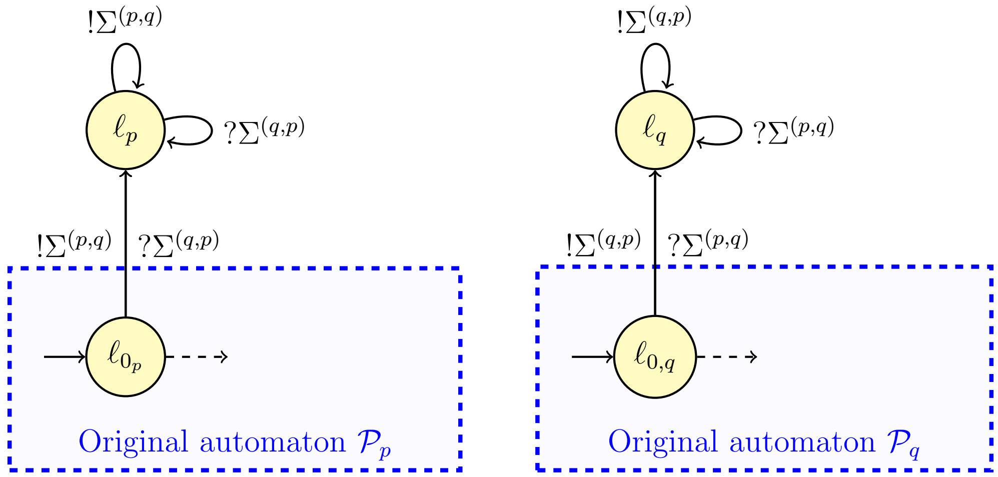


Figure 6.2: Construction of the modified system $\mathcal{A}'$, where the send/reception of every message is abbreviated by the symbol $!\Sigma$ and $?\Sigma$

we study is different, we make modifications on the original construction. The differences between the two will be discussed below.

The general idea of the construction is to reduce the reachability problem for $\mathcal{I}$-synchronizable systems to another (well-known) undecidable problem. In [FL17], the authors reduced the reachability problem to the message reception problem for a FIFO machine (with a single channel). Since we are studying the reachability problem for send-synchronizable systems, we instead consider the well-known undecidable reachability problem for FIFO machines.

First, we recall the notion of "good-for-reduction" FIFO machines from [FL17].

Definition 6.5. *A FIFO machine $\mathcal{M}$ is good-for-reduction if the only stable trace of $\mathcal{S}_{\mathcal{M}}$ is the empty trace.* *good-for-reduction*

Now, we adapt the well-known result from [BZ83] to good-for-reduction FIFO machines.

Theorem 6.6. *Reachability is undecidable for good-for-reduction FIFO machines.*

Proof sketch. We show that good-for-reduction FIFO machines may simulate the run of a Turing machine. This simulation is now sketched. Given a Turing machine TM, there exists a good-for-reduction FIFO machine $\mathcal{M}$ (with a single channel) such that any run of TM of length n can be simulated by a run of $\mathcal{M}$ of length $O(n^2)$. Intuitively, this simulation consists simply in having the entire tape of the simulated Turing machine in the channel with added end markers. To simulate a run of TM, the contents of the channel can be rotated, i.e. entirely read and immediately rewritten in the channel, with one exception, the part of the content representing the head of the Turing machine. This is modified to simulate a step of the Turing machine computation. The resulting FIFO automaton is indeed good-for-reduction since at all points of the execution, at the very minimum, at least one of the end markers will be left in the channel. □

To now prove the undecidability of reachability of $\mathcal{I}$-synchronizable systems, we use the construction from [FL17]. Let us recall this construction. Given a (good-for-reduction) FIFO machine $\mathcal{M}$, the authors first construct a CFM $\mathcal{A}_{\mathcal{M}}$ that simulates $\mathcal{M}$. Then, they complete the system in order to make it $\mathcal{I}$-synchronizable. The system $\mathcal{A}_{\mathcal{M}}$ consists of three processes p, q and r.

Intuitively, process p imitates the actions of $\mathcal{M}$, and the channel (p, q) represents the channel of $\mathcal{M}$. When $\mathcal{M}$ enqueues a letter m, process p sends a letter m to process q. When $\mathcal{M}$ dequeues a letter m, process p sends an order to dequeue to process r, which process r then forwards to process q. Then, process q executes the dequeueing, and then sends an acknowledgment back to process r, which process r then forwards to process p. In other words, process r acts as an intermediary for the dequeueing process.

Formally, let $\mathcal{M} = (Q, Ch, \Sigma, T, \ell_0)$ be a good-for-reduction FIFO machine. As $|Ch| = 1$, we will omit specifying the channel henceforth.

For the CFM $\mathcal{A}_{\mathcal{M}}$, for each channel (p, q), we define $\Sigma_{(p,q)} = \{m^{(p,q)} \mid m \in \Sigma\}$. Moreover, since the alphabet set is disjoint for each channel, we omit specifying the channel henceforth. In other words, $\langle (p, q)!m\rangle$ will now be written as $!m^{(p,q)}$.

The transition system associated to process p which we call $\mathcal{P}_p$ is obtained by replacing every $!m$-transition of $\mathcal{M}$ with $!m^{(p,r)}$, and every $?m$-transition with the sequence of two transitions $!m^{(p,r)}?m^{(r,p)}$. Formally, $\mathcal{P}_p = (Q_p, \Sigma_p, \Delta_p, \ell_{0_p})$, where

$$\begin{aligned} Q_p &= Q \uplus \{\ell_\delta \mid \delta \in T\} \text{ and} \\ \Delta_p &= \{(\ell, !m^{(p,q)}, \ell') \mid (\ell, !m, \ell' \in T)\} \\ &\cup \{(\ell, !m^{(p,r)}, \ell_\delta), (\ell_\delta, ?m^{(r,p)}, \ell') \mid \delta = (\ell, ?m, \ell') \in T\}. \end{aligned}$$

Process q receives messages from process r when it is time to dequeue, executes it and then sends a confirmation back to r. Formally, $\mathcal{P}_q = (Q_q, \Sigma_q, \Delta_q, \ell_{0_q})$ where

$$\begin{aligned} Q_q &= \{\ell_{0_q}, \ell_{1_q}\} \cup \{\ell_{m,1}, \ell_{m,2} \mid m \in \Sigma\} \\ \Delta_q &= \{(\ell_{0_q}, ?m^{(r,q)}, \ell_{m,1}), (\ell_{1_q}, ?m^{(r,q)}, \ell_{m,1}), \\ &\quad (\ell_{m,1}, ?m^{(p,q)}, \ell_{m,2}), (\ell_{m,2}, !m^{(q,r)}, \ell_{1_q}) \mid m \in \Sigma\} \end{aligned}$$

Note that the loop of $\mathcal{P}_q$ is unrolled once. This is because we need to ensure that it never comes back to the initial state (as our final system is the union of two such processes).

Process r receives a message to dequeue from process p and forwards it to process q. Once it receives an acknowledgment from process q, it forwards it to process p. The associated transition system is defined as $\mathcal{P}_r = (Q_r, \Sigma_r, \Delta_r, \ell_{0_r})$ where

$$\begin{aligned} Q_r &= \{\ell_{0_r}\} \cup \{\ell_{m,1}, \ell_{m,2}, \ell_{m,3} \mid m \in \Sigma\} \\ \Delta_r &= \{(\ell_{0_r}, ?m^{(p,r)}, \ell_{m,1}), (\ell_{m,1}, !m^{(r,q)}, \ell_{m,2}), \\ &\quad (\ell_{m,2}, ?m^{(q,r)}, \ell_{m,3}), (\ell_{m,3}, !m^{(r,p)}, \ell_{0_r}) \mid m \in \Sigma\} \end{aligned}$$

Example 30. Consider the following example from [FL17]. We have $\Sigma = \{a, m\}$ and the FIFO machine $\mathcal{M}_1 = (\{\ell_0, \ell_1\}, Ch, \Sigma, T, \ell_0)$, with transition relation $T = \{(\ell_0, !a, \ell_0), (\ell_0, !m, \ell_1), (\ell_1, ?m, \ell_0), (\ell_1, ?a, \ell_0)\}$. The CFM associated to processes p, q and r are shown in Figure 6.3.

Let $\mathbb{P} = \{p, q, r\}$, and $Ch_{\mathcal{A}} = \{(p,q) \mid p, q \in \mathbb{P}\}$, and $\Sigma_{\mathcal{A}} = \bigcup_{c \in Ch_{\mathcal{A}}} \Sigma_c$. Then, $\mathcal{A}_{\mathcal{M}} = (\mathcal{P}_p, \mathcal{P}_q, \mathcal{P}_r, \Sigma_{\mathcal{A}})$. It was shown in [FL17] that there is a tight correspondence between the k-bounded traces of $\mathcal{S}_{\mathcal{M}}$ for $k \geq 1$, and the k-bounded traces of $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$: every trace $\tau \in \mathit{Traces}_k(\mathcal{S}_{\mathcal{M}})$ induces the trace $h(\tau) \in \mathit{Traces}_k(\mathcal{S}_{\mathcal{A}_{\mathcal{M}}})$ where $h : \mathit{Act}_{\mathcal{M}} \to \mathit{Act}^*_{\mathcal{A}_{\mathcal{M}}}$ is the homomorphism from the traces of $\mathcal{S}_{\mathcal{M}}$ to the traces of $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$, defined by

$$h(!m) = !m^{(p,q)}$$

and

$$h(?m) = !?m^{(p,r)}.!?m^{(r,q)}.?m^{(p,q)}.!?m^{(q,r)}.!?m^{(r,p)}.$$

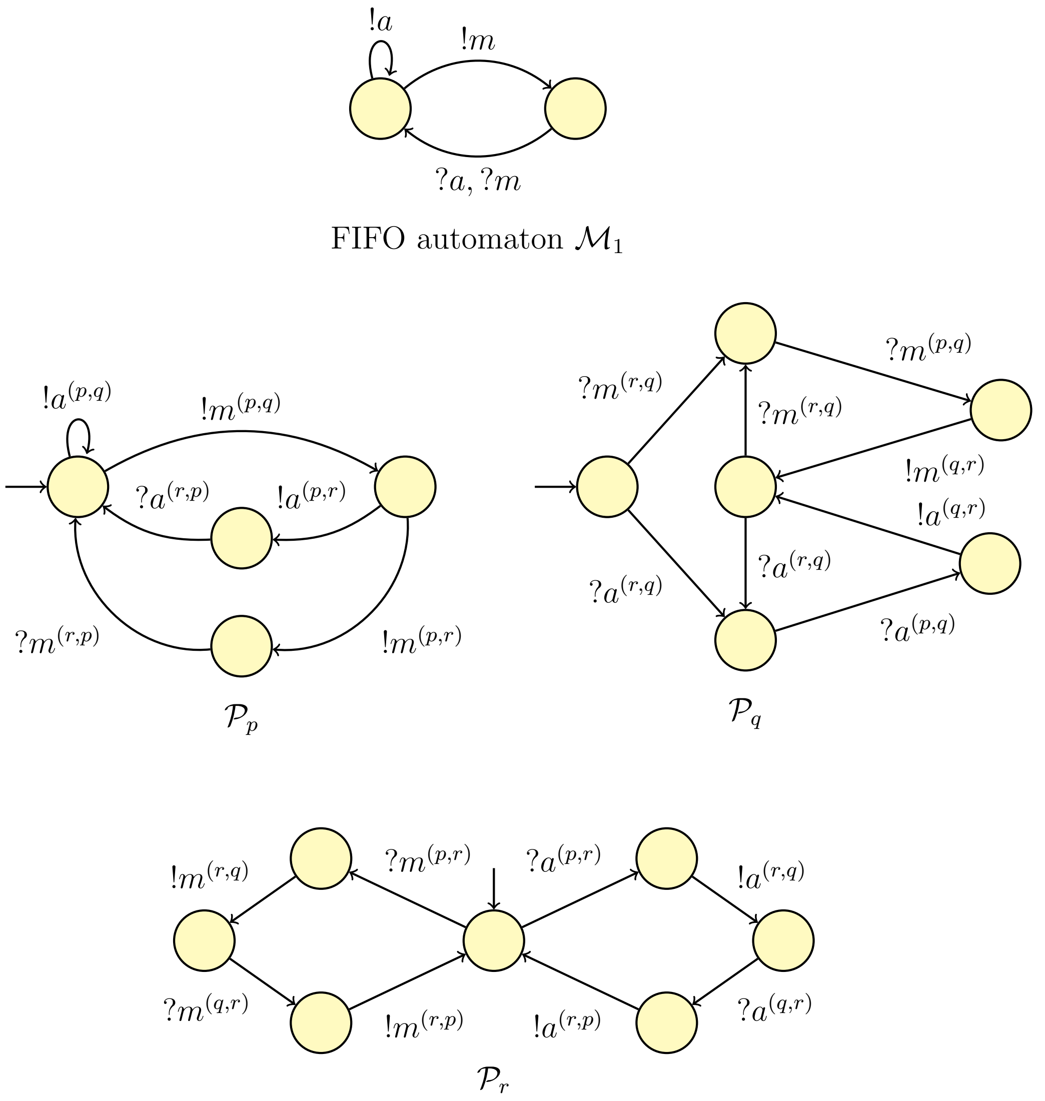
Now, we extend this idea to the sets of reachable states of $\mathcal{S}_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$. Let us denote a particular state of $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$ as $(\ell, \overline{w})$, where $\overline{w}_{(p,q)}$ denotes the channel content of (p,q).

Proposition 6.7. *For any execution $(\ell_0, \epsilon) \xrightarrow{\tau} (\ell, w)$ in $\mathcal{S}_{\mathcal{M}}$, there exists an execution $(\ell_{0_p}, \ell_{0_q}, \ell_{0_r}, \overline{\varepsilon}) \xrightarrow{h(\tau)} (\ell, \ell_q, \ell_{0_r}, \overline{w})$ in $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$, where $\ell_q \in \{\ell_{0_q}, \ell_{1_q}\}$ and $\overline{w}_{(p,q)} = w$ (resp. $\overline{\varepsilon}_{(p,q)} = \varepsilon$) and $\overline{w}_c = \varepsilon$ (resp. $\overline{\varepsilon}_c = \varepsilon$) for all $c \neq (p,q)$.*

Proof. We prove this by induction on the length of τ. For the base case, $|\tau| = 0$, i.e. $\tau = \varepsilon$, for which the proposition is trivially true.

Now, by the induction hypothesis, we assume that the proposition holds for any τ such that $|\tau| = n$. We will show that it is also true for $n+1$. Let us assume that we have an execution of length $n+1$, say ν. Let $\nu = \tau \cdot \sigma$, where τ is an execution of length n, and σ is a single action. Say $(\ell_0, \epsilon) \xrightarrow{\tau} (\ell, w) \xrightarrow{\sigma} (\ell', w')$.

Firstly, by [FL17, Lemma 6], ν induces a trace $h(\nu) = h(\tau) \cdot h(\sigma)$ from $\gamma_0 = (\ell_{0_p}, \ell_{0_q}, \ell_{0_r}, \overline{\varepsilon})$ in $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$. Moreover, by the induction hypothesis, there is a state



FIFO automaton $\mathcal{M}_1$

$\mathcal{P}_p$

$\mathcal{P}_q$

$\mathcal{P}_r$

Figure 6.3: The FIFO machine $\mathcal{M}$ and CFM $\mathcal{A}_{\mathcal{M}}$ with $\mathcal{P}_p$, $\mathcal{P}_q$ and $\mathcal{P}_r$ from Example 30

$(\ell, \ell_q, \ell_{0_r}, \overline{w})$ reachable from γ_0 via $h(\tau)$, where $\ell_q \in \{\ell_{0_q}, \ell_{1_q}\}$, $\overline{w}_{(p,q)} = w$ and $\overline{w}_c = \varepsilon$ for all $c \neq (p,q)$. Hence, we only now need to show that $(\ell, \ell_q, \ell_{0_r}, \overline{w}) \xrightarrow{h(\sigma)} (\ell', \ell'_q, \ell_{0_r}, \overline{w'})$, for some values of ℓ'_q and $\overline{w'}$.

- Let us assume σ is a write operation, say $!m$. Then, there exists a transition from the control-state ℓ in $\mathcal{M}$ to the control-state ℓ' which sends a letter m. Therefore, by construction, there exists such a control-state in $\mathcal{P}_p$ as well. Hence, we can execute the same transition in $\mathcal{P}_p$, which sends a letter m to the channel (p,q), i.e. a transition $h(\sigma)$, and moves to state ℓ'. The other processes do not make any transitions. Hence, we reach a state $(\ell', \ell'_q, \ell_{0_r}, \overline{w'})$ such that $\overline{w'}_{(p,q)} = w \cdot m$ and $\overline{w'}_c = \epsilon$ for all $c \neq (p,q)$. Moreover, $\ell'_q = \ell_q$. Hence, the proposition holds.

- Now, let us assume σ is a read operation, say $?m$. This implies $w = m \cdot w'$. Moreover, there exists a transition δ from control-state ℓ in $\mathcal{M}$ to the control-state ℓ' which reads a letter m. Therefore, by construction, there exists a control-state ℓ' in $\mathcal{P}_p$ as well, which is reached from ℓ via an intermediate control-state ℓ_δ. Let us assume $\mathcal{P}_p$ executes the transition $\ell \xrightarrow{!a^{(p,r)}} \ell_\delta$. Then, $\mathcal{P}_p$ cannot proceed any further since it needs to receive a response from $\mathcal{P}_r$, which it does not have yet as the channel (r,p) is empty (by the induction hypothesis).

  Hence, while $\mathcal{P}_p$ is blocked, $\mathcal{P}_r$ now has a message in the channel (p,r), which it can read (as it is in the initial state, such a transition is possible). It moves to an interim control-state $\ell_{m,1}$ and then sends to $\mathcal{P}_q$ an order $m^{(r,q)}$ to dequeue m from channel (p,q), and reaches $\ell_{m,2}$ where it is blocked.

  Now, $\mathcal{P}_q$ receives a new message in the channel (r,q), which was empty so far (by the induction hypothesis). It then consumes this message (again, this is possible as it is in one of the "initial states" ℓ_{0_q} or ℓ_{1_q}). It then moves to $\ell_{m,1}$ from where can read the letter m from the channel (p,q). Note that by the induction hypothesis, the channel contents in (p,q) is $w = m \cdot w'$, hence, it can read m, leaving behind w'. Then, it moves to $\ell_{m,2}$, and then sends to $\mathcal{P}_r$ an acknowledgment, and moves to ℓ_{1_q}.

  Now, $\mathcal{P}_r$ can consume the message from (q,r) and send an acknowledgment back to $\mathcal{P}_p$ and reach back to its initial control-state. And finally, $\mathcal{P}_p$ is no more blocked an can reach ℓ'. Hence, we reach a state $(\ell', \ell_{1_q}, \ell_{0_r}, \overline{w'})$ where $\overline{w'}_{(p,q)} = w'$ and $\overline{w'}_c = \epsilon$ for all $c \neq (p,q)$. Hence, the proposition holds.

The proposition holds for $n+1$, and we are done with the proof. □

Let us prove the converse direction now.

Proposition 6.8. *For any execution* $(\ell_{0_p}, \ell_{0_q}, \ell_{0_r}, \overline{\varepsilon}) \xrightarrow{\nu} (\ell, \ell_q, \ell_{0_r}, \overline{w})$ *in* $\mathcal{S}_{\mathcal{A}_\mathcal{M}}$ *such that* $\ell_q \in \{\ell_{0_q}, \ell_{1_q}\}$ *and* $\overline{w}_{(p,q)} = w \in \Sigma^*$ *and* $\overline{w}_c = \varepsilon$ *for all* $c \neq (p, q)$*, there exists an execution* $(\ell_0, \epsilon) \xrightarrow{h^{-1}(\nu)} (\ell, w)$ *in* $\mathcal{S}_\mathcal{M}$*.*

Proof. From [FL17, Lemma 6], we know that for all $k \geq 0$, $Traces_k(\mathcal{S}_{\mathcal{A}_\mathcal{M}}) = \downarrow\{h(\tau) \mid \tau \in Traces_k(\mathcal{S}_\mathcal{M})\} \cup \downarrow\{h(\tau) \cdot\ !?m^{(p,q)} \cdot\ !?m^{(r,q)} \mid \tau \in Traces_k(\mathcal{S}_\mathcal{M})\}$. Hence, for all traces $\nu \in \downarrow\{h(\tau) \mid \tau \in Traces_k(\mathcal{S}_\mathcal{M})\}$, from Proposition 6.2, we can construct an execution in $\mathcal{S}_\mathcal{M}$ that satisfies the requisite properties.

That leaves us with the set of traces $\{h(\tau) \cdot\ !?m^{(p,r)} \mid \tau \in Traces_k(\mathcal{S}_\mathcal{M})\}$ and the set $\{h(\tau) \cdot\ !?m^{(p,r)} \cdot\ !?m^{(r,q)} \mid \tau \in Traces_k(\mathcal{S}_\mathcal{M})\}$. However, for any trace belonging to either of these sets, it can be seen that the control-state reached in $\mathcal{P}_q$ is not ℓ_{0_q} and hence, the traces in these sets do not belong to the set of traces we consider in the proposition. Hence, the proposition holds true. □

While $\mathcal{S}_{\mathcal{A}_\mathcal{M}}$ simulates $\mathcal{S}_\mathcal{M}$, it is not $\mathcal{I}$-synchronizable. Hence, we try to modify $\mathcal{A}_\mathcal{M}$ so as to make the associated transition system $\mathcal{I}$-synchronizable. We define a system $\mathcal{A}'_\mathcal{M} = (\mathcal{P}_p, \mathcal{P}'_q, \mathcal{P}_r, \Sigma_\mathcal{A})$ where $\mathcal{P}_p$ and $\mathcal{P}_r$ are the same as in $\mathcal{A}_\mathcal{M}$ but $\mathcal{P}'_q$ is modified. This system will later be combined with $\mathcal{A}_\mathcal{M}$ and the whole system will be used in the reduction of the reachability problem in the FIFO automaton to the reachability in $\mathcal{I}$-synchronizable systems. The main purpose of $\mathcal{A}'_\mathcal{M}$ is to make the traces of $\mathcal{S}_{\mathcal{A}_\mathcal{M}}$ synchronizable.

A crucial difference between our construction in $\mathcal{P}'_q$ and the construction in [FL17] is that we do not fix a special message $\underline{m}$ and exclude the traces containing $\underline{m}^{(q,r)}$ from the synchronous traces. In fact, we want to make the entire system $\mathcal{I}$-synchronizable, so we do not isolate any letter from the rest. Therefore, this construction can be seen as a special case of the construction in [FL17] where the underlying FIFO machine $\mathcal{M}$ does not contain the letter $\underline{m}$ at all in its alphabet.

Intuitively, $\mathcal{P}'_q$ will always be able to receive any message from $\mathcal{P}_p$, in particular, at the time the message is sent. Furthermore, it is also able to receive orders from $\mathcal{P}_r$ to dequeue; however, instead of actually dequeueing, it ignores the order but sends an acknowledgment back to $\mathcal{P}_r$. Formally, $\mathcal{P}'_q = (Q'_q, \Sigma_q, \Delta'_q, \ell_{0_q})$ is defined as follows:

$$\begin{aligned} Q_q &= \{\ell_{0_q}, \ell'_{1_q}\} \cup \{\ell'_{m,1} \mid m \in \Sigma\} \\ \Delta_q &= \{(\ell_{0_q}, ?m^{(p,q)}, \ell'_{1_q}), (\ell, ?m^{(p,q)}, \ell) \mid m \in \Sigma, \ell \in Q'_q \setminus \{\ell_{0_q}\}\} \\ &\cup\ \{(\ell_{0_q}, ?m^{(r,q)}, \ell'_{m,1}), (\ell_{1_q}, ?m^{(r,q)}, \ell'_{m,1}), (\ell'_{m,1}, !m^{(q,r)}, \ell'_{1_q}) \mid m \in \Sigma\} \end{aligned}$$

Example 31. For $\Sigma = \{a, m\}$, and $\mathcal{P}_1$ from Figure 6.3, $\mathcal{P}'_q$ is depicted in Figure 6.4.

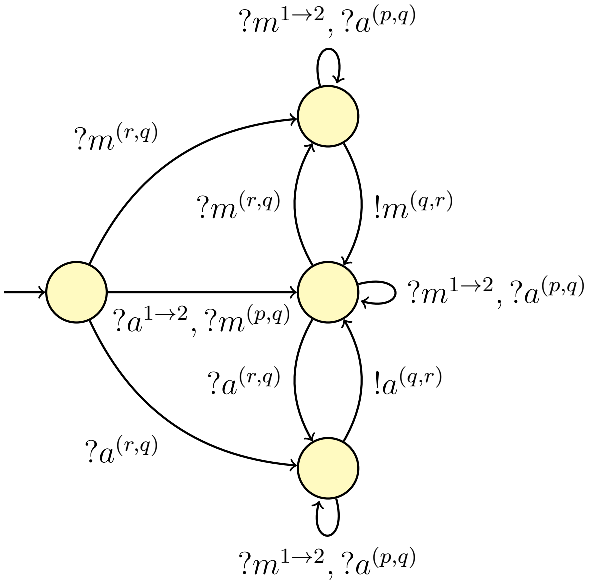

$?m^{1\to 2}, ?a^{(p,q)}$
$?m^{(r,q)}$
$?m^{(r,q)}$
$!m^{(q,r)}$
$?m^{1\to 2}, ?a^{(p,q)}$
$?a^{1\to 2}, ?m^{(p,q)}$
$?a^{(r,q)}$
$!a^{(q,r)}$
$?a^{(r,q)}$
$?m^{1\to 2}, ?a^{(p,q)}$

Figure 6.4: $\mathcal{P}'_q$ for the FIFO automata $\mathcal{M}_1$

As a direct consequence of [FL17, Lemma 8 and 9], we have the following proposition:

Proposition 6.9. *For all traces $\tau \in \mathit{Traces}(\mathcal{S}_{\mathcal{A}_\mathcal{M}})$, there is a synchronous trace $\tau' \in \mathit{Traces}_0(\mathcal{S}_{\mathcal{A}'_\mathcal{M}})$ such that $\mathit{send}(\tau) = \mathit{send}(\tau')$. Furthermore, $\mathcal{S}_{\mathcal{A}'_\mathcal{M}}$ is $\mathcal{I}$-synchronizable.*

Now, let us consider the system $\mathcal{A}''_\mathcal{M} = (\mathcal{P}_p, \mathcal{P}_q \cup \mathcal{P}'_q, \mathcal{P}_r)$, where $\mathcal{P}_q \cup \mathcal{P}'_q = (Q_q \cup Q'_q, \Delta_q \cup \Delta'_q, \Sigma_q, \ell_{0_q})$ is obtained by merging the initial state of $\mathcal{P}_q$ and $\mathcal{P}'_q$.

As a consequence of [FL17, Lemma 10], we have:

Proposition 6.10. *The system $\mathcal{S}_{\mathcal{A}''_\mathcal{M}}$ is $\mathcal{J}$-synchronizable, i.e. $\mathcal{J}(\mathcal{S}_{\mathcal{A}''_\mathcal{M}}) = \mathcal{J}_0(\mathcal{S}_{\mathcal{A}''_\mathcal{M}})$.*

From the above results, we deduce:

Proposition 6.11. *Assume that $\mathcal{M}$ is good-for-reduction. Then, the system $\mathcal{S}_{\mathcal{A}''_\mathcal{M}}$ is $\mathcal{I}$-synchronizable, i.e. $\mathcal{I}(\mathcal{S}_{\mathcal{A}''_\mathcal{M}}) = \mathcal{I}_0(\mathcal{S}_{\mathcal{A}''_\mathcal{M}})$.*

Proof. Since $\mathcal{I}_0(\mathcal{S}_{\mathcal{A}''_\mathcal{M}}) \subseteq \mathcal{I}_k(\mathcal{S}_{\mathcal{A}''_\mathcal{M}})$ is true for any system, in order to show that $\mathcal{S}_{\mathcal{A}''_\mathcal{M}}$ is $\mathcal{I}$-synchronizable, we only need to prove that $\mathcal{I}_k(\mathcal{S}_{\mathcal{A}''_\mathcal{M}}) \subseteq \mathcal{I}_0(\mathcal{S}_{\mathcal{A}''_\mathcal{M}})$ for all $k \in \mathbb{N}$. Since $\mathcal{I}_k(\mathcal{S}_{\mathcal{A}''_\mathcal{M}}) = \mathcal{I}_k(\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \cup \mathcal{I}_k(\mathcal{S}_{\mathcal{A}_\mathcal{M}})$ for all $k \in \mathbb{N}$, we have to prove that $\mathcal{I}(\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \subseteq \mathcal{I}_0(\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \cup \mathcal{I}_0(\mathcal{S}_{\mathcal{A}_\mathcal{M}})$ and $\mathcal{I}(\mathcal{S}_{\mathcal{A}_\mathcal{M}}) \subseteq \mathcal{I}_0(\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \cup \mathcal{I}_0(\mathcal{S}_{\mathcal{A}_\mathcal{M}})$.

From Proposition 6.9, we know that $\mathcal{I}(\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \subseteq \mathcal{I}_0(\mathcal{S}_{\mathcal{A}'_\mathcal{M}})$. Hence, we only need to show $\mathcal{I}(\mathcal{S}_{\mathcal{A}_\mathcal{M}}) \subseteq \mathcal{I}_0((\mathcal{S}_{\mathcal{A}'_\mathcal{M}}) \cup \mathcal{I}_0\mathcal{S}_{(\mathcal{A}_\mathcal{M}})$. We know that $\mathcal{S}_{\mathcal{A}''_\mathcal{M}}$ is $\mathcal{J}$-synchronizable from Proposition 6.10. Hence, $\mathcal{J}(\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}) \subseteq \mathcal{J}_0(\mathcal{S}_{\mathcal{A}'_{\mathcal{M}}}) \cup \mathcal{J}_0(\mathcal{S}_{\mathcal{A}_{\mathcal{M}}})$. So we only need to prove that for all stable traces τ of $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$, there is a stable synchronous trace τ' of $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$ leading to the same state such that $proj_!(\tau) = proj_!(\tau')$.

Let $\tau \in Traces(\mathcal{S}_{\mathcal{A}_{\mathcal{M}}})$ be a stable trace. By [FL17, Lemma 8], there is a trace τ_0 such that either $\tau = h(\tau_0)$, or $\tau = h(\tau_0).!?m^{(p,r)}$, or $\tau = h(\tau_0)\cdot!?m^{(p,r)}\cdot!?m^{(r,q)}$. By the definition of h, if τ is stable, τ_0 is also stable. Since $\mathcal{M}$ is good-for-reduction, τ_0 must be the empty trace. Hence, in all the above cases, τ is synchronous, and therefore, all stable traces of $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$ are synchronous, which means $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$ is $\mathcal{I}$-synchronous. □

Finally, we recall one important aspect of the construction. When defining $\mathcal{P}_q$ and $\mathcal{P}'_q$, it was enforced that the union of the two automata cannot come back to the initial state ℓ_{0_q}. In doing so, we make sure that any trace of $\mathcal{A}''_{\mathcal{M}}$ is either a trace of $\mathcal{A}_{\mathcal{M}}$ or $\mathcal{A}'_{\mathcal{M}}$. Moreover, once a control-state in $Q'_q \setminus \{\ell_{0_q}\}$ has been reached, it is no longer possible to reach a control-state in Q_q.

Using the results so far, we can now prove that reachability is undecidable for $\mathcal{I}$-synchronizable systems.

Theorem 6.12. *Reachability is undecidable for $\mathcal{I}$-synchronizable peer-to-peer systems with at least 3 processes.*

Proof. We reduce the reachability of good-for-reduction FIFO machines to reachability in $\mathcal{I}$-synchronizable peer-to-peer systems. Let $\mathcal{M}$ be a good-for-reduction FIFO machine. We construct a CFM $\mathcal{A}''_{\mathcal{M}}$ with three processes as shown in the above construction. From Proposition 6.11, we know that $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$ is $\mathcal{I}$-synchronizable.

From Proposition 6.2, we know that if a state (ℓ, w) is reachable in $\mathcal{S}_{\mathcal{M}}$, then $(\ell, \ell_q, \ell_{0_r}, \overline{w})$ is reachable in $\mathcal{S}_{\mathcal{A}_{\mathcal{M}}}$. Hence, it is also reachable in $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$.

Conversely, consider a state $(\ell, \ell_q, \ell_{0_r}, \overline{w})$ that is reachable via τ in $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$ such that $\ell_q \in \{\ell_{0_q}, \ell_{1_q}\}$, and $\overline{w}_{(p,q)} = w \in \Sigma^*$ and $\overline{w}_c = \varepsilon$ for all $c \neq (p,q)$. Since $\ell_q \in Q_q$, we know that there cannot be a trace in $\mathcal{S}_{\mathcal{A}'_{\mathcal{M}}}$ that satisfies this property. Hence, $\tau \in Traces(\mathcal{S}_{\mathcal{A}_{\mathcal{M}}})$. Moreover, by Proposition 6.2, (ℓ, w) is reachable in $\mathcal{S}_{\mathcal{M}}$.

Hence, (ℓ, w) is reachable in $\mathcal{S}_{\mathcal{M}}$ iff $(\ell, \ell_q, \ell_{0_r}, \overline{w})$ is reachable in $\mathcal{S}_{\mathcal{A}''_{\mathcal{M}}}$ where $\ell_q \in \{\ell_{0_q}, \ell_{1_q}\}$, and $\overline{w}_{(p,q)} = w \in \Sigma^*$ and $\overline{w}_c = \varepsilon$ for all $c \neq (p,q)$.

And since reachability is undecidable for general good-for-reduction FIFO machines by Theorem 6.6, we deduce that reachability is undecidable for $\mathcal{I}$-synchronizable peer-to-peer systems with at least 3 processes. □

Note that the above construction gives us, as a corollary, a result we proved earlier: reachability is undecidable for $\mathcal{J}$-synchronizable systems. Also note that the decidability question of reachability for $\mathcal{I}$-synchronizable peer-to-peer systems with 2 processes is open.

6.3 Stable reachability

As the notion of $\mathcal{I}$-synchronizability and $\mathcal{J}$-synchronizability differ only in the notion of stable states, we look at a modification of the reachability problem.

Decision Problem: STABLE REACHABILITY

Input: a CFM $\mathcal{A} = ((\mathcal{P}_p)_{p\in\mathbb{P}}, \Sigma)$,
a state $\gamma = ((\ell_p)_{p\in\mathbb{P}}, (\varepsilon)_{(p,q)})$ for $p, q \in \mathbb{P}$ and $p \neq q$
Question: is $\gamma \in \mathit{Post}^*_{\mathcal{S}_\mathcal{A}}(\mathit{init})$?

Stable state reachability is trivially decidable for $\mathcal{I}$-synchronizable systems since by definition, every stable state is reachable via a synchronous run. Hence, given a stable state, we only need to verify if there exists a path to it via a run where each channel has capacity at most 1. However, we show that this is not decidable for $\mathcal{J}$-synchronizable systems, which confirms that the restriction in $\mathcal{I}$-synchronizable systems is indeed non-trivial.

Theorem 6.13. *The stable reachability problem is undecidable for $\mathcal{J}$-synchronizable peer-to-peer systems with 2 processes.*

Proof. We reduce reachability in communicating systems with 2 processes (that is undecidable [BZ83]) to stable reachability in $\mathcal{J}$-synchronizable systems of two processes.

Let us consider a CFM $\mathcal{A} = (\mathcal{P}_p, \mathcal{P}_q, \Sigma)$ of two processes p, q such that $\mathcal{P}_i = (Q_i, \Sigma_i, \Delta_i, \ell_{0_i})$ for $i \in \{p, q\}$. Our objective is to verify if $\gamma = (\ell_{f,p}, \ell_{f,q}, w_p, w_q)$ is reachable.

We construct a new CFM $\mathcal{A}_\gamma = (\mathcal{P}'_p, \mathcal{P}'_q, \Sigma)$, with all the states and transitions as in $\mathcal{A}$. Let $\mathcal{P}'_i = (Q'_i, \Sigma_i, \Delta'_i, \ell_{0_i})$. We add new states to Q'_i as follows: a state ℓ_i, reachable from the initial state, and which has self loops to send and receive every letter of the message alphabet, and ℓ_{w_i}, which is reachable from $\ell_{f,i}$ by receiving the word w_i (this can be achieved by adding intermediate states, but we ignore this detail for brevity). Formally, for $i \in \{p, q\}$, we define $\mathcal{P}'_i = (Q'_i, \Sigma_i, \Delta'_i, \ell_{0_i})$ where

- $Q'_i = Q_i \cup \{\ell_i, \ell_{w_i}\}$
- $\Delta'_i = \Delta_i \cup \{\ell' \xrightarrow{\#m} \ell_i \mid m \in \Sigma, \# \in \{!, ?\}, \ell' \in \{\ell_i, \ell_{0_i}\}\} \cup \{\ell_{f,i} \xrightarrow{?w_i} \ell_{w_i}\}$

$\mathcal{A}_\gamma$ is shown in Figure 6.5.

The new system $\mathcal{S}_{\mathcal{A}_\gamma}$ is $\mathcal{J}$-synchronizable, since for any non-empty trace $\tau = {!m}^{(p,q)} \cdot \sigma$ where $m \in \Sigma$, $\sigma \in \Sigma^*$, there is an execution $(\ell_{0_p}, \ell_{0_q}, \epsilon, \epsilon) \xrightarrow{!?m^{(p,q)}} (\ell_p, \ell_q, \epsilon, \epsilon) \xrightarrow{\nu} (\ell_p, \ell_q, \epsilon, \epsilon)$ such that $proj_!(\tau) = m \cdot proj_!(\nu)$ and ν is of the form $!?m_1!?m_2 \ldots$ for messages $m_1, m_2, \ldots \in \Sigma$.

Furthermore, we see that if $(\ell_{f,p}, \ell_{f,q}, w_p, w_q)$ is reachable in $\mathcal{S}_\mathcal{A}$, then it is reachable in $\mathcal{S}_{\mathcal{A}_\gamma}$ (by the same trace). Then, we can reach $(\ell_{w_p}, \ell_{w_q}, \epsilon, \epsilon)$ in $\mathcal{S}_{\mathcal{A}_\gamma}$ (by construction). Conversely, if we can reach $(\ell_{w_p}, \ell_{w_q}, \epsilon, \epsilon)$ in $\mathcal{S}_{\mathcal{A}_\gamma}$, then it could only have been reached from $(\ell_{f,p}, \ell_{f,q}, w_p, w_q)$ (as it is not reachable from any other state by construction). Hence, the state $(\ell_{f,p}, \ell_{f,q}, w_p, w_q)$ is also reachable in $\mathcal{S}_\mathcal{A}$ by the same trace.

Thus, if stable reachability would be decidable for $\mathcal{S}_{\mathcal{A}_\gamma}$, we could decide the reachability of a $(\ell_{f,p}, \ell_{f,q}, w_p, w_q)$ for $\mathcal{S}_\mathcal{A}$. Hence, if stable reachability would be decidable for $\mathcal{J}$-synchronizable systems, then we could decide the reachability of a state in any system of communicating machines. □

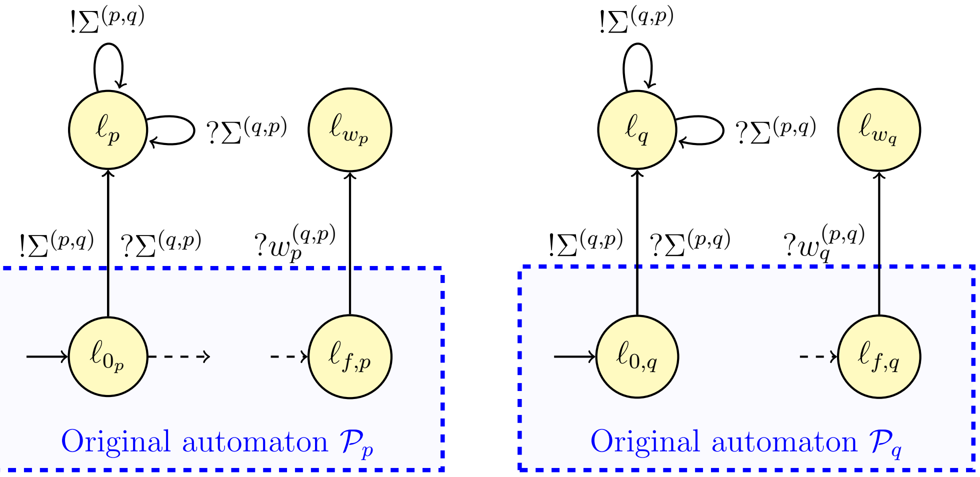


Figure 6.5: Construction of the modified system $\mathcal{A}_\gamma$, where the send/reception of every message is abbreviated by the symbol $!\Sigma$ and $?\Sigma$

6.4 Some decidable notions

Since most verification problems are undecidable for (general) synchronizable peer-to-peer systems, we study subclasses of CFMs for which some verification problems may be decidable. Let us look at some of these subclasses.

Input-bounded systems

Let us suppose that a P2P system $\mathcal{S}$ over Σ is $\mathcal{J}$-synchronizable ($\mathcal{I}$-synchronizable systems are a subclass of $\mathcal{J}$-synchronizable systems). We have: $\mathcal{J}(\mathcal{S}) = \mathcal{J}_0(\mathcal{S})$ where $\mathcal{J}_0(\mathcal{S}) = \{\mathsf{send}(\tau) \mid \tau \in \mathit{Traces}_0(\mathcal{S})\}$ and $\mathit{Traces}_0(\mathcal{S})$ is the set of synchronous traces of $\mathcal{S}$. Since $\mathit{Traces}_0(\mathcal{S})$ is computable (in polynomial time), we may also compute the language $\mathcal{J}_0(\mathcal{S}) \subseteq \Sigma^*$.

Let us recall that a language $L \subseteq \Sigma^*$ is *bounded* if there exist an integer $k \geq 1$ and k words $w_1, w_2, ..., w_k \in \Sigma^*$ such that $L \subseteq w_1^* w_2^* ... w_k^*$. The property for a rational language to be bounded is decidable, see for example [CDV10]. Let us say that a synchronizable system $\mathcal{S}$ is *bounded* if $\mathcal{J}_0(\mathcal{S})$ is a bounded language. Then given a synchronizable system, we may decide whether it is bounded.

bounded synchronizable systems

Let us recall that a system $\mathcal{S}$ is *input-bounded* [BFS20] if $\mathcal{J}(\mathcal{S}) \overset{def}{=} \bigcup_{k \geq 0} \mathcal{J}_k(\mathcal{S})$ is a bounded language. This class of systems subsumes input-letter-bounded machines, flat machines, linear FIFO nets, and monogeneous machine. We now observe, just by definition, that any bounded synchronizable system $\mathcal{S}$ is input-bounded (since $\mathcal{J}(\mathcal{S}) = \mathcal{J}_0(\mathcal{S})$) and from [BFS20], we deduce that many problems like reachability, control-state reachability, unboundedness, deadlock, etc. are decidable. With this result, we are able to find a subclass of synchronizable systems for which reachability is decidable.

Theorem 6.14. *Reachability is decidable for bounded $\mathcal{J}$-synchronizable peer-to-peer systems.*

Regular systems

Let us say that a CFM $\mathcal{A}$ is *effectively $\mathcal{J}$-regular* (resp. *effectively $\mathcal{I}$-regular*) if $\mathcal{J}(\mathcal{S}_\mathcal{A})$ (resp. $\mathcal{I}(\mathcal{S}_\mathcal{A})$) is regular and computable.

Let us remark that $\mathcal{J}(\mathcal{S}_\mathcal{A})$ can be regular when $\mathit{Traces}(\mathcal{S}_\mathcal{A})$ is not regular: suppose that $\mathcal{A}$ is a CFM with two processes, and a single unidirectional channel such that the alphabet contains an unique letter a and $\mathit{Traces}(\mathcal{S}_\mathcal{A}) = \bigcup_{n \geq p \geq 0} (!a)^n (?a)^p$ is not regular but $\mathcal{J}(\mathcal{S}_{\mathcal{A}}) = a^*$.

Systems with k-bounded FIFO channels (i.e., words in channels are in $\Sigma^{\leq k}$) and half-duplex systems [CF05] are two examples of effective $\mathcal{J}$-regular and $\mathcal{I}$-regular systems. Then $\mathcal{J}$-synchronizability (resp. $\mathcal{I}$-synchronizability) reduces to decide whether $\mathcal{J}(\mathcal{S}_{\mathcal{A}}) = \mathcal{J}_0(\mathcal{S}_{\mathcal{A}})$ (resp. $\mathcal{I}(\mathcal{S}_{\mathcal{A}}) = \mathcal{I}_0(\mathcal{S}_{\mathcal{A}})$) which is decidable since the equality of two regular languages is decidable (finite automata that recognize $\mathcal{J}_0(\mathcal{S}_{\mathcal{A}})$ and $\mathcal{I}_0(\mathcal{S}_{\mathcal{A}})$ are computable from $\mathcal{S}_{\mathcal{A}}$). We could extend this decidability result to classes of systems $\mathcal{S}_{\mathcal{A}}$ such that $\mathcal{J}(\mathcal{S}_{\mathcal{A}})$ and $\mathcal{I}(\mathcal{S}_{\mathcal{A}})$ belong to classes of languages for which testing equality with a regular language is decidable like deterministic context-free languages, visibly pushdown systems. But, regularity is an undecidable property for general FIFO systems.

6.5 Discussion

The main open question in the field remains the question of whether the notions of send-synchronizability are decidable under the mailbox semantics. As we have seen in the case of weakly synchronous MSCs (cf. Section 5.4.1), there are notions which are undecidable for the peer-to-peer semantics but become decidable for the mailbox case. It would be interesting to see if send-synchronizability could be one such example.

However, for the undecidability of reachability for $\mathcal{J}$-synchronizable systems, we have seen that it holds even for 2 processes, and hence, holds trivially for the mailbox case. This is not the case for $\mathcal{I}$-synchronizable systems. Moreover, the question of reachability for $\mathcal{I}$-synchronizable systems of 2 processes is also open as of yet.

As for the interest of the notion of send-synchronizability, the results in this chapter show that it is not a useful measure to analyse safety questions such as reachability and variations. However, the question of boundedness remains open. Moreover, it is likely that a notion based on observable behaviour could be a good choice for model checking, as one could ask the question of whether a certain message is observed along the trace, and similar questions.

Another interest point of investigate is the relationship between the notions of send-synchronizability and choreography realizability. In a recent work [SAB20], the authors have analysed the notion of choreography realisability, where a system is called realisable, if the traces of the prescribed communication coincide with those of the asynchronous system of peers. The notion of synchronizability changes slightly in the presence of choreographies, i.e. FSMs that prescribe the rendezvous synchronisation. In this case the peers are projections of a choreography, and synchronizability becomes realisability of the given choreography. The rendezvous composition of the projected peers coincides with the choreography, whereas in general projections of a rendez-vous composition of arbitrary peers may not coincide with the given peers. It would be interesting to see if there is a similar characterization of synchronizability for the mailbox case.

Chapter 7

Conclusion

In this thesis, we studied various under-approximations of transition systems, with a particular focus on subclasses of communicating finite-state systems. For these subclasses, we tried to prove the decidability (or decidability) of various verification problems, such as reachability and boundedness. Let us recall some of the significant contributions of the thesis:

Effective branch-well structured transition systems have decidable boundedness and termination. In Chapter 3, we introduced branch-WSTS, a class of systems that strictly includes well-structured transition systems where we relax both the wqo and the monotony conditions. For this class, we show that boundedness and termination are decidable under effectivity conditions.

However, we show that coverability is undecidable for branch-WSTS. Therefore, we introduce a class of (well-quasi-ordered) transition systems where the monotony condition is relaxed, and show that a variation of the coverability problem is decidable for them.

Input-bounded rational reachability is decidable for FIFO machines. In Chapter 4, given a rational set, a set of bounded languages, and a FIFO machine, the problem of checking whether a control-state is reachable along an input-bounded run, such that the channel contents belong to the rational set, is decidable. This result leads the way in deriving many other results, such as the decidability of boundedness, deadlock, reachability and so on, under the input-bounded restriction.

We also study the special case of a FIFO machine with a single channel to obtain some lower and upper bounds. Moreover, we also looked at the dual of such systems (where the reception is bounded), and observed some similarities, while also looking at the difference between the two notions. Finally, we open the way to introducing a kind of under-approximation for the general reachability (and other verification problems) of FIFO machines, where we can take larger and larger bounded languages and verify the reachability of potentially bad states.

The synchronizability problem is decidable for any class of MSCs which are MSO-definable and have bounded-special tree width. This result we show in Chapter 5 provides a framework for deciding synchronizability and reachability for a number of classes studied in the literature. Moreover, this also paves the way to unify notions of synchronizability.

We also briefly investigate the notion of send-synchronizability for peer-to-peer systems and show that a number of verification problems remain undecidable for this notion.

These questions have tried to answer some of the open questions in the literature, but there are still several unanswered questions and directions for future research.

7.1 Relaxations of well-structured transition systems

Branch-WSTS and cover-WSTS have given us insight into the fact that the conditions for WSTS can be relaxed while still keeping some of the interesting problems decidable. However, there are some open problems that this study has led to:

- The biggest gap in this study is the question of coverability, and whether we can find a class of systems for which general coverability is decidable (without necessarily having to compute the coverability set).

- Another open question is the characterisation of FIFO and counter machines which are branch-WSTS (and cover-WSTS). We were able to find some subclasses of these systems, but the larger question of whether one could precisely capture the set of all FIFO (or counter) machines, which are branch-WSTS for a given ordering, is open. Moreover, for FIFO machines, the prefix relation introduced in [JJ93] showed boundedness to be decidable for input-bounded FIFO machines. However, as it is not an ordering, we cannot use it for branch-WSTS. Strengthening this notion to an ordering would introduce a new class of FIFO machines with decidable properties.

- More generally, it would be interesting to study other models which are not WSTS and automatically verify whether boundedness and termination are decidable for such systems. We can also study other branch-WSTS (resp. cover-WSTS) and other orderings for which they hold.

Perspectives. From a theoretical perspective, a significant problem of interest is to study how much further one can relax the conditions on WSTS in order to still obtain decidable verification problems. It would also be interesting to see if this relaxation translates to better hope for the usability of WSTS and relaxations as a verification technique.

For FIFO machines, an interesting open problem is characterizing the class for which the prefix ordering behaves like a well-ordering. We saw that for branch-monotony, input-bounded systems were still too general. Another related problem would be to modify the prefix-compatible relation to make it an ordering. Since the overarching objective of the thesis was FIFO machines, a pertinent question is to see how they can be characterised as well (resp. branch-well, cover-well) under the various orderings.

7.2 Input-bounded FIFO machines

For the bounded-reachability in FIFO machines, we have a number of immediate open problems:

- It would be interesting to obtain the precise complexity of input-bounded reachability (and other verification problems) for FIFO machines with a fixed number of channels. Moreover, we could also ask what the complexity of these problems is for general FIFO machines.

- For output-bounded verification, it is interesting to see that most problems reduce from the input-bounded case. However, in order to complete the picture, it would be worthwhile to investigate whether rational reachability (and deadlock detection) are decidable for the output-bounded case. Moreover, the question of whether restricting the input-bounded language is equivalent to restricting the output-language is an interesting open question.

- We see that counter machines with restricted zero tests prove to be an interesting extension of VASS. Since each channel is modelled by more than one counter, they are not immediately comparable to VAS with hierarchical zero tests studied in [Rei08, Bon13] (as there is a total order on the counters). It would be interesting to compare these models and see whether we can derive some interesting results on CMRZs.

Perspectives. Over the long term, input-bounded reachability could provide a novel approach to verifying FIFO machines. Much like the notion of flattable systems, the notion of boundable FIFO machines could prove to be an interesting technique to verify FIFO machines. However, it is not yet known whether reachability is decidable for boundable FIFO machines, as we still would need to find the right set of bounded languages, and verify if the reachability set of the FIFO machine is equal to the input-bounded reachability set, which would a priori imply computing the reachability set.

However, it could also be used in practice as a semi-decidable under-approximate verification tool. Moreover, the size of the counter machine associated with a FIFO machine and a tuple of bounded languages is exponential, but it is only polynomial if the underlying FIFO is in normal form. Hence, it would be interesting to see if we can use existing tools for counter machines, and whether the construction of flat machines can be used to aid the verification of boundable FIFO machines.

Finally, we also observe that increments happen in an orderly fashion in the CMRZs we use for bounded verification. Hence, working backwards from counter machines to FIFO machines, it would be an interesting question to investigate whether we can extend the notions of bounded reachability to other FIFO machines such that verifying the underlying counter machine still remains decidable. This could, in turn, also be beneficial to the automated verification problems we discussed earlier.

7.3 Unifying notions of synchronizability

Many questions related to the notion of synchronizability can be studied in the future:

- We could think about the hypotheses to add to the framework to decidably answer the problem "does there exist $k \geq 0$ such that $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_k$?". This problem is already decidable for weakly and strongly systems [GLL21], but it remains to be seen if we can obtain these results as an extension of our framework.

- In [LY19], the authors introduced an *asynchronous compatibility* property. Tt would also be interesting to verify whether this property can be expressed in the framework.

- Finally, in recent works [Lav21, SZ22], more comparisons have been made between the classes studied in this thesis and in [BDGF^{+}21]. It would be interesting to complete this picture for all communication architectures (bags, mailboxes, peer-to-peer), and verify if we can include subclasses of half-duplex and reversal-boundedness to our framework.

Perspectives. To recap, the decidability of the question whether there exists a $k \geq 0$ such that $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_k$ allows us to build a model checking strategy by first deciding whether there exists such a $k \geq 0$ and testing if $L_{p2p}(\mathcal{S}) \subseteq \mathcal{C}_k$ for $k = 1, 2, \ldots$ One may use this strategy for weakly/strongly synchronizable systems but not for general existentially bounded systems or deadlock-free universally bounded systems [HMK+05, LM04].

We believe that our approach can also be used to infer the PSPACE upper bound of weakly k-synchronous MSCs from [BEJQ18] by showing bounded path-width and using finite word automata instead of tree automata. It would be interesting to also verify this approach for obtaining other upper bounds. More generally, the question of specific complexity bounds for many of these systems is also open.

A final long-term perspective would be the study of high-level MSCs within the framework. From a CFM, one can try to extract an equivalent high-level MSC. As the language of a k-synchronous CFM is finitely-generated, it may indeed be possible to extract such a high-level MSC. This would imply summarizing several asynchronous steps in a CFM into a macro step of a high-level MSC. Therefore, the question of whether one can compute a high-level MSC from a k-synchronous systems is a potential future direction to study.

7.4 Verifying send-synchronizable systems

Some immediate open questions in the study of send-synchronizable systems are as follows:

- The largest missing piece of the puzzle is the question of whether send-synchronizability is decidable under the mailbox semantics. In [FL17], the authors have shown that the earlier proof in [BBO12a] for mailbox systems fails for a counter-example. However, the decidability of this notion is still open.
- Following the earlier vein, the question of deciding other verification problems for mailbox systems is open. We know that reachability is undecidable from the proof of undecidability for the peer-to-peer case, but this does not extend to the other notions. Moreover, the question of boundedness and termination is open for both mailbox and peer-to-peer semantics.
- Finally, the question of model checking may prove to be decidable for send-synchronizable systems, as we find the definition to be conducive to verify actions along a path of execution. Decidability of model checking could add interest in this class of systems.

Perspectives. For the longer term, it appears as if this definition is not well-suited for the verification problems we are interested in. However, the notion of synchronizing observable behaviour is still an interesting notion in practise. Hence, it would be worthwhile to redefine this notion such that we keep the spirit of the existing definition, but modify it to include some information about the path taken by the system as well. This would also give us some more information on the reachability of control-states and might lead to decidable results. Moreover, as the current version of send-synchronizability is undecidable, this notion could be altered to a decidable one.

Another path of investigation in future could be the notion of choreography realisability. This involves verifying if a set of traces can be realized in a communicating system. For the peer-to-peer case, as we saw earlier, this problem reduces to synchronizability of the underlying system. However, this notion is not as explicit for the mailbox case. Moreover, the intertwined relationship between these notions could lead us to better understand how to study the notion of "slack elastic observable behaviour".

7.5 Other subclasses

We conclude this thesis with some interesting classes of systems that we believe will aid in the verification quest, but we did not have the bandwidth to cover them in this thesis.

Reversal-bounded FIFO machines. Reversal boundedness was introduced for counter machines in [Iba78] and proved to be an interesting subclass, as various properties were shown to be decidable for them. Roughly speaking, a system is k-reversal bounded if, for each run, there are at most k switches between increasing and decreasing each counter. This notion was extended to k-reversal b-bounded systems in [FS08], where the authors showed that reversal-boundedness can be restricted to counters with a value above a certain bound b, and decidability can still be retained.

For FIFO machines, however, it can be seen that reversal-boundedness is undecidable. However, for various topologies, this problem can be decidable. An interesting direction for future work can be investigating the notion of reversal bounded for FIFO machines.

Half-duplex systems. Half-duplex systems have been introduced and studied in [CF05]. Such systems add the restriction that at any given instant, only one of the unidirectional channels between a pair of processes is not empty. However, for more than two processes, more verification problems are undecidable.

However, this notion is still interesting as it can efficiently characterize reliable channel contracts [LV12]. However, since the notion is undecidable for more than two processes, it would be interesting to extend a similar notion for multiparty CFMs, like for example, in the recent work [DGGL21].

Beyond perfect channels. A significant assumption that we made in this thesis was that the channels of communication were perfect. However, this leads to ignoring many of the real-world problems in communication, such as lossiness, duplication, stuttering of messages, etc. A complete conclusion to this work would be to analyse these problems through the lens of these potential faults.

Bibliography

[ABC$^+$08] Parosh Aziz Abdulla, Ahmed Bouajjani, Jonathan Cederberg, Frédéric Haziza, and Ahmed Rezine. Monotonic Abstraction for Programs with Dynamic Memory Heaps. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 341–354, Berlin, Heidelberg, 2008. Springer.

[ABH17] Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of Ordered Multi-Pushdown Automata is 2ETIME-Complete. *International Journal of Foundations of Computer Science*, 28(08):945–975, December 2017. Publisher: World Scientific Publishing Co.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425, June 1990.

[AČJT00] Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic Analysis of Programs with Well Quasi-ordered Domains. *Information and Computation*, 160(1):109–127, July 2000.

[ACV11] Parosh Aziz Abdulla, Jonathan Cederberg, and Tomáš Vojnar. Monotonic Abstraction for Programs with Multiply-Linked Structures. In Giorgio Delzanno and Igor Potapov, editors, *Reachability Problems*, Lecture Notes in Computer Science, pages 125–138, Berlin, Heidelberg, 2011. Springer.

[ADR09] Parosh Aziz Abdulla, Giorgio Delzanno, and Ahmed Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, April 2009.

[AGK14] C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying Communicating Multi-pushdown Systems via Split-Width. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, January 1994.

[AJ93] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 160–170, June 1993.

[ANKS13] Mohamed Faouzi Atig, K. Narayan Kumar, and Prakash Saivasan. Adjacent Ordered Multi-Pushdown Systems. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory*, Lecture Notes in Computer Science, pages 58–69, Berlin, Heidelberg, 2013. Springer.

[BB11] Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 795–804, New York, NY, USA, March 2011. Association for Computing Machinery.

[BBO12a] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 191–202, New York, NY, USA, January 2012. Association for Computing Machinery.

[BBO12b] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Synchronizability for Verification of Asynchronously Communicating Systems. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, pages 56–71, Berlin, Heidelberg, 2012. Springer.

[BDGF$^+$21] Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Etienne Lozes, and Amrita Suresh. A Unifying Framework for Deciding Synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[BEJQ18] Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the Completeness of Verifying Message Passing Programs Under Bounded Asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, Lecture Notes in Computer Science, pages 372–391, Cham, 2018. Springer International Publishing.

[BFG21] Benedikt Bollig, Marie Fortin, and Paul Gastin. Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. *Journal of Computer and System Sciences*, 115:22–53, February 2021.

[BFLP08] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: acceleration from theory to practice. *International Journal on Software Tools for Technology Transfer*, 10(5):401–424, October 2008.

[BFM17] Michael Blondin, Alain Finkel, and Pierre McKenzie. Well Behaved Transition Systems. *Logical Methods in Computer Science*, Volume 13, Issue 3, September 2017. Publisher: Episciences.org.

[BFM18] Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Information and Computation*, 258:28–49, February 2018.

[BFS20] Benedikt Bollig, Alain Finkel, and Amrita Suresh. Bounded Reachability Problems Are Decidable in FIFO Machines. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[BFS22a] Benedikt Bollig, Alain Finkel, and Amrita Suresh. Bounded Reachability Problems are Decidable in FIFO Machines. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. Publisher: Episciences.org.

[BFS22b] Benedikt Bollig, Alain Finkel, and Amrita Suresh. Branch-Well-Structured Transition Systems and Extensions. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, Lecture Notes in Computer Science, pages 50–66, Cham, 2022. Springer International Publishing.

[BG21] Benedikt Bollig and Paul Gastin. Non-Sequential Theory of Distributed Systems, October 2021. arXiv:1904.06942 [cs].

[Bon13] Remi Bonnet. *Theory of Well-Structured Transition Systems and Extended Vector-Addition Systems.* PhD thesis, Ecole normale supérieure de Cachan, France, 2013.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, April 1983.

[CDV10] Christian Choffrut, Flavio D'Alessandro, and Stefano Varricchio. On Bounded Rational Trace Languages. *Theory of Computing Systems*, 46(2):351–369, February 2010.

[CF87] Annie Choquet and Alain Finkel. Simulation of linear FIFO nets by Petri nets having a structured set of terminal markings. In *Proceedings of the 8th International Conference on Applications and Theory of Petri Nets (APN'87)*, Zaragoza, Spain, June 1987.

[CF05] Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, November 2005.

[CFS11] Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward Analysis and Model Checking for Trace Bounded WSTS. In Lars M. Kristensen and Laure Petrucci, editors, *Applications and Theory of Petri Nets*, Lecture Notes in Computer Science, pages 49–68, Berlin, Heidelberg, 2011. Springer.

[CGK12] Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO Decidability of Multi-Pushdown Systems via Split-Width. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012.

[CLL$^+$19] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 24–33, New York, NY, USA, June 2019. Association for Computing Machinery.

[Cou10] Bruno Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS*, volume 8 of *LIPIcs*, pages 13–29, 2010.

[CV09] Rohit Chadha and Mahesh Viswanathan. Deciding branching time properties for asynchronous programs. *Theoretical Computer Science*, 410(42):4169–4179, September 2009.

[DF97] Catherine Dufourd and Alain Finkel. Polynomial-Time Many-One reductions for Petri nets. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 312–326, Berlin, Heidelberg, 1997. Springer.

[DFGD10] Stephane Demri, Alain Finkel, Valentin Goranko, and Govert Drimmelen. Model-Checking CTL* over Flat Presburger Counter Systems. *Journal of Applied Non-Classical Logics*, 20:313–344, January 2010.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 103–115, Berlin, Heidelberg, 1998. Springer.

[DGGL21] Cinzia Di Giusto, Loïc Germerie Guizouarn, and Etienne Lozes. Towards Generalised Half-Duplex Systems. *Electronic Proceedings in Theoretical Computer Science*, 347:22–37, October 2021. arXiv:2110.00145 [cs].

[DGLL20] Cinzia Di Giusto, Laetitia Laversa, and Etienne Lozes. On the k-synchronizability of Systems. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings*, Lecture Notes in Computer Science, pages 157–176, Cham, 2020. Springer International Publishing.

[Dic13] Leonard Eugene Dickson. Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. *American Journal of Mathematics*, 35(4):413–422, 1913. Publisher: Johns Hopkins University Press.

[DS20] Emanuele D'Osualdo and Felix Stutz. Decidable Inductive Invariants for Verification of Cryptographic Protocols with Unbounded Sessions. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[EGM12] Javier Esparza, Pierre Ganty, and Rupak Majumdar. A Perfect Model for Bounded Verification. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pages 285–294, USA, June 2012. IEEE Computer Society.

[FGL09] Alain Finkel and Jean Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 433–444, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969.

[Fin82] A. Finkel. *About monogeneous FIFO Petri Nets*. Publications du Laboratoire Informatique Theorique et Programmation. Laboratoire Informatique Theorique et Programmation, 1982.

[Fin90] Alain Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, December 1990.

[Fin94] Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, March 1994.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, April 1979.

[FL17] Alain Finkel and Étienne Lozes. Synchronizability of Communicating Finite State Machines is not Decidable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 122:1–122:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[FMP04] Alain Finkel, Pierre McKenzie, and Claudine Picaronny. A well-structured framework for analysing petri net extensions. *Information and Computation*, 195(1):1–29, November 2004.

[FP19] Alain Finkel and M. Praveen. Verification of Flat FIFO Systems. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969.

[FP20] Alain Finkel and M. Praveen. Verification of Flat FIFO Systems. *Logical Methods in Computer Science*, Volume 16, Issue 4, October 2020. Publisher: Episciences.org.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, April 2001.

[FS08] Alain Finkel and Arnaud Sangnier. Reversal-Bounded Counter Machines Revisited. In Edward Ochmański and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008*, Lecture Notes in Computer Science, pages 323–334, Berlin, Heidelberg, 2008. Springer.

[GGLR87] M. G. Gouda, E. M. Gurari, T. H. Lai, and L. E. Rosier. On deadlock detection in systems of communicating finite state machines. *Computers and Artificial Intelligence*, 6(3):209–228, July 1987.

[GKM07] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On Communicating Automata with Bounded Channels. *Fundam. Informaticae*, 80(1-3):147–167, 2007.

[GLL21] Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. In *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021.

[GMK04] Blaise Genest, Anca Muscholl, and Dietrich Kuske. A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2004.

[GS64] Seymour Ginsburg and Edwin H. Spanier. Bounded Algol-Like Languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964. Publisher: American Mathematical Society.

[Hen95] Thomas A. Henzinger. Hybrid automata with finite bisimulations. In Zoltán Fülöp and Ferenc Gécseg, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 324–335, Berlin, Heidelberg, 1995. Springer.

[Hig52] Graham Higman. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, January 1952.

[HLMS10] Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability Analysis of Communicating Pushdown Systems. In Luke Ong, editor, *Foundations of Software Science and Computational Structures*, Lecture Notes in Computer Science, pages 267–281, Berlin, Heidelberg, 2010. Springer.

[HMK$^+$05] Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, October 2005.

[Iba78] Oscar H. Ibarra. Reversal-Bounded Multicounter Machines and Their Decision Problems. *Journal of the ACM*, 25(1):116–133, January 1978.

[Jan90] Petr Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74(1):71–93, July 1990.

[Jer91] Thierry Jeron. Testing for unboundedness of Fifo channels. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91*, Lecture Notes in Computer Science, pages 322–333, Berlin, Heidelberg, 1991. Springer.

[JJ93] Thierry Jéron and Claude Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113(1):93–117, May 1993.

[JM95] Petr Jančar and Faron Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory*, Lecture Notes in Computer Science, pages 348–362, Berlin, Heidelberg, 1995. Springer.

[JR67] Claude Jard and Michel Raynal. Specification of Properties is Required to Verify Distributed Algorithms. Technical Report 651, INRIA, Centre IRISA, Rennes, February 1967.

[KM21] Dietrich Kuske and Anca Muscholl. Communicating automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 1147–1188. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[Kru72] Joseph B Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, November 1972.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lav21] Laetitia Laversa. *La synchronisabilité pour les systèmes distribués*. phdthesis, Université Côte d'Azur, December 2021.

[LM02] Markus Lohrey and Anca Muscholl. Bounded MSC Communication. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 2002.

[LM04] Markus Lohrey and Anca Muscholl. Bounded MSC Communication. *Information and Computation*, 189:160–181, 2004.

[LR11] Ming Li and Yanrui Ruan. Approach to Formalizing UML Sequence Diagrams. *Proc. 3rd International Workshop on Intelligent Systems and Applications (ISA)*, May 2011.

[LTMP08] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-Bounded Analysis of Concurrent Queue Systems. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 299–314, Berlin, Heidelberg, 2008. Springer.

[LV12] Étienne Lozes and Jules Villard. Reliable Contracts for Unreliable Half-Duplex Communications. In Marco Carbone and Jean-Marc Petit, editors, *Web Services and Formal Methods*, Lecture Notes in Computer Science, pages 2–16, Berlin, Heidelberg, 2012. Springer.

[LY19] Julien Lange and Nobuko Yoshida. Verifying Asynchronous Interactions via Communicating Session Automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 97–117, Cham, 2019. Springer International Publishing.

[May84] Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal on Computing*, 13(3):441–460, August 1984. Publisher: Society for Industrial and Applied Mathematics.

[MF85] G. Memmi and A. Finkel. An introduction to FIFO nets— monogeneous nets: A subclass of FIFO nets. *Theoretical Computer Science*, 35:191–214, January 1985.

[MM01] P. Madhusudan and B. Meenakshi. Beyond Message Sequence Graphs. In Ramesh Hariharan, V. Vinay, and Madhavan Mukund, editors, *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 256–267, Berlin, Heidelberg, 2001. Springer.

[MP11a] P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011.

[MP11b] P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. *ACM SIGPLAN Notices*, 46(1):283–294, January 2011.

[Par66] Rohit J. Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4):570–581, October 1966.

[Pel00] Doron Peled. Specification and Verification of Message Sequence Charts. In Tommaso Bolognesi and Diego Latella, editors, *Formal Methods for Distributed System Development: FORTE / PSTV 2000 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX) October 10–13, 2000, Pisa, Italy*, IFIP — The International Federation for Information Processing, pages 139–154. Springer US, Boston, MA, 2000.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[Rei08] Klaus Reinhardt. Reachability in Petri Nets with Inhibitor Arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, December 2008.

[SAB20] Klaus-Dieter Schewe, Yamine Aït Ameur, and Sarah Benyagoub. Realisability of Choreographies. In Andreas Herzig and Juha Kontinen, editors, *Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings*, volume 12012 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2020.

[Str82] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1):121–141, July 1982.

[SZ22] Felix Stutz and Damien Zufferey. Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types, August 2022. arXiv:2208.05559 [cs].

[YG83] Yao-Tin Yu and Mohamed G. Gouda. Unboundedness detection for a class of communicating finite-state machines. *Information Processing Letters*, 17(5):235–240, December 1983.

Index