ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

MÉMOIRE DE MASTER

Mécanisation de la Théorie de la Synchronisabilité pour les Automates Communicants avec Sémantique de Boîte aux Lettres

Nicole KETTLER

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention du diplôme de master
en Informatique
d'Université Côte d'Azur

Stage encadré par :
Enrico FORMENTI, Professeur, Université Côte d'Azur
Cinzia DI GIUSTO, Professeur, Université Côte d'Azur
Kirstin PETERS, Professeur, Universität Augsburg
Javier ESPARZA, Professeur, Technische Universität München

Soutenue le : 09.09.2025

Mécanisation de la Théorie de la Synchronisabilité pour les Automates Communicants avec Sémantique de Boîte aux Lettres

---

Mechanization of Synchronizability Theory for Communicating Automata with Mailbox Semantics

Nicole Kettler

⋈

Stage encadré par :

Enrico Formenti, Professeur, Université Côte d'Azur
Cinzia Di Giusto, Professeur, Université Côte d'Azur
Kirstin Peters, Professeur, Universität Augsburg
Javier Esparza, Professeur, Technische Universität München

Université Côte d'Azur

Nicole KETTLER

Mécanisation de la Théorie de la Synchronisabilité pour les Automates Communicants avec Sémantique de Boîte aux Lettres

vii+42 p.

Ce document a été préparé avec LaTeX2e et la classe `these-ISSS` version v. 2.10.

Impression : output.tex – 23/8/2025 – 8:02

Révision pour la classe : these-ISSS.cls,v 2.10 2020/06/24 14:16:37 mpelleau

Mécanisation de la Théorie de la Synchronisabilité pour les Automates Communicants avec Sémantique de Boîte aux Lettres

Résumé

Nous étudions les automates communicants, un modèle où les participants d'un protocole de communication sont représentés par des machines à états finis échangeant des messages. Ces échanges peuvent être synchrones (instantanés) ou asynchrones (via des buffers).

Dans un travail précédent (Di Giusto, Laversa, & Peters, 2024), nous avons exploré la synchronisabilité, une propriété garantissant que la communication asynchrone n'introduit pas de comportements supplémentaires par rapport à la communication synchrone. Cet article a introduit le problème de la synchronisabilité généralisée, en utilisant des automates avec états finaux, et montre qu'il est indécidable. Cependant, la décidabilité est rétablie pour certaines topologies de communication, telles que les arbres, en prouvant que les langages de buffers dans ces contextes sont réguliers et calculables. Cela étend des résultats obtenus précédemment pour les topologies en anneau (?, ?), et ouvre des perspectives pour de futurs travaux sur les structures en multi-arbres.

Un changement méthodologique clé par rapport aux approches antérieures est l'analyse directe du contenu des buffers, au lieu de la comparaison des traces d'envoi, pour évaluer la synchronisabilité (Di Giusto et al., 2024).

Au cours de ce stage, nous visons à mécaniser les principales preuves de (Di Giusto et al., 2024) à l'aide de l'assistant de preuves Isabelle (Paulson, 1994), afin de les vérifier formellement.

Mots-clés : Synchronisabilité, Automates Communicants, Isabelle, Vérification.

Mechanization of Synchronizability Theory for Communicating Automata with Mailbox Semantics

Abstract

We study communicating automata, a model where participants in a communication protocol are represented as finite state machines exchanging messages. These exchanges can be synchronous (instant) or asynchronous (via buffers).

In previous work (Di Giusto et al., 2024), we explored synchronizability, a property ensuring that asynchronous communication does not introduce behaviors beyond those allowed by synchronous communication. This paper introduced the Generalised Synchronizability Problem, using automata with final states, and shows it is undecidable. However, decidability is restored for specific communication topologies such as trees by proving that buffer languages in these settings are regular and computable. This extends results previously obtained for ring topologies (Finkel & Lozes, 2017), and opens avenues for future work on multitree structures. A key methodological shift from prior approaches is the direct analysis of buffer contents, instead of comparing send traces, to assess synchronizability. (Di Giusto et al., 2024)

During this internship, we aim to mechanize the main proofs from (Di Giusto et al., 2024) using the Isabelle proof assistant (Paulson, 1994), in order to verify them.

Keywords: Synchronizability, Communicating Automata, Isabelle, Verification.

Remerciements

I’d like to thank my supervisors, my family and all of my friends.

CHAPITRE 1

Introduction

Distributed systems consist of multiple processes that communicate with one another. When communication is synchronous — where each sent message is immediately received — the behavior of such systems can be analyzed effectively. Real-world systems, however, often rely on asynchronous communication, where messages are placed in buffers and may be received at any point, or not at all. This makes reasoning about systems and whether they behave as intended significantly harder.

The concept of synchronizability provides a way to bridge this gap. An asynchronous system satisfies synchronizability when its behavior can be represented equivalently by a synchronous system. If this is the case, the system can be analyzed as if communication were synchronous, making verification possible. For several asynchronous communication types, however, synchronizability has been shown to be undecidable, and thus research has focused on identifying subclasses of systems where it is decidable. One such result is given in (Di Giusto et al., 2024), where the authors have first proven that synchronizability is also generally undecidable for Mailbox systems, and secondly have provided and proven a procedure to decide synchronizability for Mailbox systems with tree topologies.

This line of research is particularly delicate. In the past, results on synchronizability have been published that later turned out to be flawed or even fully incorrect, identified in follow-up work by other authors. Such incidents highlight how easy it is to make subtle mistakes in reasoning about topics so complex as distributed systems, and highlight why additional measures are needed to increase trust in published results.

Formal verification with proof assistants directly addresses this exact need. By requiring that every step of a proof be written in a precise formal language so that it can then be checked by a computer, tools such as Isabelle (Paulson, 1994) eliminate ambiguity and ensure that no reasoning gaps remain. They thus provide a much higher level of confidence than handwritten proofs and intuition alone.

The goal of this internship[1] was to apply this methodology to one of the main results of (Di Giusto et al., 2024), which they listed in their discussion as a future project. We are hoping to take on this responsibility. Our work consisted of two main steps : first, gaining proficiency[2] with the Isabelle proof assistant, and second, using it to formalize and verify the decidability proof for tree-structured Mailbox systems. Beyond verifying the correctness of the source results, this work demonstrates how proof assistants can help strengthen the foundations of research in general.

1. The internship was at the I3S and in cooperation with two of the co-authors of (Di Giusto et al., 2024), more detail about the specifics of this internship is in Appendix .1.

2. Learning Isabelle could be compared in complexity to learning a programming language with no prior knowledge of that language. In some ways, Isabelle is similar to functional programming, but in others, it is very unique with its syntax for formalizations and proofs.

CHAPITRE 2

Verification with Isabelle

Ce chapitre présente le cœur de notre travail : il introduit les définitions nécessaires, situe notre démarche dans le contexte des travaux existants, puis détaille nos contributions.

This chapter presents the core of our work : it introduces the necessary definitions, provides context through related research, and then details our contributions.

2.1 Related Work

As this paper extends the work of (Di Giusto et al., 2024), the related work remains largely the same, with the addition of a small verification focus at the end. The paper (Di Giusto et al., 2024) forms the basis of this work, and as such will be referred to as the source paper throughout this work. The goal of this work is to verify the claims of the source paper with Isabelle ; thus, verification is the main topic, and the context about synchronizability, etc., is relatively short as a consequence. This is not to diminish the importance of the literature we will highlight, but due to the brief nature of this report, the focus is on our contribution, which does not need much more context other than the source paper. This is also due in part to the papers differing strongly in the definitions and approaches they use to reason about synchronizability.

While the further semantics differ strongly, the basis for the modeling of distributed systems and their communication is common ground among authors. Both synchronous and asynchronous communication protocols are commonly modeled using communicating automata (Brand & Zafiropulo, 1983), which are networks of finite state automata (the participants) where transitions can be interpreted as sending or receiving actions.

In synchronous communication, the sending and receiving of a message occur simultaneously, whereas in the asynchronous case, messages get sent to and received from buffers. In contrast to synchronous communication, the reception from buffers can happen at any time[1], and it is also possible that messages are sent but never received.

Several semantics have been proposed in the literature, e.g. (Charron-Bost, Mattern, & Tel, 1996 ; Chevrou, Hurault, & Quéinnec, 2016 ; Di Giusto, Ferré, Laversa, & Lozes, 2023), or most relevant for us, in (Di Giusto et al., 2024). One of the most commonly studied systems are those with mailbox communication, where each peer has their unique First-In-First-Out (FIFO) buffer. All messages to a certain peer get stored in its buffer (regardless of the sender's identity in the network), and it receives all messages from this buffer. This is in contrast to communication styles, such as Peer-to-Peer (P2P), where each peer has one buffer per other peer, resulting in a much greater number of buffers overall. In this work, we will exclusively focus on mailbox systems, although most other papers tend to reason about both. (Di Giusto, Laversa, & Lozes, 2020 ; Basu & Bultan, 2016 ; Di Giusto et al., 2024)

If no restrictions are imposed upon an asynchronous system, its buffers are generally considered to be unbounded. This means that there is no limit on the number of messages that can be stored in each buffer, respectively. In conjunction with the buffers being FIFO, each peer can now receive its messages at any point after they have been sent, leading to an extreme amount of possibilities that have to be considered when analyzing such systems. Verification of a distributed system can thus quickly become infeasible. In fact, asynchronous systems can encode Turing machines with as few as two peers and two FIFO buffers (Brand & Zafiropulo, 1983). Consequently, verification of asynchronous systems is generally undecidable. (Brand & Zafiropulo, 1983 ; Basu & Bultan, 2016)

Synchronous systems, on the other hand, are essentially finite state automata and can thus be easily verified. Since synchronous communication forces each sent message to be immediately received, all asynchronous cases (where the message is received any number of steps later, if at all) can be disregarded. Since many real-world systems operate asynchronously, however, research is ongoing to find subclasses of such systems that can be verified despite communicating asynchro-

1. The only requirement is that the message is in the correct place (meaning it is the next message to be read) of the buffer upon reception.

nously. To this end, asynchronous systems are generally constrained in some fashion. Two of the most prominent techniques are, firstly, bounding the size of buffers, and secondly, considering only systems that can be related in behavior to synchronous systems.

B-Bounded systems are either systems where each execution only requires buffers of size B, or where the system is causally equivalent to one only requiring buffers of size B (Genest, Kuske, & Muscholl, 2006). These types are called universally and existentially bounded systems, respectively (Genest et al., 2006). Within this categorization of systems, some problems in the realm of model checking are decidable. Thus, if the bound B is known for a system, it can be verified. If it is unknown, however, deciding whether a bound exists for a given system is undecidable for P2P (Kuske & Muscholl, 2021) and mailbox systems (Bollig et al., 2021).

In (Bouajjani, Enea, Ji, & Qadeer, 2018), k-synchronizable systems are defined, where each execution is causally equivalent to one that consists of chunks with k messages each. Within such a slice, all sends must occur first, and then the receives. In contrast to boundedness, deciding whether a system is k-synchronizable is decidable for both P2P (Bouajjani et al., 2018) and mailbox (Di Giusto et al., 2020) systems. Furthermore, if the k is not known, then for mailbox systems it is decidable whether a k exists such that it is k-synchronizable (Giusto, Laversa, & Lozes, 2023). For k-synchronizable systems, decision problems such as whether a configuration is reachable are decidable.

On k-synchronizable systems, (Di Giusto et al., 2020) derived the causal equivalence relation from the *happened before* relation (Lamport, 1978). This ensures that actions are performed in the same order, from the point of view of each participant, and consequently allows for comparing and grouping executions into classes.

The last group of systems we want to highlight, as pioneered by the authors of (Basu & Bultan, 2016), are the synchronizable systems. Such systems are asynchronous systems whose behavior can be directly related to that of a synchronous one. Unlike k-synchronizability, this does not rely on causal equivalence, but instead analyzes only the send traces, which are executions projected onto only send actions, meaning receptions are disregarded. In particular, the ordering of actions may be performed in a different order to that of a k-synchronizable system. Due to this distinction, systems belonging to either of these two groups of (k-)synchronizability are incomparable with one another. (Di Giusto et al., 2024)

Continuing with synchronizability, the authors of (Basu & Bultan, 2016) proposed a procedure to check whether a (mailbox or P2P) system is synchronizable. They claimed that by comparing the set of synchronous send traces to the set of 1-bounded [2] traces of the asynchronous system. If these sets are equal, the system is supposedly synchronizable (Basu & Bultan, 2016). Unfortunately, this was disproven in (Finkel & Lozes, 2017), for both P2P and mailbox, by counterexamples where the 1-bounded traces imply synchronizability, but the 2-bounded ones contradict it. In other words, the 2-bounded traces contained instances that were not in the set of 1-bounded traces, but which could not be created synchronously for the provided network (Finkel & Lozes, 2017). Additionally, they show that checking synchronizability of a P2P system is undecidable in (Finkel & Lozes, 2017, Theorem 3), while they left the problem open for mailbox systems.

In (Di Giusto et al., 2024), this is studied by relaxing one of the hypotheses in (Finkel & Lozes, 2017) and instead considering communicating automata with final accepting states, which is a more general case. With this, they prove that the synchronizability is undecidable, given networks with sets accepting states that are neither empty nor universal. The authors also investigate how

2. Where all buffers are constrained to store at most one message at a time.

different topologies of the communication affect the decidability of the problem. Such a topology is based on the flow of messages, i.e., the topology shows for each peer, to which peers they send and from which they receive messages. (Di Giusto et al., 2024)

In (Finkel & Lozes, 2017, Theorem 11), synchronizability was shown to be decidable for oriented ring topologies. In (Di Giusto et al., 2024), the encoding from (Finkel & Lozes, 2017) is adapted to mailbox systems, and a check to determine synchronizability of trees is proposed using this adaptation. They reach this conclusion by proving that the buffer language is regular and computable. Using this, they analyze the content of buffers, rather than the more indirect approach of comparing sets of send traces as in (Basu & Bultan, 2016). (Di Giusto et al., 2024)

It is now clear that researching the decidability of (k-)synchronizability is extremely complex, leading to human errors in the process. In fact, two of the works we mentioned in this section had flaws that were later corrected by other authors. The first instance is (Bouajjani et al., 2018), whose procedure for deciding reachability in k-synchronizable systems was corrected in (Di Giusto et al., 2020). The other instance is the wrong claim of (Basu & Bultan, 2016) concerning the decidability of synchronizability, which was disproven in (Finkel & Lozes, 2017). One way to reduce these cases is to use theorem provers such as Isabelle to automatically verify proposed theorems and their proofs. Unfortunately, few notions related to synchronizability or communicating automata in general are in publicly available in Isabelle's archive (*Archive of Formal Proofs*, 2004). This archive contains formalizations and proofs of various disciplines and topics, made publicly available by the respective authors in this manner. Concerning distributed systems, this archive contains many proofs unrelated to synchronizability, for example (Fiedler & Traytel, 2020), which concerns a very specific, but unrelated property that they show to be verifiable for a certain type of distributed systems. Another work formalizes communicating automata, but ones that use stimuli and consequently completely different syntax than the ones we use to reason about synchronizability (Buyse & Jaskolka, 2019). This exemplifies the unfortunate gap between the need for verification related to synchronizability topics especially, but the lack of formalization frameworks available for this. This means, to verify any claim, one would first need to formalize communicating automata, etc., to even begin with the truly relevant parts of the verification. This work will make a first step to bridge this gap, and attempt to formalize the claims concerning the synchronizability of trees from (Di Giusto et al., 2024).

2.2 Preliminaries

This section will detail all the necessary definitions and results from the source paper that are needed for the remainder of this work. This work exclusively relies on synchronous systems and mailbox systems with FIFO buffers and tree topologies, and thus only those will be introduced. Although it should be noted that both in the source paper and in most other related literature, additional topologies and communication protocols are routinely explored. To summarize, our preliminaries here consist of the relevant parts in the "Preliminaries" section of the source paper, as well as some of their fourth section "Synchronisability of Mailbox Communication for Tree-like Topologies". The latter of which contains the claims and results we will attempt to verify throughout this work. Some sections will be direct citations from the source paper, as we necessarily operate on the same definitions. (Di Giusto et al., 2024)

2.2.1 Communicating Automata and Formal Language

For a finite set Σ, a word $w = a_1 a_2 \dots a_n \in \Sigma^*$ is a finite sequence of symbols, such that $a_i \in \Sigma$, for all $1 \leq i \leq n$. We denote the concatenation of two words w_1 and w_2 with $w_1 \cdot w_2$, or simply $w_1 w_2$ if it is clear enough where one word starts and the other begins. The empty word is ε.

For the remainder of this work, some knowledge of non-deterministic finite state automata is necessary. A communicating automaton A is a finite state machine that performs actions of two kinds : either sends or receives, which we may also refer to as outputs or inputs, respectively. $\mathcal{L}(A)$ denotes the language accepted by automaton A.

A *network* of communicating automata, or simply a network, is the parallel composition of a finite set $\mathbb{P}$ of participants (commonly referred to as peers) that exchange messages from a finite message set $\mathbb{M}$. From here on, we consider only automata - and hence networks - where all states are final. We denote $a^{p \to q} \in \mathbb{M}$ the message sent from peer p to q with finite payload a. Note that $p \neq q$, i.e., each message must have a distinct sender and receiver. An action is the sending $!m$ or the reception $?m$ of a message $m \in \mathbb{M}$.

Définition 2.2.1 (Network of Communicating Automata)**.** $N = ((A_p)_{p \in \mathbb{P}}, \mathbb{M})$ is a network of communicating automata, where :

1. for each $p \in \mathbb{P}$, $A_p = (S_p, s_0^p, \mathbb{M}, \to_p, F_p)$ is a communicating automaton with S_p is a finite set of states, $s_0^p \in S_p$ the initial state, $\to_p \subseteq S_p Act_p S_p$ is a transition relation, and F_p a set of final states and
2. for each $m \in \mathbb{M}$, there are $p \in \mathbb{P}$ and $s_1, s_2 \in S_p$ such that $(s_1, !m, s_2) \in \to_p$ or $(s_1, ?m, s_2) \in \to_p$.

To identify the flow of messages and hierarchy between peers of a network, we define the topology as a graph with edges from senders to receivers.

Définition 2.2.2 (Topology)**.** Let $N = ((A_p)_{p \in \mathbb{P}}, \mathbb{M})$ be a network of communicating automata. Its topology is an oriented graph $G(N) = (V, E)$, where $V = \{p \mid p \in \mathbb{P}\}$ and $E = \{(p, q) \mid \exists a^{p \to q} \in \mathbb{M}\}$.

We will focus exclusively on tree topologies, where each of the inner automata (nodes) receives messages by exactly one other peer. In contrast, the root receives messages from no other peer, and also no peer can send any message to it. The source paper denotes $\mathbb{P}^p_{send}$ (resp. $\mathbb{P}^p_{rec}$) as "the set of participants sending to (resp. receiving from) p". We will specify this further for this work in Subsection 2.2.2.1, but we will first continue with the definitions as they were originally introduced.

Définition 2.2.3 (Tree topology)**.** Let $N = ((A_p)_{p \in \mathbb{P}}, \mathbb{M})$ be a network of communicating automata and $G(N) = (V, E)$ its topology. $G(N) = (V, E)$ is a tree if it is connected, acyclic, and $\mid \mathbb{P}^p_{send} \mid \leq 1$ for all $p \in \mathbb{P}$.

For networks, many different communication mechanisms and corresponding semantics exist. A *system* with a communication mechanism. We consider only synchronous systems $N_{\mathtt{sync}}$, as well as the asynchronous Mailbox systems $N_{\mathtt{mbox}}$. In synchronous communication, each send necessitates an immediate receive, whereas in the asynchronous kind, messages are sent to and received from memory (buffers), Here we only consider FIFO (First In First Out) buffers, which can be bounded or unbounded. Thus, asynchronous messages must be received in the same order that they are sent in (or not received at all). As an overview, we use the following two systems throughout this thesis :

— Synchronous (`sync`) : there are no buffers in the system, messages are immediately received when they are sent;

— Mailbox (`mbox`) : each peer has their own unique buffer, each peer receives all its messages in this buffer, regardless of who sent it.

We use *configurations* to describe the state of a system and its buffers.

Définition 2.2.4 (Configuration)**.** Let $N = ((A_p)_{p\in\mathbb{P}}, \mathbb{M})$ be a network. A `sync` configuration (respectively a `mbox` configuration) is a tuple $C = ((s^p)_{p\in\mathbb{P}}, \mathbb{B})$ such that :

— s^p is a state of automaton A_p, for all $p \in \mathbb{P}$

— $\mathbb{B}$ is a set of buffers whose content is a word over $\mathbb{M}$ with :

  — an empty tuple for a `sync` configuration,

  — and a tuple $(b_1, \dots, b_n)$ for a `mbox` configuration.

We write ε to denote an empty buffer, and $\mathbb{B}^\emptyset$ to denote that all buffers are empty. We write $\mathbb{B}\{b_i/b\}$ for the tuple of buffers $\mathbb{B}$, where b_i is substituted with b. The set of all configurations is $\mathbb{C}$, $C_0 = \left((s_0^p)_{p\in\mathbb{P}}, \mathbb{B}^\emptyset\right)$ is the *initial configuration*, and $\mathbb{C}_F \subseteq \mathbb{C}$ is the set of *final configurations*, where $s^p \in F_p$ for all participants $p \in \mathbb{P}$. Thus, in our case, all configurations are final.

We describe the behavior of a system with *runs*. A run is a sequence of transitions starting from an initial configuration C_0. Let $\mathtt{com} \in \{\mathtt{sync},\ \mathtt{mbox}\}$ be the type of communication. We define $\xrightarrow[\mathtt{com}]{}{}^*$ as the transitive reflexive closure of $\xrightarrow[\mathtt{com}]{}$.

Without loss of generality, we label the transition in the following definitions of executions and traces with the sending message $!a^{p\to q}$ for brevity.

In a synchronous communication, we consider the send and receive actions to occur together without delay; thus, the synchronous relation `sync`-send merges these two actions.

Définition 2.2.5 (Synchronous system)**.** Let $N = ((A_p)_{p\in\mathbb{P}}, \mathbb{M})$ be a network. The synchronous system $N_{\mathtt{sync}}$ associated with N is the smallest binary relation $\xrightarrow[\mathtt{sync}]{}$ over `sync`-configurations such that :

$$(\mathtt{sync}\text{-send})\ \frac{s^p \xrightarrow{!a^{p\to q}}_p s'^p \qquad s^q \xrightarrow{?a^{p\to q}}_q s'^q}{\left((s^1, \dots, s^p, \dots, s^q, \dots, s^n), \mathbb{B}^\emptyset\right) \xrightarrow[\mathtt{sync}]{!a^{p\to q}} \left((s^1, \dots, s'^p, \dots, s'^q, \dots, s^n), \mathbb{B}^\emptyset\right)}$$

Définition 2.2.6 (Mailbox system)**.** Let $N = ((A_p)_{p\in\mathbb{P}}, \mathbb{M})$ be a network. The mailbox system $N_{\mathtt{mbox}}$ associated with N is the smallest binary relation $\xrightarrow[\mathtt{mbox}]{}$ over `mbox`-configurations such that for each configuration $C = ((s^p)_{p\in\mathbb{P}}, \mathbb{B})$, we have $\mathbb{B} = (b_p)_{p\in\mathbb{P}}$ and $\xrightarrow[\mathtt{mbox}]{}$ is the smallest transition such that :

$$(\mathtt{mbox}\text{-send})\ \frac{s^p \xrightarrow{!a^{p\to q}}_p s'^p}{((s^1, \dots, s^p, \dots, s^n), \mathbb{B}) \xrightarrow[\mathtt{mbox}]{!a^{p\to q}} ((s^1, \dots, s'^p, \dots, s^n), \mathbb{B}\{b_q/b_q \cdot a\})}$$

$$\text{(mbox-rec)}\ \frac{s^q \xrightarrow{?a^{p\to q}}_q s'^q \qquad b_q = a \cdot b'_q}{((s^1,\dots,s^q,\dots,s^n),\mathbb{B}) \xrightarrow[\texttt{mbox}]{?a^{p\to q}} \big((s^1,\dots,s'^q,\dots,s^n),\mathbb{B}\{b_q/b'_q\}\big)}$$

To reason about the behavior of the introduced systems, we now define executions and traces. An execution e is a sequence of actions, and the corresponding trace t is the projection of this sequence on only the send actions. [3]

Définition 2.2.7 (Executions)**.** Let $N = ((A_p)_{p\in\mathbb{P}},\mathbb{M})$ be a network and $\texttt{com} \in \{\texttt{sync},\texttt{mbox}\}$ be the type of communication. $\mathsf{E}(N_{\texttt{com}})$ is the set of executions, with C_0 as the initial configuration, C_n a final configuration and $a_i \in Act$ for all $1 \le i \le n$, by :

$$\mathsf{E}(N_{\texttt{com}}) = \{a_1 \cdot \ \dots\ \cdot a_n \mid C_0 \xrightarrow[\texttt{com}]{a_1} C_1 \xrightarrow[\texttt{com}]{a_2} \dots \xrightarrow[\texttt{com}]{a_n} C_n\}.$$

For a given word w over *actions*, let $w{\downarrow}_!$ (resp. $w{\downarrow}_?$) be its projection on only send (resp. receive) actions, let $w{\downarrow}_P$ its projection on actions that involve only the participants in a set P, let $w{\downarrow}_p$ be its projection on sends from p (to some peer) and receptions of p (of messages from other peers), and let $w{\downarrow}_{\cancel{!?}}$ denote the word over *messages* in w, which is the result of removing all !- and ?-signs. We extend the operators ${\downarrow}_!$, ${\downarrow}_?$, ${\downarrow}_P$, ${\downarrow}_p$, and ${\downarrow}_{\cancel{!?}}$ to languages, by applying them on every word of the language. Note that, $w{\downarrow}_{\{p\}}$ is always empty, since each action involves two distinct peers and never just p, whereas $w{\downarrow}_p$ is the projection of w to actions in that p has an active role (sender in send actions and receiver in receive actions).

Définition 2.2.8 (Traces)**.** Let $N = ((A_p)_{p\in\mathbb{P}},\mathbb{M})$ be a network and $\texttt{com} \in \{\texttt{sync},\ \texttt{mbox}\}$ be the type of communication. $\mathsf{T}(N_{\texttt{com}})$ is the set of traces :

$$\mathsf{T}(N_{\texttt{com}}) = \{e{\downarrow}_! \mid e \in \mathsf{E}(N_{\texttt{com}})\}.$$

2.2.2 Synchronizability

We have provided all the general definitions of the source paper, and now move on to more in-depth definitions and notions needed for reasoning about such systems. A system is synchronizable if its asynchronous behavior can be related to its synchronous one. Thus, an asynchronous system is synchronizable if its set of traces is the same as the one obtained from the synchronous system.

Synchronizability We define the *Synchronizability Problem* as the decision problem of determining whether a given system is synchronizable or not.

The authors in (Di Giusto et al., 2024) have shown that the *General Synchronizability Problem* (where states are not constrained in any way) is undecidable for Mailbox systems. Thus, it is also undecidable for our networks where each state is final. As another contribution, they claim that for Mailbox systems with tree topologies, synchronizability is decidable, which we will aim to verify in this work. (Di Giusto et al., 2024)

We now also restate their notion of the shuffled language here, which they use in their main theorem. For a thorough explanation about the reasoning behind this construction, as well as an

3. In Definition 2.2.8, like the source paper (Di Giusto et al., 2024), we do not consider the content of buffers, in contrast to other works like (Finkel & Lozes, 2017), where the authors study stable configurations (where all buffers are empty).

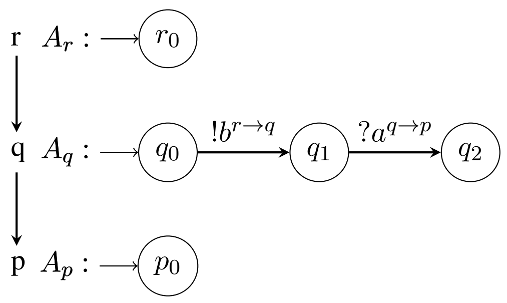

Figure 2.1 – In this network, r can be determined as root globally, but not directly from the local information alone, since both A_r and A_p contain no transitions and hence locally : $\mathbb{P}^r_{send} = \mathbb{P}^r_{rec} = \{\} = \mathbb{P}^p_{send} = \mathbb{P}^p_{rec}$.

example [4], we defer to the source paper (Di Giusto et al., 2024). The word w' is a *valid input shuffle of* w, denoted as $w' \sqcup\!\sqcup_? w$, if w' is obtained from w by a (possibly empty) number of swappings that replace some $!x?y$ within w by $?y!x$. In other words, receptions can be moved further left in a word (and thus sends further right), but the order of both (respectively) is preserved.

Définition 2.2.9 (Shuffled Language)**.** Let $p \in \mathbb{P}$. We define its shuffled language as follows :

$$\mathcal{L}^{\between}_{\sqcup\!\sqcup}(p) = \left\{ w' \mid w \in \mathcal{L}^{\between}(A_p) \wedge w' \sqcup\!\sqcup_? w \right\}$$

2.2.2.1 Formalization of Sends and Receptions

We have now introduced all definitions for which we have kept the formalization. As alluded to earlier, we now wish to specify the sets of $\mathbb{P}^p_{send}$ and resp. $\mathbb{P}^p_{rec}$ a bit further than is done in the original paper. The introduction of these two sets is slightly ambiguous, as one can understand them to either be defined on the network topology, or to be defined on each peer automaton A_p locally. In the former case, we cannot always determine the root directly, as exemplified in Figure 2.1 and elaborated more below. While this has no impact on the overall integrity of the theorem, we choose here to take the full network information into account, to allow for direct root identification and, consequently, clearer reasoning. Thus, throughout this work, we will denote $\mathbb{P}^p_{send}$ and $\mathbb{P}^p_{rec}$ as gathered from the topology.

In Figure 2.1, only A_q provides information on the messages being sent and received in this network (since both A_r and A_p only produce ϵ). Thus, locally A_r and A_p only know that they send and receive no messages, respectively. In a tree, only the root receives from no other peer, so both A_r and A_p could be the root from just their local information alone. With the addition of A_q, however, we can determine $\mathbb{P}^q_{send} = \{r\}$ and $\mathbb{P}^q_{rec} = \{p\}$, and with this, we can then label r as the root. This highlights that at some point, we can receive the same knowledge about the network locally or from the topology, but the latter is more straightforward. Put differently, locally, $\mathbb{P}^r_{send} = \{\}$ does not directly confirm that r is the root (as is the case in 2.1), whereas it does when the full topology is taken into account.

We have now introduced and clarified all definitions, which we will need in the coming parts, but which we have not altered in their formalization, throughout the verification process.

4. Note that we found a small typo in (Di Giusto et al., 2024, Example 4.3) : $!b^{q \to p}?a^{r \to q}$ should not be in the shuffled language, since shuffles can only occur in one direction and not be reversed.

2.3 Inconsistencies, Errors and Corrections

The aim of this work was to mechanize the main theorem (Di Giusto et al., 2024, Theorem 4.5) (and the required lemmas for this) concerning synchronizability in Mailbox systems with tree topologies in the proof assistant Isabelle, to verify their correctness.

Throughout this process, there were multiple moments where proving certain proof steps or lemmas was not possible with Isabelle. This can occur either when smaller lemmas are still needed, which have not yet been specified, or when the to-be-proven step is not actually correct. Because this project involves many of our own definitions and the project itself has a large scope, Isabelle is not able to find counterexamples automatically in most instances. [5] This prompted a manual search of counterexamples and other formalization errors, which ended up yielding multiple results.

The counterexamples presented in the following sections consist of small but representative instances, found after much deliberation. The initial counterexamples were larger, and thus the progression from one revision to the next might seem less intuitive here than it was in reality, but for brevity, we have chosen to showcase smaller instances that function analogously to their initial drafts, respectively.

Before detailing the verification process, a short foreword about our code and Isabelle experience will follow. The first part of this work was learning Isabelle (Paulson, 1994), for which we fully utilized all available learning materials, namely (Wenzel, s. d. ; Nipkow, Paulson, & Wenzel, s. d. ; Nipkow & Klein, 2023 ; Nipkow, 2023). For the mechanization, we applied mostly our own formalizations (and the ones we were initially provided). In addition to those, we used the "Main" library of Isabelle, which provides standard lemmas such as those for lists, etc., as well as the "Sublist" library (Nipkow, Wenzel, Sternagel, & Eberl, s. d.), which provides definitions and lemmas related to prefixes.

Our full Isabelle formalization is publicly accessible on GitHub (Kettler, 2025). As referenced in (Di Giusto et al., 2024, Discussion), the authors of the source paper have started formalizing the paper in Isabelle. We were provided this code as a starting point, which can also be found in the repository [6], as a reference point for our contribution. This initial version contained a lot of the rudimentary definitions (for formal languages, communicating automata, etc.), but was still extremely limited concerning the formalizations needed for proving the main theorem. For comparison, the initial code is around 1200 lines [7] whereas the final code is several thousand lines in the Isabelle files (including comments) and around 116 pages long when converted to a PDF. [8] For the conversion, we used Isabelle's functionality to export the code to LaTeXand then compiled the PDF for this, which contains exclusively the code, sans comments. (Nipkow, Paulson, & Wenzel, s. d.)

2.3.1 Revision of the Influenced Language

In (Di Giusto et al., 2024, Definition 4.2), the authors introduce the definition of the influenced language to "capture the influence the automata have on the language of each other". The idea is to remove all words from the language of a peer p, which it cannot reasonably produce given the

5. For easy contradictions, e.g., a lemma that assumes $x = 1$ and shows $x = 0$, Isabelle's automatic checks succeed, and it will warn that this lemma is wrong.

6. It is in the folder "Formalization_Isabelle" and is the original version provided to us at the start of the internship.

7. When counting the lines of the two initially provided Isabelle files, without converting to a PDF first.

8. This PDF can also be found in the repository, named "TreeSync".

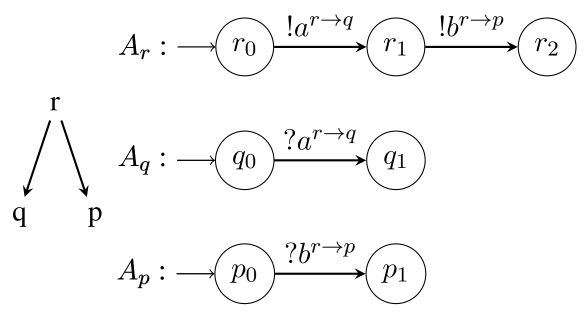

Figure 2.2 – Counterexample 0 : A network where the (original definition of the) influenced language of p contains only ϵ.

context of its network. If a peer receives some m, then it must be sent first by its parent, which in turn may have some receives that rely on its parent, and so on, until the root is reached. If somewhere along this unique path from the root to our peer p, some ancestor of p cannot provide a necessary send to its direct child, then p cannot produce its desired word in the end, either. Such words, for which there is no valid execution, are not present in the influenced language. This was formalized into the following definition :

Définition 2.3.1 (Original Influenced Language)**.** Let $p \in \mathbb{P}$. In (Di Giusto et al., 2024), the influenced language and its projections to sends/receives are defined as follows :

$$\mathcal{L}^{\between}(A_p) = \begin{cases} \mathcal{L}(A_r) & \text{if } p = r \\ \Big\{ w \mid w \in \mathcal{L}(A_p) \wedge (w{\downarrow_?}){\downarrow_{\cancel{!?}}} \in \Big(\mathcal{L}^{\between}_!(A_q) \Big){\downarrow_{\cancel{!?}}} \wedge \mathbb{P}^p_{send} = \{q\} \Big\} & \text{otherwise} \end{cases}$$

$$\mathcal{L}^{\between}_?(A_p) = \mathcal{L}^{\between}(A_p){\downarrow_?}$$

$$\mathcal{L}^{\between}_!(A_p) = \mathcal{L}^{\between}(A_p){\downarrow_!}$$

However, as it is written down, this formalization is too strong : it removes words for which there are valid executions, which is undesired. For trees where each parent sends to at most one child, respectively, this definition works as expected. However, if any parent sends a message to two children within one word, the influenced language removes this word, even if there is a valid execution for it. We want to illustrate this issue through the network in Figure 2.2.

In this synchronous network, there is an execution where p can perform $?b^{r\to p}$, since r can send the matching message as its second action. Here, r is the root and p's only ancestor, thus r requires no sends (as it has no ancestors) and there is no other peer on the path from r to p. Hence, one would expect $?b^{r\to p} \in \mathcal{L}^{\between}(A_p)$, but actually

$$\mathcal{L}^{\between}(A_r) = \mathcal{L}(A_r) = \{\varepsilon, !a^{r\to q}, !a^{r\to q}!b^{r\to p}\} = \mathcal{L}^{\between}_!(A_r) \qquad \text{since } r \text{ is root}$$

We consider all states accepting, thus $\mathcal{L}(A_p) = \{\varepsilon, ?b^{r\to p}\}$ can be inferred from the automaton in Figure 2.2. With this, and knowing that

$$(?b^{r\to p}){\downarrow_{\cancel{!?}}} = b^{r\to p},$$

we can perform the check in the second part of the conjunction of the influenced language definition :

$$\begin{aligned}\{\varepsilon,\ !a^{r\to q},\ !a^{r\to q}!b^{r\to p}\}{\downarrow_{\cancel{!?}}} &= \{\varepsilon,\ a^{r\to q},\ a^{r\to q}b^{r\to p}\},\\ b^{r\to p} &\notin \{\varepsilon,\ a^{r\to q},\ a^{r\to q}b^{r\to p}\}.\end{aligned}$$

Since this check fails, $?b^{r\to p} \notin \mathcal{L}^{\between}(A_p) = \{\varepsilon\}$, i.e. the definition is not working as intended. To summarize, the influenced language is meant to remove words from peers for which there are no valid executions anyways, i.e., to remove uninteresting [9] cases for the synchronizability decision procedure.

In fact, this issue directly harms the integrity of the original main theorem in (Di Giusto et al., 2024). The illustrated network is synchronizable, since the only traces possible are exactly the words in $\mathcal{L}^{\between}(A_r)$, and both children are ready to receive without any blocking sends. Thus, we satisfy the left side of the *if and only if* of the original theorem below :

Définition 2.3.2 (Original Mailbox Synchronizability Theorem)**.** Let N be a network such that $\mathbb{C}_F = \mathbb{C}$ and $G(N)$ is a tree. Then $\mathsf{T}(N_{\mathtt{mbox}}) = \mathsf{T}(N_{\mathtt{sync}})$ iff, for all $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$, we have $\left(\mathcal{L}^{\between}_{!}(A_q){\downarrow_{\{p,q\}}}\right){\downarrow_{\cancel{!?}}} \subseteq \mathcal{L}^{\between}(A_p){\downarrow_{\cancel{!?}}}$ and $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$.

We will refer to the rightmost part of the theorem ($\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$) as the *shuffled language condition*, and to the first condition on the right side as the *subset condition* [10].

We should consequently satisfy the right side as well. However, for p and r, where $\mathbb{P}^p_{send} = \{r\}$, we have

$$\begin{aligned}(\mathcal{L}^{\between}_{!}(A_r){\downarrow_{\{p,r\}}}){\downarrow_{\cancel{!?}}} &= \{\varepsilon,\ !b^{r\to p}\}{\downarrow_{\cancel{!?}}} = \{\varepsilon,\ b^{r\to p}\},\\ \mathcal{L}^{\between}(A_p){\downarrow_{\cancel{!?}}} &= \{\varepsilon\},\\ (\mathcal{L}^{\between}_{!}(A_r){\downarrow_{\{p,r\}}}){\downarrow_{\cancel{!?}}} &= \{\varepsilon,\ b^{r\to p}\} \not\subseteq \{\varepsilon\} = \mathcal{L}^{\between}(A_p){\downarrow_{\cancel{!?}}}.\end{aligned}$$

To conclude, the definition in the source paper not only does not fully work as intended, but in turn renders the entire theorem [11] to be false. The solution to this oversight is to project $\mathcal{L}^{\between}_{!}(A_q)$ in Definition 2.3.1 to the two peers p and q. Since $\mathcal{L}^{\between}_{!}(A_q)$ is the language - projected to only sends - of A_q, it only contains actions of q. We need to ensure that only sends from q to p are considered, thus a projection on *actions where p is the second participant*, i.e., here the recipient, is necessary. In the remainder of this work, whenever we refer to the *influenced language* of some peer and its projections (as defined in the original definition), we will refer instead to this revised version :

Définition 2.3.3 (Revised Influenced Language)**.** Let $p \in \mathbb{P}$. We define the influenced language as follows :

$$\mathcal{L}^{\between}(A_p) = \begin{cases}\mathcal{L}(A_r) & \text{if } p = r\\ \left\{w \mid w \in \mathcal{L}(A_p) \wedge (w{\downarrow_?}){\downarrow_{\cancel{!?}}} \in \left((\mathcal{L}^{\between}_{!}(A_q)){\downarrow_{\{p,q\}}}\right){\downarrow_{\cancel{!?}}} \wedge \mathbb{P}^p_{send} = \{q\}\right\} & \text{otherwise}\end{cases}$$

9. A trace requires at least one valid execution, i.e., one that is possible for the network, thus if an execution is not possible, it cannot produce a trace, either.

10. Currently, the subset condition is $(\mathcal{L}^{\between}_{!}(A_q){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} \subseteq (\mathcal{L}^{\between}(A_p)){\downarrow_{\cancel{!?}}}$

11. Both Lemma 4.4 and Theorem 4.5 in (Di Giusto et al., 2024) are affected by this, but the fix for both is to add the projection to the set $\{p, q\}$, so Lemma 4.4 is not explicitly stated for brevity.

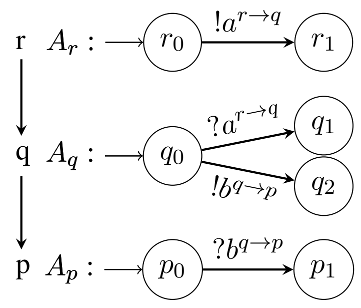

Figure 2.3 – Counterexample 1 : A network that is not synchronizable, but satisfies the original right side of the theorem

The projections of the influenced language to sends and receptions, respectively, remain the same, thus they are not repeated here. The only difference to the original definition is the added pair projection in the second case of the conjunction. Lastly, we can also apply the definition in the following manner : given a $w \in \mathcal{L}^{\between}(A_p)$ we can infer that there exists some $w' \in \mathcal{L}^{\between}(A_q)$ such that $((w'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}$.

2.3.2 Revisions of the Main Theorem

Starting with this section, we will describe the iterative process of contradicting and then revising (Di Giusto et al., 2024, Theorem 4.5). The original theorem can be referenced in Theorem 2.3.2, and we will from here on use this theorem with our revised influenced language from Definition 2.3.3. As explained below Theorem 2.3.2, we refer to the two conditions on the right side as the subset condition and the shuffled language condition, respectively. The subset condition will be revisited in the following sections, but it will always be noted which revision, i.e., which specific instance of the subset condition definition we are currently applying. The theorem will change with each revision, the general construction will remain the following, where *the subset condition* is a placeholder for any of its revisions :

Définition 2.3.4 (Mailbox Synchronizability Theorem General Scheme)**.** Let N be a network such that $\mathbb{C}_F = \mathbb{C}$ and $G(N)$ is a tree. Then $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$ iff, for all $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$, we have **the subset condition** and $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$ (the shuffled language condition).

2.3.2.1 First Counterexample

For this counterexample, we are using the original Theorem 2.3.2. Since the influenced language occurs on the right side of the original theorem, and the influenced language has already posed a problem, we explored the possibility of there being more issues related to its usage. The original subset condition only requires the sends of a parent to be receivable by the respective children. It doesn't restrict other branches, however. This can be exploited by having one branch in the child-automaton that receives exactly the sends to it from the parent, and another branch that performs a send itself, but receives none of the parent's. With such a network as in Figure 2.3, the right side of the theorem is satisfied, but we cannot perform the trace $!a^{r\to q}!b^{q\to p} \in \mathsf{T}(N_{\texttt{mbox}})$ synchronously : q cannot receive $a^{r\to q}$ before, or after sending $b^{q\to p}$, as the two actions are on different branches of A_q and thus mutually exclusive. In Mailbox, however, the message can be sent, since no message ever *needs* to be received in an execution. Thus, we have found a trace that is in $\mathsf{T}(N_{\texttt{mbox}})$ but not in $\mathsf{T}(N_{\texttt{sync}})$, which contradicts the *if and only if* in the original theorem, as the right side is satisfied. The shuffled language condition is trivially satisfied, since each possible word in any peer's language consists of exactly one action only, not allowing for a shuffle. The subset condition is also satisfied, as is inferred in the following :

$$\mathcal{L}(A_r) = \{\varepsilon, !a^{r\to q}\} = \mathcal{L}^{\between}(A_r) = (\mathcal{L}^{\between}_{!}(A_r)){\downarrow}_{\{p,q\}}$$
$$\mathcal{L}(A_q) = \{\varepsilon, ?a^{r\to q}, !b^{q\to p}\} = \mathcal{L}^{\between}(A_q) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(q)$$
$$\mathcal{L}(A_p) = \{\varepsilon, ?b^{q\to p}\} = \mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$$

Then,

$$((\mathcal{L}^{\between}_{!}(A_r)){\downarrow}_{\{p,q\}}){\downarrow}_{\cancel{!?}} \subseteq (\mathcal{L}^{\between}(A_q)){\downarrow}_{\cancel{!?}}$$

since,

$$\{\varepsilon, !a^{r\to q}\}{\downarrow}_{\cancel{!?}} \subseteq \{\varepsilon, ?a^{r\to q}, !b^{q\to p}\}{\downarrow}_{\cancel{!?}}$$

We can observe, the original subset condition is not strong enough. In particular, the problem here is that something is sent by the child, but after that initial send, it may not be able to receive everything the parent could send anymore. Put differently, prefixes consisting of only sends in the words of some peer might pose problems concerning the synchronizability. With this in mind, we revised the subset condition to the following :

Définition 2.3.5 (Subset Condition First Revision)**.** Let $outputprefixes$ denote all prefixes $w \in \mathcal{L}^{\between}(A_p)$, s.t. $w{\downarrow}_! = w$, i.e. $outputprefixes$ is the set of words in $\mathcal{L}^{\between}(A_p)$ consisting only of outputs. Then, for $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$
$\forall w \in outputprefixes\left(\mathcal{L}^{\between}(A_p)\right).\left((\mathcal{L}^{\between}_{!}(A_q){\downarrow}_{\{p,q\}})\right){\downarrow}_{\cancel{!?}} \subseteq \{x | wx \in \mathcal{L}^{\between}(A_p) \land x = x{\downarrow}_?\}{\downarrow}_{\cancel{!?}}$

The definition checks for each output prefix w of some child p with parent q, whether there is some suffix x, such that x consists only of inputs and $w \cdot x \in \mathcal{L}^{\between}(A_p)$. This is to enforce that after performing any amount of initial send actions, there is still some suffix x that p can perform, which receives all the sends its parent might send.

2.3.2.2 Second Counterexample

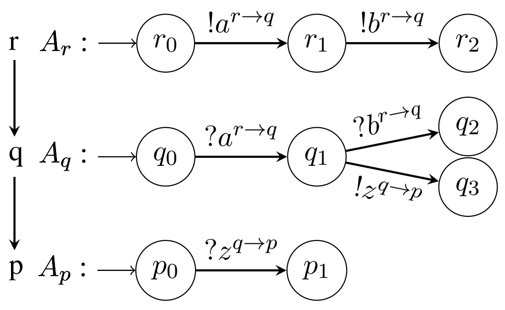

Figure 2.4 – Counterexample 2 : A network that is not synchronizable, but satisfies the right side of the theorem after the first revision of the subset condition.

Continuing on with the revised subset condition from Definition 2.3.5. This condition specifically only checks output prefixes ; thus, the idea arose to check whether prefixes starting in inputs can break this new condition. This lead turned out to be correct, resulting in the next counterexample in Figure 2.4. This new network functions analogously to the previous one, with the exception that A_r sends a second message to q, so q can first perform an input, and otherwise the automata have not changed. r and q now first exchange the message $a^{r \to q}$ and then have the same structure as the previous example (after states r_1 and q_1, respectively).

Since it functions the same as the previous example, we will not reiterate the equivalences needed to contradict the theorem here again. The problematic trace for this network is $!a^{r\to q}!b^{r\to q}!z^{q\to p} \in \mathsf{T}(N_{\mathtt{mbox}})$ and $!a^{r\to q}!b^{r\to q}!z^{q\to p} \notin \mathsf{T}(N_{\mathtt{sync}})$. For the only existing output prefix ϵ of both q and p, respectively, the revised subset condition is satisfied. This is because from the initial state, descendants of the root can receive all the sends they might receive. For example, q is able to receive all possible sends of r, namely the messages ϵ, $a^{r\to q}$ and $a^{r\to q}b^{r\to q}$. However, once q has entered the branch where it sends $z^{q\to p}$, it can no longer receive $b^{r\to q}$, even though r could have already sent it or can still send it in the future. To conclude, the issue is that words starting with receives may also clash with the possible synchronizability of the network, e.g., if some branching, like in our examples, occurs.

Taking this into account, we expanded the previous subset condition to take into account *all* possible prefixes, which are exactly all words in the language of a peer, since all states are accepting. The second iteration of the subset condition can be found in Definition 2.3.6 :

Définition 2.3.6 (Subset Condition Second Revision)**.** Let *is prefix of* refer to the non-strict standard prefix relation, e.g. $!a^{q\to p}$ is a prefix of $!a^{q\to p}$. Then, for $p, q \in \mathbb{P}$ with $\mathbb{P}^{p}_{send} = \{q\}$, have $\forall w \in \mathcal{L}^{\between}(A_p)$.
$\mathcal{L}^{\between}_{!}(A_q){\downarrow_{\{p,q\}}}{\downarrow_{\cancel{!?}}} \subseteq \{yx | wx \in \mathcal{L}^{\between}(A_p) \wedge x = x{\downarrow_?} \wedge \text{y is prefix of } (w{\downarrow_?})\}{\downarrow_{\cancel{!?}}}$

2.3.2.3 Third Counterexample

With this latest revision of the subset condition, we now enforce that parent sends can *always* be received fully, regardless of the child's current prefix. In other words, no matter which prefix we have read in the child automaton, we can still perform the matching receives that might come from the parent. Another addition to this revision is that the right set considers the receptions that have already occurred in the child, as well. This was another oversight, since otherwise we would force the child to receive something again, even if it already received it. The *prefix* relation is used to capture the ϵ, which is in each peer's language and would not be captured in this condition if $w{\downarrow_?} \neq \epsilon$.

We can observe that the right set in the subset relation is more nuanced than the left. This sparked the idea that perhaps the left set could be too general, i.e., that now the condition has become too strong. We enforce that regardless of the prefix $w \in \mathcal{L}^{\between}(A_p)$, *any* send of the parent can still be received (if it hasn't been received already). This does not consider the case in which the parent q has already entered some branch in which it can no longer send *all* the words in $\mathcal{L}^{\between}(A_q)$ anymore, and consequently also not those in $\mathcal{L}^{\between}_{!}(A_q){\downarrow_{\{p,q\}}}$. Since we now allow for inputs in the prefixes of p as well, we indirectly force q to take certain actions.

Consider the network in Figure 2.5, which illustrates the issue that arises by not considering the parent's branching. Here, r can either send $a^{r\to q}$, or $b^{r\to q}$, as its first nonempty action. Consider $w = ?a^{r\to q} \in \mathcal{L}^{\between}(A_p)$, then in all valid executions where q can perform this action, r must have

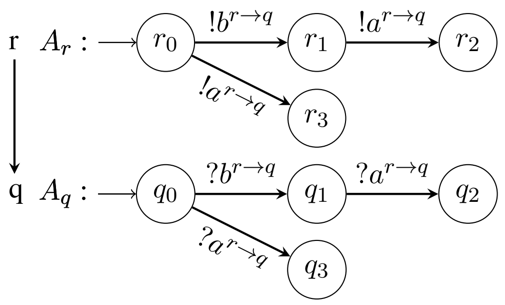

Figure 2.5 – Counterexample 3 : A network that is synchronizable, but does not satisfy the right side of the theorem after the second revision of the subset condition.

sent it already, as q can only receive when that exact message is in in the first position of its buffer. Then, the possible values for y in the right set of our subset condition are $y \in \{\epsilon, ?a^{r\to q}\}$, as those are the prefixes of our w. Since q must have entered its lower branch by performing $?a^{r\to q}$, the only possibility for x is $x \in \{\epsilon\}$. Then we can infer,

$$\{yx|wx \in \mathcal{L}^{\between}(A_q) \wedge x = (x{\downarrow_?}) \wedge \text{y is prefix of } (w{\downarrow_?})\}{\downarrow_{\cancel{!?}}} = \{\epsilon, ?a^{r\to q}\}{\downarrow_{\cancel{!?}}} = \{\epsilon, a^{r\to q}\} \tag{2.1}$$

$$\mathcal{L}^{\between}_!(A_r){\downarrow_{\{q,r\}}}{\downarrow_{\cancel{!?}}} = \{\varepsilon, !a^{r\to q}, !b^{r\to q}, !b^{r\to q}!a^{r\to q}\}{\downarrow_{\cancel{!?}}} = \{\varepsilon, a^{r\to q}, b^{r\to q}, b^{r\to q}a^{r\to q}\} \tag{2.2}$$

$$\{\varepsilon, a^{r\to q}, b^{r\to q}, b^{r\to q}a^{r\to q}\} \nsubseteq \{\epsilon, a^{r\to q}\} \tag{2.3}$$

Combining all previous equations, we then have

$$\left((\mathcal{L}^{\between}_!(A_r)){\downarrow_{\{q,r\}}}\right){\downarrow_{\cancel{!?}}} \nsubseteq \{yx|wx \in \mathcal{L}^{\between}(A_q) \wedge x = x{\downarrow_?} \wedge \text{y is prefix of } (w{\downarrow_?})\}{\downarrow_{\cancel{!?}}} \tag{2.4}$$

It is clear that the subset condition does not hold for this network. However, this network is synchronizable, contradicting the current formalization of the main theorem (2.3.4 with subset condition 2.3.6) once again. The network is synchronizable, since there are only two peers, and only the root r is performing sends, which q can all receive. In other words, the only possible traces are exactly the language of r : $\mathcal{L}(A_r) = \{\varepsilon, !a^{r\to q}, !b^{r\to q}, !b^{r\to q}!a^{r\to q}\}$. By observation, it is clear that q can receive each of the sends belonging to each respective trace in this set. Thus, $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$ and the network is synchronizable.

To conclude, this counterexample contradicts the current theorem, because the subset condition is now too strong. It does not track which actions are even still possible for the parent, but instead forces the child to be able to receive all sends to it, starting from the parent's initial state. This is undesired, as in our example, it forces q to be able to receive sends that r cannot perform anymore because it must have sent $!a^{r\to q}$ already. By sending $!a^{r\to q}$ so that q can receive $?a^{r\to q}$, both q and r must enter their respective lower branches (states r_3 and q_3). From r_3, r cannot send $!b^{r\to q}$ or $!b^{r\to q}!a^{r\to q}$ anymore, but the condition still forces q to be able to receive those two impossible sends.

This in turn means, that we must track both the parent and the child, depending on which receptions have already occurred. If the child receives a message, the parent must have sent it. Thus, we can disregard all words of the parent, where these matching sends have either not occurred (completely), or are not in the correct order. This led us to the third and final revision of the subset condition in Definition 2.3.7 and the matching rendition of the main theorem in Theorem 2.3.3.

2.3.2.4 Final Formalization and Proof

As touched upon lightly in the previous section, the final subset condition now tracks both the parent q and the child p for each parent-child pair. For each word w of the child, the condition tracks all possible w' in the parent's language, which provide *exactly* the matching sends to the receives $w{\downarrow_?}$ of w. We do not constrict these w' otherwise. In particular, w' may contain any number of sends to children other than p, or receives if q is not the root. We only choose the words that exactly provide what p needs from q, but disregard all actions to other peers. This ensures that w' really captures all possible words for q, and in turn all possible executions that might hinder the synchronizability of a network. We know that at least one such w' must exist, otherwise $w \notin \mathcal{L}^{\between}(A_p)$ by Definition 2.3.3. Now for every such pair w and w', we need to ensure that for possible suffixes x' s.t. $w'x' \in \mathcal{L}^{\between}(A_q)$, there is at least one suffix x, s.t. $((x'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (x{\downarrow_?}){\downarrow_{\cancel{!?}}}$. Put into words, we check for the current branch of q and of p, whether p can still receive everything q is still able to send after that point. With this, we have weakened the condition again slightly, so that it checks only the words that can be reached, given the current prefix of p and the resulting words of q.

Définition 2.3.7 (Subset Condition Final Revision)**.** For $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$, we have
$\forall w \in \mathcal{L}^{\between}(A_p).\ \forall w' \in \mathcal{L}^{\between}(A_q) : \Big(((w'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}\Big) \longrightarrow$
$\Big(\{(x'{\downarrow_!}){\downarrow_{\{p,q\}}} \mid w'x' \in \mathcal{L}^{\between}(A_q)\}{\downarrow_{\cancel{!?}}} \subseteq \{x{\downarrow_?} \mid wx \in \mathcal{L}^{\between}(A_p)\}{\downarrow_{\cancel{!?}}}\Big)$

Now that our final version of the theorem is formalized, which we hope is correct, we will start proving and verifying it. The original proof of the theorem in (Di Giusto et al., 2024) is now unfortunately mostly obsolete, as the change of both the influenced language and subset condition affects the previous reasoning a great amount. We still need one more result from the source paper ; in the article, the following lemma is proposed and proven, albeit unverified :

Lemme 2.3.1 (Original Lemma 4.4)**.** *Let N be a network such that $\mathbb{C}_F = \mathbb{C}$, $G(N)$ is a tree, $q \in \mathbb{P}$, and $w \in \mathcal{L}^{\between}(A_q)$. Then there is an execution $w' \in \mathsf{E}(N_{\texttt{mbox}})$ such that $w'{\downarrow_q} = w$ and $w'{\downarrow_p} = \varepsilon$ for all $p \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$.*

This lemma is built upon the original version of the influenced language in Definition 2.3.1, which we have revised to Definition 2.3.3 for this work. This has implications for the proof in the paper as well, since it has the same oversight as the original influenced language. In fact, the proof can be corrected by adding the missing projection to ${\downarrow_{\{p,q\}}}$ to the parts of the proof where the influenced language is actively used. Apart from this, the original proof follows directly from the construction of the influenced language, and thus we will not rewrite the entire proof here, as adding the projections to the original proof[12] is the only change needed. To summarize, the proof for this lemma in (Di Giusto et al., 2024, Lemma 4.4) is only missing the projections to p and q in the proof when applying the influenced language. Since we are using this lemma in the proof for the adjusted main theorem, we will now provide some intuition for the reasoning of this lemma.

The proof works as follows. First, we obtain the unique[13] path from the root r to the peer q. Since $w \in \mathcal{L}^{\between}(A_q)$, either $q = r$ and we are done, or q has some parent h, and by the definition of the influenced language, there is some $w' \in \mathcal{L}^{\between}(A_h)$ such that $((w'{\downarrow_!}){\downarrow_{\{q,h\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}$. This reasoning can be repeated until the root is reached. Simultaneously, we accumulate all the visited

12. The original lemma is "Lemma 4.4" in (Di Giusto et al., 2024), and its proof is directly below.
13. Since the network is a tree, there can only be *exactly one* path between each peer pair, respectively.

words s.t. in the end we receive a word $w_r \dots w'w$, where the leftmost component is the root word, and the rightmost component is the word of q we started with. Through this construction, all sends happen in the correct order and all existing receptions also happen in the correct order. We must make the distinction of existing receptions, since we only visit the peers on the path, and so only these peers may perform actions in the accumulated word. However, peers can send messages to multiple children, so there may be some sends in the accumulation that are never received, but this is not a problem, as we are in the asynchronous case within this lemma. To summarize, we can accumulate this word by the construction of the influenced language, and this word is a valid Mailbox execution, because all sends happen before the receives (by construction), and all receives happen in the correct order.

We verified most steps of (the proof of) this lemma in Isabelle, but some minor results are missing. Most missing parts are fairly clear ; for example, the property that a unique path from the root to q exists. This should follow directly from the tree topology, as a tree is connected, and each peer has at most one parent. Having cautioned the reader that this lemma is not completely verified, we are going to assume this lemma to be correct and use it in the proof of the main theorem.

We would also like to revise the lemma's statement to make it even stronger. The execution w' is constructed by including only peers on the path from root to q, hence $w'{\downarrow_p} = \varepsilon$ for all $p \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$ holds as the lemma claims. Note that this only means that all direct children of q do not perform actions in w'. However, any further descendants of q (children of such p, and so on) are also not actors for any action in w', as they cannot be on the traversed path to q. In fact, no peer outside of those on the path can perform actions in w' themselves (they can still be sent to, however). Thus, we strengthen the last condition and use this slightly changed version of the original lemma for the remainder of this work [14] :

Lemme 2.3.2 (Strengthened "Lemma 4.4")**.** *Let N be a network such that $\mathbb{C}_F = \mathbb{C}$, $G(N)$ is a tree, $q \in \mathbb{P}$, and $w \in \mathcal{L}^{\between}(A_q)$. Let $q, \dots, r$ be the unique [15] path from q to the root r, and let $B \subseteq \mathbb{P}$ denote the set of peers involved in that path, i.e. $q, \dots, r \in B$. Then there is an execution $w' \in \mathsf{E}(N_{\texttt{mbox}})$ such that $w'{\downarrow_q} = w$ and $w'{\downarrow_p} = \varepsilon$ for all $p \in B$.*

Next, we will present the adjusted proof for the main theorem. The Isabelle code for this can be found in Appendix .2.4, but we will detail the proof here in a more regular format. Our new proof is almost completely different in execution from the one presented in the source paper, due to the subset condition changing, with the exception of one case of the backwards direction.

To prove the direction that assumes synchronizability and shows our two conditions hold, we need to define two constructions first. In the following, we will quickly explain the main idea and show pseudocode for those constructions. The Isabelle code for them - which is quite similar in syntax to functional programming languages - can be referenced in Appendices .2.1 and .2.2. respectively. Let $w[i]$ denote word w at index i. It is important to note that the following two constructions implicitly assume that each message is unique, i.e., there is no such case where the message a is sent between more than one unique pair of peers. Additionally, within the language of one peer, all messages are also distinct ; this follows from the previous point. [16]

14. In Isabelle, we currently only use this strengthened version implicitly, but it needs to be changed to the explicit stronger version there as well, to allow for full verification.
15. Since the network is a tree.
16. If messages are not unique, they can be made unique, e.g., by adding subscripts that tag the peers involved to the message.

For => 1., we use a construction which mixes two words $w \in \mathcal{L}^{\between}(A_p)$ and $w' \in \mathcal{L}^{\between}(A_q)$, where p is q's direct child, in the following manner :

Exemple 2.3.1 – (mix_pair Pseudocode)

```
Iterate through $w$ with (index $i$) and $w'$ (with index $j$).
In the following cases, let $a$ denote an arbitrary message (from $q$ to $p$).
Cases :
  • $w'[j] = !a^{q\to p}$ :
      if $w[i] = ?a^{q\to p}$ then put $!a^{q\to p}, ?a^{q\to p}$and increment both $i$ and $j$ by one, respectively;
      else put $w[i]$ and increment $i$ until $w[i] = ?a^{q\to p}$.
  • $w'[j] \neq !a^{q\to p}$ :
      put $w'[j]$ and increment $j$.
```

We denote with (mix_pair (w', w)) a call to the function as introduced above over w' and w. Intuitively, whenever a send from q to p occurs in w', we append the matching receive of p for that message directly afterwards. All actions not concerning p and q, e.g., sends and receives from q involving other peers, or sends from p to its children, are appended s.t. that their original order is kept. In particular, (mix_pair $(w', w))\downarrow_q = w'$ and (mix_pair $(w', w))\downarrow_p = w$ hold. We alternate sends and receives of p and q. If the returned word is part of an execution e, s.t. $e \in \mathsf{E}(N_{\texttt{mbox}})$ and e contains no other actions of p or q, then a synchronous execution with the same trace as e would also satisfy $e\downarrow_q = w'$ and $\downarrow_p = w$.

For => 2., we utilize mix_pair as explained above to construct a slightly different version of a mix between two words.

Définition 2.3.8 (mix_shuf)**.** Let $p, q, x \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}, \mathbb{P}^x_{send} = \{p\}$, let $(xs{\cdot}!a^{p\to x}{\cdot}?b^{q\to p} \cdot ys) = w \in \mathcal{L}^{\between}(A_p)$ and $(xs{\cdot}?b^{q\to p}{\cdot}!a^{p\to x} \cdot ys) = w' \in \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$ s.t. $w' \sqcup\!\sqcup_? w$. Let $(as{\cdot}!b^{q\to p} \cdot bs) = w'' \in \mathcal{L}^{\between}(A_q)$.
Then we construct
mix_shuf $(w'', w', w) :=$ ((mix_pair $(as, xs)){\cdot}!b^{q\to p}{\cdot}!a^{p\to x}{\cdot}?b^{q\to p} \cdot$ (mix_pair (bs, ys)))

As intuition, w and w' in this construction are equal, except for one input-output pair in the middle, i.e., exactly one shuffle occurred from w to w'. We can infer (mix_shuf $(w'', w', w))\downarrow_q = w''$, by the construction of (mix_pair (w'', w)). We can observe that (mix_shuf (w'', w', w)) performs (mix_pair(w'', w)), except for the part where w' and w differ. However, (mix_shuf $(w'', w', w))\downarrow_p = w$, since the ordering is kept. To showcase the effect of this construction, consider an execution e, s.t. (mix_shuf (w'', w', w)) occurs as a whole in $e \in \mathsf{E}(N_{\texttt{mbox}})$ and $e\downarrow_q = w''$ and $\downarrow_p = w$. Due to the placement of $!b^{q\to p}$ before $!a^{p\to x}{\cdot}?b^{q\to p}$, a synchronous execution e' with the same trace as e would now satisfy $\downarrow_p = w'$. Otherwise, not every send would immediately be followed by the matching receive, and e' wouldn't be synchronous. We will explain how this and the first construction are relevant when they occur in the proof, as without their context, they may seem quite arbitrary.

As hinted at before, most of our proof execution is entirely different from the source, but some of the proof structure and ideas were extremely helpful in our formalization. The original proof uses a similar approach in the first direction of the theorem, but their construction is not sufficiently specific, and thus, the claim is not proven in all cases of the arbitrary words they fix.

For the first case of the second direction, we operate with the same idea, but due to the subset condition becoming stricter, this proof part also had to become more elaborate.

Apart from the condition changes needed, the formalization of the original proof was not fully formal either way. In some of the original proof, cases were missed or simply not explicitly mentioned, although they are formally required [17]. And of course in other parts of the proof, we have changed the formalization. Because of this, most of the original proof is obsolete regardless of the exact reasons ; thus, we will not detail all formalization errors or small oversights of original proof, and instead provide and explain our new version below. In general, we have made some of the implicit steps from the source explicit and provided definitions for some constructions used. To conclude, for the first direction, the proof idea is similar to the original one, although the execution is different and more detailed. For the second direction, the case distinction and the proof for the second case remain true to the original, but the first case is completely changed due to the change in the subset condition.

Théorème 2.3.3 (Final Mailbox Synchronizability Theorem)**.** *Let N be a network such that $\mathbb{C}_F = \mathbb{C}$ and $G(N)$ is a tree. Then $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$ iff, for all $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$, we have the subset condition (1) and the shuffled language condition (2) :*

(1) $\forall w \in \mathcal{L}^{\between}(A_p).\ \forall w' \in \mathcal{L}^{\between}(A_q) : \Big(((w'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}\Big) \longrightarrow$
$\Big(\{(x'{\downarrow_!}){\downarrow_{\{p,q\}}} \mid w'x' \in \mathcal{L}^{\between}(A_q)\}{\downarrow_{\cancel{!?}}} \subseteq \{x{\downarrow_?} \mid wx \in \mathcal{L}^{\between}(A_p)\}{\downarrow_{\cancel{!?}}}\Big)$

(2) $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$

Proof of Final Theorem.

⇒ Assume $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$. We have to show that

1. $\forall w \in \mathcal{L}^{\between}(A_p).\ \forall w' \in \mathcal{L}^{\between}(A_q) : \Big(((w'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}\Big) \longrightarrow$
$\Big(\{(x'{\downarrow_!}){\downarrow_{\{p,q\}}} \mid w'x' \in \mathcal{L}^{\between}(A_q)\}{\downarrow_{\cancel{!?}}} \subseteq \{x{\downarrow_?} \mid wx \in \mathcal{L}^{\between}(A_p)\}{\downarrow_{\cancel{!?}}}\Big)$

and 2. $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$ for all $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$.

1. Assume $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$. To show : the subset condition holds.
   Proof : Fix w, w' s.t. $\Big(((w'{\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = (w{\downarrow_?}){\downarrow_{\cancel{!?}}}\Big)$ as in the precondition of the subset condition ; fix x' s.t. $w'x' \in \mathcal{L}^{\between}(A_q)$. We now have to show, that there exists some x with $wx \in \mathcal{L}^{\between}(A_p)$ and $(((w'x'){\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = ((wx){\downarrow_?}){\downarrow_{\cancel{!?}}}$, which is the subset condition but rewritten on words instead of subsets. With these assumptions, we will now do a case distinction over whether q is the root or not.
   If q is the root, then $w'x'$ consists only of sends and hence $w'x' \in \mathsf{E}(N_{\texttt{mbox}})$, because $w'x'$ cannot contain receptions, and sends cannot be blocked without receptions being present.
   Then obtain $w_{mix} = (\text{mix\_pair } (w', w))$ and append x' to this to receive the execution $e = w_{mix} \cdot x' \in \mathsf{E}(N_{\texttt{mbox}})$. This is an execution, since both w' and x' cannot receive anything from any other peer (as q is the root), and w' provides all necessary sends and in the correct order for w, by assumption. Then using synchronizability, obtain a synchronous execution e' for the trace of the constructed execution, s.t. $e{\downarrow_!} = e'{\downarrow_!}$

17. For example, the first direction of the proof does a case distinction on the possibilities of the shuffle. They only prove the claim for the case where the word can be decomposed in a certain way, which does not necessarily generalize to all words (which are needed to prove the claim, however).

and $e' \in \mathsf{E}(N_{\mathtt{sync}})$. Then we can obtain x where $e'{\downarrow_p} = w \cdot x$. This is possible, since p only receives from q and by the construction of w_{mix}, each send of q to p in w' is directly followed by the reception in w. If x' now contains any further sends to p, p will perform the receptions in e' after performing the actions in w. Thus, $e'{\downarrow_p}$ consists of exactly w and some x as a peer word of p, which receives $w' \cdot x'$ exactly, and the subset condition is proven if q is the root.

This leaves open the case where q is some node in the tree with some parent r. The construction process of the two executions is analogous, but since q is not the root, finding e is more complicated. First, we obtain some w'' s.t. $\Big(((w''{\downarrow_!}){\downarrow_{\{q,r\}}}){\downarrow_{\cancel{!?}}} = ((w' \cdot x'){\downarrow_?}){\downarrow_{\cancel{!?}}}\Big)$, which must exist since $(w' \cdot x')\mathcal{L}^{\between}(A_q)$ by assumption.

Then, we apply Lemma 2.3.2 to obtain execution e for w''. By construction, p and q perform no actions here and q gets sent the necessary sends to perform $w' \cdot x'$, while p gets sent nothing, and p and q send and receive nothing in e. We could append $w' \cdot x'$ directly to e, this is possible since w'' provides all sends for $w' \cdot x'$ and $e{\downarrow_r} = w''$. Instead, we first want to "mix" w' and w to construct a certain appendix for e. So, we obtain $w_{mix} = (\text{mix\_pair}\ (w'', w))$ and in turn $e' = e \cdot w_{mix} \cdot x'$: this is possible, since e provides all the necessary sends for $w' \cdot x'$, and w_{mix} is constructed s.t. each reception of w has the corresponding send of q directly before it (and by assumption w' and w have matching sends and receptions). Furthermore, x' can be performed at the end, since $w_{mix}{\downarrow_q} = w'$ and thus q is still in the position to perform x' (peers cannot block each other's actions directly, so whether w is performed inbetween e and x' or not has no impact on q). Since the network is synchronizable, obtain the synchronous execution e'' with the same trace as e'. By construction of e', we can infer that both e'' and e' projected to only actions between p and q, before x' (i.e. sends from q to p and receptions of these sends) are equal. Since w_{mix} performs each send of q directly before the receive of p (i.e., simulating the synchronous execution between these two peers), e'' must also contain w in its execution, otherwise a different trace is performed. By construction, then $e''{\downarrow_p} = w \cdot x$ for some x, s.t. $wx \in \mathcal{L}^{\between}(A_p)$ and $(((w'x'){\downarrow_!}){\downarrow_{\{p,q\}}}){\downarrow_{\cancel{!?}}} = ((wx){\downarrow_?}){\downarrow_{\cancel{!?}}}$.

In e'', the projection on q may not be $w'x'$ (if w' contains any receptions, these will get pulled further left, where the respective sends to q are located by construction of e). The important part is, however, that $(e'{\downarrow_q}){\downarrow_!} = (e{\downarrow_q}){\downarrow_!}$, i.e. the sends of this projection are the same as the sends in $w'x'$, which is exactly what is needed for p (p's actions are not influenced by what receptions or sends to different children q performs). Finally, we have found a matching x for our arbitrary x' and thus shown the subset condition also for the case where q is not the root.

2. By definition, $\mathcal{L}^{\between}(A_p) \subseteq \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$. Assume $v' \in \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$. We have to show that $v' \in \mathcal{L}^{\between}(A_p)$. Since $v' \in \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$, then by definition of the shuffled language, there is some v such that $v \in \mathcal{L}^{\between}(A_p)$ and $v' \sqcup\!\sqcup_? v$. We prove $v' \in \mathcal{L}^{\between}(A_p)$ by induction over the shuffle $v' \sqcup\!\sqcup_? v$. There are three cases : either no shuffle occurred, exactly one occurred, or v' was received through transitivity of the shuffle relation from Definition 2.2.9. The first and last case are both trivial, and Isabelle was able to prove them directly, so the handwritten proof of these will be left as a small exercise to the reader. We will now detail the only complicated case, where exactly one shuffle occurred, so

v and v' can be decomposed into some $(xs{\cdot}?b^{q\to p}{\cdot}!a^{p\to x}\cdot ys) = v' \in \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$ and $(xs{\cdot}!a^{p\to x}{\cdot}?b^{q\to p}\cdot ys) = v \in \mathcal{L}^{\between}(A_p)$.
We can infer that p is a node in the tree with some parent q, as the root cannot receive messages, ruling out any non-trivial shuffles. Then, obtain $vq \in \mathcal{L}^{\between}(A_q)$, s.t. $((vq\downarrow_!)\downarrow_{\{p,q\}})\downarrow_{\cancel{!?}} = (v\downarrow_?)\downarrow_{\cancel{!?}}$, by applying the definition of the influenced language. Similar to the previous proof part, we now do a case distinction over whether q is the root or some node with parent r.
Case : q is the root.
Then vq has no receives and consequently vq is a valid execution in $\mathsf{E}(N_{\texttt{mbox}})$. Next, we construct $w_{mix} = (\text{mix\_shuf}\,(vq, v', v))$, i.e. mix vq with v, which then is also an execution $\in \mathsf{E}(N_{\texttt{mbox}})$ by construction, as explained beneath Definition 2.3.8. Using synchronizability, we then obtain the synchronous execution $e' \in \mathsf{E}(N_{\texttt{sync}})$ with the same trace as w_{mix}. By construction, $e'\downarrow_p = v'$ and consequently $v' \in \mathcal{L}^{\between}(A_p)$, otherwise e' is not an execution.
Case : q is a node with parent r.
Then obtain $vr \in \mathcal{L}^{\between}(A_r)$ which provides exactly matching sends for vq using the definition of the influenced language. Then apply Lemma 2.3.2 to vr and receive execution $e \in \mathsf{E}(N_{\texttt{mbox}})$ which contains no actions of p or q, respectively, by construction. Next, obtain $w_{mix} = (\text{mix\_shuf}\,(vq, v', v))$. Then, $e \cdot w_{mix} \in \mathsf{E}(N_{\texttt{mbox}})$, since both p and q are in their initial states after e, and e provides all sends for q in the correct order and w_{mix} keeps all sends of q before the receives of p. Now we again use synchronizability to receive $e' \in \mathsf{E}(N_{\texttt{sync}})$ with $e'\downarrow_! = (e \cdot w_{mix})\downarrow_!$. By construction of w_{mix} and e', we know that $e'\downarrow_p = v'$. Consequently, $v' \in \mathcal{L}^{\between}(A_p)$ must hold, or e' is no execution.
In all cases and for an arbitrary v', we have shown that $v' \in \mathcal{L}^{\between}(A_p)$, so we can conclude that $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$.

$\Leftarrow$ Assume that the subset condition (as in Definition 2.3.7) and $\mathcal{L}^{\between}(A_p) = \mathcal{L}^{\between}_{\sqcup\!\sqcup}(p)$ hold for all $p, q \in \mathbb{P}$ with $\mathbb{P}^p_{send} = \{q\}$. We have to show that $\mathsf{T}(N_{\texttt{mbox}}) = \mathsf{T}(N_{\texttt{sync}})$.

$w \in \mathsf{T}(N_{\texttt{mbox}})$ **:** Let w' be the word obtained from w by adding the matching receive action directly after every send action. [18] We show that $w' \in \mathsf{E}(N_{\texttt{mbox}})$, by an induction on the length of w.

Base Case : If $w = !a^{q\to p}$, then $w' = !a^{q\to p}?a^{q\to p}$. Since $w \in \mathsf{T}(N_{\texttt{mbox}})$, A_q is able to send $!a^{q\to p}$ in its initial state within the system $N_{\texttt{mbox}}$. Then $!a^{q\to p} \in \mathcal{L}^{\between}_{!}(A_q)$. We apply the subset condition to $\epsilon \in \mathcal{L}^{\between}(A_p)$ and $\epsilon \in \mathcal{L}^{\between}(A_q)$ and the suffix $!a^{q\to p}$. Since $\epsilon{\cdot}!a^{q\to p} = !a^{q\to p} \in \mathcal{L}^{\between}(A_q)$, p must then be able to perform some word $x \in \mathcal{L}^{\between}(A_q)$ from its initial state, which has $?a^{q\to p}$ as the only reception. We only know about x that it contains exactly this input, but it may also contain sends. This is undesired, given that we want the execution where p only receives this, so we fully shuffle x : We know that $?a^{q\to p} = x\downarrow_?$; let $ys = x\downarrow_!$ be the possibly existing sends of x, then obtain $(?a^{q\to p}\cdot ys) \sqcup\!\sqcup_? x$, by repeatedly shuffling $?a^{q\to p}$ further left until it cannot be shuffled anymore. Then, $?a^{q\to p}\cdot ys \in \mathcal{L}^{\between}(A_q)$ by the shuffled language condition. Thus, $?a^{q\to p} \in \mathcal{L}^{\between}(A_q)$, as all states of our network are accepting. Then $w' \in \mathsf{E}(N_{\texttt{mbox}})$.

18. A function in Isabelle that does this named *add_matching_recvs*, can be found in the Appendix .2.

Inductive Step : If $w = v\cdot !a^{q\to p}$ with $v\cdot !a^{q\to p} \in \mathsf{T}(N_{\mathtt{mbox}})$, then $w' = v'\cdot !a^{q\to p}.?a^{q\to p}$. By induction, $v' \in \mathsf{E}(N_{\mathtt{mbox}})$. To show : $w' \in \mathsf{E}(N_{\mathtt{mbox}})$. This can be decomposed into the following subgoals :

(1) $(v'\downarrow_q)\cdot !a^{q\to p} \in \mathcal{L}^{\between}(A_q)$,

(2) from this $v'\cdot !a^{q\to p} \in \mathsf{E}(N_{\mathtt{mbox}})$.

(3) Then $(v'\cdot !a^{q\to p})\downarrow_p\cdot ?a^{q\to p} \in \mathcal{L}^{\between}(A_p)$,

(4) hence $w' \in E_{mbox}$.

—

Proof of (1).

Case 1 : q is the root. Then $v'\downarrow_q = v\downarrow_q$, otherwise q would be receiving messages despite being the root. Since w is a valid trace by assumption and consequently q must be able to perform all its actions in w, we can conclude $w\downarrow_q = w'\downarrow_q = (v\downarrow_q)\cdot !a^{q\to p} \in \mathcal{L}^{\between}(A_q)$.

Case 2 : q is a node with parent r. Since w is a valid trace, there exists some $wq \in \mathcal{L}^{\between}(A_q)$ and $e \in \mathsf{E}(N_{\mathtt{mbox}})$, s.t. $e\downarrow_! = w$ and $e\downarrow_q = wq$. By construction, $(wq)\downarrow_! = (v'\downarrow_q\cdot !a^{q\to p})\downarrow_!$ and in general $(v'\downarrow_q\cdot !a^{q\to p})\downarrow_? = (v'\downarrow_q)\downarrow_?$, since $!a^{q\to p}$ is not an input. We infer that $(wq)\downarrow_?$ is a prefix of $(v'\downarrow_q)\downarrow_?$. If we assume the contrary, then the ordering of receives is different in v' and wq, but since v' receives each send directly afterwards, that would mean that in wq, q receives something that is not in the first position of the buffer, contradicting the assumption that e is an execution.

Moreover, $((wq)\downarrow_?)\downarrow_{\cancel{!?}}$ must be a prefix of $((w\downarrow_!)\downarrow_{\{r,q\}})\downarrow_{\cancel{!?}}$, since $((w\downarrow_!)\downarrow_{\{r,q\}})\downarrow_{\cancel{!?}} = ((w'\downarrow_?)\downarrow_{\{r,q\}})\downarrow_{\cancel{!?}}$ by construction of w' and we just showed that $(wq)\downarrow_?$ is a prefix of $(v'\downarrow_q)\downarrow_?$. Since $w'\downarrow_? = w = e\downarrow_?$, r sends the same messages to q in both e and w', but wq may receive only a prefix of them.

As a reminder, q is a node with parent r, and $wq \in \mathcal{L}^{\between}(A_q)$, so there exists $wr' \cdot x' \in \mathcal{L}^{\between}(A_r)$ such that

$$((wr' \cdot x')\downarrow_!)\downarrow_{\{r,q\}} = (w\downarrow_!)\downarrow_{\{r,q\}} \text{ and } ((wq)\downarrow_?)\downarrow_{\cancel{!?}} = (((wr')\downarrow_!)\downarrow_{\{r,q\}})\downarrow_{\cancel{!?}}.$$

Intuitively, we obtain all sends from r to q, and decompose these into wr' and x', so that wr' provides exactly the sends that wq receives from q (since wq may not receive all sends).

By the subset condition, since r can perform wr' with the x', q must also be able to perform some x after wq s.t.

$$wq \cdot x \in \mathcal{L}^{\between}(A_q) \text{ and } ((wq \cdot x)\downarrow_?)\downarrow_{\cancel{!?}} = (((wr' \cdot x')\downarrow_!)\downarrow_{\{q,r\}})\downarrow_{\cancel{!?}}.$$

We only know about x that it contains exactly the needed receptions, but it may also contain sends. This is undesired, so we fully shuffle it : let $xs = x\downarrow_?$ and $ys = x\downarrow_!$ be the sends/receives of x, then $(xs \cdot ys) \sqcup_? x$.[19] Then we prepend wq to this shuffle, to obtain $wq\cdot xs\cdot ys \sqcup_? wq\cdot xs$. Then, $wq\cdot xs\cdot ys \in \mathcal{L}^{\between}(A_q)$ by the shuffled language condition. Thus, $wq \cdot xs \in \mathcal{L}^{\between}(A_q)$ with projections matching

19. Any word can be "fully" shuffled, until all receives come first, followed by all sends of the given word. We have proven this in Isabelle, but it follows from Definition 2.2.9 relatively directly.

those of $w'{\downarrow_q}$, since all states are accepting and thus any prefix of a valid word is also valid.
We can now infer that $((v'{\downarrow_q}){\cdot}!a^{q\rightarrow p}) \sqcup\!\sqcup_? (wq \cdot xs)$ must hold ; otherwise some send/receive ordering would contradict the trace :

> Let $!x$ be an arbitrary send and $?y$ a receive. Write $!x\ <?y$ in a word w to denote that $!x$ occurs earlier (further left) in w than $?y$. Then there is some output–input pair $!x, ?y$ such that $!x\ <?y$ in $((v'{\downarrow_q}){\cdot}!a)$ but $?y\ <!x$ in $(wq{\cdot}xs)$, meaning that a conflicting shuffle from wq to $(v'{\downarrow_q}{\cdot}a)$ occurred. By construction of w', we have $!x\ <?y$ in w, but $wq \cdot xs$ has $?y\ <!x$, requiring y to be sent (and to be received) before $!x$ can be sent ; hence $wq \cdot xs$ cannot be part of an execution that produces trace w. However, $(e \cdot xs)$ is a valid execution (since xs only receives elements that are already in q's buffer after e and q cannot be blocked by any other peer at the end of e), and $(e \cdot xs){\downarrow_!} = e{\downarrow_!} = w = w'{\downarrow_!}$ by our assumptions. But $(e \cdot xs){\downarrow_!}$ has $?y\ <!x$ while w has $!x\ <?y$, yielding a different trace than assumed $\lightning$.

So $((v'{\downarrow_q}){\cdot}!a^{q\rightarrow p}) \sqcup\!\sqcup_? (wq \cdot xs)$ holds and in turn by the shuffled language condition, $(v'{\downarrow_q}{\cdot}!a^{q\rightarrow p}) \in \mathcal{L}^{\between}_{\sqcup\!\sqcup}(q) = \mathcal{L}^{\between}(A_q)$ holds as well. This completes (1).
From this, (2) follows immediately, since sends cannot be blocked by other peers and we have shown that q can perform $(v'{\downarrow_q}{\cdot}!a^{q\rightarrow p}) \in \mathcal{L}^{\between}(A_q)$, hence $v'{\cdot}!a^{q\rightarrow p} \in \mathsf{E}(N_{\mathtt{mbox}})$. Using the subset condition and shuffling out the possible sends analogously to before [20], (3) also follows. [21] Finally, (4) holds because p's buffer cannot contain $a^{q\rightarrow p}$, and receptions also cannot be blocked by other peers. This concludes the inductive step, we have shown that $w' \in E_{mbox}$. Since $w'{\downarrow_!} = w$, then $w \in \mathsf{E}(N_{\mathtt{sync}}) = \mathsf{T}(N_{\mathtt{sync}})$.

$w \in \mathsf{T}(N_{\mathtt{sync}})$ **:** For every output in w, $N_{\mathtt{sync}}$ was able to send the respective message and directly receive it. Let w' be the word obtained from w by adding the matching receive action directly after every send action. Then $N_{\mathtt{mbox}}$ can simulate the run of w in $N_{\mathtt{sync}}$ by sending every message first to the mailbox of the receiver and then receiving this message. Then $w' \in \mathsf{E}(N_{\mathtt{mbox}})$ and, thus, $w = w'{\downarrow_!} \in \mathsf{T}(N_{\mathtt{mbox}})$.

□

As a note, the last case ("$w \in \mathsf{T}(N_{\mathtt{sync}})$") is directly cited from (Di Giusto et al., 2024), but we have omitted quotation marks for visual clarity. All other cases have changed considerably through our work.

2.4 Conclusion

This work has provided another case study of why verification is so instrumental. We have verified (except for some remaining obvious helper lemmas), that synchronizability for trees is in fact decidable, as was the claim in (Di Giusto et al., 2024). Through the mechanization of the proposed algorithm and formalization in Isabelle, several oversights in the original theorem came to light and could be corrected in multiple iterations. We have rewritten the original theorem and

20. Same construction as in the base case or with $wr'x'$ and x, to receive xs.
21. We use the same reasoning as in the steps with wr' and x' where we received xs.

its proof and have thus provided a corrected algorithm to determine synchronizability of a tree, and provided a largely complete Isabelle formalization for this as well.

Through the process of mechanizing the proofs of (Di Giusto et al., 2024, Section 4) in Isabelle, we were able to catch some oversights and provide improvements. Most notably, we adjusted the subset condition and consequently the entire theorem and its proof to now formally reflect the intention of the authors. This led to us rewriting large parts of the original proof (Di Giusto et al., 2024, Theorem 4.5) entirely, as was necessary due to our changes. In addition to delivering a new rendition of the theorem and its proof by hand, we verified the majority of our lemmas and theorems in Isabelle. The only lemmas that remain are simple enough so that we can consider the main theorem verified. Each unproven lemma in the code has some comments around it, explaining the intuition behind it and often the proof idea as well. To conclude, we have learned to apply Isabelle, attempted to verify the claims in (Di Giusto et al., 2024), leading us to discover some faults, which we then corrected and mostly succeeded in verifying. With our mechanization, and some perspectives we will give for improving it, we hope to inspire other researchers in the realm of distributed systems to verify their (or others') works as well.

2.5 Perspectives

The perspectives are twofold ; we will first detail those concerning our and future Isabelle formalizations, and then those concerning the results of this work in general.

As previously mentioned, there still remain some small lemmas to be proven to fully verify our claims. However, all of these are intuitive and trivial to do on paper, but with the current formalization in Isabelle and the time constraints, this was not yet possible for us to complete. Several missing lemmas concern the prefixes of words in a language, also being in that language. Since we focus solely on automata where each state is final, this is trivially true. In other words, if we know that an automaton accepts a word $w = u \cdot v$, then we can also stop after reading u and still have an accepting run because each step starts and ends in an accepting state by assumption. The rest of the remaining lemmas are similar in complexity, thus proving them will take some additional time, mostly because of the current state of the Isabelle formalization.

As alluded to, the current Isabelle formalization is not ideal for proving all needed lemmas. The issue lies in the encoding of the tree topology, which does not directly yield any information about the peers and their messages (and consequently, whether they are the root or a node). This adds complexity to any proof in need of these relations, which is why some intuitive lemmas are still unproven. In other words, from the tree topology as it is currently defined in our Isabelle formalization, we cannot directly infer any information about the peers, especially concerning their messages or place in the topology. This formalization was present before this work began ; thus, we continued with it and built the rest of the Isabelle framework around it. Consequently, it is now difficult to adapt this tree topology definition without affecting the rest of the code. We have the following dilemma : as explained in the previous paragraph, not all lemmas can be easily proven because the tree topology's formalization in Isabelle is not ideal ; however, changing this formalization means having to, in the worst case, rewrite or reprove parts of definitions, lemmas, etc., where the tree topology currently occurs. In short, it would be best to formalize a stronger tree topology in Isabelle, which can then be used for the missing proof steps in some of the unproven lemmas.

We have described how the initial formalization may be improved, but our creations are also open to critique. In part due to the tree topology formalization, we had some issues formalizing the construction used in the proof of the original Lemma 4.4 in the source paper (see Definition 2.3.1). We used an overcomplicated and slightly inefficient construct[22]. While we have provided a possible improvement[23], we were not able to move to this version fully, in time. In general, there are surely many formalizations that we would now do differently with our newly gathered knowledge, and which can surely be improved by other Isabelle enthusiasts as well.

With or without an improved Isabelle formalization, another avenue would be to formalize the counterexamples we have found in Isabelle. We have encountered similar difficulties as in the process of proving the remaining helper lemmas, which is why the formalization was not completed. However, formalizing the counterexamples as well would contribute to the goal of fully verifying this entire project.

Lastly, this result could also possibly be extended to both Peer-to-Peer networks as well as networks with different variants of tree topologies, as was previously mentioned in the source paper. To reiterate their points, networks consisting of multiple or reversed trees may be able to be investigated in terms of their synchronizability, with a similar approach to our end result here. Since we have found a procedure to decide the synchronizability of one tree, one could apply this to each tree of a multitree network. For both of these variants, our requirement of having pairwise unique paths between nodes is still satisfied. To extend this procedure to reversed trees, some of the procedure would likely need to be reversed to account for the change in properties. Additionally, the authors claim that the set of executions of a network with tree topology should be equal for Mailbox and Peer-to-Peer systems. This remains to be formally proven and hopefully verified, however. In short, we have found the synchronizability problem to be decidable for tree topologies, and propose a similar approach for investigating the decidability of this problem in related contexts. (Di Giusto et al., 2024)

We are confident that we have achieved a good foundation of the last perspective proposed in (Di Giusto et al., 2024), namely the mechanization of the paper's proofs in Isabelle, despite its challenging nature. With this foundation and the future prospects detailed above, we hope to inspire the reader to continue improving this project or to delve into their own verification endeavors.

The entirety of this work has been an excellent case study as to why proper verification is so important ; regardless of how intuitive the reasoning may seem, formal correctness should always be respected and ensured.

22. The inductive "concat_infl" and all related lemmas.
23. In the code, above "concat_infl".

Références

Archive of formal proofs. (2004). Consulté sur https://isa-afp.org

Basu, S., & Bultan, T. (2016, décembre). On deciding synchronizability for asynchronously communicating systems. *Theoretical Computer Science*, *656*, 60–75. Consulté le 2018-06-13, sur http://www.sciencedirect.com/science/article/pii/S0304397516305102 doi: 10.1016/j.tcs.2016.09.023

Bollig, B., Di Giusto, C., Finkel, A., Laversa, L., Lozes, É., & Suresh, A. (2021). A unifying framework for deciding synchronizability. In S. Haddad & D. Varacca (Eds.), *32nd international conference on concurrency theory, CONCUR 2021, august 24-27, 2021, virtual conference* (Vol. 203, pp. 14 :1–14 :18). Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi: 10.4230/LIPICS.CONCUR.2021.14

Bouajjani, A., Enea, C., Ji, K., & Qadeer, S. (2018). On the completeness of verifying message passing programs under bounded asynchrony. In H. Chockler & G. Weissenbacher (Eds.), *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* (Vol. 10982, pp. 372–391). Springer. doi: 10.1007/978-3-319-96142-2_23

Brand, D., & Zafiropulo, P. (1983, avril). On Communicating Finite-State Machines. *Journal of the ACM*, *30*(2), 323–342. doi: 10.1145/322374.322380

Buyse, M., & Jaskolka, J. (2019, August). Communicating concurrent kleene algebra for distributed systems specification. *Archive of Formal Proofs*. (https://isa-afp.org/entries/C2KA_DistributedSystems.html, Formal proof development)

Charron-Bost, B., Mattern, F., & Tel, G. (1996). Synchronous, asynchronous, and causally ordered communication. *Distributed Comput.*, *9*(4), 173–191. doi: 10.1007/S004460050018

Chevrou, F., Hurault, A., & Quéinnec, P. (2016). On the diversity of asynchronous communication. *Formal Aspects Comput.*, *28*(5), 847–879. doi: 10.1007/S00165-016-0379-X

Constraints & applications. (s. d.). Consulté sur https://ca.i3s.unice.fr/

Di Giusto, C., Ferré, D., Laversa, L., & Lozes, É. (2023). A partial order view of message-passing communication models. *Proc. ACM Program. Lang.*, *7*(POPL), 1601–1627. doi: 10.1145/3571248

Di Giusto, C., Laversa, L., & Lozes, É. (2020). On the k-synchronizability of systems. In J. Goubault-Larrecq & B. König (Eds.), *Foundations of software science and computation structures - 23rd international conference, FOSSACS 2020, held as part of the european joint conferences on theory and practice of software, ETAPS 2020, dublin, ireland, april 25-30, 2020, proceedings* (Vol. 12077, pp. 157–176). Springer. doi: 10.1007/978-3-030-45231-5_9

Di Giusto, C., Laversa, L., & Peters, K. (2024). Synchronisability in mailbox communication. *arXiv preprint arXiv :2411.14580.*

Fiedler, B., & Traytel, D. (2020, July). A formal proof of the chandy–lamport distributed snapshot algorithm. *Archive of Formal Proofs*. (https://isa-afp.org/entries/Chandy_Lamport.html, Formal proof development)

Finkel, A., & Lozes, E. (2017). Synchronizability of Communicating Finite State Machines is not Decidable. In I. Chatzigiannakis, P. Indyk, F. Kuhn, & A. Muscholl (Eds.), *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)* (Vol. 80, pp. 122 :1–122 :14). Dagstuhl, Germany : Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Consulté le 2021-08-06, sur http://drops.dagstuhl.de/opus/volltexte/2017/7402 (ISSN : 1868-8969) doi: 10.4230/LIPIcs.ICALP.2017.122

Genest, B., Kuske, D., & Muscholl, A. (2006, juin). A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, *204*(6), 920–956. Consulté le 2018-06-13, sur http://www.sciencedirect.com/science/article/pii/S0890540106000290 doi: 10.1016/j.ic.2006.01.005

Giusto, C. D., Laversa, L., & Lozes, É. (2023). Guessing the buffer bound for k-synchronizability. *Int. J. Found. Comput. Sci.*, *34*(8), 1051–1076. doi: 10.1142/S0129054122430018

Kettler, N. (2025, août). *nicolell/isabelle-sync* [Isabelle]. Consulté sur https://github.com/nicolell/isabelle-sync

Kuske, D., & Muscholl, A. (2021). Communicating automata. In J. Pin (Ed.), *Handbook of automata theory* (pp. 1147–1188). European Mathematical Society Publishing House, Zürich, Switzerland. doi: 10.4171/AUTOMATA-2/9

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, *21*(7), 558–565. doi: 10.1145/359545.359563

Nipkow, T. (2023). *Concrete semantics : With isabelle/hol.* Consulté sur http://www.concrete-semantics.org/slides-isabelle.pdf

Nipkow, T., & Klein, G. (2023). *Concrete semantics : With isabelle/hol.* Springer International Publishing. Consulté sur http://www.concrete-semantics.org/concrete-semantics.pdf

Nipkow, T., Paulson, L. C., & Wenzel, M. (s. d.). A proof assistant for higher-order logic. *13.03.2025*. Consulté sur https://isabelle.in.tum.de/doc/tutorial.pdf

Nipkow, T., Wenzel, M., Sternagel, C., & Eberl, M. (s. d.). *Hol/library/sublist.thy.* Consulté sur https://isabelle.in.tum.de/dist/library/HOL/HOL-Library/Sublist.html

Paulson, L. C. (1994). *Isabelle : A generic theorem prover* (Vol. 828). Springer. Consulté sur https://isabelle.in.tum.de doi: 10.1007/BFb0030541

Wenzel, M. (s. d.). The isabelle/isar reference manual. *13.03.2025*. Consulté sur https://isabelle.in.tum.de/doc/isar-ref.pdf

Annexes

.1 Internship Specifications

This internship was for a full duration of six months at the I3S. It was supervised by Prof. Di Giusto, Prof. Peters and Prof. Esparza; the first two of which are co-authors of the paper which forms the basis for this work (Di Giusto et al., 2024). Prof. Di Giusto is a professor at the I3S, in the Team MDSC/C&A ((*Constraints & Applications*, s. d.)), who focus their research on applying formal methods to topics of different kinds.

.2 Isabelle Proof for Synchronizability for Trees

.2.1 First Mix Construction in Isabelle

fun mix_pair :: "('information, 'peer) action word ⇒ ('information, 'peer) action word ⇒ ('information, 'peer) action word ⇒ ('information, 'peer) action word" **where**
"mix_pair [] [] acc = acc" |
"mix_pair (a # w') [] acc = mix_pair w' [] (a # acc)" |
"mix_pair [] (a # w) acc = mix_pair [] w (a # acc)" |
"mix_pair (a # w') (b # w) acc = (if a = !⟨get_message b⟩
then (if b = ?⟨get_message a⟩ then mix_pair w' w (a # b # acc) else mix_pair (a # w') w (b # acc))
else mix_pair w' (b # w) (a # acc))"

.2.2 Second Mix Construction in Isabelle

inductive mix_shuf :: "('information, 'peer) action word ⇒ ('information, 'peer) action word ⇒ ('information, 'peer) action word ⇒ ('information, 'peer) action word ⇒ bool" **where**
mix_shuf_constr: "⟦vq$\downarrow_!\downarrow_{-p,q''}\downarrow_{!?}$ = v$\downarrow_?\downarrow_{!?}$; v' ∈ $\mathcal{L}^*{}_{\sqcup\sqcup}$(p); v' $\sqcup\sqcup_?$ v; v ∈ $\mathcal{L}^*$(p); vq ∈ $\mathcal{L}^*$(q);
vq = (as ▪ a_send # bs); v = xs ▪ b # a_recv # ys; get_message a_recv = get_message a_send; is_input a_recv; is_output a_send; is_output b⟧
⟹ mix_shuf vq v v' ((mix_pair as xs []) ▪ a_send # b # a_recv # (mix_pair bs ys []))"

.2.3 add_matching_recvs

fun add_matching_recvs :: "('information, 'peer) action word ⇒ ('information, 'peer) action word" **where**
"add_matching_recvs [] = []" |
"add_matching_recvs (a # w) = (if is_output a
then a # (?⟨get_message a⟩) # add_matching_recvs w
else a # add_matching_recvs w)"

.2.4 Main Theorem in Isabelle

```
theorem synchronisability_for_trees:
  assumes "tree_topology"
  shows "is_synchronisable ⟷ ((∀p ∈ 𝒫. ∀q ∈ 𝒫. ((is_parent_of p q) ⟶ ((subset_condition
p q) ∧ ((ℒ*(p)) = (ℒ*⊔⊔(p)))) )))" (is "?L ⟷ ?R")

proof

  assume "?L"
  show "?R"
  proof clarify
    fix p q
    assume q_parent: "is_parent_of p q"
    have sync_def: "𝒯None↓! = ℒ0" using ‹?L› by simp
    show "subset_condition p q ∧ ℒ* p = ℒ*⊔⊔ p"
    proof (rule conjI)

      show "subset_condition p q" unfolding subset_condition_def
      proof auto

        fix w w' x'
        assume w_Lp: "is_in_infl_lang p w"
          and w'_Lq: "is_in_infl_lang q w'"
          and w'_w_match: "filter (λx. is_output x ∧ (get_object x = q ∧ get_actor x = p
                ∨ get_object x = p ∧ get_actor x = q)) w'↓!? = w↓?↓!?"
          and w'x'_Lq: "is_in_infl_lang q (w' ⋅ x')"
        then show "∃wa. filter (λx. is_output x ∧ (get_object x = q ∧ get_actor x = p ∨ get_object
x = p ∧
              get_actor x = q)) x'↓!? =  wa↓!? ∧  (∃x. wa = x↓? ∧ is_in_infl_lang p (w ⋅ x))"
          using w_Lp  w'_Lq w'_w_match w'x'_Lq
        proof (cases "is_root q")
          case True
          then have "(w' ⋅ x') ∈ ℒ q" using w'x'_Lq w_in_infl_lang by auto


          then have "(w' ⋅ x') ∈ 𝒯None" using root_lang_is_mbox True by blast

          have "w'↓!↓_p,q''↓!? = w↓?↓!?" using w'_w_match by force


          let ?mix = "(mix_pair w' w [])"
          have "?mix ⋅ x' ∈ 𝒯None" sorry
            then obtain t where "t ∈ ℒ0 ∧ t ∈ 𝒯None↓! ∧ t = (?mix ⋅ x')↓!" using sync_def by
fastforce
```

then obtain xc **where** t_sync_run : "sync_run $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ t xc" **using** SyncTraces.simps **by** auto

then have "$\exists$xcm. mbox_run $\mathcal{C}_{\mathcal{I}\mathrm{m}}$ None (add_matching_recvs t) xcm" **using** empty_sync_run_to_mbox_run sync_run_to_mbox_run **by** blast

then have sync_exec: "(add_matching_recvs t) $\in \mathcal{T}_{None}$" **using** MboxTraces.intros **by** auto

then have "$\exists$x. (add_matching_recvs t)$\downarrow_p$ = w $\cdot$ x" **sorry**
then obtain x **where** x_def: "(add_matching_recvs t)$\downarrow_p$ = w $\cdot$ x" **by** blast
then have w'x'_wx_match: "(w' $\cdot$ x')$\downarrow_!\downarrow_{-p,q''}\downarrow_{!?}$ = (w $\cdot$ x)$\downarrow_?\downarrow_{!?}$" **sorry**
have "(w $\cdot$ x) $\in \mathcal{L}^*$ p" **using** sync_exec x_def **by** (metis mbox_exec_to_infl_peer_word)
have "$\exists$wa. x'$\downarrow_!\downarrow_{-p,q''}\downarrow_{!?}$ = wa$\downarrow_{!?}$ $\wedge$ ($\exists$x. wa = x$\downarrow_?$ $\wedge$ is_in_infl_lang p (w $\cdot$ x))" **using** ‹w $\cdot$ x $\in \mathcal{L}^*$ p› ‹w'$\downarrow_!\downarrow_{-p,q''}\downarrow_{!?}$ = w$\downarrow_?\downarrow_{!?}$› w'x'_wx_match **by** auto
then show ?thesis **by** simp
next
case False
then have "is_node q" **by** (metis NetworkOfCA.root_or_node NetworkOfCA_axioms assms)
then obtain r **where** qr: "is_parent_of q r" **by** (metis False UNIV_I path_from_root.simps path_to_root_exists paths_eq)

have "(w' $\cdot$ x') $\in \mathcal{L}^*$ q" **by** (simp add: w'x'_Lq)
then have "$\exists$w''. w'' $\in \mathcal{L}^*$(r) $\wedge$ (((w' $\cdot$ x')$\downarrow_?$)$\downarrow_{!?}$) = (((w''$\downarrow_{-q,r''}$)$\downarrow_!$)$\downarrow_{!?}$)"
using infl_parent_child_matching_ws[of "(w' $\cdot$ x')" q r] **using** qr **by** blast
then obtain w'' **where** w''_w'_match: "w''$\downarrow_!\downarrow_{-q,r''}\downarrow_{!?}$ = (w' $\cdot$ x')$\downarrow_?\downarrow_{!?}$" **and** w''_def: "w'' $\in \mathcal{L}^*$ r" **by** (metis (no_types, lifting) filter_pair_commutative)

have "$\exists$ e. (e $\in \mathcal{T}_{None}$ $\wedge$ e$\downarrow_r$ = w'' $\wedge$ ((is_parent_of q r) $\longrightarrow$ e$\downarrow_q$ = ε))" **using** lemma4_4[of
w'' r q] **using** ‹w'' $\in \mathcal{L}^*$ r› assms **by** blast
then obtain e **where** e_def: "e $\in \mathcal{T}_{None}$" **and** e_proj_r: "e$\downarrow_r$ = w''"
and e_proj_q: "e$\downarrow_q$ = ε" **using** qr **by** blast

let ?mix = "(mix_pair w' w [])"

have "e $\cdot$?mix $\cdot$ x' $\in \mathcal{T}_{None}$" **sorry**

then obtain t **where** "t $\in \mathcal{L}_\mathbf{0}$ $\wedge$ t $\in \mathcal{T}_{None}\lfloor_!$ $\wedge$ t = (e $\cdot$?mix $\cdot$ x')$\downarrow_!$" **using** sync_def **by** fastforce
then obtain xc **where** t_sync_run : "sync_run $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ t xc" **using** SyncTraces.simps **by** auto

then have "$\exists$xcm. mbox_run $\mathcal{C}_{\mathcal{I}\mathrm{m}}$ None (add_matching_recvs t) xcm" **using** empty_sync_run_to_mbox_run sync_run_to_mbox_run **by** blast

```
      then have sync_exec: "(add_matching_recvs t) ∈ 𝒯_None" using MboxTraces.intros by auto

      then have "∃x. (add_matching_recvs t)↓_p = w · x" sorry
      then obtain x where x_def: "(add_matching_recvs t)↓_p = w · x" by blast
      then have w'x'_wx_match: "(w' · x')↓_!↓_{−p,q″}↓_!? = (w · x)↓_?↓_!?" sorry

     have "(w · x) ∈ ℒ* p" using sync_exec x_def by (metis mbox_exec_to_infl_peer_word)
      have "w'↓_!↓_{−p,q″}↓_!? = w↓_?↓_!?" using w'_w_match by force
     have "∃wa. x'↓_!↓_{−p,q″}↓_!? = wa↓_!? ∧ (∃x. wa = x↓_? ∧ is_in_infl_lang p (w · x))" using ‹w · x ∈ ℒ* p› ‹w'↓_!↓_{−p,q″}↓_!? = w↓_?↓_!?› w'x'_wx_match by auto
      then show ?thesis by simp
    qed
   qed

   show "ℒ*(p) = ℒ*_⊔⊔(p)"
   proof

    show "ℒ*(p) ⊆ ℒ*_⊔⊔(p)" using language_shuffle_subset by auto

    show "ℒ*_⊔⊔(p) ⊆ ℒ*(p)"
    proof
      fix v'

      assume "v' ∈ ℒ*_⊔⊔(p)"
        then obtain v where v_orig: "v' ⊔⊔_? v" and orig_in_L: "v ∈ ℒ*(p)" using shuffled_infl_lang_impl_valid_shuffle by auto
      then show "v' ∈ ℒ*(p)"
      proof (induct v v')
        case (refl w)
        then show ?case by simp
      next
        case (swap b a w xs ys)

        then have "∃vq.  vq ∈ ℒ*(q) ∧ ((w↓_?)↓_!?) = (((vq↓_{−p,q″})↓_!)↓_!?)"
          using infl_parent_child_matching_ws[of w p q] orig_in_L q_parent by blast
          then obtain vq where vq_v_match: "((w↓_?)↓_!?) = (((vq↓_{−p,q″})↓_!)↓_!?)" and vq_def: "vq ∈ ℒ* q" by auto
         have lem4_4_prems: "tree_topology ∧ w ∈ ℒ*(p) ∧ p ∈ 𝒫" using assms swap.prems by auto
        then show ?case using assms swap vq_v_match vq_def lem4_4_prems
        proof (cases "is_root q")
          case True
          have "vq ∈ ℒ q" using vq_def w_in_infl_lang by auto
          then have "vq ∈ 𝒯_None" using root_lang_is_mbox True by simp
```

let ?w' = "xs · a # b # ys"
have "∃ acc. mix_shuf vq v v' acc" **sorry**
then obtain mix **where** "mix_shuf vq v v' mix" **by** blast
let ?mix = "mix"
have "?mix ∈ $\mathcal{T}_{None}$" **sorry**
then obtain t **where** "t ∈ $\mathcal{L}_{\mathbf{0}}$ ∧ t ∈ $\mathcal{T}_{None}\lfloor_!$ ∧ t = (?mix)$\downarrow_!$" **using** sync_def **by** fastforce
then obtain xc **where** t_sync_run : "sync_run $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ t xc" **using** SyncTraces.simps **by** auto

then have "∃xcm. mbox_run $\mathcal{C}_{\mathcal{I}\mathrm{m}}$ None (add_matching_recvs t) xcm" **using** empty_sync_run_to_mbox_run sync_run_to_mbox_run **by** blast

then have sync_exec: "(add_matching_recvs t) ∈ $\mathcal{T}_{None}$" **using** MboxTraces.intros **by** auto

then have "(add_matching_recvs t)$\downarrow_p$ = ?w'" **sorry**
then have "?w' ∈ $\mathcal{L}^*$ p" **using** sync_exec mbox_exec_to_infl_peer_word **by** metis
then show ?thesis **by** simp
next
case False
then have "is_node q" **by** (metis NetworkOfCA.root_or_node NetworkOfCA_axioms assms)
then obtain r **where** qr: "is_parent_of q r" **by** (metis False UNIV_I path_from_root.simps path_to_root_exists paths_eq)
then have "∃vr. vr ∈ $\mathcal{L}^*$(r) ∧ ((vq$\downarrow_?$)$\downarrow_{!?}$) = (((vr$\downarrow_{-q,r''}$)$\downarrow_!$)$\downarrow_{!?}$)"
using infl_parent_child_matching_ws[of vq q r] orig_in_L vq_def **by** blast

then obtain vr **where** vr_def: "vr ∈ $\mathcal{L}^*$(r)" **and** vr_vq_match: "((vq$\downarrow_?$)$\downarrow_{!?}$) = (((vr$\downarrow_{-q,r''}$)$\downarrow_!$)$\downarrow_{!?}$)" **by** blast

have "∃ e. (e ∈ $\mathcal{T}_{None}$ ∧ e$\downarrow_r$ = vr ∧ ((is_parent_of q r) ⟶ e$\downarrow_q$ = ε))" **using** lemma4_4[of
vr r q] **using** ‹vr ∈ $\mathcal{L}^*$ r› assms **by** blast
then obtain e **where** e_def: "e ∈ $\mathcal{T}_{None}$" **and** e_proj_r: "e$\downarrow_r$ = vr"
and e_proj_q: "e$\downarrow_q$ = ε" **using** qr **by** blast

let ?w' = "xs · a # b # ys"
have "∃ acc. mix_shuf vq v v' acc" **sorry**
then obtain mix **where** "mix_shuf vq v v' mix" **by** blast
let ?mix = "mix"

have "e · ?mix ∈ $\mathcal{T}_{None}$" **sorry**

then obtain t **where** "t $\in \mathcal{L}_{\mathbf{0}} \wedge$ t $\in \mathcal{T}_{None}\lfloor_! \wedge$ t = (e $\cdot$?mix)$\downarrow_!$" **using** sync_def **by** fastforce

then obtain xc **where** t_sync_run : "sync_run $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ t xc" **using** SyncTraces.simps **by** auto

then have "$\exists$ xcm. mbox_run $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ None (add_matching_recvs t) xcm" **using** empty_sync_run_to_mbox_run sync_run_to_mbox_run **by** blast

then have sync_exec: "(add_matching_recvs t) $\in \mathcal{T}_{None}$" **using** MboxTraces.intros **by** auto

then have "(add_matching_recvs t)$\downarrow_p$ = ?w'" **sorry**
then have "?w' $\in \mathcal{L}^*$ p" **using** sync_exec mbox_exec_to_infl_peer_word **by** metis
then show ?thesis **by** simp
qed
next
case (trans w w' w'')
then show ?case **by** simp
qed
qed
qed
qed
qed

next

assume "?R"
show "?L" — show that TMbox = TSync, i.e. L> = (i.e. the sends are equal
proof auto — cases : w in TMbox, w in TSync
fix w
show "w $\in \mathcal{T}_{None} \Longrightarrow$ w$\downarrow_!$ $\in \mathcal{L}_{\mathbf{0}}$"
proof -
assume "w $\in \mathcal{T}_{None}$"
then have "(w$\downarrow_!$) $\in \mathcal{T}_{None}\lfloor_!$" **by** blast

then have match_exec: "add_matching_recvs (w$\downarrow_!$) $\in \mathcal{T}_{None}$"
using mbox_trace_with_matching_recvs_is_mbox_exec ‹$\forall$ p$\in\mathcal{P}$. $\forall$ q$\in\mathcal{P}$. is_parent_of p q $\longrightarrow$ subset_condition p q $\wedge$ $\mathcal{L}^*$ p = $\mathcal{L}^*_{\sqcup\sqcup}$ p› assms theorem_rightside_def
by blast
then obtain xcm **where** "mbox_run $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ None (add_matching_recvs (w$\downarrow_!$)) xcm" **by** (metis MboxTraces.cases)
then show "(w$\downarrow_!$) $\in \mathcal{L}_{\mathbf{0}}$" **using** SyncTraces.simps ‹w$\downarrow_!$ $\in \mathcal{T}_{None}\lfloor_!$› matched_mbox_run_to_sync_run **by** blast
qed
next — w in TSync –> show that w' (= w with recvs added) is in EMbox
fix w

show "w $\in \mathcal{L}_{\mathbf{0}} \Longrightarrow \exists$ w'. w = w'$\downarrow_!$ $\wedge$ w' $\in \mathcal{T}_{None}$"
proof -
assume "w $\in \mathcal{L}_{\mathbf{0}}$"
— For every output in w, Nsync was able to send the respective message and directly receive it
then have "w = w$\downarrow_!$" **by** (metis SyncTraces.cases run_produces_no_inputs(1))
then obtain xc **where** w_sync_run : "sync_run $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ w xc" **using** SyncTraces.simps ‹w $\in \mathcal{L}_{\mathbf{0}}$› **by** auto
then have "w $\in \mathcal{L}_{\infty}$" **using** ‹w $\in \mathcal{L}_{\mathbf{0}}$› mbox_sync_lang_unbounded_inclusion **by** blast
obtain w' **where** "w' = add_matching_recvs w" **by** simp
— then Nmbox can simulate the run of w in Nsync by sending every mes- sage first to the mailbox of the receiver and then receiving this message
then show ?thesis
proof (cases "xc = []") — this case distinction isn't in the paper but i need it here to properly get the simulated run
case True
then have "mbox_run $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ None (w') []" **using** ‹w' = add_matching_recvs w› empty_sync_run_to_mbox_run w_sync_run **by** auto
then show ?thesis **using** ‹w $\in \mathcal{T}_{None}\downharpoonright_!$› **by** blast
next
case False
then obtain xcm **where** sim_run: "mbox_run $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ None w' xcm $\wedge$ ($\forall$ p. (last xcm p) = ((last xc) p, ε))"
using ‹w' = add_matching_recvs w› sync_run_to_mbox_run w_sync_run **by** blast
then have "w' $\in \mathcal{T}_{None}$" **using** MboxTraces.intros **by** auto
then have "w = w'$\downarrow_!$" **using** ‹w = w$\downarrow_!$› ‹w' = add_matching_recvs w› adding_recvs_keeps_send_order **by** auto
then have "(w'$\downarrow_!$) $\in \mathcal{L}_{\infty}$" **using** ‹w' $\in \mathcal{T}_{None}$› **by** blast
then show ?thesis **using** ‹w = w'$\downarrow_!$› ‹w' $\in \mathcal{T}_{None}$› **by** blast
qed
qed
qed
qed

Mécanisation de la Théorie de la Synchronisabilité pour les Automates Communicants avec Sémantique de Boîte aux Lettres

Nicole KETTLER

Résumé

Nous étudions les automates communicants, un modèle où les participants d'un protocole de communication sont représentés par des machines à états finis échangeant des messages. Ces échanges peuvent être synchrones (instantanés) ou asynchrones (via des buffers).
Dans un travail précédent (Di Giusto, Laversa, & Peters, 2024), nous avons exploré la synchronisabilité, une propriété garantissant que la communication asynchrone n'introduit pas de comportements supplémentaires par rapport à la communication synchrone. Cet article a introduit le problème de la synchronisabilité généralisée, en utilisant des automates avec états finaux, et montre qu'il est indécidable. Cependant, la décidabilité est rétablie pour certaines topologies de communication, telles que les arbres, en prouvant que les langages de buffers dans ces contextes sont réguliers et calculables. Cela étend des résultats obtenus précédemment pour les topologies en anneau (?, ?), et ouvre des perspectives pour de futurs travaux sur les structures en multi-arbres.
Un changement méthodologique clé par rapport aux approches antérieures est l'analyse directe du contenu des buffers, au lieu de la comparaison des traces d'envoi, pour évaluer la synchronisabilité (Di Giusto et al., 2024).
Au cours de ce stage, nous visons à mécaniser les principales preuves de (Di Giusto et al., 2024) à l'aide de l'assistant de preuves Isabelle (Paulson, 1994), afin de les vérifier formellement.

Mots-clés : Synchronisabilité, Automates Communicants, Isabelle, Vérification.

Abstract

We study communicating automata, a model where participants in a communication protocol are represented as finite state machines exchanging messages. These exchanges can be synchronous (instant) or asynchronous (via buffers).
In previous work (Di Giusto et al., 2024), we explored synchronizability, a property ensuring that asynchronous communication does not introduce behaviors beyond those allowed by synchronous communication. This paper introduced the Generalised Synchronizability Problem, using automata with final states, and shows it is undecidable. However, decidability is restored for specific communication topologies such as trees by proving that buffer languages in these settings are regular and computable. This extends results previously obtained for ring topologies (Finkel & Lozes, 2017), and opens avenues for future work on multitree structures. A key methodological shift from prior approaches is the direct analysis of buffer contents, instead of comparing send traces, to assess synchronizability. (Di Giusto et al., 2024)
During this internship, we aim to mechanize the main proofs from (Di Giusto et al., 2024) using the Isabelle proof assistant (Paulson, 1994), in order to verify them.

Keywords: Synchronizability, Communicating Automata, Isabelle, Verification.