


Wait-Only Broadcast Protocols Are Easier to Verify

Lucie Guillou 

IRIF, CNRS, Université Paris Cité, France

Arnaud Sangnier 

DIBRIS, Università di Genova, Italy

Nathalie Sznajder 

LIP6, CNRS, Sorbonne Université, Paris, France

Abstract

We study networks of processes that all execute the same finite-state protocol and communicate via broadcasts. We are interested in two problems with a parameterized number of processes: the synchronization problem which asks whether there is an execution which puts all processes on a given state; and the repeated coverability problem which asks if there is an infinite execution where a given transition is taken infinitely often. Since both problems are undecidable in the general case, we investigate those problems when the protocol is *Wait-Only*, i.e., it has no state from which a process can both broadcast and receive messages. We establish that the synchronization problem becomes Ackermann-complete, and the repeated coverability problem is in EXPSpace and PSPACE-hard.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Parameterised verification, Reachability, Broadcast

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.53

Related Version *Long Version:* <https://arxiv.org/abs/2505.01269> [12]

Funding ANR project PaVeDyS (ANR-23-CE48-0005).

1 Introduction

Distributed systems are at the core of modern computing, and are widely used in critical applications such as sensor networks, or distributed databases. These systems rely on protocols to enable communication, maintain coherence and coordination between the different processes. Because of their distributed nature, such protocols have proved to be error-prone. As a result, the formal verification of distributed systems has become an essential area of research. Formal verification of distributed systems presents unique challenges compared to the one of centralized systems. One of the most significant issue is the state explosion problem: the behavior of multiple processes that execute concurrently and exchange information often lead to state spaces that grow exponentially with the number of processes, making the analysis highly challenging. When the systems are parameterized, meaning designed to operate for an arbitrary number of processes, which is often the case in distributed protocols, classical techniques are not useful anymore. The difficulty shifts from state explosion to dealing with an infinite number of system configurations, which leads to undecidability in the general case [1]. Despite these challenges, verification becomes more tractable in certain restricted settings. For instance, in parameterized systems, the behavior of individual processes needs not always to be explicitly represented, which can mitigate state explosion. This has motivated researchers to focus on specific subclasses of systems or communication models where decidability can be achieved without sacrificing too much expressiveness. One such restriction involves protocols where processes are anonymous (i.e., without identities), and communicate via simple synchronous mechanisms, such as rendezvous [9], where two processes exchange a message synchronously. Several variants



© Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder;
licensed under Creative Commons License CC-BY 4.0

50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025).

Editors: Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak; Article No. 53; pp. 53:1–53:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of this model have been studied, such as communication via a shared register containing some value from a finite set [7], or non-blocking rendezvous [10, 4], where a sender is not prevented from sending a message when no process is ready to receive it. In all these cases, the processes execute the same protocol, which is described by a finite automaton.

A more expressive model is broadcast protocols, introduced by Emerson and Namjoshi [6]. In these protocols, a process can send a broadcast message that is received simultaneously by all other processes (or by all neighboring processes in a network). Several key verification problems arise in this context, including the coverability problem, which asks whether a process can eventually reach a specific (bad) state starting from an initial configuration. Another important property is synchronization, which asks whether all processes can simultaneously reach a given state. Liveness properties, like determining whether infinite executions exist, ensure that certain behaviors occur indefinitely. While broadcast protocols allow more powerful communication than rendezvous protocols, this added expressiveness comes at the cost of increased verification complexity. Indeed, rendezvous communication can be simulated using broadcasts [8]. However, verification becomes significantly harder: while coverability and synchronization can be solved in polynomial time for rendezvous protocols [9], they become respectively Ackermann-complete [8, 20] and undecidable (we show it in this paper) for broadcast protocols. Liveness properties are also undecidable for broadcast protocols [8]. The challenge in verifying broadcast protocols can be understood through their relationship with Vector Addition Systems with States (VASS). Rendezvous-based protocols can be encoded in a VASS, where counters track the number of processes in each state. A rendezvous is simulated by decreasing the counters corresponding to the sender and receiver states and increasing the counters for their post-communication states. While using a VASS for protocols using rendezvous is certainly not the way to go due to the complexity gap between problems on VASS and the existing known polynomial time algorithms to solve verification problems, it is interesting to note that this encoding fails for broadcast protocols because a broadcast message must be received by all processes in a given state, requiring a bulk transfer of a counter value – something not expressible in classical VASS without adding powerful operations such as transfer arcs, which leads to an undecidable reachability problem [21]. However, in some cases, verification of broadcast protocols can become easier, complexity-wise. One example is the setting of Reconfigurable Broadcast Networks, in which communication between processes can be lossy [5]. Liveness properties become decidable in that case [5] and even polynomial time [2].

Another such case is given by a syntactic restriction known as *Wait-Only protocols*, introduced in [10] in the context of non-blocking rendezvous communication. In Wait-Only protocols, a process cannot both send and receive a message from the same state. From a practical perspective, Wait-Only protocols were proposed to model the non-blocking rendezvous semantics used in Java's `wait/notify/notifyAll` synchronization mechanism and C's `wait/signal/broadcast` operations with conditional variables, commonly found in multi-threaded programs. In both languages, threads can be awakened by the reception of a message. This naturally leads to distinguishing between action and waiting states: a sleeping thread cannot act on its own and can only be awakened by a signal. This restriction simplifies the possible behaviours of the system, potentially reducing the complexity of the verification. Actually, the coverability problem for Wait-Only broadcast protocols becomes PSPACE-complete [11]. Indeed, when processes are in a state where they can receive broadcasts, they cannot send messages, meaning they will move together to the next state, reducing the need for precise counting. Moreover, when a process is in a broadcasting state, it remains there until it decides to send a message, simplifying execution reconstruction and enabling

better verification algorithms. By leveraging these properties, we design algorithms for the synchronization problem and liveness properties, obtaining new decidability results for these challenging problems. Proofs omitted due to space constraints can be found in [12].

2 Model and Verification Problems

In this section, we provide the description of a model for networks with broadcast communication together with some associated verification problems we are interested in. First we introduce some useful mathematical notations. We use \mathbb{N} to represent the set of natural numbers, $\mathbb{N}_{>0}$ to represent $\mathbb{N} \setminus \{0\}$, and for $n, m \in \mathbb{N}$ with $n \leq m$, we denote by $[n, m]$ the set $\{n, n+1, \dots, m\}$. For a finite set E and a natural $n > 0$, the set E^n represents the n -dimensional vectors with values in E and for $v \in E^n$ and $1 \leq i \leq n$, we use $v(i)$ to represent the i -th value of v . Furthermore, for such a vector v , we denote by $\|v\|$ its dimension n .

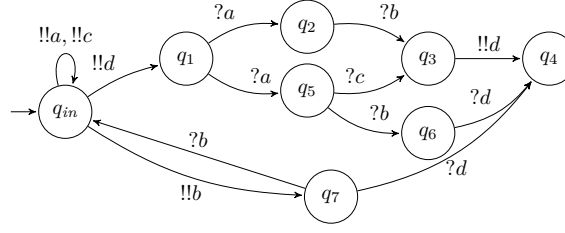
Networks of Processes using Broadcast Communication. In the networks we consider, all the processes execute the same protocol, given by a finite state machine. The actions of this machine can either broadcast a message a to the other processes (denoted by $!!a$) or receive a message a (denoted by $?a$). For a finite alphabet of messages Σ , we use the following notations: $!!\Sigma \stackrel{\text{def}}{=} \{!!a \mid a \in \Sigma\}$, $?\Sigma \stackrel{\text{def}}{=} \{?a \mid a \in \Sigma\}$ and $\text{Op}_\Sigma \stackrel{\text{def}}{=} !!\Sigma \cup ?\Sigma$.

► **Definition 2.1.** A protocol P is a tuple (Q, Σ, q_{in}, T) where Q is a finite set of states, Σ is a finite set of messages, $q_{in} \in Q$ is an initial state, and $T \subseteq Q \times \text{Op}_\Sigma \times Q$ is a set of transitions.

For all $q \in Q$, we define $R(q)$ as the set of messages that can be received from state q , given by $\{a \in \Sigma \mid \exists q' \in Q, (q, ?a, q') \in T\}$. A protocol $P = (Q, \Sigma, q_{in}, T)$ is *Wait-Only* whenever for all $q \in Q$, either $\{q' \in Q \mid (q, \alpha, q') \in T \text{ with } \alpha \in ?\Sigma\} = \emptyset$, or $\{q' \in Q \mid (q, \alpha, q') \in T \text{ with } \alpha \in !!\Sigma\} = \emptyset$. In a Wait-Only protocol, a state $q \in Q$ such that $\{q' \in Q \mid (q, \alpha, q') \in T \text{ with } \alpha \in ?\Sigma\} \neq \emptyset$ is called a *waiting state*. A state that is not a waiting state is an *action state*. We denote by Q_W the set of waiting states and by Q_A the set of action states (hence $Q_A = Q \setminus Q_W$). Observe that if $q_{in} \in Q_W$, then no process is ever able to broadcast messages, for this reason we will always assume that $q_{in} \in Q_A$.

For $n \in \mathbb{N}_{>0}$, an n -process *configuration* of a protocol $P = (Q, \Sigma, q_{in}, T)$ is a vector $C \in Q^n$ where $C(e)$ represents the state of the e -th process in C for $1 \leq e \leq n$. The configuration is said to be *initial* whenever $C(e) = q_{in}$ for all $1 \leq e \leq n$. The dimension $\|C\|$ hence characterizes the number of processes in C . For a state $q \in Q$, we use $C^{-1}(q)$ to represent the set of processes in state q , formally $C^{-1}(q) = \{1 \leq e \leq \|C\| \mid C(e) = q\}$. For a subset $A \subseteq [1, \|C\|]$ of processes, we use $C(A)$ to identify the set of states of processes in A , formally $C(A) = \{C(e) \mid e \in A\}$. We let \mathcal{C} [resp. \mathcal{I}] be the set of all configurations [resp. initial configurations] of P .

We now define the broadcast network semantics associated to a protocol $P = (Q, \Sigma, q_{in}, T)$. For configurations $C, C' \in \mathcal{C}$, transition $t = (q, !!a, q') \in T$ and $e \in [1, \|C\|]$, we write $C \xrightarrow{e, t} C'$ whenever $\|C\| = \|C'\|$, $C(e) = q$, $C'(e) = q'$ and, for all $e' \in [1, \|C\|] \setminus \{e\}$, either a reception occurs, i.e., $(C(e'), ?a, C'(e')) \in T$, or the process cannot receive a , in which case $(a \notin R(C(e'))) \text{ and } C(e') = C'(e')$. Intuitively, the e -th process of C broadcasts a message a , which is received by all processes able to receive it, while the other processes remain in their current state. We note $C \rightarrow C'$ if there exists $e \in [1, \|C\|]$ and $t \in T$ such that $C \xrightarrow{e, t} C'$ and use \rightarrow^* [resp. \rightarrow^+] for the reflexive and transitive [resp. transitive] closure of \rightarrow . A finite [resp. infinite] execution is a finite [resp. infinite] sequence of configurations



■ **Figure 1** Example of a Wait-Only protocol P .

$\rho = C_0 \dots C_\ell$ [resp. $\rho = C_0 \dots$] such that $C_0 \in \mathcal{I}$ and $C_{i-1} \rightarrow C_i$ for all $0 < i \leq \ell$ [resp. for all $i > 0$]. Its length $|\rho|$ is equal to $\ell + 1$ [resp. ω]. We write Λ [resp. Λ_ω] for the set of finite [resp. infinite] executions. For an execution $\rho = C_0 C_1 \dots \in \Lambda \cup \Lambda_\omega$ and $0 \leq i < |\rho|$, we use ρ_i to denote C_i , the i -th configuration. We use $\#\mathbf{proc}(\rho)$ to represent $\|C_0\|$, the number of processes in the execution. For $0 \leq i < j < |\rho|$, we define $\rho_{i,j} \stackrel{\text{def}}{=} \rho_i \dots \rho_j$. We also let $\rho_{\geq i} \stackrel{\text{def}}{=} \rho_i \rho_{i+1} \dots$ and $\rho_{\leq i} \stackrel{\text{def}}{=} \rho_0 \dots \rho_i$. For $e \in [1, \#\mathbf{proc}(\rho)]$, we denote by $\rho(e)$ the sequence of states $\rho_0(e) \rho_1(e) \dots$.

► **Example 2.2.** Figure 1 illustrates an example of a Wait-Only protocol. The waiting states are q_1, q_2, q_5, q_6 , and q_7 , with $R(q_5) = \{b, c\}$. Here is one of its execution with 3 processes:

$$\begin{aligned} (q_{in}, q_{in}, q_{in}) &\xrightarrow{1, (q_{in}, !!d, q_1)} (q_1, q_{in}, q_{in}) \xrightarrow{2, (q_{in}, !!d, q_1)} (q_1, q_1, q_{in}) \xrightarrow{3, (q_{in}, !!a, q_{in})} (q_5, q_2, q_{in}) \\ &\xrightarrow{3, (q_{in}, !!c, q_{in})} (q_3, q_2, q_{in}) \xrightarrow{3, (q_{in}, !!b, q_7)} (q_3, q_3, q_7). \end{aligned}$$

Verification Problems. We investigate two verification problems over broadcast networks: the synchronization problem (SYNCHRO) and the repeated coverability problem (REPCOVER). SYNCHRO asks, given a protocol $P = (Q, \Sigma, q_{in}, T)$ and a state $q_f \in Q$, whether there exist $C \in \mathcal{I}$ and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$, and $C'(e) = q_f$ for all $e \in [1, \|C'\|]$. REPCOVER asks, given a protocol $P = (Q, \Sigma, q_{in}, T)$ and a transition $t = (q, !!a, q') \in T$, whether there exist $\rho \in \Lambda_\omega$ and $e \in [1, \#\mathbf{proc}(\rho)]$, such that for all $i \in \mathbb{N}$, there exists $j > i$ verifying $\rho_j \xrightarrow{e, t} \rho_{j+1}$.

► **Theorem 2.3.** *SYNCHRO and REPCOVER are undecidable.*

We give the proof of the undecidability of SYNCHRO in [12], while the undecidability of REPCOVER was established in [8, Theorem 5.1]. In the remainder of the paper, we show that by restricting to Wait-Only protocols, one can regain decidability.

► **Remark 2.4.** Throughout the remainder of this paper, we will consider protocols without self-loop broadcast transitions of the form $(q, !!a, q)$. Such transitions can be transformed into two transitions $(q, !!a, p_q), (p_q, !!\$, q)$ where p_q is a new state added to the set of states and $\$$ is a new message added to the alphabet. This transformation of the protocol is equivalent to the original one with respect to SYNCHRO and REPCOVER.

Vector Addition System with States. In the following, we will extensively rely on the model of Vector Addition Systems with States (VASS) which we introduce below. A VASS is a tuple $\mathcal{V} = (\text{Loc}, \ell_0, \mathbf{X}, \Delta)$ where Loc is a finite set of locations, $\ell_0 \in \text{Loc}$ is the *initial* location, \mathbf{X} is a finite set of natural variables, called counters, and $\Delta \subseteq \text{Loc} \times \mathbb{Z}^{\mathbf{X}} \times \text{Loc}$ is a finite set of transitions. A configuration of \mathcal{V} is a pair (ℓ, v) in $\text{Loc} \times \mathbb{N}^{\mathbf{X}}$ where v provides a value for each counter. The initial configuration is $(\ell_0, \mathbf{0})$ where $\mathbf{0}(x) = 0$ for all $x \in \mathbf{X}$. Given

two configurations (ℓ, v) , (ℓ', v') and a transition $(\ell, \delta, \ell') \in \Delta$, we write $(\ell, v) \xrightarrow{\delta} (\ell', v')$ whenever $v'(x) = v(x) + \delta(x)$ for all counters $x \in X$. We simply use $(\ell, v) \rightsquigarrow (\ell', v')$ when there exists $(\ell, \delta, \ell') \in \Delta$ such that $(\ell, v) \xrightarrow{\delta} (\ell', v')$. We denote \rightsquigarrow^* for the transitive and reflexive closure of \rightsquigarrow . An (infinite) execution (or run) of the VASS is a (infinite) sequence of configurations $(\ell_0, v_0), (\ell_1, v_1), \dots, (\ell_n, v_n)$, starting with the initial configuration and such that $(\ell_i, v_i) \rightsquigarrow (\ell_{i+1}, v_{i+1})$ for all $0 \leq i < n$.

REACH, the well-known reachability problem for VASS, is defined as follows: given a VASS $\mathcal{V} = (\text{Loc}, \ell_0, X, \Delta)$ and a location $\ell_f \in \text{Loc}$, is there an execution from $(\ell_0, \mathbf{0})$ to $(\ell_f, \mathbf{0})$?

► **Theorem 2.5** (Membership [18], Hardness [3, 17]). *REACH is Ackermann-complete.*

3 Solving Synchro for Wait-Only Protocols

To solve SYNCHRO we show in this section that we can build a VASS that simulates the behavior of a broadcast network running a Wait-Only protocol. An intuitive way to proceed, inspired by the counting abstraction proposed for protocols communicating by pairwise rendez-vous communication [9], consists in having one counter per state of the protocol, that stores the number of processes that populate that state. Initially, the counter associated with the initial state can be incremented as much as one wants: this will determine the number of processes of the execution simulated in the VASS. Then, the simulation of broadcast messages $(q, !a, q')$ amounts to decrementing the counter associated to q and increment the counter associated to q' . However, simulating receptions of the message (e.g. a transition $(p, ?a, p')$), requires to empty the counter associated to p and transfer its value to the counter associated to q' . This transfer operation is not something that can be done in VASS unless the model is extended with special transfer transitions, leading to an undecidable reachability problem [21]. To circumvent this problem, we rely on two properties of Wait-Only protocols: (1) processes that occur to be in the same waiting state follow the same path of reception in the protocol, and end in the same action state (we show how to take care of potential non-determinism in the next section) and (2) processes in an action state cannot be influenced by the behaviour of other processes as they cannot receive any message. Thanks to property (1), instead of precisely tracking the number of processes in each waiting state, we only count the processes in a “waiting region” – a connected component of waiting states populated by processes that will all reach the same action state simultaneously. The waiting region allows us also to monitor when the processes go out from a waiting region to an action state. We will explain later how we use property (2) to handle the transfer of values of counters within the VASS semantics, and thus simulating processes leaving a waiting region.

For the rest of this section, we fix $P = (Q, \Sigma, q_{in}, T)$ a Wait-Only protocol (with Q_W [resp. Q_A] the set of waiting states [resp. action states]) for which we assume w.l.o.g. the existence of an *uncoverable state* $q_u \in Q$ verifying $q \neq q_u$ and $q' \neq q_u$ for all $(q, \alpha, q') \in T$ and $q_u \neq q_{in}$. Furthermore, we consider a final waiting state $q_f \in Q_W$. To ease the reading, we assume in this section that the final state q_f is a waiting state, but our construction could be adapted to the case where q_f is an action state. Moreover, we show in the next section that SYNCHRO in this latter case is easier to solve.

3.1 Preliminary properties

We present here properties satisfied by Wait-Only protocols, which we will rely on for our construction. We first show with Lemma 3.1 that if, during an execution, two processes fulfill two conditions: (i) they are on the same waiting state q_w , and (ii) the next action state

they reach (not necessarily at the same time) is the same (namely q_a), then one can build an execution where they both follow the same path between q_w and q_a . This is trivial for *deterministic* protocols (where for all $q \in Q$ and $\alpha \in \text{Op}_\Sigma$, there is at most one transition of the form $(q, \alpha, q') \in T$) and is also true for non-deterministic protocols.

For an execution $\rho \in \Lambda \cup \Lambda_w$ of P , an index $0 \leq j < |\rho|$, a waiting state $q \in Q_W$ and a process number $e \in [1, \#\text{proc}(\rho)]$ such that $\rho_j(e) = q$, we define $\mathbf{na-index}(\rho, j, e) = \min\{k \mid j \leq k \leq |\rho| \text{ and } \rho_k(e) \in Q_A\}$, i.e. the first moment after ρ_j where the process e reaches an action state (if such moment does not exist, it is set to $|\rho|$). We note $\mathbf{na-state}(\rho, j, e) \stackrel{\text{def}}{=} \rho_{\mathbf{na-index}(\rho, j, e)}(e)$ the next action state for process e from ρ_j if $\mathbf{na-index}(\rho, j, e) \neq |\rho|$ and otherwise we take the convention that $\mathbf{na-state}(\rho, j, e)$ is equal to q_u , the uncoverable state of the protocol P . Finally, we let $\mathbf{na}(\rho, j, e) = (\mathbf{na-state}(\rho, j, e), \mathbf{na-index}(\rho, j, e))$.

► **Lemma 3.1.** *Let $\rho \in \Lambda \cup \Lambda_w$ be an execution of P . If there exist $e_1, e_2 \in [1, \#\text{proc}(\rho)]$ and $0 \leq j < |\rho|$ such that (i) $\rho_j(e_1) = \rho_j(e_2) \in Q_W$, (ii) $\mathbf{na}(\rho, j, e_1) = (q, j_1)$, and (iii) $\mathbf{na}(\rho, j, e_2) = (q, j_2)$ with $j_1 \leq j_2 < |\rho|$, then there exists an execution ρ' of the form $\rho_{\leq j} \rho'_{j+1} \dots \rho'_{j_2} \rho_{\geq j_2+1}$ such that $\rho'_k(e_1) = \rho'_k(e_2) = \rho_k(e_1)$ for all $j+1 \leq k \leq j_1$, and $\rho'_k(e_1) = \rho_k(e_1)$ and $\rho'_k(e_2) = q$ for all $j_1 < k \leq j_2$. In particular $\mathbf{na}(\rho', j, e_1) = \mathbf{na}(\rho', j, e_2) = (q, j_1)$.*

► **Example 3.2.** Consider the protocol P in Figure 1 and the execution ρ of Example 2.2. We have $\rho_2(1) = \rho_2(2) = q_1$ and $\mathbf{na}(\rho, 2, 1) = (q_3, 4)$ and $\mathbf{na}(\rho, 2, 2) = (q_3, 5)$. Applying Lemma 3.1, we build: $\rho' = (q_{in}, q_{in}, q_{in}) \rightarrow (q_1, q_{in}, q_{in}) \rightarrow (q_1, q_1, q_{in}) \rightarrow (q_5, q_5, q_{in}) \rightarrow (q_3, q_3, q_{in}) \rightarrow (q_3, q_3, q_7)$ where process 1 and process 2 follow the same path between q_1 and q_3 . However, consider the following events from (q_1, q_1, q_{in}) : $(q_1, q_1, q_{in}) \rightarrow (q_2, q_5, q_{in}) \rightarrow (q_3, q_6, q_7) \rightarrow (q_4, q_4, q_4)$, here we cannot apply the lemma as from q_1 , processes 1 and 2 reach two different next action states (q_3 and q_4).

The above lemma enables us to focus on a specific subset of executions for SYNCHRO, which we refer to as well-formed. In these executions, at any moment, two processes in the same waiting state and with the same next action state, follow the same path of receptions. Formally, an execution ρ of a protocol P is *well-formed* iff for all $0 \leq i < |\rho|$, for all $e_1, e_2 \in [1, \#\text{proc}(\rho)]$ such that $\rho_i(e_1) = \rho_i(e_2) \in Q_W$ and $\mathbf{na-state}(\rho, i, e_1) = \mathbf{na-state}(\rho, i, e_2) = q$, it holds that $\rho_k(e_1) = \rho_k(e_2)$ for all $i \leq k \leq \mathbf{na-index}(\rho, i, e_1)$. From Lemma 3.1, we immediately get:

► **Corollary 3.3.** *There exists an execution $\rho = C_0 \dots C_n$ such that $C_n(e) = q_f$ for all $e \in [1, \#\text{proc}(\rho)]$ iff there exists a well-formed execution from C_0 to C_n .*

We finally provide a result which will allow us to bound the size of the VASS that we will build from the given Wait-Only protocol. Given an execution $\rho \in \Lambda \cup \Lambda_w$ of P , an index $0 \leq j < |\rho|$ and an action state $q_a \in Q_A$, we define $\mathbf{na-index-set}(\rho, j, q_a)$ as the set of indices where processes in a waiting state at ρ_j will reach q_a , if it is their next action state: $\mathbf{na-index-set}(\rho, j, q_a) \stackrel{\text{def}}{=} \{i \mid \text{there exists } e \in [1, \#\text{proc}(\rho)] \text{ s.t. } \rho_j(e) \in Q_W \text{ and } \mathbf{na}(\rho, j, e) = (q_a, i)\}$.

► **Lemma 3.4.** *For all well-formed executions ρ , for all $0 \leq j < |\rho|$, and for all $q_a \in Q_A$, we have $|\mathbf{na-index-set}(\rho, j, q_a)| \leq |Q_W|$.*

Proof. If we have $|\mathbf{na-index-set}(\rho, j, q_a)| > |Q_W|$, by pigeon hole principle there exist $e_1, e_2 \in [1, \#\text{proc}(\rho)]$ such that $\rho_j(e_1) = \rho_j(e_2) \in Q_W$ and such that $\mathbf{na}(\rho, j, e_1) = (q_a, i_1)$ and $\mathbf{na}(\rho, j, e_2) = (q_a, i_2)$ with $i_1 < i_2$. Consequently $\rho_{i_1}(e_1) = q_a \in Q_A$ and $\rho_{i_1}(e_2) \in Q_W$ hence $\rho_{i_1}(e_1) \neq \rho_{i_1}(e_2)$, which contradicts the definition of well-formedness. ◀

3.2 Building the VASS that Simulates a Broadcast Network

Summaries. We present here the formal tool we use to represent processes in waiting states. We begin by introducing some notations. A *print* \mathbf{pr} is a set of waiting states. Given a non empty subset of states $A = \{q_1, \dots, q_n\} \subseteq Q$, we define a configuration $\dot{A} \in Q^n$ such that $\dot{A}(e) = q_e$ for all $e \in [1, n]$. When $A = \{q\}$, we write $\dot{q} \in Q^1$ instead of $\{q\}$. Conversely, given a configuration C , we define $\mathbf{set}(C) = \{q \in Q \mid C^{-1}(q) \neq \emptyset\}$. For two configurations $C \in Q^n$ and $C' \in Q^m$, we let $C \oplus C'$ be the configuration $C'' \in Q^{n+m}$ defined by: $C''(e) = C(e)$ if $1 \leq e \leq n$ and $C''(e) = C'(e - n)$ if $n + 1 \leq e \leq m + n$. A *summary* S is a tuple (\mathbf{pr}, q_a, k) such that $\mathbf{pr} \subseteq Q_W$ is a print, $q_a \in Q_A$ is an *exit state*, and $k \in [1, |Q_W| + 1]$ is an identifier. The label of S is $\mathbf{lab}(S) = (q_a, k)$ and we denote its print by $\mathbf{print}(S)$. Finally, we define a *special* summary **Done**.

In our construction, each process in a waiting state is associated with a summary while processes in action states are handled by the counters of the VASS. Intuitively, a summary (\mathbf{pr}, q_a, k) represents a set of processes lying in the set $\mathbf{pr} \subseteq Q_W$ that will reach the same next action state q_a simultaneously (this restriction is justified by Lemma 3.1). Since the set of waiting states \mathbf{pr} evolve during the simulation, each summary must be uniquely identified. To achieve this, we use an integer $k \in [1, |Q_W| + 1]$. Hence each summary is identified with the pair (q_a, k) . We do not need more than $|Q_W| + 1$ different identifiers, because when a process enters a new summary (i.e it arrives in a waiting state q_w from an action state), aiming for next action state q_a , it either joins an existing summary with exit state q_a , or create a new one. In the latter case, well-formed executions ensure that no existing summary $S = (\mathbf{pr}, q_a, k)$ with exit state q_a is such that $q_w \in \mathbf{pr}$. Otherwise two processes would be in the same state q_w , aiming for the same action state, but at different moments, contradicting Lemma 3.1. Note that we need $|Q_W| + 1$ different identifiers, and not $|Q_W|$ for technical reasons. Finally, the special summary **Done** is used to indicate when the processes described by a summary reach the exit action state.

We now describe how summaries evolve with the sequence of transitions. Let $S = (\mathbf{pr}, q_a, k)$ and $S' = (\mathbf{pr}', q_a, k)$ be two summaries (with the same exit state and identifier) and $t = (q, !!b, q') \in T$ such that there exists a configuration C' verifying $\dot{q} \oplus \mathbf{pr} \xrightarrow{1,t} \dot{q}' \oplus C'$. We then write $S \xRightarrow{t} S'$ whenever $\mathbf{set}(C') = \mathbf{pr}' \subseteq Q_W$. This represents the evolution of a summary upon reception of message b , when the processes all stay in waiting states, and no new process joins the “waiting region” of the given summary. We write $S \xRightarrow{t, +q'} S'$ whenever $q' \in Q_W$ and $\mathbf{set}(C') \cup \{q'\} = \mathbf{pr}' \subseteq Q_W$. In that latter case, the process responsible for the transition t joins the “waiting region” represented by S . This typically occurs when the process’s next action state is q_a , and it reaches q_a simultaneously with the processes described by S . Finally, we use $S \xRightarrow{t} \mathbf{Done}$ whenever $\mathbf{set}(C') = \{q_a\}$. This represents the evolution (and disappearance) of S when all the processes reach q_a (they all reach it simultaneously).

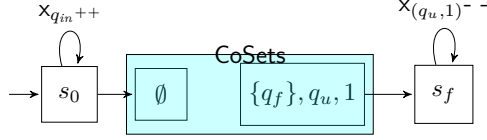
► **Example 3.5.** Returning to the protocol P of Figure 1, consider the summary $S_0 = (\{q_1\}, q_3, 1)$. Observe that $S_0 \xRightarrow{(q_{in}, !!a, q_{in})} (\{q_2\}, q_3, 1)$ and $S_0 \xRightarrow{(q_{in}, !!a, q_{in})} (\{q_5\}, q_3, 1)$. However, by definition, we do not have $S_0 \xRightarrow{(q_{in}, !!a, q_{in})} (\{q_5, q_2\}, q_3, 1)$. Indeed, processes summarized in S_0 are forced to all go either on q_2 or on q_5 upon receiving a : thanks to Corollary 3.3, we consider only well-formed executions, where all processes summarized in S_0 choose the same next state. Additionnaly, we have $(\{q_2\}, q_3, 1) \xRightarrow{(q_{in}, !!b, q_7)} \mathbf{Done}$ and

$(\{q_1\}, q_4, 1) \xrightarrow{(q_{in}, !!a, q_{in})} (\{q_5\}, q_4, 1) \xrightarrow{(q_{in}, !!b, q_7), +q_7} (\{q_6, q_7\}, q_4, 1) \xrightarrow{(q_{in}, !!d, q_1)} \text{Done}$. However, note that we do not have $(\{q_2\}, q_4, 1) \xrightarrow{(q_{in}, !!b, q_7)} \text{Done}$, since the action state q_3 reached from q_2 upon receiving b is not the exit state q_4 .

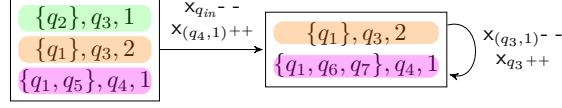
Definition of the VASS. We now explain how we use the summaries in the VASS simulating the executions in the network. First we say that a set \mathcal{S} of summaries is *coherent* if for all distinct pairs $(pr_1, q_1, k_1), (pr_2, q_5, k_2) \in \mathcal{S}$ such that $q_1 = q_5$, we have $k_1 \neq k_2$ and $pr_1 \cap pr_2 = \emptyset$. We denote by CoSets the set of coherent sets of summaries. For a set of summaries \mathcal{S} we let $\mathbf{lab}(\mathcal{S}) = \{\mathbf{lab}(S) \mid S \in \mathcal{S}\}$. Observe that, when \mathcal{S} is coherent, for a label $(q, k) \in \mathbf{lab}(\mathcal{S})$, there is a unique $S \in \mathcal{S}$ such that $\mathbf{lab}(S) = (q, k)$.

We define the VASS \mathcal{V}_P simulating the protocol P as follows: $\mathcal{V}_P = (\text{Loc}, s_0, \mathbf{X}, \Delta)$, where $\text{Loc} = \text{CoSets} \cup \{s_0\} \cup \{s_f\}$, the set of counters is $\mathbf{X} = \{x_q \mid q \in Q_A\} \cup \{x_{(q,k)} \mid q \in Q_A, k \in [1, |Q_W| + 1]\}$, and the set of transitions $\Delta = \Delta_0 \cup (\bigcup_{t \in T} \Delta_t) \cup \Delta_e$, is defined as follows:

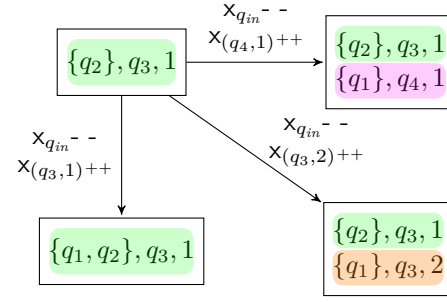
- Δ_0 contains exactly the following transitions:
 - (s_0, δ_0, s_0) where $\delta_0(x_{q_{in}}) = 1$ and $\delta_0(x) = 0$ for all other counters;
 - $(s_0, \mathbf{0}, \emptyset)$ where $\mathbf{0}$ is the null vector (note that the empty set is coherent);
 - $(\{S_f\}, \mathbf{0}, s_f) \in \Delta$ where $S_f = (\{q_f\}, q_u, 1)$;
 - $(s_f, \delta_f, s_f) \in \Delta$ where $\delta_f(x_{(q_u, 1)}) = -1$ and $\delta_f(x) = 0$ for all other counters.
- For $t = (q, !!a, q') \in T$, we have $(\mathcal{S}, \delta, \mathcal{S}') \in \Delta_t$ iff one of the following conditions holds:
 - a. $q' \in Q_A$ and $\mathcal{S} = \{S_1, \dots, S_k\}$. Then, for all $1 \leq i \leq k$, there exists S'_i such that $S_i \xrightarrow{t} S'_i$ and $\mathcal{S}' = \{S'_i \mid 1 \leq i \leq k\} \setminus \{\text{Done}\}$. Moreover, $\delta(x_q) = -1$, $\delta(x_{q'}) = 1$ and $\delta(x) = 0$ for all $x \in \mathbf{X} \setminus \{x_q, x_{q'}\}$. Here we simply update the sets of states populated by processes in waiting states after the transition t . Some summaries may have disappeared from \mathcal{S} if the processes represented by this summary have reached their exit action state when receiving a . For now, we only modify the counters associated to the states q and q' . Note that this is well-defined, since we assume in Remark 2.4 that there is no self-loop of the form $(q, !!a, q)$.
 - b. $q' \in Q_W$, $\mathcal{S} = \{S_1, \dots, S_k\}$, and there exists $1 \leq i \leq k$ and S'_i such that $S_i \xrightarrow{t, +q'} S'_i$. For all $j \neq i$, there exists S'_j such that $S_j \xrightarrow{t} S'_j$. Then $\mathcal{S}' = \{S'_j \mid 1 \leq j \leq k\} \setminus \{\text{Done}\}$. Moreover, $\delta(x_q) = -1$, $\delta(x_{\mathbf{lab}(S_i)}) = 1$ and $\delta(x) = 0$ for all $x \in \mathbf{X} \setminus \{x_q, x_{\mathbf{lab}(S_i)}\}$. In that case, the process having performed the transition joins an existing summary S_i . We have then modified accordingly the counters associated to q and to the appropriate summary.
 - c. $q' \in Q_W$ and $\mathcal{S} = \{S_1, \dots, S_k\}$. For all $1 \leq j \leq k$, there exists S'_j such that $S_j \xrightarrow{t} S'_j$. Then $\mathcal{S}' = (\{S'_1, \dots, S'_k\} \setminus \{\text{Done}\}) \cup \{(\{q'\}, q_a, k)\}$ for some $q_a \in Q_A$ and $k \in [1, |Q_W| + 1]$ such that $(q_a, k) \notin \mathbf{lab}(\mathcal{S})$. Moreover, $\delta(x_q) = -1$, $\delta(x_{(q_a, k)}) = 1$, and $\delta(x) = 0$ for all $x \in \mathbf{X} \setminus \{x_q, x_{(q_a, k)}\}$. This case happens when the process having sent the message a reaches a waiting state, and its expected next action pair (index and state) does not correspond to any existing summary. In that case, it joins a new summary, the counter associated to the state q is decremented and the counter of this new summary is incremented.
- Finally, Δ_e allows to empty the counters associated to summaries that have disappeared from the set of locations. It is defined as the set of transitions $(\mathcal{S}, \delta, \mathcal{S}')$ such that: $\mathcal{S} = \mathcal{S}'$ and there exists $(q, k) \notin \mathbf{lab}(\mathcal{S})$ with $\delta(x_q) = +1$ and $\delta(x_{(q, k)}) = -1$ and for all other counters, $\delta(x) = 0$.



■ **Figure 2** The structure of the VASS \mathcal{V} .



■ **Figure 3** One successor of location $\{(\{q_2\}, q_3, 1), (\{q_1\}, q_3, 2), (\{q_1, q_5\}, q_4, 1)\}$ (left location) after the broadcast transition $(q_{in}, !!b, q_7)$.



■ **Figure 4** Three successors of location $\{(\{q_2\}, q_3, 1)\}$ (top left location) after the broadcast transition $(q_{in}, !!d, q_1)$.

Transitions from Δ_e allow the transfer of counter values from summaries that have disappeared to the counter of their exit action state. This transfer is not immediate, as it is done through iterative decrements and increments of counters. In particular, it is possible for the execution of the VASS to continue without the counter of a disappeared summary being fully transferred. The processes represented by the counter of a disappeared summary behave in the same way as processes in the exit action state that do not take any action. These counters can be emptied later, but always from a location from where no summary with the same label exists. This ensures that there is no ambiguity between processes in a waiting region and those on an action state that have not yet been transferred to the appropriate counter.

► **Example 3.6.** Figures 2–4 illustrate the VASS \mathcal{V}_P built for the protocol P from Figure 1. Figure 2 shows its overall structure: any run reaching $(s_f, \mathbf{0})$ starts by incrementing the counter of the initial state and ends by decrementing the counter of the summary $(q_f, q_u, 1)$. Here, q_u is an artificial, unreachable state used to ensure that the counted processes must end in q_f (since they cannot be in q_u). Figure 4 illustrates how message receptions and summary creations are handled. The top-left location contains a single summary $(\{q_2\}, q_3, 1)$, representing configurations where all processes on Q_W are in q_2 , all progressing to q_3 . After the broadcast $(q_{in}, !!d, q_1)$, three behaviors may follow. In the first case (right), the sender creates a new summary $(\{q_1\}, q_4, 1)$ (pink). In the second (diagonal), it creates another new summary $(\{q_1\}, q_3, 2)$ (orange), indicating a different arrival time at q_3 . In the third (below), it joins an existing summary, and q_1 is added to the print. Well-formed executions restrict the number of summaries with the same exit state, as there are at most $|Q_W|$ distinct moments where processes can reach a given action state (Lemma 3.4). Figure 3 illustrates summary deletion. The transition $(q_{in}, !!b, q_7)$ causes the sender to join the pink summary (bottom one), incrementing its counter. Processes in the green summary $(\{q_2\}, q_3, 1)$ receive the message via $(q_2, ?b, q_3)$, reaching their exit state q_3 . This summary is deleted in the next location. From there, value of $x_{(q_3,1)}$ is transferred to the counter x_{q_3} with the loop transition. If it is not taken enough times, $x_{(q_3,1)}$ may remain non-zero. This does not affect correctness: $x_{(q_3,1)}$ will be decremented eventually from a state in which there is no summary labeled with $(q_3, 1)$. Not decrementing soon enough only delay the moment the corresponding processes will move from the action state q_3 .

► **Remark 3.7.** In the sequel of this paper, the size of \mathcal{V}_P will be of interest to us. Hence, observe that $|\text{Loc}| = |\text{CoSets}| + 3 \leq 2^{|Q_A| \times (|Q_W| + 1) \times 2^{|Q_W|}} + 3$ (as one summary is composed of a state in Q_A , one label in $[1, |Q_W| + 1]$ and one set of waiting states), and $|X| = |Q_A| + |Q_A| \times (|Q_W| + 1)$.

Soundness of the construction. We show now that if there exists a run in the VASS \mathcal{V}_P from $(s_0, \mathbf{0})$ to $(s_f, \mathbf{0})$, then in the network built from P , there exist $C \in \mathcal{I}$ and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$ and $C'(e) = q_f$ for all $e \in [1, \|C'\|]$.

We say that a configuration (ℓ, v) of \mathcal{V}_P is an *S-configuration* if $\ell = \mathcal{S}$ for some $\mathcal{S} \in \text{CoSets}$. The *implementation* of an S-configuration $\lambda = (\mathcal{S}, v)$ is the set of network configurations $\llbracket \lambda \rrbracket \subseteq \mathcal{C}$ defined as follows: $C \in \llbracket \lambda \rrbracket$ if and only if there exists a function $f : [1, \|C\|] \rightarrow \mathbf{X}$ such that $|f^{-1}(x)| = v(x)$ for all $x \in \mathbf{X}$, and

CondImpl1 for all $x_q \in \mathbf{X}$ with $q \in Q_A$, we have $C(e) = q$ for all $e \in f^{-1}(x_q)$;

CondImpl2 for all $x_{(q,k)} \in \mathbf{X}$, if there exists $(\text{pr}, q, k) \in \mathcal{S}$ for some $\text{pr} \subseteq Q_W$, then $C(e) \in \text{pr} \cup \{q\}$ for all $e \in f^{-1}(x_{(q,k)})$;

CondImpl3 for all $x_{(q,k)} \in \mathbf{X}$, if $(q, k) \notin \text{lab}(\mathcal{S})$, then $C(e) = q$ for all $e \in f^{-1}(x_{(q,k)})$.

In an implementation of λ , the processes populate states according to the values of the counters. All processes associated with a counter x_q of an action state will populate this exact state (**CondImpl1**). However, processes associated with a counter $x_{(q,k)}$ of a summary do not necessarily populate waiting states. This occurs when the label (q, k) does not appear in \mathcal{S} while the associated counter remains strictly positive. Such a situation arises when a summary has been previously deleted, but its counter has not been emptied. Since all associated processes have already reached the exit action state, the processes associated to the corresponding counter have to populate this active state (**CondImpl3**). If the summary with label (q, k) is active (i.e. its label appears in \mathcal{S}), the processes associated with $x_{(q,k)}$ will be in the corresponding waiting region, or in the exit action state (**CondImpl2**). This may seem contradictory, as all processes are supposed to reach their exit state simultaneously. However, the processes already on the exit action state are “old” processes, that remained after a previous deletion of this summary (cf. **CondImpl3**). These processes will be allowed to send messages only when they are identified with the counter of the active state, meaning when the corresponding transition in Δ_e will be taken. The next lemma allows to build an execution of the protocol P from an execution of the VASS \mathcal{V}_P .

► **Lemma 3.8.** *Let λ, λ' be two S-configurations of \mathcal{V}_P and $C \in \llbracket \lambda \rrbracket$. If $\lambda \xrightarrow{\delta} \lambda'$ in \mathcal{V}_P then there exists $C' \in \llbracket \lambda' \rrbracket$ such that $C \rightarrow^* C'$ for P .*

Sketch of Proof. Assume $\lambda = (\mathcal{S}, v)$ and $\lambda' = (\mathcal{S}', v')$. Then either $(\mathcal{S}, \delta, \mathcal{S}') \in \Delta_e$ or $(\mathcal{S}, \delta, \mathcal{S}') \in \Delta_t$ for some $t \in T$. In the first case, we show that if $C \in \llbracket \lambda \rrbracket$, then $C \in \llbracket \lambda' \rrbracket$. Otherwise, let $t = (q, !a, q')$, there is a process e such that $C(e) = q$. The transition δ in the VASS makes all the summaries and counters evolve according to the reception of the message a . According to the different cases a., b. or c., one can build a configuration $C' \in \llbracket \lambda' \rrbracket$ such that $C \rightarrow C'$. ◀

► **Lemma 3.9.** *If $(s_0, \mathbf{0}) \xrightarrow{\sim}^* (s_f, \mathbf{0})$ in \mathcal{V}_P , then there exist $C \in \mathcal{I}$ and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$ and $C'(e) = q_f$ for all $e \in [1, \|C'\|]$.*

Proof. From the definition of \mathcal{V}_P , any run from $(s_0, \mathbf{0})$ to $(s_f, \mathbf{0})$ is of the form $(s_0, \mathbf{0}) \xrightarrow{\sim}^* (\emptyset, v_{init}) \xrightarrow{\sim}^* (\{S_f\}, v_f) \xrightarrow{\sim}^* (s_f, \mathbf{0})$ where $v_{init}(x_{q_{in}}) = n$ for some $n \in \mathbb{N}$ and $v_{init}(x) = 0$ for all $x \in \mathbf{X} \setminus \{x_{q_{in}}\}$, whereas $v_f(x_{(q_u, 1)}) = v_{init}(x_{q_{in}}) = n$ and $v_f(x) = 0$ for all $x \in \mathbf{X} \setminus \{x_{(q_u, 1)}\}$. Define C as the initial configuration in \mathcal{I} with n processes (i.e. $\|C\| = n$). Trivially, $C \in \llbracket (\emptyset, v_{init}) \rrbracket$ and with Lemma 3.8 and a simple induction, we get that there exists $C' \in \llbracket (\{S_f\}, v_f) \rrbracket$ such that $C \rightarrow^* C'$. Observe that by definition of v_f , and since $C' \in \llbracket (\{S_f\}, v_f) \rrbracket$, there exists $f : [1, n] \rightarrow \mathbf{X}$ such that $f^{-1}(x_{(q_u, 1)}) = [1, n]$. Using **CondImpl2**, we have $C'(e) \in \{q_f, q_u\}$ for all $e \in [1, n]$ with $n = \|C'\|$. Since q_u is unreachable by construction, we deduce that $C'(e) = q_f$ for all $e \in [1, \|C'\|]$. ◀

Completeness of the construction. Let us first introduce some new definitions. Given a well-formed execution $\rho \in \Lambda \cup \Lambda_\omega$, two indices $0 \leq i < j < |\rho|$ and an action state $q_a \in Q_A$, we define $E_{q_a,j}^{\rho,i}$, the set of processes in waiting states in ρ_i , and whose next action state is q_a , reached at ρ_j . Formally, $E_{q_a,j}^{\rho,i} = \{e \in [1, \#\text{proc}(\rho)] \mid \rho_i(e) \in Q_W \text{ and } \mathbf{na}(\rho, i, e) = (q_a, j)\}$. Furthermore, an S-configuration $\lambda = (\mathcal{S}, v)$ is a *representative* of the configuration ρ_i in ρ iff the following conditions are respected:

- CondRepr1** for all $q \in Q_A$, $v(x_q) = |\rho_i^{-1}(q)|$, and
for all $q_a \in Q_A$, there is an injective function $r_{q_a} : \mathbf{na-index-set}(\rho, i, q_a) \rightarrow [1, |Q_W| + 1]$ s.t.:
- CondRepr2** $\mathcal{S} = \{(\rho_i(E_{q_a,j}^{\rho,i}), q_a, r_{q_a}(j)) \mid q_a \in Q_A, j \in \mathbf{na-index-set}(\rho, i, q_a)\}$
- CondRepr3** for all $q_a \in Q_A$, we have $v(x_{(q_a, r_{q_a}(j))}) = |E_{q_a,j}^{\rho,i}|$ for all
 $j \in \mathbf{na-index-set}(\rho, i, q_a)$ and $v(x_{(q_a, k)}) = 0$ for all $k \notin r_{q_a}(\mathbf{na-index-set}(\rho, i, q_a))$.

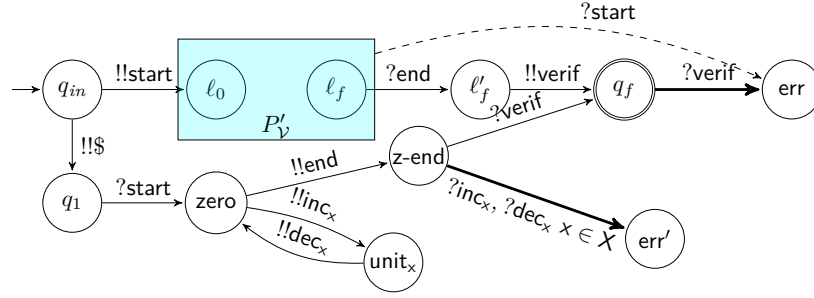
In a representative of the configuration ρ_i , the counters faithfully count the number of processes on active states (**CondRepr1**), and on waiting regions. The set $E_{q_a,j}^{\rho,i}$ gathers exactly the set of processes that populate a same waiting region in a representative of ρ_i in ρ : they all reach the same next action state q_a at the same instant j . The set $\mathbf{na-index-set}(\rho, i, q_a)$ being bounded by $|Q_W|$ (cf. Lemma 3.4), we can associate with each of these indices a unique identifier, given by the injective function r_{q_a} . We use a spare identifier for technical reasons, to handle more easily situations when $|Q_W|$ summaries exist in the location of the VASS, and a new summary is created while one of the previous summaries is deleted at the next step. Then, **CondRepr2** and **CondRepr3** require that the counters corresponding to the summaries and the summaries themselves reflect faithfully the situation in the configuration ρ_i with respect to ρ . Observe that such a representative is tightly linked to the simulated execution, since we need to know the future behavior of the different processes to determine the different summaries. Finally, if we let $\mathbf{repr}(\rho, i)$ be the set of all S-configurations that are representatives of ρ_i in ρ , we establish the following result before proving the main lemma of this subsection (Lemma 3.11).

► **Lemma 3.10.** *Let ρ be a well-formed execution, and $0 \leq i < |\rho|$. For all $\lambda_i \in \mathbf{repr}(\rho, i)$, there exists $\lambda_{i+1} \in \mathbf{repr}(\rho, i+1)$ such that $\lambda_i \rightsquigarrow^* \lambda_{i+1}$ in \mathcal{V}_P .*

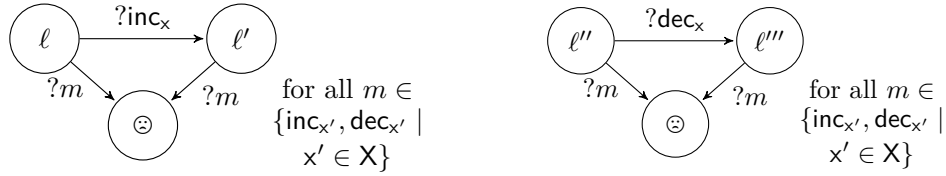
Sketch of Proof. From an S-configuration $\lambda_i = (\mathcal{S}_i, v_i)$ in $\mathbf{repr}(\rho, i)$, we can compute the effect on \mathcal{S}_i of the transition $t = (q, !!a, q')$ taken in ρ to go from ρ_i to ρ_{i+1} , and find a matching transition $(\mathcal{S}_i, \delta, \mathcal{S}_{i+1}) \in \Delta_t$ (according to whether q' is a waiting state or not, and in the latter case, whether it joins an existing summary or a new one, depending of the execution ρ). We then obtain a new configuration $\lambda' = (\mathcal{S}_{i+1}, v')$ such that $\lambda_i \rightsquigarrow^\delta \lambda'$. Note that λ' is not necessarily in $\mathbf{repr}(\rho, i+1)$: if some summaries have been deleted between λ_i and λ' , the transition $(\mathcal{S}_i, \delta, \mathcal{S}_{i+1})$ in \mathcal{V}_P has not updated the corresponding counters. Hence we need to apply transitions from Δ_e to empty counters corresponding to deleted summaries and accordingly increase corresponding counters on matching action states to obtain $\lambda_{i+1} = (\mathcal{S}_{i+1}, v_{i+1})$ in $\mathbf{repr}(\rho, i+1)$. ◀

► **Lemma 3.11.** *If there exist $C \in \mathcal{I}$, and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$ in P and $C'(e) = q_f$ for all $e \in [1, ||C'||]$, then $(s_0, \mathbf{0}) \rightsquigarrow^* (s_f, \mathbf{0})$ in \mathcal{V}_P .*

Proof. Thanks to Corollary 3.3, we know there exists a well-formed execution ρ from C to C' . Let $K = ||C||$ and $n = |\rho| - 1$. First, consider the sequence $(s_0, \mathbf{0}) \rightsquigarrow^* (s_0, v_0) \rightsquigarrow (\emptyset, v_0)$ where $v_0(x_{q_{in}}) = K$ and for all other counters x , $v_0(x) = 0$. Observe that, since $q_{in} \in Q_A$, $(\emptyset, v_0) \in \mathbf{repr}(\rho, 0)$. By applying inductively Lemma 3.10, we get that $(\emptyset, v_0) \rightsquigarrow^* (\mathcal{S}_1, v_1) \rightsquigarrow^* \dots \rightsquigarrow^* (\mathcal{S}_n, v_n)$ with $(\mathcal{S}_n, v_n) \in \mathbf{repr}(\rho, n)$. By definition, every time that $\mathbf{na-state}(\rho, i, e) = q_u$ for some process e , then $\mathbf{na-index}(\rho, i, e) = n + 1$,



■ **Figure 5** Protocol P associated to a VASS \mathcal{V} . The dashed edge $(\ell_f, ?start, err)$ will only be considered in Section 4.2.



■ **Figure 6** Part of P'_V that simulates respectively the transition $(\ell, \delta, \ell') \in \Delta$ with $\delta(x) = 1$ at the left, and $(\ell'', \delta, \ell''') \in \Delta$ with $\delta(x) = -1$ at the right.

hence there will always be at most one summary with exit state q_u . So in the execution we build, every time that some summary $(pr, q_u, k) \in \mathcal{S}_i$, we chose $k = 1$. We then have $v_n(x_{q_u,1}) = K$ and $v_n(x) = 0$ for all other $x \in X \setminus \{x_{q_u,1}\}$. Hence, by construction of \mathcal{V}_P , we have $(\mathcal{S}_n, v_n) \rightsquigarrow (s_f, v_n) \rightsquigarrow^* (s_f, \mathbf{0})$. ◀

Using Lemma 3.10 and Lemma 3.11 and the fact that the reachability problem for VASS is decidable (see Theorem 2.5) we get the main theorem of this section.

► **Theorem 3.12.** *SYNCHRO restricted to Wait-Only protocols is decidable.*

3.3 Lower Bound for Synchro in Wait-Only Protocols

In this subsection we reduce the reachability problem for VASS to SYNCHRO. We fix a VASS $\mathcal{V} = (\text{Loc}, \ell_0, X, \Delta)$ and a final location $\ell_f \in \text{Loc}$. W.l.o.g., we suppose that any transition in Δ either increments or decrements of 1 exactly one counter. Hence, for a transition $(\ell, \delta, \ell') \in \Delta$, we might describe δ by giving only $\delta(x)$ for the only $x \in X$ such that $\delta(x) \neq 0$.

We explain how to build a protocol P_V that simulates \mathcal{V} : it is depicted in Figure 5 in which we don't consider the dashed edge. The states zero and $\{\text{unit}_x \mid x \in X\}$ represent the evolution of the different counters of the VASS while the blue box P'_V describes the evolution of \mathcal{V} with respect to its locations. Formally, P'_V consists of a set of states $Q_V = \text{Loc} \cup \{\odot\}$ and a set of transitions $T_V = \{(\ell, ?inc_x, \ell') \mid (\ell, \delta, \ell') \in \Delta \text{ and } \delta(x) = 1\} \cup \{(\ell, ?dec_x, \ell') \mid (\ell, \delta, \ell') \in \Delta \text{ and } \delta(x) = -1\} \cup \{(\ell, ?m, \odot) \mid m \in \{inc_x, dec_x \mid x \in X\}\}$. Some transitions of T_V are depicted in Figure 6. We then obtain the following theorem.

► **Theorem 3.13.** *SYNCHRO with Wait-Only protocols is Ackermann-complete.*

Sketch of Proof. The upper bound follows from Theorem 3.12 and Theorem 2.5. For the lower bound, we give the ideas that prove that the reduction described above is correct. In particular, there exist $C_0 \in \mathcal{I}$ and $C_f \in \mathcal{C}$ such that $C_0 \rightarrow^* C_f$ in P_V and $C_f(e) = q_f$ for all $e \in [1, \|C_f\|]$ if and only if $(\ell_0, \mathbf{0}) \rightsquigarrow^* (\ell_f, \mathbf{0})$ in \mathcal{V} .

Assume that $(\ell_0, \mathbf{0}) \rightsquigarrow^* (\ell_f, \mathbf{0})$ in \mathcal{V} , and let $K \in \mathbb{N}$ be the maximum value of $\sum_{x \in X} v(x)$ reached during the run. We choose C_0 such that $\|C_0\| = K + 1$. The execution proceeds as follows: all processes except one (the leader) broadcast a dummy message $\$$. The leader then broadcasts **start** and reaches ℓ_0 , while the others move to **zero**. These simulate the counter values, and the leader simulates the VASS by moving through the states of P'_V . The value of counter x is represented by the number of processes in unit_x . To simulate an increment of x from (ℓ_i, v_i) to (ℓ_{i+1}, v_{i+1}) , a process in **zero** sends inc_x , which the leader receives to transition to ℓ_{i+1} . If the current configuration correctly represents (ℓ_i, v_i) , the new one faithfully represents (ℓ_{i+1}, v_{i+1}) . At the end of the simulation, the leader reaches ℓ_f , and the other processes remain in **zero**. They then broadcast **end**, and the leader (now in ℓ'_f) broadcasts **verif**, gathering all processes in q_f .

To prove the other direction, we must show that the processes cannot cheat in simulating the VASS. First, note that reaching q_f requires exactly one broadcast of **verif**, a second would send the original sender to **err**. This ensures that only one process (the leader) simulates the VASS by moving through P_V . Second, counter updates must follow the VASS transitions: any unauthorized update would cause the leader to receive an unexpected message and transition to the losing state \odot . Finally, when the leader reaches ℓ_f , all other processes must be in **zero**. If not, problems occur during the final steps: one process (the helper) broadcasts **end**, which sends the leader to ℓ'_f . Then, all remaining processes must reach **z-end** before the unique **verif** broadcast. If a process in unit_x moves to **z-end**, it must broadcast dec_x , which is received by other processes in **z-end**, sending them, including the helper, to **err'**. As a result, the helper misses **verif** and cannot reach q_f , preventing a successful execution. Thus, the simulation must end with the leader in ℓ_f and all others in **zero**. ◀

4 Synchro is ExpSpace-complete when the Target State is an Action State

Upper bound. We show that SYNCHRO when $q_f \in Q_A$ can be reduced to the mutual reachability problem on VASS, defined as follows: given a VASS $\mathcal{V} = (\text{Loc}, \ell_0, X, \Delta)$ and a location ℓ_f , do there exist a run from $(\ell_0, \mathbf{0})$ to $(\ell_f, \mathbf{0})$ and one from $(\ell_f, \mathbf{0})$ to $(\ell_0, \mathbf{0})$. This problem is shown to be in EXPSPACE [16]. More precisely, Leroux establishes the EXPSPACE-membership and provides a bound on the lengths of the runs, for the mutual reachability problem in VAS (Vector Addition Systems). However, by applying the VASS-to-VAS transformation presented in [14], we get the following result.

► **Theorem 4.1** ([16] Corollary 10.6, Transformation of [14]). *Given a VASS $\mathcal{V} = (\text{Loc}, \ell_0, X, \Delta)$, if there are two runs $(\ell_0, \mathbf{0}) \rightsquigarrow^* (\ell_f, \mathbf{0})$ and $(\ell_f, \mathbf{0}) \rightsquigarrow^* (\ell_0, \mathbf{0})$, then there are two runs $(\ell_0, \mathbf{0}) \rightsquigarrow^* (\ell_f, \mathbf{0})$ and $(\ell_f, \mathbf{0}) \rightsquigarrow^* (\ell_0, \mathbf{0})$ whose lengths are bounded by $17(|X| + 3)^2 x^{15(|X|+3)^{|X|+5}}$ where $x = (1 + 2(|\text{Loc}| + 1)^2)^2$.*

Given a Wait-Only protocol $P = (Q, \Sigma, q_{in}, T)$ and a target state $q_f \in Q_A$, we explain how to transform it into a VASS where mutual reachability is equivalent to SYNCHRO. We build the VASS $\mathcal{V}_P = (\text{Loc}, X, s_0, \Delta)$ as described in Section 3.2, with some modifications. The first two modifications simply encode the fact that the target state is not a waiting state anymore, while the third one ensures the mutual reachability.

- We add a state s'_f and the transition $(\{S_f\}, \mathbf{0}, s_f)$ is replaced by two transitions: $(\emptyset, \delta, s'_f)$ and (s'_f, δ', s_f) . Here, $\delta(x_{q_f}) = -1$, $\delta'(x_{q_f}) = 1$, and for all other counters x , $\delta(x) = \delta'(x) = 0$. It prevents the reachability of $(s_f, \mathbf{0})$ to be trivial with the run $(s_0, \mathbf{0}) \rightsquigarrow (\emptyset, \mathbf{0}) \rightsquigarrow (s_f, \mathbf{0})$.

- The transition (s_f, δ_f, s_f) is replaced by (s_f, δ'_f, s_f) , where $\delta'_f(x_{q_f}) = -1$ and $\delta'_f(x) = 0$ for all other counters x .
- We add a transition $(s_f, \mathbf{0}, s_0)$ to allow the resetting of \mathcal{V}_P to s_0 after reaching s_f .

► **Lemma 4.2.** *There exists $C \in \mathcal{I}$ and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$ and, for all $e \in [1, ||C'||]$, $C'(e) = q_f$ if and only if there are two runs $(s_0, \mathbf{0}) \rightsquigarrow^* (s_f, \mathbf{0})$ and $(s_f, \mathbf{0}) \rightsquigarrow^* (s_0, \mathbf{0})$ in \mathcal{V}_P .*

Sketch of Proof. Assume that there exist $C \in \mathcal{I}$ and $C' \in \mathcal{C}$ such that $C \rightarrow^* C'$ and, for all $e \in [1, ||C'||]$, $C'(e) = q_f$. Since the main body of \mathcal{V}_P is the same as in Section 3.2 we can, like in Lemma 3.11, build a run of \mathcal{V}_P from $(s_0, \mathbf{0})$ to $(s_f, \mathbf{0})$. By construction of \mathcal{V}_P , $(s_f, \mathbf{0}) \rightsquigarrow (s_0, \mathbf{0})$. Assume now that $(s_0, \mathbf{0}) \rightsquigarrow^* (s_f, \mathbf{0})$ (and $(s_f, \mathbf{0}) \rightsquigarrow (s_0, \mathbf{0})$). Observe that it might be the case that the run looks like $(s_0, \mathbf{0}) \rightsquigarrow^* (s_f, v_1) \rightsquigarrow (s_0, v_1) \rightsquigarrow^* (s_f, v_2) \dots (s_0, v_k) \rightsquigarrow^* (s_f, \mathbf{0})$. By construction of the VASS, we know that there is an execution $(s_0, \mathbf{0}) \rightsquigarrow^* (\emptyset, v'_0) \rightsquigarrow^* (\emptyset, v''_0) \rightsquigarrow (s'_f, v'''_0) \rightsquigarrow (s_f, v''_0) \rightsquigarrow^* (s_f, v_1)$. Here v'_0 is the valuation obtained after having increased the counter $x_{q_{in}}$, v''_0 is the valuation obtained just before going to s'_f , and v_1 is obtained after having decremented counter x_{q_f} . Adapting the proof of Lemma 3.9 to this case, we deduce the existence of an execution of the protocol from an initial configuration C_0 to a configuration C'_0 such that $|C'^{-1}_0(q)| = 0$ for all $q \in Q_W$, and $|C'^{-1}_0(q)| = v''_0(x_q) + \sum_{i=1, \dots, |Q_W|+1} v''_0(x_{(q,i)})$ for all $q \in Q_A$. From the portion of run of the VASS $(s_0, v_1) \rightsquigarrow^* (\emptyset, v'_1) \rightsquigarrow^* (\emptyset, v''_1) \rightsquigarrow (s'_f, v'''_1) \rightsquigarrow (s_f, v''_1) \rightsquigarrow^* (s_f, v_2)$, we similarly get another sequence of configurations of the protocol from a configuration C_1 (not necessarily initial because v_1 might not be equal to 0 for some counters other than $x_{q_{in}}$) to a configuration C'_1 such that $C_1 \rightarrow^* C'_1$ and $|C'^{-1}_1(q)| = 0$ for all $q \in Q_W$. Observe however that these two sequences can be merged into a single one, of size $v'_0(x_{q_{in}}) + (v'_1(x_{q_{in}}) - v_1(x_{q_{in}}))$ (the second part corresponds to processes added in q_{in} with transition (s_0, δ_{in}, s_0) , and $\delta_{in}(x_{q_{in}}) = 1$). The execution then first behaves like the execution from C_0 to C'_0 , potentially leaving some processes in the initial state (in particular processes from $v'_1(x_{q_{in}}) - v_1(x_{q_{in}})$). Once this first part is over, all the processes are either in the initial state, or in an action state (because $C'^{-1}_0(q) = 0$ for all $q \in Q_W$). Then, one can simulate the second sequence, (processes already in q_f from the first execution won't be affected because they are in an action state, so they won't receive any message. This would not be true if $q_f \in Q_W$). Since the execution of \mathcal{V}_P eventually reaches $(s_f, \mathbf{0})$, it means that it reaches (s_f, v_{k+1}) where $v_{k+1}(x) = 0$ for all $x \neq x_{q_f}$. Then, by iterating the construction described above, one obtains an execution of P from an initial configuration to a configuration C_f such that $C_f(e) = q_f$ for all $e \in [1, ||C_f||]$. ◀

As runs of doubly exponential length can be guessed in exponential space in the size of the VASS and $\text{EXPSPACE} = \text{NEXPSPACE}$, we get the following theorem using Remark 3.7 and Theorem 4.1. Observe that even if the number of locations is doubly exponential in the size of the protocol, the bound of Theorem 4.1 is polynomial in $|\text{Loc}|$ and doubly exponential in $|\text{X}|$, hence lengths of the runs to guess remain doubly exponential in the size of the protocol.

► **Theorem 4.3.** *SYNCHRO for Wait-Only protocols is in EXPSPACE when $q_f \in Q_A$.*

Lower bound. EXPSPACE-hardness follows from a reduction of the coverability problem in VASS, which is EXPSPACE-hard [19], and stated as follows: given a VASS $\mathcal{V} = (\text{Loc}, \ell_0, \text{X}, \Delta)$ of dimension $d \in \mathbb{N}$, and a location $\ell_f \in \text{Loc}$, decide whether there is an execution from $(\ell_0, \mathbf{0})$ to (ℓ_f, v) for some $v \in \mathbb{N}^d$.

The reduction uses the protocol depicted in Figure 5, this time including the dashed edge but excluding the thick edges: q_f is now an action state. Then ℓ_f is coverable iff there exists an execution $C_0 \rightarrow^* C_f$ in which all processes reach q_f . The execution of the VASS can be simulated in the protocol like in Section 3.3. The processes that may still remain in the states unit_x can reach z-end even when ℓ'_f is populated (recall that all the thick edges have been removed). Once this is done, all processes can gather in q_f . Conversely, if there exists an execution in which all processes reach q_f , it means that ℓ_f is coverable. Let e_0 be the first process to broadcast **start**, then an execution of the VASS covering ℓ_f can be built based on the sequence of states visited by e_0 between ℓ_0 and ℓ_f . This relies on three keys observations: (1) The process e_0 must visit ℓ_f , otherwise it cannot reach q_f . (2) When e_0 is between ℓ_0 and ℓ_f , e_0 is the only process in this region. If another process attempts to join by broadcasting **start**, e_0 will receive the message and go to **err**, preventing it from reaching q_f . (3) When e_0 reaches ℓ_0 for the first time, no other process is in the states $\{\text{unit}_x \mid x \in X\}$, since **start** is sent for the first time at this point. Hence, with Theorem 4.3, it establishes the following result.

► **Theorem 4.4.** *SYNCHRO for Wait-Only protocols is EXPSPACE-complete when $q_f \in Q_A$.*

5 Solving the Repeated Coverability problem for Wait-Only Protocols

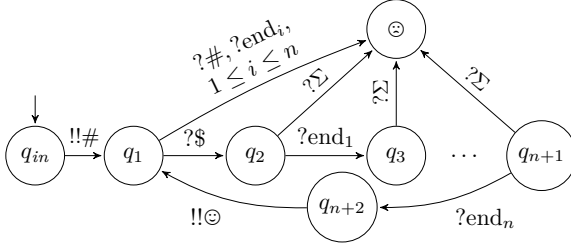
Upper Bound. We show that REPCOVER on Wait-Only protocols is in EXPSPACE, via the repeated coverability problem in VASS: given a VASS $\mathcal{V} = (\text{Loc}, \ell_{\text{init}}, X, \Delta)$ and a location $\ell_f \in \text{Loc}$, is there an infinite execution $(\ell_{\text{init}}, \mathbf{0}) \rightsquigarrow (\ell_1, v_1) \rightsquigarrow \dots$ such that, for all $i \in \mathbb{N}$, there exists $j > i$ such that $\ell_j = \ell_f$? We rely on the following theorem.

► **Theorem 5.1** (Theorem 3.1 of [13]). *For a VASS $\mathcal{V} = (\text{Loc}, \ell_{\text{init}}, X, \Delta)$, the repeated coverability problem is solvable in $O(\log(l) + \log(|\text{Loc}|))2^{c|X|\log(|X|)}$ nondeterministic space for some constant c independent of \mathcal{V} , and where the absolute values of components of vectors in Δ are smaller than l .*

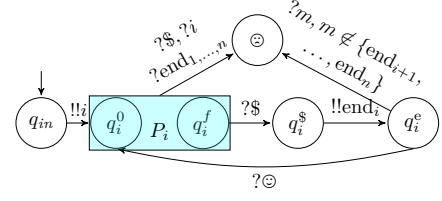
Let $P = (Q, \Sigma, q_{\text{in}}, T)$ be a Wait-Only protocol and $t_f = (q, !!a, q')$ the transition that has to occur infinitely often. We build a VASS \mathcal{V}_P based on the construction presented in Section 3.2. This time, we add a new set of states $\text{CoSets}_{\odot} = \{(\mathcal{S}, \odot) \mid \mathcal{S} \in \text{CoSets}\}$, and the set of transitions $\{(\mathcal{S}, \delta, (\mathcal{S}', \odot)) \mid (\mathcal{S}, \delta, \mathcal{S}') \in \Delta_{t_f}\} \cup \{(\mathcal{S}, \odot, \mathbf{0}, \mathcal{S}) \mid \mathcal{S} \in \text{CoSets}\}$. Hence, when a process takes the transition t_f , it is highlighted in \mathcal{V}_P by going in a location tagged with \odot , before continuing the execution. Taking the protocol of Figure 1, with $t_f = (q_{\text{in}}, !!d, q_1)$, the previous transition from the location of the VASS $\{(\{q_3\}, q_4, 1)\}$ to the location $\{(\{q_3\}, q_4, 1), (\{q_1\}, q_6, 1)\}$ (see Figure 4) is now transformed into two transitions in the VASS we build here: one from $\{(\{q_3\}, q_4, 1)\}$ to the location $\{(\{q_3\}, q_4, 1), (\{q_1\}, q_6, 1), \odot\}$ and one from $\{(\{q_3\}, q_4, 1), (\{q_1\}, q_6, 1), \odot\}$ to $\{(\{q_3\}, q_4, 1), (\{q_1\}, q_6, 1)\}$. There is an execution of P with a process that takes t_f infinitely often iff there is an execution of \mathcal{V}_P where one state in CoSets_{\odot} is visited infinitely often. The procedure to decide REPCOVER is then: guess a location ℓ_f in CoSets_{\odot} and solve the repeated coverability problem for \mathcal{V}_P and ℓ_f . From Remark 3.7 and Theorem 5.1, this can be done in $O(\log(2) + (|Q| + 1)^2 \times 2^{|Q|} + 2)2^{c(|Q|+1)^3 \log((|Q|+1)^3)}$ nondeterministic space. The overall procedure is then in $\text{NEXPSPACE} = \text{EXPSPACE}$. Hence, the following theorem.

► **Theorem 5.2.** *REPCOVER is in EXPSPACE.*

► **Remark 5.3.** One could define REPCOVER with a reception transition that has to occur infinitely often. The VASS we described hereabove can be adapted by enlarging each location with the current state of the witness process. This can also be done in EXPSPACE.



■ **Figure 7** The part of P with transition $t_f = (q_{n+2}, !!@, q_1)$. We write $?\Sigma$ for $?m, \forall m \in \Sigma$.



■ **Figure 8** The part of P simulating automaton i .

Lower bound. We reduce the intersection non-emptiness problem for deterministic finite automata, which is known to be PSPACE-complete [15], to REPCOVER. Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be deterministic, complete finite automata, with $\mathcal{A}_i = (A, Q_i, q_i^0, q_i^f, \Delta_i)$ for $1 \leq i \leq n$. The reduction assumes a single accepting state per automaton, which does not affect complexity. We denote by A^* the set of words over the alphabet A , and extend each transition function Δ_i to A^* by defining: $\Delta_i^*(q, \varepsilon) = q$, and $\Delta_i^*(q, wa) = \Delta_i(\Delta_i^*(q, w), a)$ for all $w \in A^*$ and $a \in A$.

We define a protocol $P = (Q, \Sigma, q_{in}, T)$ composed of several components. For each automaton \mathcal{A}_i , P includes a component $P_i = (Q^i, T^i)$ (see Figure 8), where $Q^i = Q_i$ and $T^i = \{(q_1^i, ?a, q_2^i) \mid (q_1^i, a, q_2^i) \in \Delta_i\}$. The main control component (see Figure 7) includes the transition $t_f = (q_{n+2}, !!@, q_1)$, which must occur infinitely often. A process taking t_f infinitely often also receives infinitely many sequences of messages $\$ \cdot \text{end}_1 \cdot \text{end}_2 \dots \text{end}_n$, where each end_i is broadcast by the component simulating \mathcal{A}_i . In addition to transitions in Figures 7 and 8, T also includes: $\{(q_{in}, !!a, q_{in}) \mid a \in A\} \cup \{(q_{in}, !!$, $q_{in})\}$. One can show that there exists a word $w = a_1 \dots a_n$ in the intersection of the n automata iff there is an execution in which t_f is taken infinitely often.$

► **Theorem 5.4.** *REPCOVER is PSPACE-hard.*

References

- 1 K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986. doi:10.1016/0020-0190(86)90071-2.
- 2 Peter Chini, Roland Meyer, and Prakash Saivasan. Liveness in broadcast networks. *Computing*, 104(10):2203–2223, 2022. doi:10.1007/S00607-021-00986-Y.
- 3 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.
- 4 Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the automated verification of multithreaded java programs. In *TACAS 2002*, volume 2280 of *LNCS*, pages 173–187. Springer, 2002. doi:10.1007/3-540-46002-0_13.
- 5 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 313–327, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15375-4_22.
- 6 E.A. Emerson and K.S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998. doi:10.1109/LICS.1998.705644.

- 7 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140. Springer-Verlag, 2013. doi:10.1007/978-3-642-39799-8_8.
- 8 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 352–359. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782630.
- 9 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.
- 10 Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder. Safety analysis of parameterised networks with non-blocking rendez-vous. In *CONCUR'23*, volume 279 of *LIPICs*, pages 7:1–7:17. loss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CONCUR.2023.7.
- 11 Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder. Safety verification of wait-only non-blocking broadcast protocols. In *Application and Theory of Petri Nets and Concurrency - 45th International Conference, PETRI NETS 2024*, volume 14628 of *Lecture Notes in Computer Science*, pages 291–311. Springer, 2024. doi:10.1007/978-3-031-61433-0_14.
- 12 Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder. Wait-only broadcast protocols are easier to verify, 2025. arXiv:2506.22144.
- 13 Peter Habermehl. On the complexity of the linear-time μ -calculus for petri-nets. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 1997. doi:10.1007/3-540-63139-9_32.
- 14 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.
- 15 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.
- 16 Jérôme Leroux. Vector addition system reversible reachability problem. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 2011. doi:10.1007/978-3-642-23217-6_22.
- 17 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.
- 18 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.
- 19 R.J. Lipton. *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science). Department of Computer Science, Yale University, 1976. URL: <https://books.google.fr/books?id=7iSbGwAACAAJ>.
- 20 S. Schmitz and P. Schnoebelen. The power of well-structured systems. In *CONCUR'13*, volume 8052 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2013. doi:10.1007/978-3-642-40184-8_2.
- 21 Rüdiger Valk. Self-modifying nets, a natural extension of petri nets. In *ICALP'78*, volume 62 of *LNCS*, pages 464–476. Springer, 1978. doi:10.1007/3-540-08860-1_35.