Global State Estimates for Distributed Systems

Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, Thierry Massart

Global State Estimates for Distributed Systems

Gabriel Kalyon[1], Tristan Le Gall[2], Hervé Marchand[3] and Thierry Massart[1⋆]
[1] Université Libre de Bruxelles (U.L.B.), Campus de la Plaine, Bruxelles, Belgique
[2] CEA LIST, LMeASI, boîte 94, 91141 Gif-sur-Yvette, France
[3] INRIA, Rennes - Bretagne Atlantique, France

Abstract. We consider distributed systems modeled as *communicating finite state machines* with reliable unbounded FIFO channels. As an essential sub-routine for control, monitoring and diagnosis applications, we provide an algorithm that computes, during the execution of the system, an estimate of the current global state of the distributed system for each local subsystem. This algorithm does not change the behavior of the system; each subsystem only computes and records a symbolic representation of the state estimates, and piggybacks some extra information to the messages sent to the other subsystems in order to refine their estimates. Our algorithm relies on the computation of reachable states. Since the reachability problem is undecidable in our model, we use abstract interpretation techniques to obtain regular overapproximations of the possible FIFO channel contents, and hence of the possible current global states. An implementation of this algorithm provides an empirical evaluation of our method.

1 Introduction

During the execution of a computer system, the knowledge of its global state may be crucial information, for instance to control which action can or must be done, to monitor its behavior or perform some diagnostic. Distributed systems, are generally divided into two classes, depending on whether the communication between subsystems is *synchronous* or not. When the synchrony hypothesis [1] can be made, each local subsystem can easily know, at each step of the execution, the global state of the system (assuming that there is no internal action). When considering *asynchronous* distributed systems, this knowledge is in general impossible, since the communication delays between the components of the system must be taken into account. Therefore, each local subsystem can *a priori* not immediately know either the local state of the other subsystems or the messages that are currently in transfer.

In this paper, we are interested in the asynchronous framework where we consider that the system is composed of n subsystems that asynchronously communicate through reliable *unbounded* FIFO channels. These subsystems are modeled by *communicating finite state machines* (CFSM) [4] that explicitly express the work and communications of a distributed system. This model appears to be essential for concurrent systems in which components cooperate via asynchronous message passing through unbounded buffers (they are e.g. widely used to model communication protocols). We thus assume that the distributed system is already built and the architecture of communication between the different subsystems is fixed. Our aim is to provide an algorithm that allows us to compute, in each

⋆ This work has been done in the MoVES project (P6/39), part of the IAP-Phase VI Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy.

subsystem of a distributed system $\mathcal{T}$, an estimate of the current state of $\mathcal{T}$. More precisely, each subsystem or a local associated *estimator* computes a set of possible global states, including the contents of the channels, in which the system $\mathcal{T}$ can be; it can be seen as a particular case of monitoring with partial observation. In our framework, we assume that the subsystems (or associated estimators) can record their own state estimate and that some extra information can be piggybacked to the messages normally exchanged by the subsystems. Indeed, without this additional information, since a local subsystem cannot observe the other subsystems nor the FIFO channel contents, the computed state estimates might be too rough. Our computation is based on the use of the reachability operator, which cannot always be done in the CFSM model for undecidability reasons. To overcome this obstacle, we rely on the abstract interpretation techniques we presented previously in [14]. They ensure the termination of the computations of our algorithm by overapproximating in a symbolic way the possible FIFO channel contents (and hence the state estimates) by *regular languages*. Computing state estimates is useful in many applications. For example, this information can be used to locally control the system in order to prevent it from reaching some given forbidden global states [5], or to perform some diagnosis to detect some faults in the system [8, 20]. For these two potential applications, a more precise state estimate allows the controller or the diagnoser to take better decisions.

This problem differs from the synthesis problem (see e.g. [16, 10, 6]) which consists in synthesizing a distributed system (together with its architecture of communication) equivalent to a given specification. It also differs from the methodology described in [11] in the sense that in their framework, the authors try to infer from a distributed observation of a distributed system (modeled by a High Level Message Sequence Chart) the set of sequences that explains this observation. It is also different from *model checking* techniques [2, 3, 9, 12] that proceed to a symbolic exploration of all the possible states of the system, without running it. We however use the same symbolic representation of queue contents as in [2, 12]. In [22], Kumar and Xu propose a distributed algorithm which computes an estimate of the current state of a system. More precisely, local estimators maintain and update local state estimates from their own observation of the system and information received from the other estimators. In their framework, the local estimators communicate between them through reliable FIFO channels with delays, whereas the system is monolithic and therefore in their case, a global state is simpler than for our distributed systems composed of several subsystems together with communicating FIFO channels. In [21], Tripakis studies the decidability of the existence of controllers such that a set of responsiveness properties is satisfied in a decentralized framework with communication delays between the controllers. He shows that the problem is undecidable when there is no communication or when the communication delays are unbounded. He conjectures that the problem is decidable when the communication delays are bounded. Other works dealing with communication (with or without delay) between agents can be found in [19, 15].

Below, in section 2, we define the formalism of *communicating finite state machines*, that we use. We formally define, in section 3, the state estimate mechanisms and the notion of state estimators. In section 4, we provide an algorithm to compute an estimate of the current state of a distributed system and prove its correctness. We explain, in section 5, how we can ensure the termination of this algorithm by using abstract interpretation techniques. Finally, section 6 gives some experimental results. Technical proofs are given in Appendix A.

2 Communicating Finite State Machines as a Model of the System

We model a distributed system by the standard formalism of *communicating finite state machines* [4] which use reliable unbounded FIFO channels (also called *queues*) to communicate. A *global state* in this model is given by the local state of each subsystem together with the content of each FIFO queue. Therefore, since no bound is given either in the transmission delay, or on the length of the FIFO queues, the global state space of the distributed system is *a priori* infinite.

Definition 1 (Communicating Finite State Machines). *A* communicating finite state machine *(*CFSM*) $\mathcal{T}$ is defined as a 6-tuple $\langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$, where (i) L is a finite set of locations, (ii) $\ell_0 \in L$ is the initial location, (iii) Q is a set of queues that $\mathcal{T}$ can use, (iv) M is a finite set of messages, (v) $\Sigma \subseteq Q \times \{!, ?\} \times M$ is a finite set of actions, that are either an* output $a!m$ *to specify that the message $m \in M$ is written on the queue $a \in Q$ or an* input $a?m$ *to specify that the message $m \in M$ is read on the queue $a \in Q$, (vi) $\Delta \subseteq L \times \Sigma \times L$ is a finite set of transitions.*

An output transition $\langle \ell, i!m, \ell'\rangle$ indicates that when the system moves from the location ℓ to ℓ', a message m is added at the end of the queue i. An input transition $\langle \ell, i?m, \ell'\rangle$ indicates that, when the system moves from ℓ to ℓ', a message m must be present at the beginning of the queue i and is removed from this queue. Moreover, throughout this paper, we assume that $\mathcal{T}$ is deterministic, meaning that for all $\ell \in L$ and $\sigma \in \Sigma$, there exists at most one location $\ell' \in L$ such that $\langle \ell, \sigma, \ell'\rangle \in \Delta$. For $\sigma \in \Sigma$, $\mathsf{Trans}(\sigma)$ denotes the set of transitions of $\mathcal{T}$ labeled by σ. The occurrence of a transition will be called an *event* and given an event e, δ_e denotes the corresponding transition. Note that the model could also have included internal events; but, since it does not bring any particular difficulty, to simplify the presentation, we have not integrated them here. The semantics of a CFSM is defined as follows:

Definition 2 (Semantics). *The semantics of a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$ is given by an infinite Labeled Transition System (LTS) $[\![\mathcal{T}]\!] = \langle X, \boldsymbol{x}_0, \Sigma, \rightarrow\rangle$, where (i) $X \stackrel{\mathbf{def}}{=} L \times (M^*)^{|Q|}$ is the set of states, (ii) $\boldsymbol{x}_0 \stackrel{\mathbf{def}}{=} \langle \ell_0, \epsilon, \ldots, \epsilon\rangle$ is the initial state, (iii) Σ is the set of actions, and $(iv) \rightarrow \stackrel{\mathbf{def}}{=} \bigcup_{\delta\in\Delta} \stackrel{\delta}{\rightarrow} \subseteq X \times \Sigma \times X$ is the transition relation where $\stackrel{\delta}{\rightarrow}$ is defined by:*

$$\frac{\delta = \langle \ell, i!m, \ell'\rangle \in \Delta \quad w_i' = w_i \cdot m}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|}\rangle \stackrel{\delta}{\rightarrow} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|}\rangle}$$

$$\frac{\delta = \langle \ell, i?m, \ell'\rangle \in \Delta \quad w_i = m \cdot w_i'}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|}\rangle \stackrel{\delta}{\rightarrow} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|}\rangle}$$

A global state of a CFSM $\mathcal{T}$ is thus a tuple $\langle \ell, w_1, ..., w_{|Q|}\rangle \in X = L \times (M^*)^{|Q|}$ where ℓ is the current location of $\mathcal{T}$ and $w_1, ..., w_{|Q|}$ are finite words on M^* which give the content of the queues in Q. At the beginning, all queues are empty, so the initial state is $\boldsymbol{x}_0 = \langle \ell_0, \varepsilon, \cdots, \varepsilon\rangle$. Given a CFSM $\mathcal{T}$, two states $\boldsymbol{x}, \boldsymbol{x}' \in X$ and an event e, to simplify the notations we sometimes denote $\boldsymbol{x} \stackrel{\delta_e}{\rightarrow} \boldsymbol{x}'$ by $\boldsymbol{x} \stackrel{e}{\rightarrow} \boldsymbol{x}'$. An *execution* of $\mathcal{T}$ is a sequence $\boldsymbol{x}_0 \stackrel{e_1}{\longrightarrow} \boldsymbol{x}_1 \stackrel{e_2}{\longrightarrow} \ldots \stackrel{e_m}{\longrightarrow} \boldsymbol{x}_m$ where $\boldsymbol{x}_i \stackrel{e_{i+1}}{\longrightarrow} \boldsymbol{x}_{i+1} \in \rightarrow\ \forall i \in [0, m-1]$. Given a set of states

$Y \subseteq X$, $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(Y)$ corresponds to the set of states that are reachable in $[\![\mathcal{T}]\!]$ from Y only firing transitions of $\Delta' \subseteq \Delta$ in $\mathcal{T}$. It is defined by $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \bigcup_{n\geq 0}(\mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y))^n$ where $(\mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y))^n$ is the n^{th} functional power of $\mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y)$, defined by: $\mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \{\boldsymbol{x}' \in X | \exists \boldsymbol{x} \in Y, \exists \delta \in \Delta' : \boldsymbol{x} \stackrel{\delta}{\rightarrow} \boldsymbol{x}'\}$. Although there is no general algorithm that can exactly compute the reachability set in our setting [4], there exist some techniques that allow us to compute an overapproximation of this set (see section 5). Given a sequence of actions $\overline{\sigma} = \sigma_1 \cdots \sigma_m \in \Sigma^*$ and two states $x, x' \in X$, $x \stackrel{\overline{\sigma}}{\rightarrow} x'$ denotes that the state x' is reachable from x by executing $\overline{\sigma}$.

Asynchronous Product. A distributed system $\mathcal{T}$ is generally composed of several subsystems $\mathcal{T}_i$ ($\forall i \in [1, n]$) acting in parallel. In fact, $\mathcal{T}$ is defined by a CFSM resulting from the asynchronous (interleaved) product of the n subsystems $\mathcal{T}_i$, also modeled by CFSMs. This can be defined through the asynchronous product of two subsystems.

Definition 3. *Given 2 CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, Q_i, M_i, \Sigma_i, \Delta_i\rangle$ ($i = 1, 2$), their asynchronous product, denoted by $\mathcal{T}_1||\mathcal{T}_2$, is defined by a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$, where $L \stackrel{\mathbf{def}}{=} L_1 \times L_2$, $\ell_0 \stackrel{\mathbf{def}}{=} \ell_{0,1} \times \ell_{0,2}$, $Q \stackrel{\mathbf{def}}{=} Q_1 \cup Q_2$, $M \stackrel{\mathbf{def}}{=} M_1 \cup M_2$, $\Sigma \stackrel{\mathbf{def}}{=} \Sigma_1 \cup \Sigma_2$, and $\Delta \stackrel{\mathbf{def}}{=} \{\langle\langle \ell_1, \ell_2\rangle, \sigma_1, \langle \ell'_1, \ell_2\rangle\rangle | (\langle \ell_1, \sigma_1, \ell'_1\rangle \in \Delta_1) \wedge (\ell_2 \in L_2)\}$ $\cup \{\langle\langle \ell_1, \ell_2\rangle, \sigma_2, \langle \ell_1, \ell'_2\rangle\rangle | (\langle \ell_2, \sigma_2, \ell'_2\rangle \in \Delta_2) \wedge (\ell_1 \in L_1)\}$.*

Note that in the previous definition, Q_1 and Q_2 are not necessarily disjoint; this allows the subsystems to communicate between them via common queues. Composing the various subsystems $\mathcal{T}_i$ ($\forall i \in [1, n]$) two-by-two in any order and again for their results gives the global distributed system $\mathcal{T}$ whose semantics (up to state isomorphism) does not depend on the order of grouping.

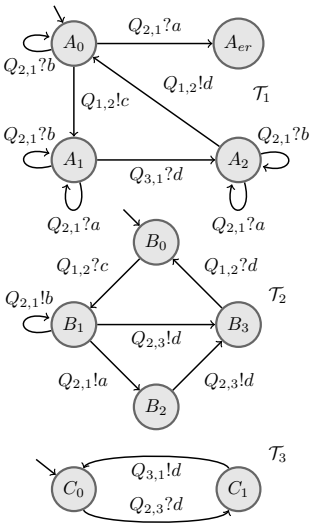
Definition 4 (Distributed system). *A distributed system $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$ is defined by the asynchronous product of n CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, Q_i, M, \Sigma_i, \Delta_i\rangle$ ($\forall i \in [1, n]$) acting in parallel and exchanging information through FIFO channels.*

Note that a distributed system is also modeled by a CFSM, since the asynchronous product of several CFSMs is a CFSM. To avoid the confusion between the model of one process and the model of the whole system, in the sequel, a CFSM $\mathcal{T}_i$ always denotes the model of a single process, and a distributed system $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$ always denotes the model of the global system, as in Definition 4. Below, unless stated explicitly, $\mathcal{T} = \mathcal{T}_1|| \ldots ||\mathcal{T}_n$ is the considered distributed system.

Communication Architecture of the System. We consider an architecture for the system $\mathcal{T} = \mathcal{T}_1|| \ldots ||\mathcal{T}_n$ defined in Definition 4 with *point-to-point* communication i.e., any subsystem $\mathcal{T}_i$ can send messages to any other subsystem $\mathcal{T}_j$ through a queue[4] $Q_{i,j}$. Thus, only $\mathcal{T}_i$ can write a message m on $Q_{i,j}$ (this is denoted by $Q_{i,j}!m$) and only $\mathcal{T}_j$ can read a message m on this queue (this is denoted by $Q_{i,j}?m$). Moreover, we suppose that the queues are unbounded, that the message transfers between the subsystems are reliable and may suffer from arbitrary non-zero delays, and that no *global clock* or *perfectly synchronized local clocks* are available. With this architecture, the set Q_i of $\mathcal{T}_i$ ($\forall i \in [1, n]$) can be

[4] To simplify the presentation of our method, we suppose there is one queue from $\mathcal{T}_i$ to $\mathcal{T}_j$. But, our implementation is more permissive: there can be zero, one or more queues from $\mathcal{T}_i$ to $\mathcal{T}_j$.

rewritten as $Q_i = \{Q_{i,j}, Q_{j,i} \,|\, (1 \leq j \leq n) \wedge (j \neq i)\}$ and $\forall j \neq i \in [1,n]$, $\Sigma_i \cap \Sigma_j = \emptyset$. Let $\delta_i = \langle \ell_i, \sigma_i, \ell'_i \rangle \in \Delta_i$ be a transition of $\mathcal{T}_i$, $\mathsf{global}(\delta_i) \stackrel{\text{def}}{=} \{\langle \langle \ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{i+1}, \ldots, \ell_n \rangle, \sigma_i, \langle \ell_1, \ldots, \ell_{i-1}, \ell'_i, \ell_{i+1}, \ldots, \ell_n \rangle \rangle \in \Delta \,|\, \forall j \neq i \in [1,n] : \ell_j \in L_j\}$ is the set of transitions of Δ that can be built from δ_i in $\mathcal{T}$. We extend this definition to sets of transitions $D \subseteq \Delta_i$ of the subsystem $\mathcal{T}_i$: $\mathsf{global}(D) \stackrel{\text{def}}{=} \bigcup_{\delta_i \in D} \mathsf{global}(\delta_i)$. We abuse notation and write $\Delta \setminus \Delta_i$ instead of $\Delta \setminus \mathsf{global}(\Delta_i)$ to denote the set of transitions of Δ that are not built from Δ_i. Given the set Σ_i of $\mathcal{T}_i$ ($\forall i \in [1,n]$) and the set Σ of $\mathcal{T}$, the projection P_i of Σ onto Σ_i is standard: $P_i(\varepsilon) = \varepsilon$ and $\forall w \in \Sigma^*$, $\forall a \in \Sigma$, $P_i(wa) = P_i(w)a$ if $a \in \Sigma_i$, and $P_i(w)$ otherwise. The inverse projection P_i^{-1} is defined, for each $L \subseteq \Sigma_i^*$, by $P_i^{-1}(L) = \{w \in \Sigma^* \mid P_i(w) \in L\}$.



Example 1. Let us illustrate the concepts of distributed system and CFSM with our running example depicted on the right hand side. It models a factory composed of three components $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$. The subsystem $\mathcal{T}_2$ produces two kinds of items, a and b, and sends these items to $\mathcal{T}_1$ to finish the job. At reception, $\mathcal{T}_1$ must immediately terminate the process of each received item. $\mathcal{T}_1$ can receive and process b items at any time, but must be in a *turbo mode* to receive and process a items. The subsystem $\mathcal{T}_1$ can therefore be in *normal* mode modeled by the location A_0 or in *turbo* mode (locations A_1 and A_2). In normal mode, if $\mathcal{T}_1$ receives an item a, an error occurs (transition in location A_{er}). Since $\mathcal{T}_1$ cannot always be in turbo mode, a protocol between $\mathcal{T}_1$ and $\mathcal{T}_2$ is imagined. At the beginning, $\mathcal{T}_1$ informs (*connect* action, modeled by $\stackrel{Q_{1,2}!c}{\rightarrow}$) $\mathcal{T}_2$ that it goes in a turbo mode, then $\mathcal{T}_2$ sends a and b items. At the end of a working session, $\mathcal{T}_2$ informs $\mathcal{T}_1$ (*disconnect* action, modeled by $\stackrel{Q_{2,3}!d}{\rightarrow}$) that it has completed its session, so that $\mathcal{T}_1$ can go back in normal mode. However, this information has to transit through $\mathcal{T}_3$ via queues $Q_{2,3}$ and $Q_{3,1}$, as $\mathcal{T}_3$ must also record this end of session. Since the message d can be transmitted faster than some items a and b, one can easily find a scenario where $\mathcal{T}_1$ decides to go back to A_0 and ends up in the A_{er} location by reading the message a. It is due to the fact that the subsystems cannot observe the content of the queues and thus $\mathcal{T}_1$ does not know whether there is a message a in queue $Q_{2,1}$ when it arrives in A_0. This motivates the interest of computing good state estimates of the current state of the system. Indeed, if each subsystem maintains good estimates of the current state of the system, then $\mathcal{T}_1$ can know whether there is a message a in $Q_{2,1}$, and reach the location A_0 only if it is not the case. $\diamond$

3 State Estimates of Distributed Systems

We introduce here the framework and the problem we are interested in.

Local View of the Global System. A global state of $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$ is given by a tuple of locations (one for each subsystem) and the content of all the FIFO queues. Informally our problem consists in defining one local estimator per subsystem, knowing that each of

them can only observe the occurrences of actions of its own local subsystem, such that these estimators compute *online* (i.e., during the execution of the system) estimates of the global state of $\mathcal{T}$. We assume that each local *estimator* $\mathcal{E}_i$ has a precise observation of subsystem $\mathcal{T}_i$, and that the model of the global system is known by all the estimators (i.e., the structure of each subsystem and the architecture of the queues between them). Each estimator $\mathcal{E}_i$ must determine online the *smallest possible set* of global states E_i that contains the actual current global state. Note that if $\mathcal{E}_i$ observes that the location of $\mathcal{T}_i$ is ℓ_i, a very rough state estimate is $L_1 \times \ldots \times \{\ell_i\} \times \cdots \times L_n \times (M^*)^{|Q|}$. In other words all the global states of the system such that location of $\mathcal{T}_i$ is ℓ_i; however, this rough estimate does not provide a very useful information.

Online State Estimates. The estimators must compute the state estimates online. Since each estimator $\mathcal{E}_i$ is local to its subsystem, we suppose that $\mathcal{E}_i$ synchronously observes the actions fired by its subsystem; hence since each subsystem is deterministic, each time an event occurs in the local subsystem, it can immediately infer the new location of $\mathcal{T}_i$ and use this information to define its new state estimate. In order to have better state estimates, we also assume that the estimators can communicate with each other by adding some information (some timestamps and their state estimates) to the messages exchanged by the subsystems. Notice that, due to the communication delay, the estimators cannot communicate synchronously, and therefore the state estimate attached to a message might be out-of-date. A classical way to reduce this uncertainty is to timestamp the messages, e.g., by means of vector clocks (see section 4.1).

Estimates Based on Reachability Sets. Each local estimator maintains a symbolic representation of all global states of the distributed system that are compatible with its observation and with the information it received previously from the other estimators. In section 4.2, we detail the algorithms which update these symbolic representations whenever an event occurs. But first, let us explain the intuition behind the computation of an estimate. We consider the simplest case: the initial state estimate before the system begins its execution. Each FIFO channel is empty, and each subsystem $\mathcal{T}_i$ is in its initial location $\ell_{i,0}$. So the initial global state is known by every estimator $\mathcal{E}_i$. A subsystem $\mathcal{T}_j$ may however start its execution, while $\mathcal{T}_i$ is still in its initial location, and therefore $\mathcal{E}_i$ must thus take into account all the global states that are reachable by taking the transitions of the other subsystems $\mathcal{T}_j$. The initial estimate E_i is this set of reachable global states. This computation of reachable global states also occurs in the update algorithms which take into account any new local event occurred or message received (see section 4.2). The reachability problem is however undecidable for distributed FIFO systems. In section 5, we explain how we overcome this obstacle by using abstract interpretation techniques.

Properties of the Estimators. Estimators may have two important properties: soundness and completeness. Completeness refers to the fact that the current state of the global system is always included in the state estimates computed by each state estimator. Soundness refers to the fact that all states included in the state estimate of $\mathcal{E}_i$ ($\forall i \in [1, n]$) can be reached by one of the sequences of actions that are compatible with the observation of $\mathcal{T}_i$ performed by $\mathcal{E}_i$.

Definition 5 (Completeness and Soundness). *The estimators* $(\mathcal{E}_i)_{i \leq n}$ *are* (i) complete *if and only if, for any execution* $\boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$ *of* $\mathcal{T}$, $\boldsymbol{x}_m \in \bigcap_{i=1}^{n} E_i$, *and* (ii) sound *if and only if, for any execution* $\boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$ *of* $\mathcal{T}$, $E_i \subseteq \{x' \in$

$X|\exists\overline{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x'\}$ *(*$\forall i \leq n$*) where* σ_{e_k} *(*$\forall k \in [1,m]$*) is the action that labels the transition corresponding to* e_k*.*

4 Algorithm to Compute the State Estimates

We now present our algorithm that computes estimates of the current state of a distributed system. But first, we recall the notion of *vector clocks* [13], a standard concept that we shall use to compute a more precise state estimates.

4.1 Vector Clocks

To allow the estimators to have a better understanding of the concurrent execution of the distributed system, it is important to determine the causal and temporal relationship between the events that occur in its execution. In a distributed system, events emitted by the same process are ordered, while events emitted by different processes are generally not. When the concurrent processes communicate, additional ordering information can however be obtained. In this case, the communication scheme can be used to obtain a partial order on the events of the system. In practice, vectors of logical clocks, called *vector clocks* [13], can be used to time-stamp the events of the distributed system. The order of two events can then be determined by comparing the value of their respective vector clocks. When these vector clocks are incomparable, the exact order in which the events occur cannot be determined. Vector clocks are formally defined as follows:

Definition 6 (Vector Clocks). *Let* $\langle D, \sqsubseteq\rangle$ *be a partially ordered set, a* vector clock mapping *of width* n *is a function* $V : D \to \mathbb{N}^n$ *such that* $\forall d_1, d_2 \in D : (d_1 \sqsubseteq d_2) \Leftrightarrow (V(d_1) \leq V(d_2))$.

In general, for a distributed system composed of n subsystems, the partial order on events is represented by a vector clock mapping of width n. The method for computing this vector clock mapping depends on the communication scheme of the distributed system. For CFSMs, this vector clock mapping can be computed by the Mattern's algorithm [17], which is based on the causal and thus temporal relationship between the sending and reception of any message transferred through any FIFO channel. This information is then used to determine a partial order, called *causality (or happened-before) relation* $\prec_c$, on the events of the distributed system. This relation is actually the smallest transitive relation satisfying the following conditions: (i) if the events $e_i \neq e_j$ occur in the same subsystem $\mathcal{T}_i$ and if e_i comes before e_j in the execution, then $e_i \prec_c e_j$, and (ii) if e_i is an output event occurring in $\mathcal{T}_i$ and if e_j is the corresponding input event occurring in $\mathcal{T}_j$, then $e_i \prec_c e_j$. In Mattern's algorithm [17], each process $\mathcal{T}_i$ ($\forall i \in [1,n]$) has a vector clock $V_i \in \mathbb{N}^n$ of width n and each element $V_i[j]$ ($\forall j \in [1,n]$) is a counter which represents the knowledge of $\mathcal{T}_i$ regarding $\mathcal{T}_j$ and which means that $\mathcal{T}_i$ knows that $\mathcal{T}_j$ has executed at least $V_i[j]$ events. Each time an event occurs in a subsystem $\mathcal{T}_i$, the vector clock V_i is updated to take into account the occurrence of this event (see [17] for details). When $\mathcal{T}_i$ sends a message to some subsystem $\mathcal{T}_j$, this vector clock is piggybacked and allows $\mathcal{T}_j$, after reception, to update its own vector clock. Our state estimate algorithm uses vector clocks and follows Mattern's algorithm, which ensures the correctness of the vector clocks that we use (see section 4.2).

4.2 Computation of State Estimates

Our state estimate algorithm computes, for each estimator $\mathcal{E}_i$ and for each event occurring in the subsystem $\mathcal{T}_i$, a vector clock V_i and a state estimate E_i that contains the current state of $\mathcal{T}$ and any future state that can be reached from this current state by firing actions that do not belong to $\mathcal{T}_i$. This computation obviously depends on the information that $\mathcal{E}_i$ receives. As a reminder, $\mathcal{E}_i$ observes the last action fired by $\mathcal{T}_i$ and can infer the fired transition. $\mathcal{T}_i$ also receives from the other estimators $\mathcal{E}_j$ their state estimate E_j and their vector clock V_j. Our state estimate algorithm proceeds as follows :

- When the subsystem $\mathcal{T}_i$ sends a message m to $\mathcal{T}_j$, $\mathcal{T}_i$ attaches the vector clock V_i and the state estimate E_i of $\mathcal{E}_i$ to this message. Next, $\mathcal{E}_i$ receives the action fired by $\mathcal{T}_i$, and infers the fired transition. It then uses this information to update its state estimate E_i.
- When the subsystem $\mathcal{T}_i$ receives a message m from $\mathcal{T}_j$, $\mathcal{E}_i$ receives the action fired by $\mathcal{T}_i$ and the information sent by $\mathcal{T}_j$ i.e., the state estimate E_j and the vector clock V_j of $\mathcal{E}_j$. It computes its new state estimate from these elements.

In both cases, the computation of the new state estimate E_i depends on the computation of reachable states. In this section, we assume that we have an operator that can compute an *approximation* of the reachable states (which is *undecidable* is the CFSM model). We will explain in section 5 how such an operator can be computed effectively.

State Estimate Algorithm. Our algorithm, called *SE-algorithm*, computes estimates of the current state of a distributed system. It is composed of three sub-algorithms: (i) the initialization algorithm, which is only used when the system starts its execution, computes, for each estimator, its initial state estimate (ii) the outputTransition algorithm computes online the new state estimate of $\mathcal{E}_i$ after an output of $\mathcal{T}_i$, and (iii) the inputTransition algorithm computes online the new state estimate of $\mathcal{E}_i$ after an input of $\mathcal{T}_i$.

INITIALIZATION *Algorithm:* According to the Mattern's algorithm [17], each component of the vector V_i is set to 0. To take into account that, before the execution of the first action of $\mathcal{T}_i$, the other subsystems $\mathcal{T}_j$ ($\forall j \neq i \in [1, n]$) could perform inputs and outputs, the initial state estimate of $\mathcal{E}_i$ is given by $E_i = \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\langle \ell_{0,1}, \ldots, \ell_{0,n}, \epsilon, \ldots, \epsilon \rangle)$.

Algorithm 1: initialization($\mathcal{T}$)

input : $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$.
output: The initial state estimate E_i of the estimator $\mathcal{E}_i$ ($\forall i \in [1, n]$).

```
1 begin
2   for i ← 1 to n do  for j ← 1 to n do  V_i[j] ← 0
3   for i ← 1 to n do  E_i ← Reach^T_{Δ\Δ_i}(⟨ℓ_{0,1}, ..., ℓ_{0,n}, ϵ, ..., ϵ⟩)
4   return (E_1, ..., E_n)
5 end
```

OUTPUT *Algorithm:* Let E_i be the current state estimate of $\mathcal{E}_i$. When $\mathcal{T}_i$ wants to execute a transition $\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$ corresponding to an output on the queue $Q_{i,j}$, the following instructions are computed to update the state estimate E_i:

- according to the Mattern's algorithm [17], $V_i[i]$ is incremented (i.e., $V_i[i] \leftarrow V_i[i] + 1$) to indicate that a new event has occurred in $\mathcal{T}_i$.
- $\mathcal{T}_i$ tags message m with $\langle E_i, V_i, \delta \rangle$ and writes this information on the queue $Q_{i,j}$. The state estimate E_i tagging m contains the set of states in which $\mathcal{T}$ can be *before* the

Algorithm 2: outputTransition$(\mathcal{T}, V_i, E_i, \delta)$

input : $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$, the vector clock V_i of $\mathcal{E}_i$, the current state estimate E_i of $\mathcal{E}_i$, and a transition $\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$.
output: The state estimate E_i after the output transition δ.

begin
  $V_i[i] \leftarrow V_i[i] + 1$
  $\mathcal{T}_i$ tags message m with $\langle E_i, V_i, \delta \rangle$ and it writes this tagged message on $Q_{i,j}$
  $E_i \leftarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$
  return (E_i)
end

execution of δ. The additional information $\langle E_i, V_i, \delta \rangle$ will be used by $\mathcal{T}_j$ to refine its state estimate.

- the state estimate E_i is updated as follows, to contain the current state of $\mathcal{T}$ and any future state that can be reached from this current state by firing actions that do not belong to $\mathcal{T}_i$: $E_i \leftarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$. More precisely, $\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$ gives the set of states in which $\mathcal{T}$ can be after the execution of δ. After the execution of this transition, $\mathcal{T}_j$ $(\forall j \neq i \in [1, n])$ could however read and write on their queues. Therefore, we define the state estimate E_i by $\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$.

Algorithm 3: inputTransition$(\mathcal{T}, V_i, E_i, \delta)$

input : $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$, the vector clock V_i of $\mathcal{E}_i$, the current state estimate E_i of $\mathcal{E}_i$ and a transition $\delta = \langle \ell_1, Q_{j,i}?m, \ell_2 \rangle \in \Delta_i$. Message m is tagged with the triple $\langle E_j, V_j, \delta' \rangle$ where (i) E_j is the state estimate of $\mathcal{E}_j$ before the execution of δ' by $\mathcal{T}_j$, (ii) V_j is the vector clock of $\mathcal{E}_j$ after the execution of δ' by $\mathcal{T}_j$, and (iii) $\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2 \rangle \in \Delta_j$ is the output corresponding to δ.
output: The state estimate E_i after the input transition δ.

begin
  \\ We consider three cases to update E_j
  if $V_j[i] = V_i[i]$ **then** $Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$
  else if $V_j[j] > V_i[j]$ **then** $Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j))))$
  else $Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$
  $E_i \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$ \\ We update E_i
  $E_i \leftarrow E_i \cap Temp_1$ \\ $E_i =$ updates of $E_i \cap$ update of E_j (i.e., $Temp_1$)
  $V_i[i] \leftarrow V_i[i] + 1$
  for $k \leftarrow 1$ ***to*** n **do** $V_i[k] \leftarrow \mathbf{max}(V_i[k], V_j[k])$
  return (E_i)
end

INPUT *Algorithm:* Let E_i be the current state estimate of $\mathcal{E}_i$. When $\mathcal{T}_i$ executes a transition $\delta = \langle \ell_1, Q_{j,i}?m, \ell_2 \rangle \in \Delta_i$, corresponding to an input on the queue $Q_{j,i}$, it also reads the information $\langle E_j, V_j, \delta' \rangle$ (where E_j is the state estimate of $\mathcal{E}_j$ before the execution of δ' by $\mathcal{T}_j$, V_j is the vector clock of $\mathcal{E}_j$ after the execution of δ' by $\mathcal{T}_j$, and

$\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2 \rangle \in \Delta_j$ is the output corresponding to δ) tagging m, and the following operations are performed to update E_i:

- we update the state estimate E_j of $\mathcal{E}_j$ (this update is denoted by $Temp_1$) by using the vector clocks to guess the possible behaviors of $\mathcal{T}$ between the execution of the transition δ' and the execution of δ. We consider three cases :
  - if $V_j[i] = V_i[i] : Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$. In this case, thanks to the vector clocks, we know that $\mathcal{T}_i$ has executed no transition between the execution of δ' by $\mathcal{T}_j$ and the execution of δ by $\mathcal{T}_i$. Thus, only transitions in $\Delta \setminus \Delta_i$ could have occurred during this period. We then update E_j as follows. We compute (i) $\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)$ to take into account the execution of δ' by $\mathcal{T}_j$, (ii) $\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j))$ to take into account the transitions that could occur between the execution of δ' and the execution of δ, and (iii) $\mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$ to take into account the execution of δ.
  - else if $V_j[j] > V_i[j] : Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j))))$. Indeed, in this case, we can prove (see the proof of Theorem 1) that if we reorder the transitions executed between the occurrence of δ' and the occurrence of δ in order to execute the transitions of Δ_i before the ones of Δ_j, we obtain a correct update of E_i. Intuitively, this reordering is possible, because there is no causal relation between the events of $\mathcal{T}_i$ and the events of $\mathcal{T}_j$, that have occurred between δ' and δ. So, in this reordered sequence, we know that, after the execution of δ, only transitions in $\Delta \setminus \Delta_j$ could occur followed by transitions in $\Delta \setminus \Delta_i$.
  - else[5] $Temp_1 \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$. Indeed, in this case, the vector clocks do not allow us to deduce information regarding the behavior of $\mathcal{T}$ between the execution of δ' and the execution of δ. Therefore, to have a correct state estimate, we update E_j by taking into account all the possible behaviors of $\mathcal{T}$ between the execution of δ' and the execution of δ.
- we update the state estimate E_i to take into account the execution of the input transition δ: $E_i \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$.
- we have two different state estimates: $Temp_1$ and E_i. Thus, we intersect them to obtain a better state estimate: $E_i \leftarrow E_i \cap Temp_1$.
- according to the Mattern's algorithm [17], the vector clock V_i is incremented to take into account the execution of δ and subsequently is set to the component-wise maximum of V_i and V_j. This last operation allows us to take into account the fact that any event that precedes the sending of m should also precede the occurrence of δ.

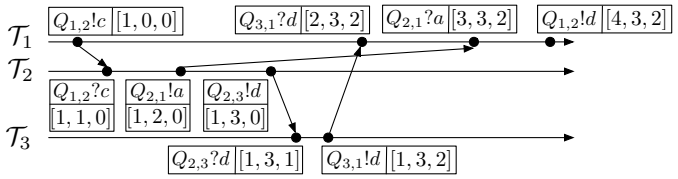


Fig. 1. An execution of the running example.

Example 2. We illustrate SE-algorithm with a sequence of actions of our running example depicted in Figure 1 (the vector clocks are given in the figure). A state of the

[5] It can be shown that the set $Temp_1$ computed in the second case is better that the one computed in the third case.

global system is denoted by $\langle \ell_1, \ell_2, \ell_3, w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1} \rangle$ where ℓ_i is the location of $\mathcal{T}_i$ (for $i = 1, 2, 3$) and $w_{1,2}$, $w_{2,1}$, $w_{2,3}$ and $w_{3,1}$ denote the content of the queues $Q_{1,2}$, $Q_{2,1}$, $Q_{2,3}$ and $Q_{3,1}$.At the beginning of the execution, the state estimates of the three subsystems are (i) $E_1 = \{\langle A_0, B_0, C_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$, (ii) $E_2 = \{\langle A_0, B_0, C_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, C_0, c, \epsilon, \epsilon, \epsilon \rangle\}$, and (iii) $E_3 = \{\langle A_0, B_0, C_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, C_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, C_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, C_0, \epsilon, b^*(a+\epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, C_0, \epsilon, b^*(a+\epsilon), d, \epsilon \rangle\}$. After the first transition $\langle A_0, Q_{1,2}!c, A_1 \rangle$, the state estimate of the estimator $\mathcal{E}_1$ is not really precise, because a lot of events may have happened without the estimator $\mathcal{E}_1$ being informed: $E_1 = \{\langle A_1, B_0, C_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, C_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, C_0, \epsilon, b^*a, \epsilon, \epsilon \rangle, \langle A_1, B_3, C_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle, \langle A_1, B_3, C_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, C_0, \epsilon, b^*(a + \epsilon), \epsilon, d \rangle\}$. However, after the second transition $\langle B_0, Q_{1,2}?c, B_1 \rangle$, the estimator $\mathcal{E}_2$ has an accurate state estimate: $E_2 = \{\langle A_1, B_1, C_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$. We skip a few steps and consider the state estimates before the sixth transition $\langle C_1, Q_{3,1}!d, C_0 \rangle$: E_1 is still the same, because the subsystem $\mathcal{T}_1$ did not perform any action, $E_3 = \{\langle A_1, B_3, C_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle\}$, and we do not indicate E_2, because $\mathcal{T}_2$ is no longer involved. When $\mathcal{T}_3$ sends the message d to $\mathcal{T}_1$ (the transition $\langle C_1, Q_{3,1}!d, C_0 \rangle$), it attaches E_3 to this message. When $\mathcal{T}_1$ reads this message, it computes $E_1 = \{\langle A_2, B_3, C_0, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle\}$ and when it reads the message a, it updates $E_1 : E_1 = \{\langle A_2, B_3, C_0, \epsilon, b^*, \epsilon, \epsilon \rangle\}$. Thus, $\mathcal{E}_1$ knows, after this action, that there is no message a in $Q_{2,1}$, and that after writing d on $Q_{1,2}$, it cannot reach A_{er} from A_0. This example shows the importance of knowing the content of the queues as without this knowledge, $\mathcal{E}_1$ may think that there is an a in the queue, so an error might occur if the transition $\langle A_2, Q_{1,2}!d, A_0 \rangle$ is enabled. $\diamond$

Properties. As explained above, we assume that we can compute an approximation of the reachable states. In this part, we present the properties of our state estimate algorithm w.r.t. the kind of approximations that we use.

Theorem 1. *SE-algorithm is complete, if the* Reach *operator computes an overapproximation of the reachable states.*

The proof is given in Appendix A. If we compute an underapproximation of the reachable states, our state estimate algorithm is not complete.

Theorem 2. *SE-algorithm is sound, if the* Reach *operator computes an underapproximation of the reachable states.*

The proof is given in Appendix A. If we compute an overapproximation of the reachable states, our state estimate algorithm is not sound.

Depending on the used approximations, our algorithm is either complete or sound. Completeness is a more important property, because it ensures that the computed state estimates always contains the current global state. Therefore, in section 5, we define an effective algorithm for the state estimate problem by computing overapproximations of the reachable states. Finally, note that our method proposes that we only add information to existing transmitted messages. We can show that increasing the information exchanged between the estimators (for example, each time an estimator computes a new state estimate, this estimate is sent to all the other estimators) improves their state estimate. This can be done only if the channels and the subsystems can handle this extra load.

5 Effective Computation of State Estimates by Means of Abstract Interpretation

The algorithm described in the previous section requires the computation of reachability operators. But they cannot always be computed for undecidability (or complexity) reasons. In this section, we explain how to overcome this obstacle by using abstract interpretation techniques (see e.g. [7, 14]). In our case, abstract interpretation allows us to compute, in a finite number of steps, an overapproximation of the reachability operators and thus of the state estimates E_i.

Computation of Reachability Sets by the Means of Abstract Interpretation. For a given set of global states $X' \subseteq X$ and a given set of transitions $\Delta' \subseteq \Delta$, the reachability set from X' can be characterized by the least fixpoint: $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(X') = \mu Y.X' \cup \mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y)$. Abstract interpretation provides a theoretical framework to compute efficient overapproximation of such fixpoints. The concrete domain (i.e., the sets of states 2^X), is substituted by a simpler abstract domain Λ, linked by a *G*alois Connection $2^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$ [7], where α (resp. γ) is the abstraction (resp. concretization) function. The fixpoint equation is transposed into the abstract domain. So, the equation to solve has the form: $\lambda = F^{\sharp}_{\Delta'}(\lambda)$, with $\lambda \in \Lambda$ and $F^{\sharp}_{\Delta'} \sqsupseteq \alpha \circ F_{\Delta'} \circ \gamma$. In this setting, a standard way to ensures that the fixpoint computation converges after a finite number of steps to some overapproximation λ_∞, is to use a *widening operator* ∇. The concretization $c_\infty = \gamma(\lambda_\infty)$ is an overapproximation of the least fixpoint of the function $F_{\Delta'}$.

Choice of the Abstract Domain. In abstract interpretation based techniques, the quality of the approximation we obtain depends on the choice of the abstract domain Λ. In our case, the main issue is to abstract the content of the FIFO channels. Since the CFSM model is Turing-powerful, the language that represents all the possible contents of the FIFO channels may be recursively enumerable. As discussed in [14], a good candidate, that abstracts the contents of the queues, to use is the class of regular languages, which can be represented by finite automata. Let us recall the main ideas of this abstraction.

Finite Automata as an Abstract Domain. We first assume that there is only one queue in the distributed system $\mathcal{T}$; we explain later how to handle a distributed system with several queues.

With one queue, the concrete domain of the system $\mathcal{T}$ is defined by $X = 2^{L \times M^*}$. A set of states $Y \in 2^{L \times M^*}$ can be viewed as a map $Y : L \mapsto 2^{M^*}$ that associates a language $Y(\ell)$ with each location $\ell \in L$; $Y(\ell)$ therefore represents the possible contents of the queue in the location ℓ. To simplify the computation, we substitute the concrete domain $\langle L \mapsto 2^{M^*}, \subseteq, \cup, \cap, L \times M^*, \emptyset \rangle$ by the abstract domain $\langle L \mapsto \mathrm{Reg}(M), \subseteq, \cup, \cap, L \times M^*, \emptyset \rangle$, where $\mathrm{Reg}(M)$ is the set of *regular languages* over the alphabet M. This substitution consists in abstracting, for each location, the possible contents of the queue by a regular language. Since regular languages have a canonical representation given by finite automata, each operation (union, intersection, left concatenation,...) in the abstract domain can be performed on finite automata.

Widening Operator. With our abstraction, the widening operator we use to ensure the convergence of the computation, is also performed on a finite automaton, and consists

in quotienting the nodes[6] of the automaton by the *k-bounded bisimulation relation* $\equiv_k$; $k \in \mathbb{N}$ is a parameter which allows us to tune the precision, since increasing k improves the quality of the abstractions in general. Two nodes are equivalent w.r.t. $\equiv_k$ if they have the same outgoing path (sequence of labeled transitions) up to length k. While we merge the equivalent nodes, we keep all transitions and we obtain an automaton recognizing a larger language. Note that for a fixed k, the class of automata which results from such a quotient operation from any original automaton, is finite and its cardinality is bounded by a number which is only function of k. This is the reason why when we apply this widening operator regularly, the fixpoint computation terminates (see [14] for more details).

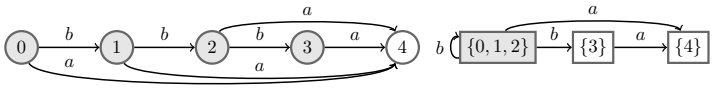


Fig. 2. Illustration of the 1-bounded bisimulation relation $\equiv_1$ for $\mathcal{A}$.

Example 3. We consider the automaton $\mathcal{A}$ depicted in Figure 2, whose recognized language is $a + ba + bba + bbba$. We consider the 1-bounded bisimulation relation i.e., two nodes of the automaton are equivalent if they have the same outgoing transitions. So, nodes $0, 1, 2$ are equivalent, since they all have two transitions labeled by a and b. Nodes 3 and 4 are equivalent to no other node since 4 has no outgoing transition whereas only a is enabled in node 3. When we quotient $\mathcal{A}$ by this equivalent relation, we obtain the automaton on the right side of Figure 2, whose recognized language is b^*a. $\diamond$

When the system contains several queues $Q = \{Q_1, \ldots, Q_r\}$, their content can be represented by a concatenated word $w_1\sharp \ldots \sharp w_r$ with one w_i for each queue Q_i and $\sharp$, a delimiter. With this encoding, we represent a set of queue contents by a finite automaton of a special kind, namely a QDD [2]. Since QDDs are finite automata, classical operations (union, intersection, left concatenation,...) in the abstract domain are performed as was done previously. We must only use a slightly different widening operator not to merge the different queue contents [14].

Effective SE-algorithm. The Reach operator is computed using those abstract interpretation techniques: we proceed to an iterative computation in the abstract domain of regular languages and the widening operator ensures that this computation terminates after a finite number of steps [7]. So the Reach operator always gives an overapproximation of the reachable states regardless the distributed system. The efficiency of these approximations is measured in the experiments of section 6. Because of Theorem 1, our SE-algorithm is complete.

6 Experiments

We have implemented the SE-algorithm as a new feature of the McScM tool [18], a model checker for distributed systems modeled by CFSM. Since it represents queue contents by QDDs, this software provides most of the functionalities needed by our algorithm, like effective computation of reachable states. We have also added a mechanism to manage vector clocks, and an interactive simulator. This simulator first computes and displays the initial state estimates. At each step, it asks the user to choose a possible transition.

[6] The states of an automaton representing the queue contents are called nodes to avoid the confusion with the states of a CFSM.

If the chosen transition is an output, it attaches the current state estimate of the active subsystem and its vector clock to the message sent, and updates the local state estimate. If the transition is an input, the local estimator reads the information attached to the received message and updates its state estimate.

We proceeded to an evaluation of our algorithm measuring the size of the state estimates. Note that this size is not the number of global states of the state estimate (which may be infinite) but the number of nodes of its QDD representation. We generated random sequences of transitions for our running example and some other examples of [12]. Table 1 shows the average execution time for a random sequence of 100 transitions, the memory required (heap size), the average and maximal size of the state estimates. Default value of the widening parameter is $k = 1$. Experiments were done on a standard MacBook Pro with a 2.4 GHz Intel core 2 duo CPU.

example	# subsystems	# channels	time [s]	memory [MB]	maximal size	average size
running example	3	4	7.13	5.09	143	73.0
c/d protocol	2	2	5.32	8.00	183	83.2
non-regular protocol	2	1	0.99	2.19	172	47.4
ABP	2	3	1.19	2.19	49	24.8
sliding window	2	2	3.26	4.12	21	10.1
POP3	2	2	3.08	4.12	22	8.5

Table 1. Experiments

These results show that the computation of state estimates takes about 50ms per transition and that the symbolic representation of state estimates we add to messages are automata with a few dozen nodes. A sensitive element in the method is the size of the computed and transmitted information. It can be improved by the use of compression techniques to reduce the size of this information. A more evolved technique would consist in the offline computation of the set of possible estimates. Estimates are indexed in a table, available at execution time to each local estimator. If we want to keep an online algorithm, we can use the memoization technique. When a state estimate is computed for the first time, it is associated with an index that is transmitted to the subsystem which records both values. If the same estimate must be transmitted, only its index can be transmitted and the receiver can find from its table the corresponding estimate. Those techniques are not yet implemented.

We also highlight that our method works better on the real-life communication protocols we have tested (alternating bit protocol, sliding window, POP3) than on the examples we introduced to test our tool. More details about the tool and experiments are given in Appendix B.

7 Conclusion and Future Work

We have proposed an effective algorithm to compute online, locally to each subsystem, an estimate of the global state of a running distributed system, modeled as *communicating finite state machines* with reliable unbounded FIFO queues. With such a system, a global state is composed of the current location of each subsystem together with the channel contents. The principle is to add a local estimator to each subsystem such that most of the system is preserved; each local estimator is only able to compute information and

in particular symbolic representations of state estimates and to piggyback some of this computed information to the transmitted messages. Since these estimates may be infinite, a crucial point of our work has been to propose and evaluate the use of regular languages to abstract sets of FIFO queues. In practice, we have used k-bisimilarity relations, which allows us to represent each (possibly infinite) set of queue contents by the minimal and canonical k-bisimilar finite automaton which gives an overapproximation of this set. Our algorithm transmits state estimates and vector clocks between subsystems to allow them to refine and preserve consistent state estimates. More elaborate examples must be taken to analyze the precision of our algorithm and see, in practice, if the estimates are sufficient to solve diagnosis or control problems. Anyway, it appears important to study the possibility of reducing the size of the added communication while preserving or even increasing the precision in the transmitted state estimates.

References

1. G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.
2. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS '97: Proceedings of the 4th International Symposium on Static Analysis*, pages 172–186, 1997.
3. Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.
4. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
5. C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
6. T. Chatain, P. Gastin, and N. Sznajder. Natural specifications yield decidability for distributed synthesis of asynchronous systems. In *SOFSEM*, volume 5404 of *LNCS*, pages 141–152, 2009.
7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.
8. R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamical Systems: Theory and Applications*, 10:33–79, 2000.
9. Alain Finkel, S. Purushothaman Iyer, and Grégoire Sutre. Well-abstracted transition systems: application to fifo automata. *Information and Computation*, 181(1):1–31, 2003.
10. B. Genest. On implementation of global concurrent systems with local asynchronous controllers. In *CONCUR*, volume 3653 of *LNCS*, pages 443–457, 2005.
11. L. Hélouet, T. Gazagnaire, and B. Genest. Diagnosis from scenarios. In *proc. of the 8th Int. Workshop on Discrete Events Systems, WODES'06*, pages 307–312, 2006.
12. A. Heußner, T. Le Gall, and G. Sutre. Extrapolation-Based Path Invariants for Abstraction Refinement of Fifo Systems. In *Proc. Model Checking Software, SPIN Workshop 2009*, volume 5578 of *LNCS*, pages 107–124. Springer, 2009.
13. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
14. T. Le Gall, B. Jeannet, and T. Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In *AMAST '06*, volume 4019 of *LNCS*, July 2006.
15. F. Lin, K. Rudie, and S. Lafortune. Minimal communication for essential transitions in a distributed discrete-event system. *IEEE Trans. on Automatic Control*, 52(8):1495–1502, 2007.
16. T. Massart. A calculus to define correct tranformations of lotos specifications. In *FORTE*, volume C-2 of *IFIP Transactions*, pages 281–296, 1991.

17. F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.
18. The McScM library, 2009. http://www.labri.fr/perso/heussner/mcscm/.
19. L. Ricker and B. Caillaud. Mind the gap: Expanding communication options in decentralized discrete-event control. In *46th IEEE Conference on Decision and Control*, New Orleans, LA, USA, 2007.
20. M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, March 1996.
21. S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. *IEEE Trans. on Automatic Control*, 49(9):1489–1501, 2004.
22. S. Xu and R. Kumar. Distributed state estimation in discrete event systems. In *ACC'09: Proc. of the 2009 conference on American Control Conference*, pages 4735–4740, 2009.

A Proofs of Theorems 1 and 2

In this section, we prove Theorems 1 and 2. These proofs use the lemmas given in section A.1 and the following notations.

Let $s = \boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \boldsymbol{x}_m$ be an execution of the global system $\mathcal{T}$. When the subsystem $\mathcal{T}_i$ executes an event e_k (with $k \in [1, m]$) of this sequence, the state estimate, computed by $\mathcal{E}_i$ at this step, is denoted by E_i^t, where t is the number of events executed by $\mathcal{T}_i$ in the subsequence $\boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_k} \boldsymbol{x}_k$. However, to be more concise in our presentation, we use an abuse of notation: when an event e_k is executed in the sequence s, the state estimate of *each* state estimator $\mathcal{E}_i$ is denoted by[7] E_i^k. This state estimate is defined in the following way: if e_k has not been executed by $\mathcal{T}_i$, then $E_i^k \stackrel{\mathbf{def}}{=} E_i^{k-1}$. Otherwise, the value of E_i^k is computed by the state estimator from E_i^{k-1}, the observed event, and the information that possibly tags this event.

The vector clock V_i, computed after the occurrence of an event e in the subsystem $\mathcal{T}_i$, is denoted by $V_i(e)$.

A.1 Lemmas

The following lemma proves the correctness of the vector clock mapping computed by the Mattern's algorithm for the relation $\prec_c$:

Lemma 1 ([17]). *Given n subsystems $\mathcal{T}_i$ ($\forall i \in [1, n]$) and two events $e_1 \neq e_2$ occurring respectively in $\mathcal{T}_i$ and $\mathcal{T}_j$ (i can be equal to j), we have the following equivalence: $e_1 \prec_c e_2$ if and only if $V_i(e_1) \leq V_j(e_2)$.*

Lemma 2. *Given n subsystems $\mathcal{T}_i$ ($\forall i \in [1, n]$) and three events $e_i \neq e_j \neq e_k$ occurring respectively in $\mathcal{T}_i$, $\mathcal{T}_j$ and $\mathcal{T}_k$, if $e_k \not\prec_c e_j$ and $e_i \prec_c e_j$, then $e_k \not\prec_c e_i$.*

Proof. Let us assume that $e_k \prec_c e_i$. Since $e_k \not\prec_c e_j$, there exists $\ell \in [1, n]$ such that $V_k(e_k)[\ell] > V_j(e_j)[\ell]$. Moreover, $V_k(e_k)[\ell] > V_i(e_i)[\ell]$, because $V_i(e_i)[m] \leq V_j(e_j)[m]$ for each $m \in [1, n]$ (due to $e_i \prec_c e_j$). But it is a contradiction with $e_k \prec_c e_i$, because this relation implies that $V_k(e_k)[m] \leq V_i(e_i)[m]$ for each $m \in [1, n]$. □

[7] In this way, we do not need to introduce, for each subsystem, a parameter giving the number of events that has been executed so far by each subsystem

Lemma 3. *Given a distributed system $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n$, and a sequence $se_1 = \boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{i-1}} \boldsymbol{x}_{i-1} \xrightarrow{e_i} \boldsymbol{x}_i \xrightarrow{e_{i+1}} \boldsymbol{x}_{i+1} \xrightarrow{e_{i+2}} \dots \xrightarrow{e_m} \boldsymbol{x}_m$ executed by $\mathcal{T}$, if $e_i \not\prec_c e_{i+1}$, then the sequence $se_2 = \boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{i-1}} \boldsymbol{x}_{i-1} \xrightarrow{e_{i+1}} \boldsymbol{x}'_i \xrightarrow{e_i} \boldsymbol{x}_{i+1} \xrightarrow{e_{i+2}} \dots \xrightarrow{e_m} \boldsymbol{x}_m$ can also occur in the system $\mathcal{T}$.*

Proof. We suppose that $\delta_{e_i} = \langle \ell_{e_i}, \sigma_{e_i}, \ell'_{e_i} \rangle \in \Delta_i$ and $\delta_{e_{i+1}} = \langle \ell_{e_j}, \sigma_{e_j}, \ell'_{e_j} \rangle \in \Delta_j$. Note that $i \neq j$; otherwise, we would have $e_i \prec_c e_{i+1}$ (by definition of $\prec_c$). We can prove this property by showing that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}))$. For that, we consider two cases:

1) δ_{e_i} and $\delta_{e_{i+1}}$ act on different queues: we suppose that δ_{e_i} and $\delta_{e_{i+1}}$ respectively act on the queues Q_{k_i} and Q_{k_j}. We also suppose that $\boldsymbol{x}_{i-1} = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, w_{k_i}, \dots, w_{k_j}, \dots, w_{|Q|} \rangle$ (where w_{k_i} and w_{k_j} respectively denote the content of the queues Q_{k_i} and Q_{k_j}), and that the action σ_{e_i} (resp. σ_{e_j}), which acts on the content w_{k_i} (resp. w_{k_j}), modifies it to give w'_{k_i} (resp. w'_{k_j}). In consequence, $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1}) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w_{k_j}, \dots, w_{|Q|} \rangle$ and $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w'_{k_j}, \dots, w_{|Q|} \rangle$. Since $e_i \not\prec_c e_{i+1}$, we have that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}) = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w_{k_i}, \dots, w'_{k_j}, \dots, w_{|Q|} \rangle$ and $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w'_{k_j}, \dots, w_{|Q|} \rangle$, which implies that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}))$.

2) δ_{e_i} and $\delta_{e_{i+1}}$ act on the same queue Q_k: we consider two cases:

a) $\sigma_i = Q_k!m_i$ is an output and $\sigma_{i+1} = Q_k?m_j$ is an input: the message written by δ_{e_i} cannot be read by the transition $\delta_{e_{i+1}}$, because, in this case, we would have $e_i \prec_c e_{i+1}$. Thus, $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_i, \dots, w_{|Q|} \rangle$ where $w.m_i$ is the content of the queue Q_k. Therefore, the state $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1}) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, m_j.w.m_i, \dots, w_{|Q|} \rangle$ and the state $\boldsymbol{x}_{i-1} = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, m_j.w, \dots, w_{|Q|} \rangle$. Next, we compute the state $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}) = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w, \dots, w_{|Q|} \rangle$ and the state $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_i, \dots, w_{|Q|} \rangle$. In consequence, $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}))$.

b) $\sigma_i = Q_k?m_i$ is an input and $\sigma_{i+1} = Q_k!m_j$ is an output: the state $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_j, \dots, w_{|Q|} \rangle$ where $w.m_j$ is the content of the queue Q_k. Next, similarly to the previous case, we can prove that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\boldsymbol{x}_{i-1})) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{i+1}}}(\boldsymbol{x}_{i-1}))$.

The cases, where δ_{e_i} and $\delta_{e_{i+1}}$ are both an input or an output, are not possible, because these transitions would then be executed by the same process and hence we would have $e_i \prec_c e_{i+1}$. □

This property means that if two consecutive events e_i and e_{i+1} are such that $e_i \not\prec_c e_{i+1}$, then these events can be swapped without modifying the reachability of $\boldsymbol{x}_m$. We finally prove the following lemma.

Lemma 4. *Given a distributed system* $\mathcal{T} = \mathcal{T}_1||\dots||\mathcal{T}_n$ *, a transition* $\delta_i = \langle \ell_i, Q_{t,i}?m_i, \ell_i' \rangle \in \Delta_i$ *(with* $t \neq i$*), and a set of states* $B \subseteq X$*, then* $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B))) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B))$.

Proof. First, the inequality $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B)) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B)))$ holds trivially.

To prove the other inclusion, we have to show that if a state $\boldsymbol{x}_m \in \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B)))$, then $\boldsymbol{x}_m \in \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B))$. We actually prove a more general result. We show that each sequence $\boldsymbol{x}_1 \xrightarrow{e_2} \boldsymbol{x}_2 \xrightarrow{e_3} \dots \xrightarrow{e_{k-1}} \boldsymbol{x}_{k-1} \xrightarrow{e_k} \boldsymbol{x}_k \xrightarrow{e_{k+1}} \boldsymbol{x}_{k+1} \xrightarrow{e_{k+2}} \dots \xrightarrow{e_m} \boldsymbol{x}_m$ (where (i) $\boldsymbol{x}_1 \in B$, (ii) the event e_k corresponds to the transition $\delta_{e_k} = \delta_i \in \Delta_i$, and (iii) the event e_b, for each $b \neq k \in [2, m]$, corresponds to a transition $\delta_{e_b} \in \Delta \setminus \Delta_i$) can be reordered to execute e_k at the end of the sequence without modifying the reachability of $\boldsymbol{x}_m$ i.e., the following sequence can occur: $\boldsymbol{x}_1 \xrightarrow{e_2} \boldsymbol{x}_2 \xrightarrow{e_3} \dots \xrightarrow{e_{k-1}} \boldsymbol{x}_{k-1} \xrightarrow{e_{k+1}} \boldsymbol{x}'_{k+1} \xrightarrow{e_{k+2}} \dots \xrightarrow{e_m} \boldsymbol{x}'_m \xrightarrow{e_k} \boldsymbol{x}_m$. This reordered sequence can be obtained thanks to Lemma 3, but to use this lemma, we must prove that $e_k \not\prec_c e_b$ $(\forall b \in [k+1, m])$. The proof is by induction on the length of the sequence of events that begins from $\boldsymbol{x}_k$:

- **Base case**: we must prove that $e_k \not\prec_c e_{k+1}$. By definition of $\prec_c$, since e_k and e_{k+1} occur in different subsystems and are consecutive events, there is one possibility to have $e_k \prec_c e_{k+1}$: it is when e_k is an output and e_{k+1} is the corresponding input. But e_k is an input and hence $e_k \not\prec_c e_{k+1}$.

- **Induction step**: we suppose that $e_k \not\prec_c e_{k+r}$ $(\forall r \in [1, j])$ and we prove that $e_k \not\prec_c e_{k+j+1}$. By definition of $\prec_c$, since e_k and e_{k+1} occur in different subsystems, there are two possibilities to have $e_k \prec_c e_{k+j+1}$:

  1) e_k is an output and e_{k+j+1} is the corresponding input. However, e_k is an input and thus this case is impossible.

  2) $e_k \prec_c e_{k+r}$ (with $r \in [1, j]$) and $e_{k+r} \prec_c e_{k+j+1}$. But by induction hypothesis, $e_k \not\prec_c e_{k+r}$ $(\forall r \in [1, j])$ and thus this case is impossible.

  Therefore, $e_k \not\prec_c e_{k+j+1}$. □

A.2 Proof of Theorem 1

To show that this theorem holds, we prove by induction on the length m of an execution $\boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \boldsymbol{x}_m$ of the system $\mathcal{T}$ that $\forall i \in [1, n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m) \subseteq E_i^m$. Since $\boldsymbol{x}_m \in \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)$, we have then that $\boldsymbol{x}_m \in E_i^m$. In this proof, when $e_i \prec_c e_j$, we say that e_j *causally depends* on e_i (or e_i *happened-before* e_j).

- **Base case ($m = 0$):** For each $i \in [1, n]$, the set $E_i^0 = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle)$ (see Algorithm 1). Therefore, we have that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_0) = E_i^0$ $(\forall i \in [1, n])$, because $\boldsymbol{x}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$.
- **Induction step:** We suppose that the property holds for the executions of length $k \leq m$ (i.e., $\forall\, 0 \leq k \leq m, \forall i \in [1, n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_k) \subseteq E_i^k$) and we prove that the property also holds for the executions of length $m + 1$. For that, we suppose that the event e_{m+1} has been executed by $\mathcal{T}_i$. We must consider two cases:

1) $\delta_{e_{m+1}}$ is an output on the queue $Q_{i,k}$ (with $k \neq i \in [1,n]$): We must prove that $\forall j \in [1,n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^{m+1}$ and we consider again two cases to prove this property:
   a) $j \neq i$: By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m) \subseteq E_j^m$. Moreover, we have that:

   $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m)$, by definition of Reach
   $\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m))$, as Post is monotonic
   $\Rightarrow \boldsymbol{x}_{m+1} \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m)$, because $\delta_{e_{m+1}} \in \Delta \setminus \Delta_j$ (as $\delta_{e_{m+1}} \in \Delta_i$) and $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) = \boldsymbol{x}_{m+1}$
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m))$, as Reach is monotonic
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_m)$
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^m$
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^{m+1}$, because $E_j^m = E_j^{m+1}$ (due to the fact that e_{m+1} has not been executed by $\mathcal{T}_j$)

   b) $j = i$: By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m) \subseteq E_i^m$. The set $E_i^{m+1} = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m))$ (see Algorithm 2). Moreover, we have that:

   $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)$
   $\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m))$
   $\Rightarrow \boldsymbol{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m))$, as $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) = \boldsymbol{x}_{m+1}$
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)))$
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m))$, by induction hypothesis
   $\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq E_i^{m+1}$, by definition of E_i^{m+1}

   Thus, for each $j \in [1,n]$, we have that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^{m+1}$. Moreover, since we compute an overapproximation of E_j^{m+1} ($\forall j \in [1,n]$), this inclusion remains true[8].

2) $\delta_{e_{m+1}}$ is an input on the queue $Q_{k,i}$ (with $k \neq i \in [1,n]$): We must prove that $\forall j \in [1,n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^{m+1}$ and we consider again two cases:
   a) $j \neq i$: The proof is similar to the one given in the case where $\delta_{e_{m+1}}$ in an output.
   b) $j = i$: By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m) \subseteq E_i^m$. By Algorithm 3, the set $E_i^{m+1} = Temp_1 \cap \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m)$ (in our algorithm, the set $Temp_1$ can have three possible values). To prove that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq E_i^{m+1}$, we first prove that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m)$ and next we show that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq Temp_1$. The first inclusion is proved as follows:

[8] Note that if we compute an underapproximation of E_j^{m+1}, the inclusion does not always hold.

$$
\begin{aligned}
&\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)\\
\Rightarrow\ &\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m))\\
\Rightarrow\ &\boldsymbol{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)), \text{ because } \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) = \boldsymbol{x}_{m+1}\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)))\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_m)), \text{ by Lemma 4}\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m), \text{ by induction hypothesis}
\end{aligned}
$$

To prove the second inclusion, we must consider three cases which depend on the definition of $Temp_1$. Let e_t (with $t \leq m$) be the output (executed by $\mathcal{T}_k$ with $k \neq i \in [1, n]$) corresponding to the input e_{m+1}:

A) $Temp_1 = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$ and $V_k[i] = V_i[i]$ (as a reminder, V_k represents the vector clock of $\mathcal{T}_k$ after the occurrence of the event e_t and V_i represents the vector clock of $\mathcal{T}_i$ before the occurrence of the event e_{m+1}): By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1}) \subseteq E_k^{t-1}$. Moreover, we have that:

$$
\begin{aligned}
&\boldsymbol{x}_{t-1} \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1})\\
\Rightarrow\ &\boldsymbol{x}_{t-1} \subseteq E_k^{t-1}, \text{ by induction hypothesis}\\
\Rightarrow\ &\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})\\
\Rightarrow\ &\boldsymbol{x}_t \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}), \text{ as } \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) = \boldsymbol{x}_t\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_t) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})) \qquad (\beta)
\end{aligned}
$$

However, since $V_k[i] = V_i[i]$, we know that, between the moment where e_t has been executed and the moment where e_m has been executed, the vector clock $V_i[i]$ has not been modified. Thus, during this period no transition of $\mathcal{T}_i$ has been executed. In consequence, we have that $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_t)$ and hence $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))$ by (β). From this inclusion, we deduce that:

$$
\begin{aligned}
&\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))\\
\Rightarrow\ &\boldsymbol{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))),\\
&\text{because } \boldsymbol{x}_{m+1} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m)\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))))\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))),\\
&\text{by Lemma 4}\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq Temp_1, \text{ by definition of } Temp_1
\end{aligned}
$$

B) $Temp_1 = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))))$ and $V_k[k] > V_i[k]$ (as a reminder, V_k represents the vector clock of $\mathcal{T}_k$ after the occurrence of the event e_t and V_i represents the vector clock of $\mathcal{T}_i$ before the occurrence of the event e_{m+1}): By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1}) \subseteq E_k^{t-1}$. Moreover, we have that:

$$\begin{aligned}
&\boldsymbol{x}_{t-1} \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1}) \Rightarrow \boldsymbol{x}_{t-1} \subseteq E_k^{t-1}, \text{ by induction hypothesis}\\
&\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})\\
&\Rightarrow \boldsymbol{x}_t \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}), \text{ because } \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) = \boldsymbol{x}_t \qquad (\gamma)
\end{aligned}$$

This inclusion is used further in the proof. Now, we prove that $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))$. For that, let us consider the subsequence $se = \boldsymbol{x}_t \xrightarrow{e_{t+1}} \boldsymbol{x}_{t+1} \xrightarrow{e_{t+2}} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$ of the execution $\boldsymbol{x}_0 \xrightarrow{e_1} \boldsymbol{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$. Let e_{K_1} be the first event of the sequence se executed[9] by $\mathcal{T}_k$ and $s_I = e_{I_1}, \ldots, e_{I_\ell}$ (with $I_1 < \ldots < I_\ell$) be the events of the sequence se executed[10] by $\mathcal{T}_i$. If $I_\ell < K_1$ (i.e., e_{I_ℓ} has been executed before e_{K_1}), then $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))$, because all the events of the sequence se executed by $\mathcal{T}_i$ have been executed before the first event e_{K_1} of $\mathcal{T}_k$. Otherwise, let $s_{I'} = e_{I_d}, \ldots, e_{I_\ell}$ be the events of s_I executed after e_{K_1}. We must reorder the sequence se to obtain a new sequence where all the actions of $\mathcal{T}_i$ are executed before the ones of $\mathcal{T}_k$ and $\boldsymbol{x}_m$ remains reachable. Lemma 3 allows us to swap two consecutive events without modifying the reachability when these events are not causally dependent. To use this lemma, we must prove that the events $e_{I_d}, \ldots, e_{I_\ell}$ do not causally depend on e_{K_1}. For that, we first prove that $e_{K_1} \not\prec_c e_{I_\ell}$. By assumption, we know that $V_k[k] > V_i[k]$. V_k represents the vector clock of $\mathcal{T}_k$ after the execution of e_t and V_i represents the vector clock of $\mathcal{T}_i$ before the execution of e_{m+1}, which gives $V_k(e_t)[k] > V_i(e_m)[k]$. Moreover, $V_i(e_m)[k] \geq V_i(e_{I_\ell})[k]$ (because e_{I_ℓ} has been executed before[11] e_m) and $V_k(e_{K_1})[k] \geq V_k(e_t)[k] + 1$ (because e_{K_1} is the event which follows e_t in the execution of the subsystem $\mathcal{T}_k$). Thus, $V_k(e_{K_1})[k] > V_i(e_{I_\ell})[k]$, and hence $e_{K_1} \not\prec_c e_{I_\ell}$. Next, since $e_{I_c} \prec_c e_{I_\ell}$ ($\forall e_{I_c} \neq e_{I_\ell} \in s_{I'}$) and since $e_{K_1} \not\prec_c e_{I_\ell}$, we have by Lemma 2 that $e_{K_1} \not\prec_c e_{I_c}$. Now, in the sequence se, we will move the events $e_{I_d}, \ldots, e_{I_\ell}$ to execute them before e_{K_1} without modifying the reachability of $\boldsymbol{x}_m$. We start by moving the element e_{I_d}. To obtain a sequence where e_{I_d} precedes e_{K_1}, we swap e_{I_d} with the events which precede it and we repeat this operation until the event e_{K_1}. Lemma 3 ensures that $\boldsymbol{x}_m$ remains reachable if e_{I_d} is swapped with an element e' such that $e' \not\prec_c e_{I_d}$. However, between e_{K_1} and e_{I_d} there can be some events, that *happened-before* e_{I_d}. We must thus move these events before moving e_{I_d}. More precisely, let $s_b = e_{b_1}, \ldots, e_{b_p}$

[9] If this element does not exist, then the transitions executed in this sequence do not belong to Δ_k; thus, $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t)$ and hence $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))$

[10] If the sequence s_I is empty, then the transitions executed in the sequence se do not belong to Δ_i; thus, $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_t)$ and hence $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))$, because $\boldsymbol{x}_t \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t)$

[11] Note that e_{I_ℓ} may be equal to e_m.

(with $b_1 < \ldots < b_p$) be the greatest sequence of events such that (i) these events are executed between the occurrence of e_{K_1} and the occurrence of e_{I_d} and (ii) $\forall e_{b_c} \in s_b : e_{b_c} \prec_c e_{I_d}$ (note that the events of the sequence s_b are not executed by $\mathcal{T}_k$; otherwise, we would have $e_{K_1} \prec_c e_{I_d}$). The sequence of events $s = e_{K_1}, e_{K_1+1}, e_{K_1+2}, \ldots, e_{b_1-1}$ executed between e_{K_1} and e_{b_1} is such that $\forall e_{t'} \in s : e_{t'} \not\prec_c e_{b_1}$. Indeed, if $e_{t'} \prec_c e_{b_1}$, then by transitivity we would have $e_{t'} \prec_c e_{I_d}$, but this is not possible, because $e_{t'} \notin s$. Thus, by Lemma 3, in the sequence $\boldsymbol{x}_t \xrightarrow{e_{t+1}} \ldots \xrightarrow{e_{K_1}} \boldsymbol{x}_{K_1} \xrightarrow{e_{K_1+1}} \boldsymbol{x}_{K_1+1} \xrightarrow{e_{K_1+2}} \ldots \xrightarrow{e_{b_1-1}} \boldsymbol{x}_{b_1-1} \xrightarrow{e_{b_1}} \boldsymbol{x}_{b_1} \xrightarrow{e_{b_1+1}} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$, we can *safely* swap the events e_{b_1-1} and e_{b_1}. We then obtain a reordered sequence where $\boldsymbol{x}_m$ remains reachable i.e., we obtain $\boldsymbol{x}_t \xrightarrow{e_{t+1}} \ldots \xrightarrow{e_{K_1}} \boldsymbol{x}_{K_1} \xrightarrow{e_{K_1+1}} \boldsymbol{x}_{K_1+1} \xrightarrow{e_{K_1+2}} \ldots \xrightarrow{e_{b_1-2}} \boldsymbol{x}_{b_1-2} \xrightarrow{e_{b_1}} \boldsymbol{x}'_{b_1} \xrightarrow{e_{b_1-1}} \boldsymbol{x}_{b_1} \xrightarrow{e_{b_1+1}} \ldots \xrightarrow{e_m} \boldsymbol{x}_m$. By repeating this swap with the events $e_{b_1-2}, e_{b_1-3}, \ldots, e_{K_1+1}, e_{K_1}$, we obtain a reordered sequence where (i) e_{b_1} is executed before e_{K_1} and (ii) $\boldsymbol{x}_m$ remains reachable (by Lemma 3). We repeat the operations performed for e_{b_1} with the events $e_{b_2}, \ldots, e_{b_p}$ and e_{I_d} to obtain a reordered sequence where (i) e_{I_d} is executed before e_{K_1} and (ii) $\boldsymbol{x}_m$ is reachable. Finally, we repeat the operations performed for e_{I_d} with the other elements of the sequence $s_{I'}$ to obtain a reordered sequence where (i) $\boldsymbol{x}_m$ is reachable from $\boldsymbol{x}_t$ and (ii) the events of $\mathcal{T}_i$ are executed before the ones of $\mathcal{T}_k$, which implies that $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))$. Next, from this inclusion, we deduce that:

$\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t)))$

$\Rightarrow \boldsymbol{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t)))$,

because $\boldsymbol{x}_{m+1} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m)$

$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t))))$

$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_t)))$,

by Lemma 4

$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))))$,

by (γ)

$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq Temp_1$, by definition of $Temp_1$

C) $Temp_1 = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$: By induction hypothesis, we know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1}) \subseteq E_k^{t-1}$. Moreover, we have that:

$\boldsymbol{x}_{t-1} \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\boldsymbol{x}_{t-1}) \Rightarrow \boldsymbol{x}_{t-1} \subseteq E_k^{t-1}$, by induction hypothesis

$\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})$

$\Rightarrow \boldsymbol{x}_t \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})$, as $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(\boldsymbol{x}_{t-1}) = \boldsymbol{x}_t$

$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\boldsymbol{x}_t) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))$ $\qquad (\alpha)$

However, the events $e_{t+1}, \ldots, e_m$ leading to $\boldsymbol{x}_m$ from the state $\boldsymbol{x}_t$ correspond to transitions which belong to Δ. Thus, $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\boldsymbol{x}_t)$ and hence $\boldsymbol{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))$ by (α). From this inclusion, we deduce that:

$$
\begin{aligned}
&\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))\\
\Rightarrow\ &\boldsymbol{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))), \text{ as } \boldsymbol{x}_{m+1} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}_m)\\
\Rightarrow\ &\boldsymbol{x}_{m+1} \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})), \text{ because } \delta_{e_{m+1}} \in \Delta\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})), \text{ because } \Delta \setminus \Delta_i \subseteq \Delta\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))),\\
&\text{because } \delta_{e_{m+1}} \in \Delta\\
\Rightarrow\ &\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq Temp_1, \text{ by definition of } Temp_1
\end{aligned}
$$

In conclusion, we have proven, for each definition of $Temp_1$, that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq Temp_1$ and hence $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_{m+1}) \subseteq E_i^{m+1}$.

Thus, for each $j \in [1, n]$, we have that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\boldsymbol{x}_{m+1}) \subseteq E_j^{m+1}$. Moreover, since we compute an overapproximation of E_j^{m+1} $(\forall j \in [1, n])$, this inclusion remains true.

A.3 Proof of Theorem 2

To show that this theorem holds, we prove by induction on the length m of the sequences of events $e_1, \ldots, e_m$ (let $\delta_{e_k} = \langle \ell_{e_k}, \sigma_{e_k}, \ell'_{e_k} \rangle$ be the transition corresponding to e_k, for each $k \in [1, m]$) executed by the system that $\forall i \in [1, n] : E_i^m \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1}.\sigma_{e_2} \ldots \sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$:

- **Base case ($\boldsymbol{m = 0}$):** The initial state $\boldsymbol{x}_0 = \langle \ell_{0,1}, \ldots, \ell_{0,n}, \epsilon, \ldots, \epsilon \rangle$ and we must prove that $\forall i \in [1, n] : E_i^0 \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\epsilon)) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. The set $E_i^0 = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_0)$ (see Algorithm 1) and $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\boldsymbol{x}_0) = \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\epsilon)) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$, which implies that $E_i^0 = \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\epsilon)) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Moreover, since we compute an underapproximation of E_i^0 $(\forall j \in [1, n])$, this inclusion remains true[12].
- **Induction step:** We suppose that the property holds for the sequences of events of length $k \leq m$ and we prove that the property remains true for the sequences of length $m + 1$. We suppose that e_{m+1} has been executed by $\mathcal{T}_i$. We consider two cases:
  1) $\delta_{e_{m+1}}$ is an output: We must prove that $\forall j \in [1, n] : E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2} \ldots \sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$ and we consider again two cases:
     a) $i \neq j$: By induction hypothesis, we know that $E_j^m \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2} \ldots \sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Since $E_j^{m+1} = E_j^m$ (by definition), we have that $E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2} \ldots \sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$.

[12] Note that if we compute an overapproximation of the reachable states, the inclusion does not always hold.

Moreover, $\{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\} = \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$, as $P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m}) = P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})$ (because $\sigma_{e_{m+1}} \notin \Sigma_j$). Therefore, we have that $E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Moreover, since we compute an underapproximation of E_j^{m+1}, this inclusion remains true.

b) $i = j$: the set $E_j^{m+1} = \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m))$ and $P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}}) = P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m}).\sigma_{e_{m+1}}$, because $\sigma_{e_{m+1}} \in \Sigma_j$. We prove that if $\boldsymbol{x} \in E_j^{m+1}$, then $\boldsymbol{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. If $\boldsymbol{x} \in E_j^{m+1}$, then there exists a state $\boldsymbol{x}' \in E_j^m$ such that $\boldsymbol{x} \in \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}'))$. Let $\langle \ell_{e_{m+1}}, \sigma_{e_{m+1}}, \ell'_{e_{m+1}} \rangle$, $\langle \ell_{t_1}, \sigma_{t_1}, \ell'_{t_1} \rangle, \dots, \langle \ell_{t_k}, \sigma_{t_k}, \ell'_{t_k} \rangle$ be the sequence of transitions which leads to $\boldsymbol{x}$ from $\boldsymbol{x}'$ i.e., $\boldsymbol{x}' \xrightarrow{\sigma_{e_{m+1}}.\sigma_{t_1}\dots\sigma_{t_k}} \boldsymbol{x}$. The transition $\langle \ell_{t_b}, \sigma_{t_b}, \ell'_{t_b} \rangle \in \Delta \setminus \Delta_j$ (for each $b \in [1, k]$), which implies that $\sigma_{e_{m+1}}.\sigma_{t_1}\dots\sigma_{t_k} \in P_j^{-1}(\sigma_{e_{m+1}})$. Moreover, by induction hypothesis, the state $\boldsymbol{x}' \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$, which implies that $\exists \overline{\sigma'} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma'}} \boldsymbol{x}'$. Since $P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) = [P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})).P_j^{-1}(\sigma_{e_{m+1}})]$, the sequence $\overline{\sigma''} = \overline{\sigma'}.\sigma_{e_{m+1}}.\sigma_{t_1}\dots\sigma_{t_k}$ belongs to $P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}}))$. Moreover, $\boldsymbol{x}_0 \xrightarrow{\overline{\sigma''}} \boldsymbol{x}$ (because $\boldsymbol{x}_0 \xrightarrow{\overline{\sigma'}} \boldsymbol{x}'$ and $\boldsymbol{x}' \xrightarrow{\sigma_{e_{m+1}}.\sigma_{t_1}\dots\sigma_{t_k}} \boldsymbol{x}$) which implies that $\boldsymbol{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Hence, $E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Again, since we compute an underapproximation of E_j^{m+1}, this inclusion remains true.

2) $\delta_{e_{m+1}}$ is an input: We must prove that $\forall i \in [1, n] : E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$ and we consider again two cases:

a) $i \neq j$: the proof is similar the one given in the case where $\delta_{e_{m+1}}$ is an output.

b) $i = j$: The set $E_j^{m+1} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m) \cap Temp_1$ (see Algorithm 3). Thus, we have that $E_j^{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m)$ and it then suffices to prove that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m) \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. For that, we show that if $\boldsymbol{x} \in \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m)$, then $\boldsymbol{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. If $\boldsymbol{x} \in \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m)$, then there exists a state $\boldsymbol{x}' \in E_j^m$ such that $\boldsymbol{x} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}')$. By induction hypothesis, the state $\boldsymbol{x}' \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$, which implies that $\exists \overline{\sigma'} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma'}} \boldsymbol{x}'$. Since $P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}})) = [P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_m})).P_j^{-1}(\sigma_{e_{m+1}})]$, the sequence $\overline{\sigma''} = \overline{\sigma'}.\sigma_{e_{m+1}}$ belongs to $P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\dots\sigma_{e_{m+1}}))$. Moreover, $\boldsymbol{x}_0 \xrightarrow{\overline{\sigma''}}$

$\boldsymbol{x}$ (because $\boldsymbol{x}_0 \xrightarrow{\overline{\sigma'}} \boldsymbol{x}'$ and $\boldsymbol{x} = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\boldsymbol{x}')$) which implies that $\boldsymbol{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2} \dots \sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Therefore, we have that $E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2} \dots \sigma_{e_{m+1}})) : \boldsymbol{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Again, since we compute an underapproximation of E_j^{m+1}, this inclusion remains true. □

B Experiments

This appendix presents some additional details about the current implementation of our algorithm. The source code is available: [18] version "0.02 Control". Note that this tool is still under development.

B.1 McScM tool - control version

McScM [18] is a model checker for distributed systems modeled by CFSMs. It has been developed by Alexander Heussner, Gregoire Sutre and Tristan Le Gall, and is distributed under the LGPL license. We added, to this tool, an implementation of the state estimate algorithm presented in this paper. McScM offers several model checking engines. The module we implemented is considered as a "new" engine. We call it by the command line:

```
mcscm.native -mc-engine control -simulation-only [other
options] <input>
```

The input is the model of the distributed system. The engine is called "control" because we aim at the control of distributed systems. By default, it starts with an interactive simulation of the model. At each step, the current state and the possible transitions are displayed. The user chooses one of these transitions and the tool updates the state of the system and the state estimates according to our algorithm. By default, the state estimates are not displayed, but their size is displayed.

B.2 Experiment on the running example

Here are some of the state estimates computed while the system executes the sequence of actions of Example 2. The widening parameter is $k = 1$. Each estimate is a map from a global location to a queue content. We first give the initial estimate of each process. With the option *-display-estimate*, the tool displays, for each estimate and each location, the internal representation of the queue content - a QDD. We give here a more readable representation as regular expression. We also give here the queue names as they appear in the paper - the implementation uses numbers instead of names.

$\mathcal{E}_1$

Location	$Q_{1,2}$	$Q_{2,1}$	$Q_{2,3}$	$Q_{3,1}$
$A_0 \times B_0 \times C_0$	ε	ε	ε	ε

$\mathcal{E}_2$

Location	$Q_{1,2}$	$Q_{2,1}$	$Q_{2,3}$	$Q_{3,1}$
$A_0 \times B_0 \times C_0$	ε	ε	ε	ε
$A_1 \times B_0 \times C_0$	c	ε	ε	ε

$\mathcal{E}_3$

Location	$Q_{1,2}$	$Q_{2,1}$	$Q_{2,3}$	$Q_{3,1}$
$A_0 \times B_0 \times C_0$	ε	ε	ε	ε
$A_1 \times B_0 \times C_0$	c	ε	ε	ε
$A_1 \times B_1 \times C_0$	ε	b^*	ε	ε
$A_1 \times B_2 \times C_0$	ε	$b^*a + \varepsilon$	ε	ε
$A_1 \times B_3 \times C_0$	ε	$b^*a + \varepsilon$	d	ε

At the beginning, $\mathcal{E}_1$ knows exactly the current state, because nothing can happen if $\mathcal{T}_1$ does not send message c to $\mathcal{T}_2$. $\mathcal{E}_2$ is almost as precise. However $\mathcal{E}_3$ is less precise, because many events could occur before $\mathcal{T}_3$ receives its first message. After the first transition $\langle A_0, Q_{1,2}!c, A_1\rangle$, the state estimate of the estimator $\mathcal{E}_1$ is similar to $\mathcal{E}_3$, but:

$Location$	$Q_{1,2}$	$Q_{2,1}$	$Q_{2,3}$	$Q_{3,1}$
$A_1 \times B_0 \times C_0$	c	ε	ε	ε
$A_1 \times B_1 \times C_0$	ε	b^*	ε	ε
$A_1 \times B_2 \times C_0$	ε	b^*a	ε	ε
$A_1 \times B_3 \times C_0$	ε	b^*a	d	ε
$A_1 \times B_3 \times C_0$	ε	b^*a	ε	d
$A_1 \times B_3 \times C_1$	ε	b^*a	ε	ε

1. $\mathcal{E}_1$ does not know if $\mathcal{T}_3$ received d and sent another message d in channel $Q_{3,1}$.
2. the content of channel $Q_{2,1}$ is a bit more precise, b^*a instead of $b^*a + \varepsilon$, because $\mathcal{E}_1$ knows that $\mathcal{T}_1$ did not receive any message.

Note that location $A_1 \times B_3 \times C_0$ appears two times. The actual regular expression associated to this location is $\varepsilon\#b^*a\#(d\#\varepsilon + \varepsilon\#d)$ where $\#$ is a special symbol to separate the different queues. So we write it as $\varepsilon\#b^*a\#d\#\varepsilon + \varepsilon\#b^*a\#\varepsilon\#d$ in this state estimate.

If we continue, the tool calculates and displays the estimate as in Example 2. We only give, in the table below, the size of the estimates.

$Subsystem$	$Event$	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$
$-$	$start$	4	9	35
$\mathcal{T}_1$	$Q_{1,2}!c$	42	$-$	$-$
$\mathcal{T}_2$	$Q_{1,2}?c$	$-$	4	$-$
$\mathcal{T}_2$	$Q_{2,1}!a$	$-$	9	$-$
$\mathcal{T}_2$	$Q_{2,3}!d$	$-$	46	$-$
$\mathcal{T}_3$	$Q_{2,3}?d$	$-$	$-$	7
$\mathcal{T}_3$	$Q_{3,1}!d$	$-$	$-$	132
$\mathcal{T}_1$	$Q_{2,1}?a$	14	$-$	$-$
$\mathcal{T}_1$	$Q_{3,1}?d$	4	$-$	$-$