Anonymity, Failures, Detectors and Consensus

Zohir Bouzid, Corentin Travers

Anonymity, Failures, Detectors and Consensus

Zohir Bouzid
UPMC - LIP6-CNRS, France.
zohir.bouzid@lip6.fr

Corentin Travers
UB1 - LaBRI-CNRS, France.
travers@labri.fr

Abstract

The paper investigates the consensus problem in anonymous, failures prone and asynchronous message passing systems. It determines the weakest failure detector in anonymous message passing systems for solving consensus. Two failure detector classes $A\Sigma'$ and $A\mathcal{L}$ are introduced, which may be seen as the anonymous counterparts of Σ and Ω, the weakest failure detectors for consensus in non-anonymous system. An anonymous consensus algorithm that relies on $A\Sigma'$ and $A\mathcal{L}$ is presented. The paper then establishes that $A\Sigma'$ and $A\mathcal{L}$ can be distributively emulated given any failure detector D that can be used to solve consensus, thereby proving the necessity of $A\Sigma'$ and $A\mathcal{L}$ to solve crash-tolerant consensus in anonymous systems.

1 Introduction

Anonymous systems A common, often implicit, assumption in distributed computing is that the system is *eponymous*: each process is provided with an unique identifier. On the other hand, in *anonymous* systems, processes have no identity and are programmed identically. When provided with the same input, processes in such systems are indistinguishable. Anonymity adds a new, challenging, difficulty to distributed computing.

From a practical point of view, anonymity is sometimes unavoidable. Consider, for example, a system composed of many tiny nodes, e.g., sensors networks. Each node may have very limited storage and computational capability, and might not have been provided with unique identifier [2]. Anonymity might also be a desirable property when privacy is concerned [13, 19]. Starting from the pioneering work of Angluin [1], the computational power of anonymous networks in the failure-free case has been investigated for particular or general graph topology, e.g., [4, 5, 25, 26].

Consensus and failure detectors in eponymous systems Besides anonymity, a major difficulty is coping with *failures* and *asynchrony*. Many simple problems cannot be solved in asynchronous and failures-prone distributed system. A prominent example is consensus, which is a fundamental problem in fault-tolerant distributed computing. Informally, n processes, each starting with a private value, are required to agree on one value chosen among their initial values. In message-passing systems, it is well known that *asynchronous fault tolerant* consensus is impossible as soon as at least one process may fail by crashing [20]. Trivially, consensus is thus impossible in anonymous, asynchronous and failure-prone message passing system. Several approaches have been identified to overcome this impossibility, including randomization (e.g., [7]), strengthening the model with timing assumptions (e.g., [18]) and failure detectors (e.g., [12]).

A *failure detector* is a distributed device that provides processes with possibly unreliable information about failures. According to the quality of the information, several classes of failure detectors can be defined. Given a distributed problem P, a natural question is to determine the *weakest failure detector* for P, that is

a failure detector D which is both *sufficient* to solve the problem – there is an asynchronous, fault tolerant algorithm based on D that solves P – and *necessary*, in the sense that any failure detector D' that allows solving P can be used to emulate D.

For consensus, it has been shown that the combination of failures detectors Ω and Σ (denoted $\Omega \times \Sigma$) is both sufficient and necessary [11, 15]. A failure detector Ω maintains at each process p a read-only variable LEADER_p that contains a process identity. After an unbounded, but finite, time the value of each variable LEADER_p is permanently the identity of the same non-faulty process. Failure detector Σ maintains at each process a set of process identities such every two sets intersect, and after an unbounded, but again finite, time, each set includes only non-faulty processes.

Consensus and failures detector in eponymous systems The failure detector based approach has been investigated recently in anonymous systems. Although it has been introduced in the context of eponymous systems, the loneliness failure detector $\mathcal{L}$ introduced in [16] may be used in anonymous systems as its output is identity-free. Similarly, the failure detector Ψ^x introduced in [23] outputs an estimation of the number of failures and is thus adapted to anonymous systems. A class of anonymously perfect failure detector is defined in [8] as an adaptation of the perfect failure detector [12] to the anonymous context. There, it is shown that the round-complexity of solving consensus with perfect failure detection essentially doubles when moving from eponymous to anonymous systems. A fault-tolerant consensus algorithm for asynchronous systems is presented in [17]. The algorithm does not rely on failure detectors but instead on partial synchrony assumptions on the underlying network.

Bonnet and Raynal present in [9] several anonymous variants of classic failure detector and study their relative power. In particular, anonymous variants of Ω and Σ, called $A\Omega$ and $A\Sigma$ are defined, and a consensus algorithm based on these two failure detectors is described. The anonymous counterpart of Ω and Σ introduced in the present paper are strictly weaker. Bonnet and Raynal also asked "Consensus in anonymous distributed systems: is there a weakest failure detector?"[10]. The present paper answers positively to this question.

Content of the paper The paper generalizes the tight bound on failure detection for consensus in eponymous systems [11, 15] to the case of anonymous systems. It defines two new classes of failure detector that generalizes Ω and Σ and shows that are both necessary and sufficient to solve consensus in anonymous systems subject to any number of crash failures. In more details, the paper has the following contributions:

1. It first introduces two new classes of failure detectors, called $A\mathcal{L}$ and $A\Sigma'$, suited to anonymous systems. $A\mathcal{L}$ and $A\Sigma'$ may be seen as anonymous counterparts of the classes Ω and Σ. Differently from previous attempts to generalize Ω to anonymous systems, $A\mathcal{L}$ eventually distinguishes a *set* L of non-faulty, possibly identical, processes. For processes in L, the output $A\mathcal{L}$ eventually converges to the same value, from which the size of L can be inferred. Each other process is eventually informed that it is not part of this set. The output of $A\Sigma'$ has two components: a label and a set of quorums, each quorum being a multi-set of labels. At each process, both components may never stabilize. Labels may be seen as temporary identifiers assigned to the processes by the failure detector. The label component allows to map each quorums Q to a collection of set of processes (we call such set an *instance* of Q). Instances of quorums have similar properties as the set of processes identities output by Σ: every two instances intersect and each set of quorums eventually includes a quorum that has an instance containing only correct processes. Although similar to the class $A\Sigma$ introduced in [9], $A\Sigma'$ is strictly weaker than $A\Sigma$

2. The paper then presents a $(A\Sigma' \times A\mathcal{L})$-based consensus algorithm for anonymous and asynchronous system. The algorithm tolerates an arbitrary number of failures and is "genuinely anonymous" [9], as

processes are not required to be aware of the total number n of processes. Although the general structure of the algorithm is the same as the well-known Ω-based eponymous consensus in [24], anonymity and the weak guarantees offered by $A\Sigma' \times A\mathcal{L}$ impose to modify the algorithm of [24] in non-trivial ways. In particular, the safety part of the protocol is encapsulated in a simple abstraction called *intersecting-sets* [22], that we implement separately using $A\Sigma'$. An intersecting-sets abstraction supports a propose() primitive that takes as parameter a value and returns a set of values. Each set that is returned is valid, in the sense that it includes only values that have been proposed, and intersects any returned set. By combining two intersecting sets abstraction, one can implement an adopt-commit abstraction [27], which is a basic building block in many agreement algorithms, e.g., [3].

3. Finally, the paper shows how to emulate $A\mathcal{L}$ and $A\Sigma'$ from any pair $(\mathcal{A}, D)$, where $\mathcal{A}$ is a consensus algorithm that uses failure detector. The extraction is performed in a slightly stronger communication model: the channels are assumed to be FIFO (this assumption is not required for the sufficiency part).

A standard procedure in extracting weakest failure detectors is the construction of a *precedence graph* (DAG, [11, 21]) that describes temporal relationships as well as ownership between failure detector outputs. Typically, each node of the graph is labeled with a failure detector output and a process id; a path then represents a sequence of values that may be output by the failure detector in the chronological order induced by the path in some execution of $\mathcal{A}$. The precedence graph is used to simulate execution of $\mathcal{A}$, from which information about the underlying failure pattern can be inferred.

The main challenge lies in extending the precedence graph construction to the anonymous case. Due to anonymity, it may not be possible to distinguish failure detector values output at distinct processes. This further complicates the tracking of precedence relations between failures detector outputs. Part of these difficulties is resolved by relying on *reliable causal broadcast* abstraction, which may be of independent interest, to exchange failure detector values among processes.

Roadmap The paper is made of 6 sections. The main features of the computational model are presented in Section 2. Section 3 introduces the failure detector framework and the new classes $A\Sigma'$ and $A\mathcal{L}$ of anonymous failure detectors. Section 4 establishes the sufficiency of $A\Sigma' \times A\mathcal{L}$ for solving consensus in anonymous systems. An implementation of a causal reliable broadcast is given in Section 5. Section 6 deals with the necessity of $A\mathcal{L} \times A\Sigma'$. A comparison between $A\Sigma'$ and the failure detector $A\Sigma$ introduced in [9] appears in Appendix A. Due to space constraints, proofs have been moved to appendixes.

2 Model

We consider an *anonymous* and *asynchronous* system in which processes communicate by *message-passing* and may fail by *crashing*. Except for anonymity, the model is the same as in the failure detector literature, e.g., [9, 11, 14]. We recall here the main features of this model, and refer to [11] for a more detailed and accurate description.

Anonymous processes The system consists in a set Π of n processes. Processes are anonymous: they do not do not have identifiers and they execute identical algorithms. The total number of processes n is however known by the processes. To simplify the exposition, it is sometimes convenient to denote $\Pi = \{p_1, \ldots, p_n\}$ the processes in the system. The index i is not known by the processes and only available to an external observer. Processes are asynchronous in the sense that each process runs at its own speed, independently of the other processes. The system is equipped with a global clock whose ticks range is the set of positive

integer $\mathbb{N}$. Similarly to processes indexes, the global clock is not available to the processes. It is used from an external point of view to state and prove properties about executions.

Crash failures Processes may fail by *crashing*, i.e. may prematurely halt. A process is *correct* in an execution if it never crashes in this execution; otherwise it is *faulty*. $Correct \subseteq \Pi$ denote the set of correct processes. If not otherwise specified, we assume that every process may fail in an execution. A *failure pattern* is an increasing function $F : \mathbb{N} \to \Pi$, where $F(\tau)$ is the set of processes that have failed by time τ. An *environment* is a set of failure pattern. If not otherwise specified, we consider in this paper the *wait-free* environment in which every process may crash in an execution.

Communication Processes communicate via sending and receiving messages over an asynchronous network. Each pair of processes is connected by a bi-directional channel. The channels are *asynchronous* but *reliable*. Reliable means that there is no creation, alteration or loss of messages whereas asynchronous means that message transfer delays are finite but unbounded. When extracting $A\mathcal{L} \times A\Sigma'$, we assume in addition that channels are FIFO.

The system provides a simple *broadcast* communication primitives. This primitive allows each process to send a message m to every process in the system including itself. Whereas channels are reliable, *broadcast* is not: if a process calls *broadcast*(m) and fails before returning from that call, m is sent to an arbitrary, possibly empty, set of processes. We show in Section 5 how to implement a *reliable broadcast* abstraction on top of the simple broadcast primitive offered by the system.

Consensus In the consensus problem, each process proposes a value and has to decide a value such that the following properties are satisfied: (*Validity*) A decided value is a proposed value; (*Termination*) Every correct process eventually decides a value; (*Agreement*) No two distinct values are decided.

3 Failure detectors

As indicated in the introduction, a 2failure detector is a distributed oracle that provides (perhaps inaccurate) hints on the current failure pattern. Operationally, a failure detector provide at each process p a read-only variable FD_p. We denote by FD_p^τ the value of the variable at time τ. This variable is the output of the failure detector for process p at time τ. We recall next the main features of formal framework in which failure detectors are defined, as introduced in [11]. We then present the definition of the two new failure detectors $A\Sigma'$ and $A\mathcal{L}$, which are anonymous counterparts of the failure detectors Ω and Σ, respectively. The new failure detectors are similar, but strictly weaker than anonymous equivalent of Σ and Ω that have been previously proposed [9]. See Appendix A.

Definition of failures detector A *failure detector history H with range $\mathcal{R}$* is a function $H : \Pi \times \mathbb{N} \to \mathcal{R}$. $H(p_i, \tau)$ may be seen as the output of the local failure detector module of process p_i at time τ. A *failure detector D with range $\mathcal{R}_D$* is a function that maps each failure pattern to set of failure detector histories with range $\mathcal{R}_D$. Given a failure pattern F, $D(F)$ denotes the set of failure detector histories allowed by D in when the failure pattern is F.

For example, the range of the failure detector Ω, defined for eponymous systems [12], is Π. $H : \Pi \times \mathbb{N} \to \Pi \in \Omega(F)$ if there exists $p_\ell \in Correct(F), \tau$ such that $H(p_i, \tau') = p_\ell$, for all $p_i \in \Pi, \tau' \geq \tau$. The range of the failure detector Σ, defined also for eponymous systems [14], is 2^Π. $H : \Pi \times \mathbb{N} \to 2^\Pi \in \Sigma(F)$ if

(1) $H(p_i, \tau_i) \cap H(p_j, \tau_j) \neq \emptyset$, for all $p_i, p_j \in \Pi, \tau_i, \tau_j \in \mathbb{N}$ and (2) there exist τ such that $H(p_i, \tau') \subseteq Correct(F)$ for all $p_i \in \Pi, \tau' \geq \tau$.

Let $D1, D2$ denote two failure detectors. Failure detector $D1$ is *weaker than* $D2$ if there exists a distributed algorithm $\mathcal{T}_{D2 \rightarrow D1}$ that uses $D2$ to emulate the output of $D1$. More specifically, algorithm $\mathcal{T}_{D2 \rightarrow D1}$ maintains at each process p_i a variable out_{D1} intended to emulate the output of $D1$ at p_i; The variable can be used at each process to replace the actual output of $D1$: in any execution, p_i cannot distinguish between reading the variable out_{D1} or querying the failure detector $D1$. If $D1$ is weaker than $D2$ and $D2$ weaker than $D1$, $D1$ and $D2$ are said to be *equivalent*. On the contrary, if $D2$ is not weaker than $D1$, $D1$ is *strictly weaker* than $D2$.

Given a distributed task T, such as consensus, failure detector D is a *weakest failure detector* for T if (1) there exists an algorithm $\mathcal{A}_D$ for T that uses D and (2) for every failure detector D' that can be used to solve T, there exists an algorithm $\mathcal{T}_{D' \rightarrow D}$ that uses D' to solve D. Note that if $D1$ and $D2$ are weakest failure detector for T, then $D1$ is equivalent to $D2$.

The failure detector $A\Sigma'$ Let $\mathbb{L}$ be a (possibly infinite) set of *labels* endowed with a *partial order* relation $\sqsubseteq$. A failure detector of the class $A\Sigma'$ outputs at each process p a pair $(\text{LABEL}_p, \text{QUORUMS}_p)$ that consists of a label and a set of multi-set of labels.

Intuitively, the value of LABEL_p may be seen as the current identifier assigned to p by the failure detector and a multi-set of labels $Q \in \text{QUORUMS}_p$ may be seen as a quorum. However, as we are about to see, the successive values LABEL_p may not necessarily converge to a stable label. We associate sets of processes with quorum (= multi-set of labels) through the notion of an *instance* of a quorum.

Let $Q = \{\ell_1, \ldots, \ell_x\}$ be a multi-set of labels. The set of instances of Q, denoted by $I(Q)$, is defined as follows:

$$P \in I(Q) \iff \exists q_1, \ldots, q_x \in \Pi, \exists \tau_1, \ldots, \tau_x \in \mathbb{N} : P = \{q_1, \ldots, q_x\} \text{ and } \forall i, 1 \leq i \leq x, \ell_i \sqsubseteq \text{LABEL}_{q_i}^{\tau_i}$$

That is, each process q_i in set P is uniquely mapped to a label $\ell_i \in Q$ in such way that, at some time τ_i, the label LABEL_{q_i} assigned by $A\Sigma'$ to q_i is larger that ℓ_i.

The outputs of a failure detector of the class $A\Sigma'$ satisfy the following properties :

- *Monotony*: The successive labels output at each process forms an increasing sequence according to the order $\sqsubseteq$. Formally, $\forall p, \forall \tau, \tau' : \tau < \tau' \implies \text{LABEL}_p^{\tau} \sqsubseteq \text{LABEL}_p^{\tau'}$.
- *Liveness*: Eventually, the set of quorums output by the failure detector includes a quorum Q whose set of instances $I(Q)$ contains a set of correct processes. Formally, $\exists \tau$ such that $\forall \tau' \geq \tau$, $\forall p$, $\exists Q$ multi-set of labels, $\exists P \subseteq Correct : Q \in \text{QUORUMS}_p^{\tau'} \wedge P \in I(Q)$.
- *Intersection*: This last property states that any instances of any pair of quorums intersect: $\forall \tau, \tau', \forall p, p', \forall P \in I(\text{QUORUM}_p^{\tau}), \forall P' \in I(\text{QUORUM}_{p'}^{\tau'}), P \cap P' \neq \emptyset$.

The failure detector family $\{A\mathcal{L}^k\}_{1 \leq k \leq n}$ A failure detector $A\mathcal{L}^k$ provides for each process p a variable LEADER_p that consists in two components denoted RANK_p and MULTIPLICITY_p. RANK_p stores an integer and the value of MULTIPLICITY_p is always an integer in $\{0, \ldots, n\}$. Intuitively, when $\text{LEADER}_p = (r, m)$, the failure detector indicates, perhaps erroneously, that m non-faulty processes are ranked r. However, there is a time after which the indication is accurate for at least one rank r. More formally, the outputs of a failure detector of the class $A\mathcal{L}^k$ satisfies the following property : $\exists r \in \{1, \ldots, k\}, \exists L \subseteq Correct, \exists \tau$ such that $\forall \tau' \geq \tau$:

- $\forall p \in L : (\text{RANK}_p^{\tau'}, \text{MULTIPLICITY}_p^{\tau'}) = (r, |L|)$

- $\forall p \notin L : \text{RANK}_p^{\tau'} \neq r$

When $k = 1$, we simply note $A\mathcal{L}$ instead of $A\mathcal{L}^1$. In that case, the content of RANK_p may be interpreted as a binary value indicating whether p is a leader ($\text{RANK}_p = 1$) or not ($\text{RANK}_p \neq 1$). When $\text{RANK}_p = 1$, MULTIPLICITY_p then converges towards the actual number of correct leader.

4 A $(A\Sigma' \times A\mathcal{L})$-based consensus algorithm

We present an asynchronous consensus algorithm for anonymous system that relies on a failures detector of the class $A\Sigma' \times A\mathcal{L}$. The algorithm builds upon from previous failure detectors-based consensus algorithms for eponymous system. It is in particular inspired from the three-phases consensus protocol of Mostefaoui and Raynal [24].

Intersecting sets A key ingredient in our consensus algorithm is an implementation in an anonymous system of an *intersecting sets* abstraction, which has been defined by Friedman et al. [22]. An intersecting sets abstraction $INTSET$ supports a single primitive denoted $\mathsf{propose}(\cdot)$, that can be invoked at most once by each process. A $\mathsf{propose}(\cdot)$ operation takes as input parameter a value v in some set $\mathbb{V}$ and returns a set of values. The name of the abstraction comes from the property that every two sets returned by $\mathsf{propose}(\cdot)$ operations intersect. Formally, the $INTSET$ abstraction is specified by the following properties:

- *Termination*. Every invocation of $\mathsf{propose}(\cdot)$ by a correct process returns.
- *Validity*. If V is the set returned by an invocation of $\mathsf{propose}(\cdot)$, then then some process previously invoked $\mathsf{propose}(v)$, for each value $v \in V$.
- *Intersection*. For every pair of sets V, V' returned by invocations of $\mathsf{propose}(\cdot)$, $V \cap V' \neq \emptyset$.

$A\Sigma'$-based implementation of intersecting sets An algorithm that implements an intersecting sets abstraction in an anonymous system is described in Figure 1. The algorithm relies on an underlying failure detector of the class $A\Sigma'$. It tolerates any number of processes crash failures.

For each process p, the algorithm consists in two tasks $T1$ and $T2$ that run in parallel.Task $T2$ is triggered each time a new message is received by p. In task $T1$, the execution proceeds in asynchronous rounds. In each round r, p queries its failure detector of the class $A\Sigma'$ to obtain a new set of quorum Q and a label my_label (line 3). The set of quorums p has obtained so far is stored in the variable $\mathcal{Q}$. Then, p broadcasts its initial value $prop$ and its current label my_label together with its current round number r (message $EST(prop, my\_label, r)$, line 4). This is repeated until p has received a multi-set M of EST messages sent in the same round by each process of an *instance* of a quorum Q stored in $\mathcal{Q}$ (line 5). p then broadcasts a message $DEC(V)$, where V is the set of values carried by the messages in M, to indicate that V can be safely decided. Indeed, each set of values V obtained in this way is the set of proposals of a set processes that form an instance of some quorum Q. Suppose that $DEC(V')$ is broadcast by another process. Since for any two quorums Q and Q' output by the failure detector, any two instances S and S' of Q and Q' respectively have at least a process in common (intersection property of the class $A\Sigma'$), the intersection of sets V and V' is non-empty. Therefore, when a process receives a message $DEC(V)$, it decides V (line 8).

In order to identify the proposals of a set of processes that form an instance of a quorum, process p maintains in the variable $rec[i]$ the pairs $(value, label)$ carried by the messages EST tagged with round number i it receives (line 7). As several processes may broadcast in round i the same message $EST(value, label, i)$, $rec[i]$ is a multi-set. Note however that each process sends at most one message $EST(*, *, i)$ in round i and thus no two messages in $rec[i]$ are sent by the same process. Let $Q = \{\ell_1, \ldots, \ell_x\}$ be a quorum obtained

by p from its failure detector. Recall that a set of processes $I = \{p_1, \ldots, p_x\}$ forms an instance of a quorum Q if for each process p_i eventually $\ell_i \sqsubseteq label_{p_i}$, where $label_{p_i}$ is the label output by the failure detector at process p_i. Hence, if $rec[i]$ contains a multi-set $M = \{(v_1, \lambda_1), \ldots, (v_x, \lambda_x)\}$ with $\ell_j \sqsubseteq \lambda_j$, for each $j, 1 \leq j \leq x$ (line 5), then $v_1, \ldots, v_x$ are the values sent by a set of process that forms an instance of Q, as desired.

```
INTSET.propose(prop):
(1)  foreach i ≥ 1 do rec[i] ← ∅ endfor; /* multi-sets of messages, initially empty */
     Q ← ∅; /* set of quorums */ r ← 0; /* round number */
     start T1, T2

task T1:
(2)  repeat r ← r + 1;
(3)       (my_label, Q) ← (LABEL, QUORUM);                    /* query failure detector AΣ' */
(4)       Q ← Q ∪ Q; broadcast EST(prop, my_label, r);
(5)  until ∃i, ∃ multi-set M = {(v_1, λ_1), ..., (v_x, λ_x)} , ∃Q = {ℓ_1, ..., ℓ_x} ∈ Q
          such that (M ⊆ rec[i]) ∧ (∀i, 1 ≤ i ≤ x, ℓ_i ⊑ λ_i)
(6)  let V = {v : (v, *) ∈ M}; broadcast DEC(V)

task T2: when a message m is received :
(7)  case m = EST(v, λ, i) then add (λ, v) to the multi-set rec[i]
(8)       m = DEC(V)    then broadcast DEC(V); stop T1; decide(V); return;
(9)  endcase
```

Figure 1: $A\Sigma'$-based implementation of an intersecting-set object.

The proof of the algorithm can be found in Appendix B.1

$A\mathcal{L} \times A\Sigma'$-based consensus A consensus algorithm for an anonymous system equipped with failure detectors $A\mathcal{L}$ and $A\Sigma'$ is described in Figure 2. The algorithm tolerate an arbitrary number of failures.

A process p proposing v to the consensus initiates the algorithm by invoking propose(v). The algorithm consists in two tasks $T1$ and $T2$ that are ran in parallel. Task $T2$ is triggered whenever p receives a message. When a message $DEC(u)$ is received, p first relays this messages, decides u (line 17) and halts. Relaying $DEC(u)$ ensures that if p decides, every correct process eventually receives a message of type DEC and consequently also decides. Other types of messages ($EST1$ and $EST2$) are tagged with a round number; when a message tagged with round i is received, it is stored in the multi-set $rec[i]$ (line 18).

In task $T1$, p maintains a local estimate est_p whose content is the value p currently favors. Initially, the value of est_p is the value proposed by p. The task proceeds in rounds. Each round r is divided in two phases. The goal of the first phase is for the processes to adopt the same estimate while in the second phase processes check whether the first phase was successful. If the processes succeed in adopting the same estimate v in phase 1, the check performed in phase 2 is passed and messages $DEC(v)$ are broadcast (line 12), ultimately resulting in the decision of v (line 17). However, as we will see, a process may see the first phase successful even if processes have distinct estimates at the end of the phase. In that case, however, the algorithm ensures that every process that has not failed adopt the same estimate at the end of the second phase, thus preventing distinct values to be decided in subsequent rounds.

In the first phase (lines 3–8), process p strives to adopt the smallest estimate of the processes that are currently appointed as leaders by $A\mathcal{L}$. To that end, p periodically queries the failure detector $A\mathcal{L}$. If it is a leader, namely, the value of the variable LEADER provided by the failure detector is $(1, m)$, for some $m \geq 1$, it broadcasts a messages $EST1(v, r)$ where v is p's current estimate (line 4). A flag ($sent1$) is then set to true to prevent p form sending more than once message $EST1$ in round r. p then waits until

```
CONS.propose(prop):
(1)  est ← prop; /* estimate */ r ← 0; /* round number */
     rec ← ∅; /* multi-sets of messages, initially empty */
     INTS[1... + ∞][2]; /* array of intersecting-sets objects */
     start T1, T2

task T1:
(2)  repeat r ← r + 1; sent1 ← false; sent2 ← false;
(3)     repeat (isleader, m) ← LEADER;                                          /* phase 1 */
(4)        if (isleader) ∧ (¬sent1) then broadcast EST1(est, r); sent1 ← true endif
(5)        if (isleader) ∧ (¬sent2) ∧ (|{EST1(∗, r′) ∈ rec : r′ = r}| ≥ m)
(6)        then let v = min{v : EST1(v_i, r) ∈ rec}; broadcast EST2(v, r); sent2 ← true
(7)        endif
(8)     until ∃v : EST2(v, r) ∈ rec
(9)     broadcast EST2(v, r); est ← v; V ← INTS[r, 1].propose(v);               /* phase 2 */
(10)    if V = {u} then aux ← u else aux ← ⊥ endif
(11)    U ← INTS[r, 2].propose(aux);
(12)    case U = {u} ∧ u ≠ ⊥ then est ← u; broadcast DEC(u)
(13)         U = {u, ⊥}      then est ← u
(14)         default         then nope
(15)    endcase
(16) endrepeat

task T2: when a message m is received:
(17) if m = DEC(v) then broadcast DEC(v); decide(v); return;
(18)               else add m to the multi-set rec
(19) endif
```

Figure 2: $A\mathcal{L} \times A\Sigma'$-based implementation of consensus.

it receives a message $EST1(*, r)$ from each of the processes that is currently leader. As processes are anonymous, p does not known which these processes are. However, the second component of LEADER$_p$ gives an indication, which is eventually accurate, on the number of correct leaders. As each process sends at most one message $EST1$ in each round, and only if it considers itself as a leader, p suspects it has received an $EST1$ message form each leader when the total number of $EST1$ messages it has received is larger than or equal to the number of leaders indicated by the failure detector (line 5). When this occurs, p broadcasts a message $EST2(v', r)$ (line 6), with v' equal to the smallest value carried by the messages $EST1(*, r)$ received by p.

The first phase terminates when a message $EST2(v, r)$ is received. When this occurs, $EST2(v, r)$ is first broadcast to ensure that every process eventually exits the first phase. The estimate is then updated to v (line 9). A process that is not leader simply waits until it receives a message $EST2(*, r)$. As the output of $A\mathcal{L}$ may be arbitrary for some time, several processes may have distinct estimate at the end of the first phase.

The second phase of round r is associated with two instances $INTS[r, 1]$ and $INTS[r, 2]$ of the intersecting-sets abstraction. Each process first proposes its current estimate to $INTS[r, 1]$ (line 9) and gets back a set V. An auxiliary local variable aux is set to u if $V = \{u\}$ and to $\perp$ otherwise. Note that due to the intersection property of the abstraction, if $aux_p = v \neq \perp$ then, for all processes p', $aux_{p'} = v$ or $\perp$. Moreover, if v is the estimated of each process at the end of the first phase, then $aux_p = v$ for every process p.

The value of aux_p is then proposed to the second intersecting-sets abstraction $INTS[2, r]$. If the returned set U contained a value $u \neq \perp$, the estimate is updated to that value (lines12–13). In addition, if $U = \{u\}$, a message $DEC(u)$ is broadcast. In that case, note that by the intersection property of

$INTS[r, 2]$, every set returned by the abstraction contains u. Hence, every process that has not failed has the same estimate at the end of round r, ensuring agreement.

The proof of the algorithm can be found in Appendix B.2.

5 Reliable broadcast

An important ingredient in the minimality proof is a *reliable causal broadcast* abstraction. Such an abstraction is made up of two primitives denoted *RC_broadcast*() and *RC_deliver*() that allow processes to broadcast and deliver messages (we say accordingly that a message is *RC_broadcast* or *RC_delivered* by a process). Since processes are anonymous, we cannot assume that messages are unique. To disambiguate between messages, we use a mapping κ from the set of *RC_deliver*(m) events to the set of *RC_broadcast*(m) events, as in [6]. κ maps each *RC_deliver*(m) event to an earlier *RC_broadcast*(m) event with the same message. For each process p, let κ_p denote the restriction of κ to the set of *RC_deliver*(m) events occurring at p. The reliable causal broadcast abstraction satisfies the following properties:

1. *Integrity*. κ is well defined. That is, if message m is RC_delivered k times by process p, then k invocations of *RC_broadcast*(m) (possibly by distinct processes) occur before m is RC_delivered for the kth time by p, for all k.
2. *No Duplicates*. κ_p is one-to-one: If m is RC_broadcast k times, m is RC_delivered at most k times at each process, for all k.
3. *Non-faulty Liveness*. If p is a correct process, the restriction of κ_p to *RC_broadcast*() events that occur at correct processes is onto. That is, each *RC_broadcast*(m) event that occurs at a correct process is the image by κ_p of a *RC_deliver*(m) event.
4. *Faulty Liveness*. If for some faulty process q, a *RC_deliver*(m) event is mapped by κ_q to *RC_broadcast*(m) event, then this particular event is the image by κ_p of a *RC_deliver*(m) event occurring at p, for all correct process p. That is, if m is RC_delivered k times by a process, then m is RC_delivered at least k times by every correct process.
5. *Causal ordering*. If *RC_broadcast*(m) = κ_p(*RC_deliver*(m)), *RC_broadcast*(m') = κ_p(*RC_deliver*(m')) and *RC_broadcast*(m) causally precedes *RC_broadcast*(m'), then *RC_deliver*(m) happens before *RC_deliver*(m') at process p, for all m, m'.

Reliable broadcast We first present an algorithm that implements the reliable abstraction in the anonymous model. The reliable causal abstraction satisfies properties 1–4 above. The algorithm is described in Figure 3. It assumes that no processes send twice the same message. This can easily be achieved by appending a sequence number to each message.

The straightforward idea of relaying each message before delivering fails to ensure the non-faulty liveness property. Consider the following protocol: each process maintains a counter $rcv[m]$ for each possible message m. When a message m is received, the counter $rcv[m]$ is incremented and a message $RELAY(m, rcv[m])$ is sent. When $RELAY(m, k)$ is received, m is delivered a number of times equal to the difference between k and the number of times m has previously been delivered. Suppose that the same message m is RC_broadcast by a faulty process p and later by a correct process q. p sends m to process $p' \neq q$ and crashes immediately after. p' sends $RELAY(m, 1)$ to all according to the protocol, but fails immediately after. Process q sends m after p and p' has crashed. Hence, no processes receive m twice, and consequently no messages $RELAY(m, 2)$ are sent. Therefore, a process q' may receive a message $RELAY(m, 1)$, and thus delivers m before q calls *RC_broadcast*(m). Since no messages $RELAY(m, 2)$

are sent, q' never delivers m after the call of *RC_broadcast*(m) by q, which violates the non-faulty liveness property.

We improve the naive protocol as follows: each time a process invokes *RC_broadcast*(m), it sends n copies of m labeled $1, \ldots, n$ (messages $MSG(m, i)$, line 1). At each process p, and for each message m, the counter $RCV[m][i]$ keeps tracks of the number of copies labeled i that p has received. p sends $RELAY(m, k)$ messages only if it has not done so previously and it has received k times $MSG(m, i)$, for each $1 \leq i \leq n - k + 1$ (lines 3–4). Suppose that when a correct process q invokes *RC_broadcast*(m), the maximum ℓ such that $RELAY(m, \ell)$ has been sent previously is k. Note that if a process has received $MSG(m, i)$ k times, then every process receives $MSG(m, j)$ at least k times, for each $j : 1 \leq j \leq i - 1$. Hence, $RELAY(m, k + 1)$ is eventually sent after the call of *RC_broadcast*(m) by q, ensuring that *RC_deliver*(m) occurs at each correct process after the call of *RC_broadcast*(m) by q.

init: foreach $m \in \mathcal{M}$ **do** $RCV[m][1..n] \leftarrow [0, \ldots, 0]$ /* array of n counters */
$DLV[m] \leftarrow 0$ /* counter */
$level(m) \leftarrow 0$ **endfor**

when *RC_broadcast*(m) is invoked:
(1) **for** $k = 1$ **to** n **do** *broadcast* $MSG(m, k)$ **endfor** /* broadcast message n times */

when $MSG(m, k)$ is received:
(2) $RCV[m][k] \leftarrow RCV[m][k] + 1$; /* increment kth counter */
(3) **let** $\ell = \max\{i : \forall j, 1 \leq j \leq n - i + 1, RCV[m][j] \geq i\}$
(4) **if** $level(m) < \ell$ **then** $level(m) \leftarrow \ell$; *broadcast* $RELAY(m, level(m))$ **endif**

when $RELAY(m, k)$ is received:
(5) **if** $k > DLV[m]$ **then** *broadcast* $RELAY(m, k)$;
(6) **repeat** $(k - DLV[m])$ times **do** *RC_deliver*(m); $DLV[m] \leftarrow DLV[m] + 1$ **enddo endif**

Figure 3: Reliable broadcast

The proof of the protocol can be found in Appendix C.

From reliable to reliable causal broadcast. To obtain the $5th$ property, namely causality, we assume that channels are FIFO. Causality then results from the assumption of FIFO channels and the fact that each message is relayed (line 5) before being received.

6 Necessity of $(A\Sigma' \times A\mathcal{L})$

Theorem 6.1. *$A\Sigma' \times A\mathcal{L}$ is necessary for consensus in anonymous message passing systems.*

The full proof appears in Appendixes D–G.

References

[1] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[2] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[3] James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 460–467. ACM, 2010.

[4] Hagit Attiya and Marc Snir. Better computing on the anonymous ring. *J. Algorithms*, 12(2):204–238, 1991.

[5] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.

[6] Hagit Attiya and Jennifer Welch. *Distributed Computing*. Wiley, 2004.

[7] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[8] François Bonnet and Michel Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash and anonymity. In *Proceedings of the 23rd International Symposium on Distributed Computing (DISC)*, volume 5805 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2009.

[9] François Bonnet and Michel Raynal. Anonymous asynchronous systems: The case of failure detectors. In *Proceedings of the 24th International Symposium Distributed Computing (DISC)*, volume 6343 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 2010.

[10] François Bonnet and Michel Raynal. Consensus in anonymous distributed systems: Is there a weakest failure detector? In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 206–213, 2010.

[11] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[12] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[13] Tom Chothia and Konstantinos Chatzikokolakis. A survey of anonymous peer-to-peer file-sharing. In *Proceedings Workshops on Embedded and Ubiquitous Computing (EUC)*, volume 3823 of *Lecture Notes in Computer Science*, pages 744–755. Springer, 2005.

[14] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Tight failure detection bounds on atomic object implementations. *J. ACM*, 57(4), 2010.

[15] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 338–346. ACM, 2004.

[16] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The weakest failure detector for message passing set-agreement. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC)*, volume 5218 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2008.

[17] Carole Delporte-Gallet, Hugues Fauconnier, and Andreas Tielmann. Fault-tolerant consensus in unknown and anonymous networks. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 368–375, Washington, DC, USA, 2009. IEEE Computer Society.

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[19] Hannes Federrath, editor. *Proceedings Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*. Springer, 2001.

[20] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[21] Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9, 2011.

[22] Roy Friedman, Achour Mostéfaoui, and Michel Raynal. Intersecting sets: a basic abstraction for asynchronous agreement problems. In *Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 15–22. IEEE Computer Society, 2005.

[23] Achour Mostéfaoui, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3):201–222, 2008.

[24] Achour Mostéfaoui and Michel Raynal. Leader-based consensus. *Parallel Processing Letters*, 11(1):95–107, 2001.

[25] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part i-characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):69–89, 1996.

[26] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part ii-decision and membership problems. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):90–96, 1996.

[27] Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 297–306. ACM, 1998.

A The failure detector class $A\Sigma$

In this section we compare the relative power of the failure detector classes $A\Sigma$ and $A\Sigma'$. The class has been introduced by Bonnet and Raynal in [9] as an anonymous counterpart of the quorum failure detector Σ [14]. We establish that $A\Sigma$ is strictly stronger than $A\Sigma'$: a failure detector of the class $A\Sigma'$ can be emulated in an asynchronous and anonymous system equipped with a failure detector of the class $A\Sigma$. On the contrary, one cannot emulate the output of a failure detector of the class $A\Sigma$ in an anonymous and asynchronous system equipped with a failure detector of the class $A\Sigma'$. That is,

Theorem A.1. *In anonymous and asynchronous systems made of $n \geq 5$ processes, the failure detector class $A\Sigma'$ is strictly weaker than the failure detector class $A\Sigma$.*

The failure detector class $A\Sigma$ The failure detector class $A\Sigma$ has been introduced in [9] as the counterpart for anonymous system of the quorum failure detector Σ. It provides each process p with a variable ASIGMA_p that contains pairs of integers (ℓ, m). Intuitively, ℓ is a label and m is the number of processes that know label ℓ. Formally, the output of a failure detector of the class $A\Sigma$ satisfies the following properties [9]:

- *Validity*. For every process p and every time τ, $\text{ASIGMA}_p = \{(\ell_1, m_1), \ldots, (\ell_x, m_x)\}$ where each (ℓ_i, m_i) is a pair of integer and $\ell_i \neq \ell_j$ for all $i, j : 1 \leq i \neq j \leq x$.
- *Monotonicity*. If $(\ell, m) \in \text{ASIGMA}_p^\tau$, $\forall \tau' \geq \tau, \exists m' \leq m : (\ell, m') \in \text{ASIGMA}_p^{\tau'}$.

For each label ℓ, let $S(\ell)$ denote the set of processes whose output contains ℓ at some point, i.e. $S(\ell) = \{p : \exists \tau, \exists m, (\ell, m) \in \text{ASIGMA}_p^\tau\}$.

- *Liveness*. Eventually, ASIGMA_p contains a label (ℓ, m) whose associated set $S(\ell)$ includes at least m correct process: $\exists (\ell, m), \exists \tau, \exists \ell$ such that $\forall \tau' \geq \tau, (\ell, m) \in \text{ASIGMA}_p^{\tau'}$ and $|S(\ell) \cap Correct| \geq m$.
- *Safety*. If $(\ell, m) \in \text{ASIGMA}_p^\tau$ and $(\ell', m') \in \text{ASIGMA}_{p'}^{\tau'}$, then for each pair of sets $Q \in S(\ell), Q' \in S(\ell')$ with $|Q| = m$ and $|Q'| = m'$ respectively, $Q \cap Q' \neq \emptyset$.

$A\Sigma$ emulates $A\Sigma'$ We present an algorithm that emulates the output of a failure detector of the class $A\Sigma'$ in an anonymous system equipped with a failure detector of the class $A\Sigma$. The pseudo-code appears in Figure 4.

Let $\mathbb{L}$ denote the set of all possible sequences of integers. We define a partial order $\sqsubseteq$ on $\mathbb{L}$ as follows: for all $u, v \in \mathbb{L}$, $u \sqsubseteq v$ if and only if there exists a prefix v_1 and a suffix v_2 of v, both possibly empty, such that $v = v_1 \cdot u \cdot v_2$. That is, $u \sqsubseteq v$ is u is a sub-sequence of v. In particular, if u consists in a single element $u \sqsubseteq v$ if that element occurs in the sequence v.

The failure detector of the class $A\Sigma'$ we emulate is defined over $\mathbb{L}$: each label output by the simulated failure detector is a sequence of integers, and each quorum consists in a multi-set of sequence of integers.

The algorithm that emulates a failure detector $A\Sigma'$ is described in Figure 4. At each process, the algorithm maintains two variables LABEL and QUORUMS whose values are the current output of the simulated failure detector. When the emulated failure detector is queried, the current values of these two variables is returned (task T2, line 9).

In task $T1$, the values of LABEL and QUORUMS are periodically updated according to the output of the underlying failure detector of the class $A\Sigma$. In each iteration of the **repeat** loop (lines 1–8), process p queries its local failure detector $A\Sigma$ and stores the resulting output in the local variable as (line 2). Whenever a new

pair (ℓ, m) is output for the first time, ℓ is appended to the current value of the sequence stored in LABEL if no pairs $(\ell, *)$ have been previously output (line 4), and a new quorum consisting in a multi-set of m copies of ℓ (line 5) is added to the set of quorums QUORUMS (line 6).

init: QUORUMS $\leftarrow \emptyset$; LABEL $\leftarrow \epsilon$; /* the output of the emulated failure detector */
/* ϵ is the empty sequence */

task $T1$:
(1) **repeat** forever
(2) $as \leftarrow$ ASIGMA /* query failure detector $A\Sigma$ */
(3) **for each** $(\ell, m) \in as$ **do**
(4) **if** $(\ell \not\sqsubseteq$ LABEL$)$ **then** LABEL $\leftarrow$ LABEL $\cdot\, \ell$ /* append ℓ to the sequence LABEL */
(5) **let** $Q_{\ell,m}$ the multi-set that consists in m occurrences of ℓ;
(6) QUORUMS $\leftarrow$ QUORUMS $\cup \{Q_{\ell,m}\}$ **endif** /* add $\{\underbrace{\ell, \ldots, \ell}_{m \text{ times}}\}$ to the set of quorums */
(7) **end for**
(8) **end repeat**

task $T2$: **when** $A\Sigma'$ is queried
(9) *return* (QUORUMS, LABEL);

Figure 4: $A\Sigma$-based emulation of a failure detector of the class $A\Sigma'$

Next, we show that the algorithm described in Figure 4 is a correct emulation of a failure detector of the class $A\Sigma'$. We use as in the pseudo-code $Q_{\ell,m}$ to denote a multi-set made of m copies of ℓ.

Observation A.2. *Let p be a correct process. If $(\ell, m) \in$ ASIGMA$_p^\tau$, there exists a time $\tau' \geq \tau$ and an integer $m' \leq m$ such that after τ', $\ell \sqsubseteq$ LABEL$_p$ and $Q_{\ell,m'} \in$ QUORUMS$_p$.*

Proof. As $(\ell, m) \in$ ASIGMA$_p^\tau$, it follows from the monotonicity of the class $A\Sigma$ that for all $\tau' \geq \tau$, ASIGMA$_p$ contains a pair (ℓ, m'), with $m' \leq m$. Consider the first time τ_1 following τ at which p queries the failure detector $A\Sigma$ (line 2). The variable as_p then contains a pair (ℓ, m_1) with $m_1 \leq m$. By the code, the quorum Q_{ℓ,m_1} is then added to QUORUMS$_p$ (line 6) and, if the current value of LABEL$_p$ does not contain ℓ, ℓ is appended at the end of the sequence (line 4). □

Lemma A.3. *The algorithm described in Figure 4 emulates a failure detector of the class $A\Sigma'$ in an anonymous system equipped with a failure detector of the class $A\Sigma$.*

Proof. We show that at each process, the values of the variables QUORUMS$_p$ and LABEL$_p$ satisfy the properties of the class $A\Sigma'$, namely, monotony, liveness and intersection.

- Monotony. Each time the variable LABEL$_p$ is updated (line 4), the old sequence is a prefix of the new value. Hence, by definition of the partial order $\sqsubseteq$, LABEL$_p^\tau \sqsubseteq$ LABEL$_p^{\tau'}$, for all p, $\tau \leq \tau'$.

- Liveness. This property concerns the existence of instances of quorums made of correct processes. Since the property is only required to hold eventually, we can focus on correct processes. Let p denote a correct process. By the liveness property of the class $A\Sigma$, there exists a time τ_p and a pair of integers (ℓ, m) such that (1) $(\ell, m) \in$ ASIGMA$_p^{\tau'}$, for all $\tau' \geq \tau_p$ and (2) $|S(\ell) \cap Correct| \geq m$. That is, after some time the output of $A\Sigma$ at process p always contains the pair (ℓ, m). Moreover, the output of $A\Sigma$ at m correct processes contains at some point a pair $(\ell, *)$.

  Since p periodically queries the underlying failure detector $A\Sigma$, (1) implies that it eventually obtains an output that includes the pair (ℓ, m). Therefore, after some time τ_1, $Q_{\ell,m} \in$ QUORUMS$_p$.

Let $q \in I = S(\ell) \cap Correct$. By definition of $S(\ell)$, $\exists \tau_q, m_q : (\ell, m_q) \in \text{ASIGMA}_q^{\tau_q}$. It thus follows from observation A.2 that eventually $\ell \sqsubseteq \text{LABEL}_q$. Therefore, for each process q in I, we have eventually $\ell \sqsubseteq \text{LABEL}_q$. As I contains at least $m = |Q_{\ell,m}|$ processes and each of them is a correct process, the quorum $Q_{\ell,m}$ has thus an instance made only of correct processes. Therefore, the liveness property of $A\Sigma'$ is satisfied since $Q_{\ell,m}$ is eventually always contained in the collection of quorums QUORUMS_p.

- Intersection. Let us first notice that for each quorum $Q \in \text{QUORUMS}_p$, there exists a pair (ℓ, m) such that (1) $Q = Q_{\ell,m}$ and (ℓ, m) is contains at some time in the output of the failure detector $A\Sigma$ at process p, i.e. $\exists \tau : (\ell, m) \in \text{ASIGMA}_p^{\tau}$.

  Let p, p' denote two processes, let τ, τ' denote two time instants and let $Q \in \text{LABEL}_p^{\tau}, Q' \in \text{LABEL}_{p'}^{\tau'}$ two quorums. Denote by T and T' two instances of Q and Q' respectively.

  By the remark above, there exits a pair (ℓ, m) and a time τ_1 such that $Q = Q_{\ell,m}$ and $(\ell, m) \in \text{ASIGMA}_p^{\tau_1}$. Note that, be the definition of instances of a quorum, for each process $q \in T$, the following holds eventually: $\ell \sqsubseteq \text{LABEL}_q$. By the code (line 4), ℓ is included in the sequence of labels LABEL_q of process q if and only if a query to $A\Sigma$ by process q returns a set containing a pair of the form $(\ell, *)$. Therefore, $q \in S(\ell)$, and consequently, $T \subseteq S(\ell)$.

  Similarly, there exits a pair (ℓ', m') and a time τ_1' such that $Q' = Q_{\ell',m'}$, $(\ell', m') \in \text{ASIGMA}_{p'}^{\tau_1'}$ and $T' \subseteq S(\ell')$. Note also that, since the sets of processes T and T' are instances of the quorums $Q_{\ell,m}$ and $Q_{\ell',m'}$ respectively, we have $|T| = m$ and $|T'| = m'$. Therefore, by the safety property of the failure detector class $A\Sigma$, $T \cap T' \neq \emptyset$, as required. □

$A\Sigma'$ does not emulate $A\Sigma$ We show that failure detector $A\Sigma'$ is not powerful enough to implement a failure detector of the class $A\Sigma$.

We first make a simple observation regarding the output of a failure detector of the class $A\Sigma$ in execution in which every process but 2 fails.

Observation A.4. *Consider an infinite run in which every process but 2, denoted p_1 and p_2, is faulty in a system equipped with a failure detector of the class $A\Sigma$. There exists an integer ℓ and time τ such that $\forall \tau' \geq \tau, (\ell, 2) \in \text{ASIGMA}_i^{\tau'}$, for some $i \in \{1, 2\}$.*

Proof. Let $i \in \{1, 2\}$. By the liveness property of $A\Sigma$, there exists a time τ_i and a pair (ℓ_i, m_i) such that (1) $(\ell_i, m_i) \in \text{ASIGMA}_i^{\tau'}$, for all $\tau' \geq \tau$, and (2) $|S(\ell_i) \cap Correct| \geq m_i$. As two processes are correct in the considered run, $m_i \leq 2$.

Assume for contradiction that $m_1 = m_2 = 1$. Let us consider the sets of processes $T_1 = \{p_1\} \subseteq S(\ell_1)$ and $T_2 = \{p_2\} \subseteq S(\ell_2)$. In addition, we have $|T_1| = m_1$ and $T_2 = |m_2|$. By the safety property of $A\Sigma$, it should then be the case that $T_1 \cap T_2 \neq \emptyset$: a contradiction. □

Lemma A.5. *There is no algorithm that emulates a failure detector of the class $A\Sigma$ in an n-processes anonymous system equipped with a failure detector of the class $A\Sigma'$, for all $n \geq 5$*

Proof. Assume for contradiction that there exists an algorithm $\mathcal{T}$ that emulates the output of a failure detector of the class $A\Sigma$ in a anonymous systems equipped with a failure detector of the class $A\Sigma'$. We construct an execution e_2 of $\mathcal{T}$ in which the safety property of $A\Sigma$ is violated. To that end, we first examine the output of $\mathcal{T}$ in an execution e_1 in which two processes are correct. e_2 is then an execution with no faulty processes, but is indistinguishable from e_1 for $n - 1$ processes. We show that while the output of $\mathcal{T}$ is valid in e_1, this is no longer true in e_2 because more processes are correct. Executions e_1 and e_2 are defined next.

- Execution e_1. In e_1, only two processes that we denote p_1 and p_2 are correct. Other processes are faulty and do not take any step. The output of the underlying failure detector $A\Sigma'$ is $(\text{QUORUMS}_1, \text{LABEL}_1) = (\{\{1,2\}\}, 1)$ at process p_1 and $(\text{QUORUMS}_2, \text{LABEL}_2) = (\{\{1,2\}\}, 2)$ at process p_2. The output of $A\Sigma'$ does not change at each process during e_1. As the only instance of quorum $\{1,2\}$ is $\{p_1, p_2\}$, on can check that the monotony, intersection and liveness properties of $A\Sigma'$ are satisfied in e_1.

  Let us consider the output of the simulated failure detector ASIGMA_1 and ASIGMA_2 at process p_1 and p_2 respectively. As $\mathcal{T}$ correctly emulate a failure detector $A\Sigma$, it follows from Observation A.4 that there exists a time τ_1 and an integer ℓ such that for some $i \in \{1,2\}$, $(\ell, 2) \in \text{ASIGMA}_i^{\tau'}, \forall \tau' \geq \tau$. Without loss of generality, assume that $i = 2$.

- Execution e_2. In e_2, no processes fail. Similarly to execution e_1, the output of the failure detector $A\Sigma'$ is $(\text{QUORUMS}_1, \text{LABEL}_1) = (\{\{1,2\}\}, 1)$ at process p_1 and $(\text{QUORUMS}_j, \text{LABEL}_j) = (\{\{1,2\}\}, 2)$ at each process $p_j \neq p_1$. The output does not change during the execution. One can also check that the output is valid, as the set of instances of the quorum $\{1,2\}$ is $I(\{1,2\}) = \{\{p_1, p_2\}, \{p_1, p_3\}, \ldots, \{p_1, p_n\}\}$. Hence, any two instances intersect. Furthermore, each instance contains only correct processes.

  Up to time τ_1, p_1 and p_2 execute the same steps in the same order as in e_1. The messages that are sent to them by other processes in $\Pi \setminus \{p_1, p_2\}$ are delayed until after τ_1. Each process $p_j \in \Pi \setminus \{p_1, p_2\}$ executes the same steps as p_2 in the same order. In particular, p_j receives only the messages that are sent by p_1. This is possible since p_2 and p_j are initially in the same state (they are anonymous), obtain the same output from the underlying failure detector $A\Sigma'$ and receive in the same order the same messages from p_1.

  Therefore, the state of each process $p_j, j \geq 2$ at time τ_1 is the same as the state of process p_2 at the same time in e_2. Hence, at time τ_1 we have $(\ell, 2) \in \text{ASIGMA}_j$, for all $j \geq 2$. It thus follows that $S(\ell) = \{p_2, \ldots, p_n\}$, and thus, since $n \geq 5$, $S(2)$ contains two non-intersecting sets of size 2. Since $(\ell, 2)$ is contained in the output of the emulated failure of at least two processes, this violates the safety property of the class $A\Sigma$. □

Theorem A.1 then immediately follows from Lemma A.3 and Lemma A.5.

B Missing proofs of Section 4

B.1 Proof of the intersecting-set algorithm

Consider an execution α of the algorithm described in Figure 1 in which every correct process calls $INTSET$.propose$(\cdot)$.

Lemma B.1 (Intersection and validity)**.** *Let V, V' denote two sets returned by* propose$(\cdot)$ *operations.*

1. *(Validity) $\forall v \in V \cup V'$, v is the input value of a* propose$()$ *operation.*
2. *(Intersection) $V \cap V' \neq \emptyset$.*

Proof. (Validity) The property directly follows from the code. Let v denote a value in the set V returned by process p. By the code, this implies that p has received a message $EST(v, \lambda, i)$ broadcast by some process q where v is the input value of the of the propose$()$ operation by q.

(Intersection) Consider set V. Since V is decided, there is a process p that, after exiting the **repeat** loop of lines 2–5, broadcasts a message $DEC(V)$ (on line 6). To exit the loop, p identifies a multi-set

$M = \{(v_1, \lambda_1), \ldots, (v_x, \lambda_x)\}$ of pairs (value,label), and a quorum $Q = \{\ell_1, \ldots, \ell_x\}$. Moreover, M is contained in $rec[i]$, for some $i \geq 1$.

Consider a pair $(v_j, \lambda_j) \in M$. (v_j, λ_j) is added to $rec[i]$ each time p receives a message $EST(v_j, \lambda_j, i)$ (line 7). Such a message is sent by a process that proposes v_j and receives label λ_j from the failure detector in the ith iteration of the **repeat** loop (lines 3 – 4). Since each process broadcasts at most one message $EST(*, *, *)$ in each iteration, there exists a set $I = \{q_1, \ldots, q_x\}$ of x processes and x times $\tau_1, \ldots, \tau_x$ such that q_j proposes v_j and at time τ_j, $\text{LABEL}_{q_j}^{\tau_j} = \lambda_j$.

Since p exits the loop, the test on line 5 is successful. Therefore, for each process $q_j \in I$, $\text{LABEL}_{q_j}^{\tau_j} = \lambda_j \sqsubseteq \ell_j$. I is thus an instance of Q, and V is the set of values proposed by the processes in that instance.

Similarly, there exists a quorum Q' and an instance I' of Q' such that V' is the set of values proposed by the processes in I'. By the intersection property of the failure detector class $A\Sigma'$, $I \cap I' \neq \emptyset$. Therefore, $V \cap V' \neq \emptyset$. □

A call to propose() by a process p *terminates* when p executes the *return* statement on line 8.

Lemma B.2 (Termination)**.** *Every call of* propose() *by every non-faulty process terminates.*

Proof. Let p be a correct process. Assume for contradiction that p never returns from the call to propose$(\cdot)$. This means that no messages $DEC(\cdot)$ are received by p (line 8). Therefore, no correct processes broadcast a messages $DEC(\cdot)$, since such a message will be received by p. Hence, no correct processes including p exit the **repeat** loop at lines 2–5.

By the liveness property of the failure detector class $A\Sigma'$, eventually every quorum output by the failure detector has at least one instance that contains only correct processes. In each iteration of the **repeat** loop (lines 2–5), p obtains a new quorum Q from its failure detector (on line 3), which is then included in the set $\mathcal{Q}$. Therefore, $\mathcal{Q}$ eventually includes a quorum $Q = \{\ell_1, \ldots, \ell_x\}$ for which there exists an instance $I = \{q_1, \ldots, q_x\} \subseteq Correct$. By definition of instance, for each $j, 1 \leq j \leq x$, there exists a time τ_j at which $\text{LABEL}_{q_j}^{\tau_j} \sqsupseteq \ell_j$.

Since the labels output at each process $q_j \in I$ are non decreasing (monotony property of the failure detector class $A\Sigma'$), there exists an iteration $i_j \geq 1$ such that in each iteration $i \geq i_j$, q_j obtains a label $\lambda_j \sqsupseteq \ell_j$. Let $i = \max\{i_j, 1 \leq j \leq x\}$. As no process decides, each correct $q_j \in I$ performs iteration i, and thus broadcasts $EST(v_j, \lambda_j, i)$, where $\lambda_j \sqsupseteq \ell_j$ is the label output by the failure detector at q_j in iteration i. These messages are received by p, and hence eventually $rec_p[i]$ contains the multi-set $M = \{(v_1, \lambda_1), \ldots, (v_x, \lambda_x)\}$ (line 7). Therefore, the test on line 5 is eventually successful, from which we conclude that p exits the **repeat** loop and thus then broadcasts a $DEC()$ message: a contradiction. □

Lemma B.3. *The algorithm described in Figure 1 implements an intersecting-set abstraction.*

Proof. Immediate from Lemma B.1 and Lemma B.2. □

B.2 Proof of the consensus algorithm

Proof of the algorithm We say that a process *enters* x a round when it sets its local variable r to x (line 2). A round x *starts* the first time a process enters round x. We first establish that a correct process that enters a round r eventually enters the next round $r + 1$ unless it decides.

Lemma B.4. *Every correct process that enters round r eventually enters round $r + 1$ or decides.*

Proof. Assume for contradiction that some correct process enters a round but does not enter the next round or decide. Observe that before deciding, any process broadcasts an $DEC(\cdot)$ messages, and upon receiving such a message, any process decides (line 17). Therefore, no process decides in the execution we consider.

Let R be the smallest round at which this occurs. Let p then denote a non-deciding correct process that enters round R but never enters round $R+1$. Observe that as p is correct and does not decide, p does not enter round $R+1$ only if it never exits the inner **repeat** loop of lines 3–8. This implies that p does not receive any message $EST2(*, R)$.

When a process exits the inner **repeat** loop in round R, it first broadcasts a messages $EST2(*, R)$ (line 9). Hence, in round R no correct process exits the inner **repeat** loop. Otherwise, the message $EST2(*, R)$ sent by such a process would be received by p, allowing p to pass the exit condition of the inner loop. As R is the smallest round in which a correct process is stuck, it thus follows that every correct process enters round R.

By definition of the failure detector class $A\mathcal{L}$, there exists a time τ and a set of process $L \subseteq Correct$ such that after τ, the output of the failure detector at process every process $p \in L$ is $(true, |L|)$. In each iteration of the inner **repeat** loop of round R, a process queries its failure detector gets back a pair (b, m) (line 3) and broadcasts a message $EST1(*, R)$ if $b = true$ and it has not done so before (line 4). As $L \subseteq Correct$, and every correct process eventually enters round R, it follows that each process in L broadcasts a message $EST1(*, R)$. Thus, p eventually receives $|L|$ messages $EST1(*, R)$, which are stored in the multi-set rec_p. It thus follows from the code, and the fact that the failure detector output at p is eventually $(true, |L|)$ that p eventually broadcasts a message $EST2(*, R)$ (lines 5–6). Therefore, p eventually receives a message $EST2(*, R)$, and thus exits the inner **repeat** loop: a contradiction. □

Lemma B.5. *Suppose that no processes decide. There exists a round R and a value V such that every message $EST2$ (if any) broadcast in round R carries value V. That is, for every sent message $EST2(r, v)$, $r = R \implies v = V$.*

Proof. Suppose that no processes decide. Assume for contradiction that in every round r, at least two messages $EST2(v_r, r)$ and $EST2(u_r, r)$ are sent with $v_r \neq u_r$.

By definition of the failure detector class $A\mathcal{L}$, there exists a set L of correct processes and a time τ, such that, after time τ, we always have $\text{LEADER}_p = (true, |L|)$ for every process $p \in L$ and $\text{LEADER}_p = (false, *)$ for processes $p \notin L$. Let us consider the first round R that starts after τ. Since no processes decide, every correct process enters that round by Lemma B.4.

Let $p \in L$. Since round R starts after time τ, each time p queries its local failure detector $A\mathcal{L}$ during that round, it obtains the same output $= (true, |L|)$. It thus follows from the code that p broadcasts a message $EST1(*, R)$ (line 4). On the other hand, for any process $q \notin L$, the output of the failure detector is $(false, *)$ in round R. Therefore, no processes other than those in L broadcast a message $EST1$ in round R. Since immediately after broadcasting a message $EST1$, the flag $sent1$ is set to true (line 4), each process in L broadcasts $EST1(*, R)$ exactly once. For each process $p \in L$, let v_p denote the estimate of p when it enters round R. The (multi)-set of messages $EST1$ broadcast during round R is thus $M = \{EST1(v_p, R) : p \in L\}$ which has size $|L|$. Since $L \subseteq Correct$, every message in M is received by each correct process q, i.e. $M \subseteq rec_q$, as each message $EST1$ received by a process q is stored in the local multi-set rec_q (line 18).

In round R, a message $EST2$ may be broadcast by a process q if it gets back $(true, m)$ from the failure detector $A\mathcal{L}$ and rec_q contains a multi-set of at least m messages $EST1$ sent in round R (line 5). In round R, each query to the failure detector $A\mathcal{L}$ by a process $p \in L$ outputs $(true, |L|)$. Moreover, eventually $M \subseteq rec_p$ since p is a correct process, and we have $|M| = |L|$. Therefore, the two conditions

above are eventually satisfied at each process $p \in L$. On the other hand, for a process $q \notin L$, each query to the failure detector $A\mathcal{L}$ returns $(false, *)$ in round R, thus no $EST2(*, R)$ messages are sent by any process $q \notin L$. Suppose that some process $p \in L$ broadcast a message $EST2$. Let V denote the value $\min\{v_p : p \in L\} = \min\{v : EST1(v, R) \in M\}$. Since the multi-set of $EST1$ messages broadcast in round R is M and $|M| = |L|$, the multi-set of messages $EST1(*, R)$ received by p when the condition on line 5 is satisfied is M. By the code, the message $EST2(V, R)$ is then broadcast by p (line 6), since V is the smallest value carried the messages $EST1(*, R)$ received by p. □

Lemma B.6 (Termination)**.** *Every correct process decides.*

Proof. Assume for contradiction that there exists an execution in which a correct process does not decide. Since before deciding, any process broadcasts an $DEC(\cdot)$ messages, and upon receiving such a message, any process decides (line 17), this means that no correct processes decide.

By Lemma B.5, there exists a round R and a value V such that for every message $EST2(v, r)$, if $R = r$ then $V = v$. Since no processes decide, it follows from Lemma B.4 that every correct process enters round $R + 1$. Therefore, messages $EST2(*, R)$ are broadcast in round R. Otherwise, in round R, no process would exit the inner **repeat** loop (line 3–8) and thus no processes would enter round $R + 1$.

Consider the first intersecting-set abstraction of round R ($INTS[R, 1]$, line 9). If value v is proposed to $INTS[R, 1]$, some process has received a message $EST2(v, R)$ carrying that value. Therefore, only value V is proposed to $INTS[R, 1]$. Hence, by the validity property of the abstraction, every invocation of $INTS[R, 1]$.propose() returns the singleton $\{V\}$. By the code (lines 10–11), it then follows that only value V is proposed to the second intersecting-set abstraction of round R ($INTS[R, 2]$), from which we have that every invocation of $INTS[R, 2]$.propose() also returns the singleton $\{V\}$. A process that obtains the singleton $\{V\}$ from $INTS[R, 2]$ broadcasts a $DEC(V)$ messages (line 12).

Let p denote a correct process. By Lemma B.4 p enter round R. Before entering round $R+1$, p performs round R and thus proposes a value and decides in each intersecting-set abstractions of round R. Hence, p broadcasts a message $DEC(*)$. By the code, any correct process receiving this message decides (line 17): a contradiction. □

In each round r, according to the output it obtains from first intersecting-set abstraction, a process p updates its local variable aux (line 10). Let aux_p^r denote the value of the aux variable of process p immediately after the update occurred and let $AUX[x]$ denote the set of values $\{aux_p^r : p$ performs the updates of the local variable aux in

Lemma B.7. *For every round r, $AUX[r] \subseteq \{\perp, v\}$ for some value v.*

Proof. Assume for contradiction there exists two processes p and q such that $aux_p^r = v \neq u = aux_q^r$ with $u, v \neq \perp$. Process p writes v to aux_p if it gets back the singleton $\{v\}$ from the intersecting-set abstraction $INTS[r, 1]$. Similarly, q decides the singleton $\{u\}$ in $INTS[r, 1]$. This contradicts the intersection property of the abstraction. □

Lemma B.8 (Agreement)**.** *No two processes decide distinct values.*

Proof. A process p decides the first time it receives a messages $DEC(v)$. The decided value is then v.

Let R be the first round in which a message $DEC(*)$ is broadcast. Let est_p^r denote the value of the estimate est of p at the end of round r, i.e. after p has possibly written to the local variable est at line 12 or at line 13.

Let p and q denote two processes. We show that $est_p^R = est_q^R$. By definition of R, at least one process s sends a decision message DEC in round R. Let V be the value carried by that message. Note

that $V \neq \perp$. It follows from the code that s decides the singleton $\{V\}$ in the second intersecting-set $INTS[R, 2]$ abstraction of round R (line 12). As each process proposes the value of its aux variable in $INTS[R, 2]$ (line 11), it follows from the validity property of intersecting-sets that $V \in AUX[R]$. Therefore, by Lemma B.7 and the intersection property of intersecting-sets, the possible answers returned by the invocations of $EST[R, 2]$.propose() are $\{V\}$ or $\{V, \perp\}$.

Since a process that decides $\{V\}$ or $\{V, \perp\}$ in $INTS[R, 2]$ sets its estimate to V (lines 12–13), we have $est_p^R = est_q^R = V$. Hence, V is the only estimate that remains at the end of round R. Since the set of estimates at the end of a round is a subset of the estimates at the end of the previous round, no process changes its estimates in any round following R. Finally, note that if p sends $DEC(v)$ in some round r, then $v = est_p^r$. Therefore, for every message $DEC(v)$ that is sent in round R or in a later round, $V = V$. We thus conclude that V is the unique value that is decided. □

Theorem B.9. *The algorithm described 2 solves consensus in anonymous system equipped with a failure detector of the class $A\mathcal{L} \times A\Sigma'$.*

Proof. Validity immediately follows from the code and the validity property of intersecting-sets. The agreement and termination properties follow from Lemma B.6 and Lemma B.8 respectively. □

C Missing proofs of Section 5

C.1 Proof of the reliable broadcast algorithm

Proof of the protocol The main difficulty is to establish that the non-faulty-liveness property is ensured. We prove that the protocol ensures this property in a separate Lemma.

Lemma C.1. *If m is RC_broadcast k times by the correct processes, then for each correct process p, there exists k occurrences of RC_deliver(m) denoted $d_1, \ldots, d_k$ such that the ith call to RC_broadcast(m) by a correct process precedes d_i, for each $i : 1 \leq i \leq k$.*

Proof. Suppose that message m is RC_broadcast at least once by the correct processes. Let τ^k be the time immediately before *RC_broadcast*(m) is invoked by the correct processes for the kth time. Denote by Q^k the set of (correct or faulty) processes that have invoked *RC_broadcast*(m) before or at time τ^k.

Let p be a process. When a process q calls *RC_broadcast*(m), it broadcasts $MSG(m, 1), \ldots, MSG(m, n)$ to each process (lines 1). Depending on whether q is correct or not, each of these messages is eventually received by p. We denote by $ST_p^k[m][i]$ the number of messages $MSG(m, i)$ that are sent to process p by the processes in Q^k. That is, if p does not fail, it will eventually receive $ST_p^k[m][i]$ messages $MSG(m, i)$ that have been sent by processes in Q^k. Since by assumption, each process invokes *RC_broadcast*(m) at most once, $ST_p^k[m][i] \leq |Q^k|$. Moreover, since *RC_broadcast*(m) is invoked $k - 1$ times by the correct processes before τ, $k - 1 \leq ST_p^k[m][i]$.

Let $L_p^k(m) = \max\{i : \forall j, 1 \leq j \leq n + 1 - i, ST_p^k[m][j] \geq i\}$ and let $M^k = \max\{L_p(m) : p \in \Pi\}$. Note that $L_p^k(m) \geq k - 1$, as $ST_p^k[m][i] \geq k - 1$ for every $i, 1 \leq i \leq n$. It thus follows that $M^k \geq k - 1$. Notice also that $M^k < M^{k+1}$ since between τ^k and τ^{k+1}, *RC_broadcast*(m) is called by a correct process $\notin Q^k$. Hence, for each process p and each $i : 1 \leq i \leq n$, $ST_p^{k+1}[m][i] > ST_p^k[m][i]$ and therefore $M^{k+1} > M^k$.

We show that for every correct process p, m is RC_delivered at least $M^k + 1$ times by p, and that the $(M^k + 1)$th occurrence of *RC_deliver*(m) occurs after *RC_broadcast*(m) has been called kth times by the correct processes.

By definition of M^k, there exists a process r such that $ST_r^k[m][j] \geq M^k$, for every $j, 1 \leq j \leq n + 1 - M^k$. In particular, we have $ST_r^k[m][n + 1 - M^k] \geq M^k$ which means that $MSG(m, n + 1 - M^k)$ are sent to r at least M^k times by processes in Q^k. Let p denote a correct process. Since each broadcast of $MSG(m, n + 1 - M^k)$ is preceded by a broadcast of $MSG(m, 1), \ldots, MSG(m, n + 1 - M^k - 1)$ (line 1), M^k messages $MSG(m, j)$ for every $j : 1 \leq j \leq n - M^k$ are sent to p by processes in Q. Since p is correct, each of these messages is eventually received by p. Consider the process q that invokes *RC_broadcast*(m) at time τ^k. This process is correct, and does not belong to Q. Hence, p eventually receives in addition the messages $MSG(m, 1), \ldots, MSG(m, n)$ sent by q. Therefore, we have eventually at process p $RCV_p[m][j] \geq M^k + 1$ for every $j : 1 \leq j \leq n - M^k = (n + 1) - (M^k + 1)$. It thus follows that at process p, eventually $level(m) \geq M^k + 1$, from which we conclude that a message $RELAY(m, M^k + 1)$ is sent to every process (line 4). As p is a correct process, this message is eventually received by every correct process. By the code, m is thus RC_delivered at least $(M^k + 1)$ times by each correct process (line 5–6).

m cannot be RC_delivered $(M^k + 1)$ times before τ^k, since no messages $RELAY(m, \ell)$ with $\ell \geq M^k + 1$ can be sent before time τ^k. This follows from the definition of M^k. Before time τ^k, and for every process p, $RCV[m][j] \leq ST_p^k[m][j]$, as $ST_p^k[m][j]$ is an upper bound on the number of messages $MSG(m, j)$ that have been sent to p before time τ^k. Hence, before τ^k, $level_p(m) \leq \max\{i : \forall j, 1 \leq j \leq n + 1 - jST_p^k[m][j] \geq i\} = L_p^k(m) \leq M^k$, and thus no $RELAY(m, \ell)$ with $\ell \geq M^k + 1$ can be sent before time τ^k.

Suppose that m is RC_broadcast k times in total by the correct processes. Let p be a correct process. We have shown that m is RC_delivered at least $M^k + 1 \geq k$ times by p. Moreover, for each i, $1 \leq i \leq k$, when m is RC_deliveredby p for the $(M^i + 1)$th time, *RC_broadcast*(m) has previously been called i time by the correct processes.

We have shown that if m is RC_broadcast k times by the correct processes, then for every correct p, m is RC_deliveredat least $M^k + 1 \geq k$ times by p. Moreover, the $(M^k + 1)$ occurrence of *RC_deliver*(m) occurs after *RC_broadcast*(m) is called for the kth time by the correct processes. □

Theorem C.2. *The algorithm described in Figure 3 implements a reliable broadcast abstraction in an n-processes anonymous system.*

Proof.

- Integrity. Suppose that m is RC_delivered k times by process p. By the pseudo-code (lines 5–6), p receives a message $RELAY(m, k')$ with $k' \geq k$ before m is RC_delivered for the kth time. Let q denote the process that sends this message. When q sends $RELAY(m, k')$, $level_q(m) = k'$ (lines 3–4). In particular, this implies that q has previously received k' messages $MSG(m, 1)$. As a process sends $MSG(m, 1)$ only immediately after invoking *RC_broadcast*(m) (line 1), $k' (\geq k)$ invocations of *RC_broadcast*(m) occur before m is RC_delivered by p for the kth time.

- No duplicates. Suppose that m is RC_broadcast k times. Notice that a message $MSG(m, 1)$ is sent at most once for each call of *RC_broadcast*(m) (line 1). Therefore, at each process the counter $MSG[m][1]$ is upper bounded by k, since this counter is incremented when a message $MSG(m, 1)$ is received (line 2). It follows that $level(m) \leq k$ (line 3), and thus every message $RELAY(m, k')$ that is sent at line 4 or 5 is such that $k' \leq k$. This concludes the proof of this property, since in order for m to be RC_delivered more than k times by a process, a message $RELAY(m, k')$ with $k' > k$ has to be received by this process (lines 5–6).

- Non-faulty Liveness. Immediately follows from Lemma C.1.

- Faulty Liveness. Suppose that m is RC_delivered$k$ time by some process p. By the code, before m is delivered for the kth time by p, p broadcasts a message $RELAY(m, \ell)$ with $\ell \geq k$ (line 5). Since p

does not fail while broadcasting this message, $RELAY(m, \ell)$ is eventually received by every correct process q. Upon reception of $RELAY(m, \ell)$, m is delivered sufficiently many times to make sure that m is delivered $\ell \geq k$ times (lines 5–6). If a message is RC_delivered k times by p and RC_broadcast $k' \leq k$ time by the correct processes, we consider that only the first $(k - k')$ events ***RC_broadcast***(m) are mapped by κ. □

D System Model for Necessity

Communication Processes communicate with each other through a **reliable causal** broadcast primitive.

Graphs and Sets: Given a graph G, let $V(G)$ and $E(G)$ denote its node set and edge set respectively. Given $v \in V(G)$, the predecessor set of v, $N_G^-(v)$, is the set of vertices having edges going into v. That is, $N_G^-(v) = \{x \in V(G) \mid (x, v) \in E(G)\}$. Define $G[v]$ to be the subgraph of G induced by $N_G^-(v) \cup \{v\}$. Given $U \subseteq V(G)$, denote by $G[U]$ the subgraph of G induced by U. $X \subseteq G$ is a **closed** subgraph of G iff $\forall u \in V(X) : G[u] \subseteq X$. All subgraphs we consider in this paper are closed. Given a multiset S and element x, let $\mathbf{1}_x(S)$ denotes the multiplicity of x in S. $X \overset{C}{\cup} Y$ is equal to $X \cup Y$ if $C = true$, X otherwise.

Algorithms An *algorithm* $\mathcal{A}$ is a collection of deterministic automata $\mathcal{A}_i$, one for each process p_i in the network. Computation proceeds in *steps* of $\mathcal{A}$. In each step of $\mathcal{A}$, process p_i performs *atomically* the following actions: (1) It receives a single message addressed to it by some process p_j and which is present in the buffer, or a null message, denoted $\perp$. In the latter case, we say that p_i receives a message from a fictional process p_{n+1}. (2) It queries and gets a value d from its failure detector module. (3) It changes its state according to $\mathcal{A}_i$ and sends a message to *all* other processes.

Schedules A step of $\mathcal{A}$ is thus identified by the tuple (p_i, p_j, d) where $p_i \in \Pi$ is the process that *takes* the step, $p_j \in \Pi \cup \{p_{n+1}\}$ is the process from which p_i receives a message and d is the output of the failure detector module (possibly equal to $\perp$). A **schedule** S of algorithm A is a (finite or infinite) sequence of steps of A. We assume w.l.o.g that the first step of each process p_i in a schedule is of the form (p_i, p_{n+1}, d), that is, p_i does not receive a message in the first step it executes.

Given a schedule S, denote by $S[k]$ the k-th step in S. Let $S^k(p_i)$ be the k-th step taken by $p_i \in \Pi$ in S if such a step exists, $\perp$ otherwise. Similarly, let $S^k(p_i, p_j)$ be the k-th step of S in which $p_i \in \Pi$ receives a message from $p_j \in (\Pi \cup \{p_{n+1}\}) \setminus \{p_i\}$ if such a step exists, $\perp$ otherwise. If $s = S[k]$, let $\text{TIME}(s) = k$.

Let $Correct(S)$ be the set of process which take infinitely many steps in S. If S is finite, $Correct(S) = \emptyset$.

We define the **relation** $\underset{S}{\rightsquigarrow}$ between steps of S that captures the causality relations between them. It is defined as follows. Given $s = S^k(p_i)$ and $s' = S^{k'}(p_j)$ two steps of S, we write $s \underset{S}{\rightsquigarrow} s'$ (or simply $s \rightsquigarrow s'$) and we say that s *precedes* s' (in S) iff one of the following conditions is satisfied:

(p1) $(p_j = p_i)$ and $(k' > k)$.

(p2) $(s' = S^k(p_j, p_i))$ or $(S^k(p_j, p_i) \rightsquigarrow s')$.

The following property follows from the definition of $\underset{S}{\rightsquigarrow}$:

Property D.1. *For every $s \in S$ with $S^k(p_i) = S^{k'}(p_i, p_j)$ for $p_i, p_j \in \Pi$ with $p_i \neq p_j$ and $k, k' > 0$, for every $s' \in S$:*

$$\begin{aligned}(s' \rightsquigarrow s) \quad \Rightarrow \quad & ((k > 1) \wedge (s' = S^k(p_i, k-1) \vee s' \rightsquigarrow S^k(p_i, k-1))) \\ & \vee \\ & ((p_j \neq p_{n+1}) \wedge (s' = S^{k'}(p_j)))\end{aligned}$$

A schedule S is **well formed** iff:

(w1) For every $s, s' \in S$, if $s \rightsquigarrow s'$ then $\text{TIME}(s) < \text{TIME}(s')$.

(w2) $\underset{S}{\rightsquigarrow}$ is transitively closed (since the underlying broadcast primitive is causal).

(w3) For every $p_i \in Correct(S)$, every $k > 0$, if $S^k(p_i) \neq \perp$, then for every $p_j \in Correct(S) \setminus \{p_i\}$: $S^k(p_j, p_i) \neq \perp$. That is, each sent message is eventually received by all other processes. This follows from the non faulty liveness property of the broadcast service.

(w4) For every $p_i, p_l \in \Pi$, every $k > 0$, if $S^k(p_i, p_l) \neq \perp$, for every $p_j \in Correct(S)$, $S^k(p_j, p_l) \neq \perp$. This follows from the faulty liveness property of the broadcast primitive.

All schedules we consider in this paper are well formed. A schedule S **complies with** $\mathcal{F}$ iff (i) $Correct(S) = Correct$ and (ii) for each step $s = (p_i, -, -) \in S$, it holds that $p_i \notin \mathcal{F}(\text{TIME}(s))$. S **complies with** $\mathcal{H}$ iff for each step $(p_i, -, d) \in S$, it holds that $\mathcal{H}(p_i, \text{TIME}(s)) = d$.

Runs A *configuration* defines the state of each process and each message buffer in the system. An initial configuration consists in an initial state of every automaton A_i and empty message buffers. A *run* of algorithm $\mathcal{A}$ using a failure detector $\mathcal{D}$ is a tuple $R = \langle \mathcal{F}, \mathcal{H}, I, S \rangle$ where $\mathcal{F} \in \xi$ is a failure pattern, $\mathcal{H} \in \mathcal{D}(\mathcal{F})$ is a failure detector history, I is an initial configuration of the network, S is an infinite schedule of $\mathcal{A}$ that complies with both $\mathcal{F}$ and $\mathcal{H}$.

The full information state of p_i at the end its k-th step, denoted by $\text{STATE}(p_i, k)$, is a recursive data structure consisting in a pair of two arrays of zie k, $\mathcal{Q}[1 \ldots k]$ and $\mathcal{R}[1 \ldots k]$. For each $j \in [1, k]$, $\mathcal{Q}[i]$ stores the output of the failure detector module at p_i during its i-th step and $\mathcal{R}[i]$ is the full-information state of the process from which p_i receives a message during this step if any, $\perp$ otherwise.

Formally, assume that $S^k(p_i) = S^{k'}(p_i, p_j) = (p_i, p_j, d)$ with $k \geq k' > 0$. Then,

1. $\text{STATE}(p_i, k).\mathcal{Q}[k] = d$.
2. If $(p_j \neq p_{n+1})$ then $\text{STATE}(p_i, k).\mathcal{R}[k] = \text{STATE}(p_j, k')$. Otherwise, $\text{STATE}(p_i, k).\mathcal{R}[k] = \perp$.
3. If $k > 1$ then $\forall i \in [1, k-1] : (\text{STATE}(p_i, k).\mathcal{Q}[i] = \text{STATE}(p_i, k-1).\mathcal{Q}[i])$ and $(\text{STATE}(p_i, k).\mathcal{R}[i] = \text{STATE}(p_i, k-1).\mathcal{R}[i])$.

For convenience, let $\text{STATE}(p_i, 0) = \perp$. Let $\mathbb{S}$ be the set of all possible states. Given $\texttt{st} \in \mathbb{S}$, let $size(\texttt{st})$ denote the size of the arrays $\texttt{st}.\mathcal{Q}[]$ and $\texttt{st}.\mathcal{R}[]$ if ($\texttt{st} \neq \perp$, 0 otherwise. Define $\texttt{st}^j$ to be the the pair of arrays $(\texttt{st}.\mathcal{Q}[1 \ldots j])$ and $(\texttt{st}.\mathcal{R}[1 \ldots j])$ where Note that if $\texttt{st} = \text{STATE}(p_i, k)$ then $\texttt{st}^j = \text{STATE}(p_i, j)$ for $1 \leq j \leq k$.

Given $\texttt{st}, \texttt{st}' \in \mathbb{S}$, $\texttt{st} \sqsupseteq \texttt{st}'$ iff $\texttt{st}' = \texttt{st}^j$ with $1 \leq j \leq size(\texttt{st})$. $\texttt{st} \sqsupset \texttt{st}'$ iff $(\texttt{st} \sqsupseteq \texttt{st}') \wedge (\texttt{st} \neq \texttt{st}')$. Note that $\sqsupseteq$ is a partial order in $\mathbb{S}$. We define also the *global* order $>$ on $\mathbb{S}$ which corresponds to the lexicographical order $(\mathcal{Q}[1], \mathcal{R}[1], \mathcal{Q}[2], \mathcal{R}[2], \mathcal{Q}[3] \ldots)$. Outputs of the failure detector are totally ordered according to their binary representation.

When a process p_i takes its k-th step in S (*i.e.* $S^k(p_i)$), it computes $\text{STATE}(p_i, k)$ by calling the function $getState()$ described in Figure 5. It consists in a simple full-information protocol. First, p_i queries its failure detector module and stores the obtained output in $\mathcal{Q}[k]$ (line 2). Then it calls RCB-DELIVER() and stores the return value in $\mathcal{R}[k]$ (line 2). If the reception buffers of p_i do not contain a message that is ready

to be delivered to it, RCB-DELIVER() returns $\perp$. Otherwise, it delivers a message previously sent by another process which consists in its full-information state. The current value of $(\mathcal{Q}[1 \dots k], \mathcal{R}[1 \dots k])$ corresponds to $state(p_i, k)$. Before being returned to p_i (line 6), the algorithm has to broadcast it first to other processes (line 5) so they can use it in the computation of their own states.

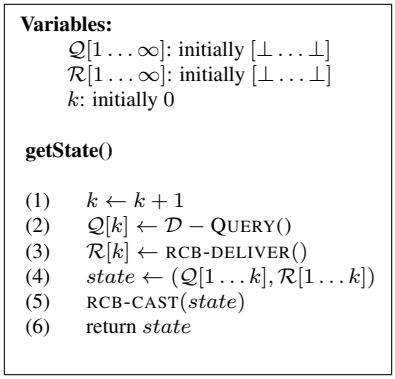

Figure 5: Full Information Protocol

Definition D.2. *Given $X \subseteq \mathcal{G}$ and $p_i \in \Pi$, let $maxState(p_i, X)$ denote the maximal state of p_i in X if any, $\perp$ otherwise. Formally, if p_i has a vertex in X then $maxState(p_i, X) = \underset{\text{STATE}(p_i,k)}{\text{argmax}} \{k \mid V(p_i, k) \in X\}$. Otherwise, $maxState(p_i, X) = \perp$.*

We order the processes according to the decreasing order of $maxState(p_i, X)$, ties are broken using processes identities. Formally, we say that $(p_i \underset{X}{>} p_j)$ iff $(maxState(p_i, X) \sqsupseteq maxState(p_j, X))$ or $(maxState(p_i, X) = maxState(p_j, X)) \wedge (i > j)$. Let $\mathcal{O}(p_i, X)$ denote the order of p_i. That is, $\mathcal{O}(p_i, X) = |\{p_j \in \Pi \mid p_i >_X p_j\}| + 1$.

*Define $\chi[X] : \Pi \mapsto \Pi$ a **permutation** of processes identities which depend on the vertices they have in X. It is such that for every $j \in [1, n] : \chi[X](p_i) = p_j$ iff $\mathcal{O}(p_i, X) = j$. Let π_i^k denotes $\chi[\mathcal{G}[p_i, k]]$ and let $\chi = \chi[\mathcal{G}]$.*

Precedence Graph Steps of S and the precedence relation between them can be represented through a *precedence* graph, denoted $\mathcal{G}$, and defined as follows. To each step $s = S^k(p_i)$ of S corresponds a unique vertex $\Phi(s)$ in $V(\mathcal{G})$ which consists in the pair $(p_i, \text{STATE}(p_i, k))$. $(\Phi(s), \Phi(s')) \in E(\mathcal{G})$ iff $s \underset{S}{\rightsquigarrow} s'$. Let us define the time map: $\text{TIME} : V(\mathcal{G}) \mapsto \mathbb{T}$ such that $\text{TIME}(\Phi(s)) = \text{TIME}(s)$. Let $\mathcal{V}(p_i, k)$ denote $\Phi(S^k(p_i))$, If $S^k(p_i) = \perp$ then $\mathcal{V}(p_i, k) = \perp \notin V(\mathcal{G})$.

Given $v = V(p_i, k) \in V(\mathcal{G})$, let $id(v)$, $ord(v)$, $state(v)$ and $fd(v)$ denote respectively p_i, k, $\text{STATE}(p_i, k)$ and $\text{STATE}(p_i, k).\mathcal{Q}[k]$. Let $\mathcal{G}[p_i, k] = \mathcal{G}[\mathcal{V}(p_i, k)]$. Given $V \subseteq V(\mathcal{G})$, and v a vertex of $V(\mathcal{G})$, denote by $S(v, V)$ the the vertices of in V that have the same state as v. Formally, $S(v, X) = \{v' \in V \mid state(v') = state(v)\}$. Given $X \subseteq \mathcal{G}$, let $S(v, X) = S(v, V(X))$.

Our precedence graph corresponds to the limit DAG in [12] to which converge the local DAGs computed by processes. However, our definition is more abstract and non operational as it depends only on S. We think that our approach gives more intuition about the structure of $\mathcal{G}$ and makes proofs simpler to do.

After proving the properties of $\mathcal{G}$ in the next lemma, we show later that processes can construct ever-growing subgraphs of it.

Lemma D.3. *The following properties of $\mathcal{G}$ follow from its definition.*

(d1) *For each vertex $v = (p_i, st) \in V(\mathcal{G})$, it holds that $p_i \notin \mathcal{F}(\text{TIME}(v))$ and $st.\mathcal{Q}[size(st)] = \mathcal{H}(p_i, \text{TIME}(v))$. Moreover, for every edge $(v, v') \in E(\mathcal{G})$ we have $\text{TIME}(v) < \text{TIME}(v')$.*

(d2) *$\mathcal{G}$ is transitively closed.*

(d3) *If $\mathcal{V}(p_i, k), \mathcal{V}(p_i, k') \in V(\mathcal{G})$ with $k' > k$, then $(\mathcal{V}(p_i, k), \mathcal{V}(p_i, k')) \in E(\mathcal{G})$.*

(d4) *For every finite subgraph X of $\mathcal{G}$, for every correct process p_i, there exists a $k > 0$ such that $\mathcal{V}(p_i, k) \in V(\mathcal{G})$ and for every vertex v of X, $(v, \mathcal{V}(p_i, k)) \in E(\mathcal{G})$.*

Proof.

- (d1) $(p_i \notin \mathcal{F}(\text{TIME}(v)))$ and $(st.\mathcal{Q}[size(st)] = \mathcal{H}(p_i, \text{TIME}(v)))$ follow from the fact that S complies with $\mathcal{F}$ and $\mathcal{H}$ respectively. Let $(v, v') \in E(\mathcal{G})$ with $v = \Phi(s)$ and $v' = \Phi(s')$. This means according to the definition of $E(\mathcal{G})$ that $s \underset{S}{\rightsquigarrow} s'$. But S is well-formed, which implies according to (w1) that $\text{TIME}(s) < \text{TIME}(s')$. Consequently $\text{TIME}(\Phi(s)) < \text{TIME}(\Phi(s'))$. That is, $\text{TIME}(v) < \text{TIME}(v')$.
- (d2) Follows from (w2).
- (d3) Follows from (p1) (Definition of $\underset{S}{\rightsquigarrow}$).
- (d4) Fix $X \subseteq \mathcal{G}$. Let p_i be any correct process. For each $v \in V(X)$, let $\theta(v) = min\{k > 0 | (v, V(p_i, k) \in V(\mathcal{G})\}$. Let $v = V(p_j, k_j)$ for some $p_j \in \Pi$ and $k_j > 0$. The definition of $\theta(v)$ depends on whether $(p_j = p_i)$ or not.

  1. If $(p_j = p_i)$ then $\theta(v) = k_j + 1$. Since $p_j = p_i$ is correct, it holds that $S^{k+1}(p_j) \neq \perp$. Therefore, $\mathcal{V}(p_j, k_j + 1) \in V(\mathcal{G})$ and $(\mathcal{V}(p_j, k_j), \mathcal{V}(p_j, k_j + 1)) \in E(\mathcal{G})$ (according to (d3)).
  2. Assume$(p_j \neq p_i)$. Note that $V(p_j, k_j) = \Phi(S^{k_j}(p_j))$. Since $V(p_j, k_j) \in V(\mathcal{G})$ it follows that $(S^{k_j}(p_j) \neq \perp)$. Since we assumed that p_i is correct, this implies according to property (w3) that $S^{k_j}(p_i, p_j) \neq \perp$. Observe that $S^{k_j}(p_j) \rightsquigarrow S^{k_j}(p_i, p_j)$ (According to (p2)) and $S^{k_j}(p_i, p_j) = S^k(p_i)$ for some $k \geq k_j$. Hence $S^{k_j}(p_j) \rightsquigarrow S^k(p_i)$. Consequently $\mathcal{V}(p_i, k) \in V(\mathcal{G})$ and $(\mathcal{V}(p_j, k_j), \mathcal{V}(p_i, k)) \in E(\mathcal{G})$. In this case, set $\theta(v) = k$.

  Let $K = \underset{v \in V(X)}{\text{argmax}}\ \theta(v)$. By definition of K, it holds that $\mathcal{V}(p_i, K) \in V(\mathcal{G})$ and $\forall v \in V(X) : (v, \mathcal{V}(p_i, K)) \in E(\mathcal{G})$. This proves the claim.

□

Minimal Precedence Graph In *non-anonymous* systems, a process p_i executing its k-th step is able to compute $\mathcal{G}[p_i, k]$ using a trivial adaptation of the DAG construction algorithm of [12]. Unfortunately, the extension of this algorithm to *anonymous* systems is not trivial at all due to the ambiguities that may arise from anonymity. To understand this, consider the following two schedule S and S':

$$S = (p_1, p_{n+1}, d_1)(p_2, p_{n+1}, d_1)(p_1, p_2, d_1')\underline{(p_3, p_1, d_3)}(p_1, p_3, d_1'')$$

$$S' = (p_1, p_{n+1}, d_1)(p_2, p_{n+1}, d_1)(p_1, p_2, d_1')\underline{(p_3, p_2, d_3)}(p_1, p_3, d_1'')$$

The two schedules are similar except that $S[4] \neq S'[4]$. However, due to anonymity, no process can distinguish between them. More precisely, it is impossible to distinguish between the two causality chains

$S[1] \rightsquigarrow S[4] \rightsquigarrow S[5]$ and $S'[2] \rightsquigarrow S'[4] \rightsquigarrow S'[5]$. The difference between them is important as the first chain involves only two processes ($S[1]$ and $S[5]$ are executed by the same process) while the second implicates three different processes.

To cope with this problem, we keep only the precedence relations that do not create ambiguity, and we ignore those that are ambiguous. This is formalized in the following definition:

Definition D.4. *Let $X \subseteq \mathcal{G}$. Define (X), the minimal subgraph of X, using the following three rules. The first one defines its vertex set while the last two define its edge set:*

- ***(r1)*** *$(V(X)) = V(X)$. That is, the set of vertices remains unchanged under the action of operator $\mathcal{M}$.*

For every $u, w \in V(\mathcal{M}(X))$, $(u, w) \in E(\mathcal{M}(X))$ iff at least one of *the following conditions is satisfied:*

- ***(r2)*** *$id(u) = id(w)$. That is, both u and v are created by the same process.*
- ***(r3)*** *$\forall x \in S(u, X) : (x, w) \in E(X)$. That is, all the vertices that have the same state as u have outgoing edges to w in X.*

The following definition formalizes a notion of relabelling of subgraphs of $\mathcal{G}$.

Definition D.5. *Given X a subgraph of $\mathcal{G}$, given $\pi : \Pi \mapsto \Pi$ a permutation of processes identities, Define $\pi(X)$ to be the following graph:*

1. *For each $v = (p_i, s) \in V(X)$, let $\pi(v) = (\pi(p_i), s)$.*
2. *$V(\pi(X)) = \pi(V(X)) = \{\pi(v) \mid v \in V(X)\}$.*
3. *$E(\pi(X)) = \{(\pi(v), \pi(v') \mid (v, v') \in E(X)\}$.*

The following property follows from Definitions D.4 and D.5:

Property D.6. *For every $X \subseteq \mathcal{G}$, for every permutation π: $\mathcal{M}(\pi(X)) = \pi(\mathcal{M}(X))$.*

In the next section we prove the following theorem:

Theorem D.7. *There exists an algorithm that allows each process p_i executing $S^k(p_i)$ to compute $\mathcal{M}(\pi_i^k(\mathcal{G}[p_i, k]))$ (π_i^k is defined in Definition D.2).*

The following theorem proves some properties of minimal precedence graphs:

Theorem D.8. *Let $X \subseteq \mathcal{G}$. $\mathcal{M}(X)$ satisfies the properties (d1) and (d3) of Lemma D.3, let us denote them by (m1) and (m3) in this case.*
Moreover:

(m4) *For every $V \subseteq V(\mathcal{M}(\mathcal{G}))$, for every correct process p_i, there exists $k_i > 0$ such that $\forall k \geq k_i : \forall v \in V : (v, V(p_i, k)) \in E(\mathcal{M}(\mathcal{G}))$.*

(m5) *(i) For every $Y \subseteq X$ with $Y = X[V(Y)]$ it holds that $\mathcal{M}(X)[V(Y)] \subseteq \mathcal{M}(Y)$. (ii) If X is closed than $\mathcal{M}(\mathcal{G})[V(X)] \subseteq \mathcal{M}(X)$.*

Proof.
- (m1) & (m3): Follow from rules (r1) and (r3) of Definition D.4.

- (m4): Fix $V \subseteq V(\mathcal{M}(\mathcal{G}))$. Take $V' \supseteq V$ such that $\forall v \in V : S(v, \mathcal{G}) \subseteq V'$ and V' is minimal. According to property (d4) of $\mathcal{G}$, $\exists k_i > 0$ such that $V(p_i, k_i) \in V(\mathcal{G})$ and $\forall v \in V' : (v, V(p_i, k_i)) \in E(\mathcal{G})$. Since p_i is correct it holds that $\forall k > k_i : V(p_i, k) \in V(\mathcal{G})$. Hence, according to property (d3) of $\mathcal{G}$, it holds that $\forall k > k_i : (V(p_i, k_i), V(p_i, k)) \in E(\mathcal{G})$. But we showed that $\forall v \in V' : (v, V(p_i, k_i)) \in E(\mathcal{G})$. By transitivity (d2) of $\mathcal{G}$, we get: $\forall k \geq k_i : \forall v \in V' : (v, V(p_i, k)) \in E(\mathcal{G})$. Since $\forall v \in V : S(v, \mathcal{G}) \subseteq V'$, It follows that:

  $$\forall k \geq k_i : \forall v \in V : \forall x \in S(v, \mathcal{G}) : (x, V(p_i, k)) \in E(\mathcal{G})$$

  This implies according to rule (r3) of Definition D.4 that: $\forall k \geq k_i : \forall v \in V : (v, V(p_i, k)) \in E(\mathcal{M}(\mathcal{G}))$.

- (m5): (ii) is a simple corollary of (i). In the following we prove (i). Note that $V(\mathcal{M}(X)[V(Y)]) = V(\mathcal{M}(Y)) = V(Y)$. Hence it remains to show that $E(\mathcal{M}(X)[V(Y)]) \subseteq E(\mathcal{M}(Y))$.

  Let $u, w \in V(Y)$ with $(u, w) \in E(\mathcal{M}(X))$. We show that $(u, x) \in E(\mathcal{M}(Y))$. The fact that $(u, w) \in E(\mathcal{M}(X))$ implies that either (i) $(id(u) = id(w))$ or (ii) $\forall x \in S(u, X) : (x, w) \in E(X)$. Since $Y \subseteq X$, it holds that $S(u, Y) \subseteq S(u, X)$. Hence (ii) implies that $\forall x \in S(u, Y) : (x, w) \in E(X)$. But $Y = X[V(Y)]$ and $u, w \in V(Y)$. It follows that (ii') $\forall x \in S(u, Y) : (x, w) \in E(Y)$. From (i) and (ii') we conclude that $(u, w) \in E(\mathcal{M}(Y))$. This proves the claim.

□

Property D.9. *Two important properties of $\mathcal{M}$ are absent in $\mathcal{M}(\mathcal{G})$: it is not transitively closed and $\mathcal{M}(\mathcal{G}[v])$ is not necessarily a subgraph of $\mathcal{M}(\mathcal{G})$ (while $\mathcal{G}[v] \subset \mathcal{G}$). We illustrate this with the following example. Consider the graph X defined as follows:*

- $V(X) = \{v_1 = (p_1, x), v_1' = (p_1, x'), v_2 = (p_2, x), v_3 = (p_3, w), v_3' = (p_3, w')\}$
- $E(X) = \{(v_1, v_1'), (v_3, v_3'), (v_1, v_3), (v_1', v_3), (v_1, v_3'), (v_1', v_3'), (v_2, v_3')\}$.

Hence:

- $E(\mathcal{M}(X)) = \{(v_1, v_1'), (v_1', v_3), (v_1, v_3'), (v_1', v_3'), (v_2, v_3'), (v_3, v_3')\}$
- $E(\mathcal{M}(X[v_3])) = \{(v_1, v_1'), (v_1', v_3), (v_1, v_3)\}$
- *Observe that X is transitively closed while $\mathcal{M}(X)$ is not because $(v_1, v_3) \notin E(\mathcal{M}(X))$ whereas $(v_1, v_1'), (v_1', v_3) \in E(\mathcal{M}(X))$.*
- *Note that $E(\mathcal{M}(X[v_3])) \not\subseteq E(\mathcal{M}(X))$ since $(v_1, v_3) \in E(\mathcal{M}(X[v_3]))$ while $(v_1, v_3) \notin E(\mathcal{M}(X))$.*

E Minimal Precedence Graphs: Construction & Properties

In this section we show how processes construct minimal precedence graphs; then we prove some their properties.

E.1 Contruction of Minimal Graphs

This section is devoted the the proof of the following theorem which is an equivalent formulation of Theorem D.7:

Theorem E.1. *Given* STATE(p,k) *for* $p \in \Pi, k > 0$ *and* $S^k(p) \neq \perp$*, it is possible to construct* $\mathcal{M}(\pi_p^k(\mathcal{G}[p,k]))$.

Fix $p \in \Pi, k > 0$ such that $S^k(p) \neq \perp$. Let Q be the set of processes having at least one vertex in $\mathcal{G}[p,k]$. For simplicity, we shall drop the subscript p and the exponent k from π_p^k. If $\pi(p_i) = p_j$, we may refer to p_i as p_{ij}.

Let V_j be the subset of vertices in $V(\mathcal{G}[p,k))$ that were created by p_{ij}. That is $V_j = \{v : (v \in V(\mathcal{G}[p,k])) \wedge (id(v) = p_{ij})\}$. Note that $V(\mathcal{G}[p,k]) = \bigcup_{j=1}^{|Q|} V_j$.

Lemma E.2. *Given* STATE(p,k) *for* $p \in \Pi, k > 0$ *and* $S^k(p) \neq \perp$*, it is possible to construct* $V(\mathcal{M}(\pi(\mathcal{G}[p,k]))))$.

Proof. According to Definition D.5, $V(\pi(\mathcal{G}[p,k])) = \pi(\bigcup_{j=1}^{|Q|} V_j)$. Since $V(\mathcal{G}[p,k]) = \bigcup_{j=1}^{|Q|} V_j$, it follows that $V(\pi(\mathcal{G}[p,k])) = \pi(\bigcup_{j=1}^{|Q|} V_j)$. Thus, $V(\pi(\mathcal{G}[p,k])) = \bigcup_{j=1}^{|Q|} \pi(V_j)$. But Property (r1) of Definition D.4 says that $V(\mathcal{M}(\pi(\mathcal{G}[p,k])))) = V(\pi(\mathcal{G}[p,k]))$. Hence, $V(\mathcal{M}(\pi(\mathcal{G}[p,k])))) = \bigcup_{j=1}^{|Q|} \pi(V_j)$. To finish the proof it suffices to show how to construct V_j for every $j \in [1, |Q|]$. This is done in Lemma E.6. □

Definition E.3 (bagStates)**.** *Given a state* $st \in \mathbb{S}$*, let* $bagStates(st)$ *denote the multiset of all states appearing in* $(st.\mathcal{R}[])$ *together with all prefixes of* st*. Formally,* $bagStates(st)$ *is the multiset* $\{(s,m) : (s \in \mathbb{S}) \wedge (m > 0) \wedge (m = (|\{1 \leq k \leq size(st) : st.\mathcal{R}[k] = s\}| + |\{1 \leq k \leq size(st) : st^k = s\}|))\}$

Similarly, given V *a subset of* $V(\mathcal{G})$*, let* $bagStates(V)$ *denote the multiset of all states of the vertices of* V*. That is,* $bagStates(V) = \{(s,m) : (s \in \mathbb{S}) \wedge (m > 0) \wedge (m = |\{v \in V : state(v) = s)\}|)\}$

If X *a subgraph of* $\mathcal{G}$*, then let* $bagStates(X) = bagStates(V(X))$.

Lemma E.4. $bagStates(\mathcal{G}[p,k]) = bagStates($STATE$(p,k))$ *for every* $p \in \Pi$ *and every* $k > 0$ *such that* $S^k(p) \neq \perp$

Proof. It follows from Property D.1 and the definition of $E(\mathcal{G})$ that:

$$N_{\mathcal{G}}^{-}(\phi(S^k(p))) = \begin{cases} N_{\mathcal{G}}^{-}(\phi(S^{k-1}(p))) \cup \{\phi(S^{k'}(p_j)\} & \textbf{if} \quad (k > 1) \wedge (p_j \neq p_{n+1}) \\ N_{\mathcal{G}}^{-}(\phi(S^{k-1}(p))) & \textbf{if} \quad (k > 1) \wedge (p_j = p_{n+1}) \\ \emptyset & \textbf{if} \quad (k = 1) \end{cases}$$

Hence,

$$V(\mathcal{G}[p,k]) = \{V[p,k]\} \overset{k>1}{\cup} V(\mathcal{G}[p,k-1]) \overset{p_j \neq p_{n+1}}{\cup} \{V(p_j,k')\}$$

This implies that:

$$bagStates(\mathcal{G}[p,k]) = \{\text{STATE}(p_i,k)\} \overset{k>1}{\underset{m}{\cup}} bagStates(\mathcal{G}[p_i,k-1]) \overset{p_j \neq p_{n+1}}{\cup} \{\text{STATE}(p_j,k')\} \quad (1)$$

On the other hand, by definition of $bagStates()$ we have:

$$bagStates(\text{STATE}[p,k]) = \{\text{STATE}[p,k]\} \overset{k>1}{\underset{m}{\cup}} bagStates(\text{STATE}(p,k-1)) \overset{\text{STATE}[p,k].\mathcal{R}[k] \neq \perp}{\underset{m}{\cup}} \{\text{STATE}[p,k].\mathcal{R}[k]\}$$

But by definition of full-information states, $(\text{STATE}[p_i,k].\mathcal{R}[k] \neq \perp)$ iff $(p_j \neq \perp)$ and in this case, $\text{STATE}[p,k].\mathcal{R}[k] = \text{STATE}(p_j,k')$. Hence,

$$bagStates(\text{STATE}[p,k]) = \{\text{STATE}[p,k]\} \overset{k>1}{\underset{m}{\cup}} bagStates(\text{STATE}(p,k-1)) \overset{p_j \neq p_{n+1}}{\cup} \{\text{STATE}(p_j,k')\} \quad (2)$$

From equations (1) and (2), we conclude that (i) $bagStates(\mathcal{G}[p,1]) = bagStates(\text{STATE}(p,1))$ and (ii) $bagStates(\mathcal{G}[p,k]) = bagStates(\text{STATE}[p,k])$ provided that $bagStates(\mathcal{G}[p,k-1]) = bagStates(\text{STATE}(p,k-1))$. This proves the lemma by induction. □

In the following lemma, we show how to construct $bagStates(V_j), j \in [1,|Q|]$ in a recursive way:

Lemma E.5. *For every $j \in [1,|Q|]$,*

$$bagStates(V_j) = \{st^x \mid (st = \max(bagStates(V) \setminus \bigcup_{y=1}^{j-1} bagStates(V_y))) \wedge (x \in [1,size(st)])\}$$

Proof. Fix $j \in [1,|Q|]$. Let $st = maxState(p_{ij},\mathcal{G}[p,k])$. That is, $st = \text{STATE}(p_{ij},k_{ij})$. Note that $bagStates(V_j) = \{\text{STATE}(p_{ij},x) \mid x \in [1,k_{ij}]\}$. But for every $x \in [1,k_{ij}]$, it holds that $\text{STATE}(p_{ij},x) = (\text{STATE}(p_{ij},k_{ij}))^x = st^x$. Consequently,

$$bagStates(V_j) = \{st^x \mid (x \in [1,k_{ij}]) \wedge (k_{ij} = size(st))\}$$

Thus, to prove the lemma it suffices to show that $st = maxState(p_{ij},\mathcal{G}[p,k]) = max(bagStates(V) \setminus \bigcup_{y=1}^{j-1} bagStates(V_y))$.

Note that $(bagStates(V) \setminus \bigcup_{y=1}^{j-1} bagStates(V_y)) = \bigcup_{y=j}^{|Q|} bagStates(V_y)$. Hence,

$$\begin{aligned} max(bagStates(V) \setminus \bigcup_{y=1}^{j-1} bagStates(V_y)) &= max(\bigcup_{y=j}^{|Q|} bagStates(V_y)) \\ &= max(\{max(bagStates(V_y)) \mid j \leq y \leq |Q|\}) \\ &= max(\{maxState(p_{iy},\mathcal{G}[p,k]) \mid j \leq y \leq |Q|\}) \end{aligned}$$

But by definition of p_{ij}, for every $y \in [j,|Q|] : maxState(p_{ij},\mathcal{G}[p,k]) \geq maxState(p_{iy},\mathcal{G}[p,k])$. It follows that:

$$max(bagStates(V) \setminus \bigcup_{y=1}^{j-1} bagStates(V_y)) = maxState(p_{ij},\mathcal{G}[p,k]) = st$$

This proves the lemma.

□

The following lemma gives a recursive formula to compute $\pi(V_j)$ (provided that $V_y, 1 \leq y < j$ are known).

Lemma E.6. *$\pi(V_j) = \{(p_j, st) \mid st \in bagStates(V_j)\}$ where $bagStates(V_j)$ is computed recursively as indicated in Lemma E.5.*

Proof. Follows from the fact that $V_j = \{(p_{ij}, st) \mid st \in bagStates(V_j)\}$ and $\pi(p_{ij}) = p_j$ by definition of p_{ij}. □

Lemma E.7. *Given S a multiset, let $\mathbf{1}_x S$ denote the multiplicity of x in S.*

Given $u = (p_u, stateU)$ and $v = (p_v, stateV)$ any two vertices of $V(\mathcal{M}(\pi(\mathcal{G}[p,k])))$, $(u,v) \in E(\mathcal{M}(\pi(\mathcal{G}[p,k])))$ iff at least one of the following conditions is satisfied:

(1) $(p_u = p_v) \wedge (stateU \sqsubset stateV)$.

(2) $\mathbf{1}_{stateU} bagStates(stateV) = \mathbf{1}_{stateU} bagStates(\text{STATE}(p,k)) \neq 0$.

Proof. Fix $u = (p_u, stateU)$ and $v = (p_v, stateV)$ two vertices of $V(\mathcal{M}(\pi(\mathcal{G}[p,k]))) = V(\pi(\mathcal{G}[p,k]))$. According to Definition D.4, $(u,v) \in E(\mathcal{M}(\pi(\mathcal{G}[p,k])))$ iff (i) $(u,v) \in E(\pi(\mathcal{G}[p,k]))$ and either (ii.1) $p_u = p_v$ or (ii.2) $\forall x \in V(\mathcal{M}(\pi(\mathcal{G}[p,k]))) : (state(x) = stateU) \Rightarrow (x,v) \in E(\pi(\mathcal{G}[p,k]))$.

Note that $(i) \wedge (ii.1)$ is equivalent to condition (1). Hence, to prove the lemma it suffices to show that $(i) \wedge (ii.2)$ is equivalent to condition (2).

$$
\begin{aligned}
(i) \wedge (ii.2) \quad &\Leftrightarrow \quad (u \in V(\pi(\mathcal{G})[v])) \wedge (\forall x \in S(u, V(\mathcal{M}(\pi(\mathcal{G}[p,k])))) : x \in V(\pi(\mathcal{G})[v])) \\
&\Leftrightarrow \quad (stateU \in bagStates(\pi(\mathcal{G})[v])) \wedge (\mathbf{1}_{stateU} bagStates(\pi(\mathcal{G}[p,k])) \geq \mathbf{1}_{stateU} bagStates(\pi(\mathcal{G})[v])) \\
&\Leftrightarrow \quad (stateU \in bagStates(\pi(\mathcal{G})[v])) \wedge (\mathbf{1}_{stateU} bagStates(\pi(\mathcal{G}[p,k])) = \mathbf{1}_{stateU} bagStates(\pi(\mathcal{G})[v])) \\
&\qquad \textit{Proof.} \text{ Since } v \in \pi(\mathcal{G}[p,k]) \text{ it follows that } \pi(\mathcal{G})[v] \subseteq \pi(\mathcal{G}[p,k]). \\
&\qquad \text{Hence } bagStates(\pi(\mathcal{G})[v]) \subseteq bagStates(\pi(\mathcal{G}[p,k])). \\
&\Leftrightarrow \quad (\mathbf{1}_{stateU} bagStates(\pi(\mathcal{G}[p,k])) = \mathbf{1}_{stateU} bagStates(\pi(\mathcal{G})[v]) \neq 0) \\
&\Leftrightarrow \quad (\mathbf{1}_{stateU} bagStates(\pi(\mathcal{G}[p,k])) = \mathbf{1}_{stateU} bagStates(stateV) \neq 0) \\
&\qquad \textit{Proof.} \text{ According to Lemma E.4 } bagStates(\pi(\mathcal{G})[v]) = bagState(state(v)) = bagState(stateV) \\
&\Leftrightarrow \quad (\mathbf{1}_{stateU} bagStates(\text{STATE}(p,k)) = \mathbf{1}_{stateU} bagStates(stateV) \neq 0) \\
&\qquad \textit{Proof.} \text{ Note that } bagStates(\pi(\mathcal{G}[p,k])) = bagState(\mathcal{G}[p,k]). \\
&\qquad \text{According to Lemma E.4 } bagStates(\mathcal{G}[p,k]) = \text{STATE}(p,k).
\end{aligned}
$$

□

Proof of Theorem E.1. The theorem follows from Lemmas E.2 and E.7. □

E.2 Properties of Minimal Graphs

The following propery is a consequence of Theorem E.1:

Lemma E.8. *Let $X \subseteq \mathcal{G}$. $\forall (u,v) \in E(\mathcal{M}(X))$, for every permutation of processes identities π, if $\pi(u), \pi(v) \in V(\mathcal{M}(X))$, then $(\pi(u), \pi(v)) \in E(\mathcal{M}(X))$.*

Proof. Let $u = (p_u, s_u), v = (p_v, s_v)$. We distinguish between the case in which (i) ($p_u = p_v$) or (ii) not.

- ($p_u = p_v$). Since $(u, v) \in E(\mathcal{M}(X))$ and $E(\mathcal{M}(X)) \subseteq E(X)$ it follows that $(u, v) \in E(X)$. This combined with the assumption that $id(u) = id(v)$ implies that $s_u \sqsubset s_v$. Note that $\pi(u), \pi(v) \in V(\mathcal{M}(X)) = V(X)$ and $\pi(u) = (\pi(p_u), s_u)$ and $\pi(v) = (\pi(p_u), s_v)$. Since $s_u \sqsubset s_v$, it follows according to property (d2) of $\mathcal{G}$ that $(\pi(u), \pi(v)) \in E(X)$. By applying rule (r2) of Definition D.4, we conclude that $(\pi(u), \pi(v)) \in E(\mathcal{M}(X))$.

- ($p_u = p_v$). Let $v' = \pi(v)$. Note that $state(v') = state(v)$. It follows according to Theorem E.1 that $\mathcal{M}(\mathcal{G}[v']) = \phi(\mathcal{M}(\mathcal{G}[v]))$ where ϕ is any permutation of processes identities such that $\phi(v) = \pi(v) = v'$. (Note that $\phi(u)$ is not necessarily equal to $\pi(u)$).

  Since $(u, v) \in E(\mathcal{M}(X))$ and $id(u) \neq id(v)$ it follows according rule (r3) of Definition D.4 that $\forall x \in S(u, X) : (x, v) \in E(X)$. By isomorphism, it holds also that $\forall x \in S(u, X) : (x, v') \in E(X)$. Hence, according to rule (r3) of Definition D.4, we have: $\forall x \in S(u, X) : (x, v') \in E(\mathcal{M}(X))$. But $\pi(u) \in S(u, X) = S(u, \mathcal{M}(X))$. Hence $(\pi(u), v') \in E(\mathcal{M}(X))$. Remember that we have set $v' = \pi(v)$. Thus $(\pi(u), \pi(v)) \in E(\mathcal{M}(X))$.

□

Lemma E.9. *Let Y, X two subgraphs of $\mathcal{G}$ such that $X \subseteq Y \subseteq \mathcal{G}$ and $Y = X[V(Y)]$. If $\forall v \in V(X) : S(v, X) = S(v, Y)$ then $\mathcal{M}(Y) = \mathcal{M}(X)[V(Y)]$.*

Proof. According to Property (m4) of Theorem D.8, it holds that $\mathcal{M}(X)[V(Y)] \subseteq \mathcal{M}(Y)$. To prove the lemma we have to show that $\mathcal{M}(Y) \subseteq \mathcal{M}(X)[V(Y)]$. Note that $V(\mathcal{M}(Y)) = V(\mathcal{M}(X)[V(Y)]) = V(Y)$. Hence, it suffices to prove that $E(\mathcal{M}(Y)) \subseteq E(\mathcal{M}(X)[V(Y)])$.

Let $(u, v) \in E(\mathcal{M}(Y))$. We show that $(u, v) \in E(\mathcal{M}(X)[V(Y)])$.

$$
\begin{array}{rcl}
(u, v) \in E(\mathcal{M}(Y)) & \Rightarrow & ((id(u) = id(v)) \vee (\forall w \in S(u, Y) : (w, v) \in E(Y)) \\
& \Rightarrow & ((id(u) = id(v)) \vee (\forall w \in S(u, X) : (w, v) \in E(Y)) \\
& & \textit{Proof.} \text{ We assumed that } S(u, X) = S(u, Y) \\
& \Rightarrow & ((id(u) = id(v)) \vee (\forall w \in S(u, X) : (w, v) \in E(X)) \\
& & \textit{Proof.} \text{ By definition of } Y : E(Y) \subseteq E(X) \\
& \Rightarrow & (u, v) \in E(\mathcal{M}(X)) \\
& \Rightarrow & (u, v) \in E(\mathcal{M}(X)[V(Y)]) \\
& & \textit{Proof.} \ u, v \in V(Y)
\end{array}
$$

□

Corollary E.10. *Let X a (closed) subgraph of $\mathcal{G}$. If $\forall v \in V(X) : S(v, X) = S(v, \mathcal{G})$ then $\mathcal{M}(X) = \mathcal{M}(\mathcal{G})[V(X)]$.*

F Spans

In this section we introduce the notion of spans and we prove two important properties about them.

Definition F.1. *Let $X \subseteq \mathcal{G}$. A set of vertices $U \subseteq V(\mathcal{M}(X))$ is a **span** in $\mathcal{M}(X)$ iff: for every $X \subseteq \mathcal{G}$, for every $X' \subseteq \mathcal{G}$ with $X' = \pi(X)$ for some permutation π, $\exists x \in U : \exists v' \in \pi(U) : id(x) = id(x')$. Note that $\pi(U)$ is a span also.*

Let $\text{SPAN}(\mathcal{M}(X)) = \{U \subseteq V(\mathcal{M}(X)) \mid U \text{ is a span in } \mathcal{M}(X)\}$.

*Let $v_1, v_2 \in V(G)$ with $id(v_1) = id(v_2)$. We say that v_2 U-**validates** v_1 iff $\exists U \in \text{SPAN}(\mathcal{M}(\mathcal{G}[v_2]))$ such that: $\forall u \in U : (v_1, u) \in E(\mathcal{M}(\mathcal{G}[v_2])) \wedge (u, v_2) \in E(\mathcal{M}(\mathcal{G}[v_2]))$.*

*Let $X \subseteq \mathcal{G}$ a subgraph of $\mathcal{G}$. We say that $v_2 \in V(X)$ is U-**stable in** $\mathcal{M}(X)$ iff $\exists U \in \text{SPAN}(\mathcal{M}(\mathcal{G}[v_2])) : \exists v_1 \in V(\mathcal{M}(X))$ such that: (i) v_2 U-validates v_1 and (ii) $|S(v_2, \mathcal{M}(X))| = |S(v_1, \mathcal{M}(X))|$.*

In the following sections, we give two examples of spans in Lemmas G.4 and H.1.

Lemma F.2. *For every $v_1, v_2 \in V(\mathcal{G})$ with $id(v_1) = id(v_2)$, if v_2 validates v_1 then:*

$$\forall v_2' \in S(v_2, \mathcal{G}) : \exists v_1' \in S(v_1, \mathcal{M}(\mathcal{G}[v_2])) \text{ with } id(v_1') = id(v_2')$$

Proof. Assume that $v_1 = V(p, k_1)$ and $v_2 = V(p, k_2)$ with $p \in \Pi$ and $0 < k_1 < k_2$. Assume that v_2 U-validates u_1 for some $U \in \text{SPAN}(\mathcal{M}(\mathcal{G}[v_2]))$.

Assume *towards contradiction* that $\exists v_2' \in S(v_2, \mathcal{G})$ and $\nexists v \in S(v_1, \mathcal{M}(\mathcal{G}[v_2]))$ with $id(v) = id(v_2')$. Since $v_2 = V(p, k_2)$ and $v_2' \in S(v_2, \mathcal{G})$ then $v_2' = V(p', k_2)$ for some $p' \in \Pi$. Let $v_1' = V(p', k_2)$. Since $v_2' \in S(v_2, \mathcal{G})$ it follows that $v_1' \in S(v_1, \mathcal{G})$. But the contradiction assumption says that $\nexists v \in S(v_1, \mathcal{M}(\mathcal{G}[v_2]))$ with $id(v) = id(v_2')$. Since $id(v_1') = id(v_2')$, it follows that $v_1' \notin V(\mathcal{M}(\mathcal{G}[v_2]))$. But $V(\mathcal{M}(\mathcal{G}[v_2])) = V(\mathcal{G}[v_2]))$ according to rule (r1) of Definition D.4. Hence, $v_1' \notin V(\mathcal{G}[v_2])$ which means that $(v_1', v_2) \notin E(\mathcal{G})$.

According to Theorem E.1 and Property D.6, $state(v_2)$ determines $\pi_p(\mathcal{M}(\mathcal{G}[v_2]))$ for some permutation π_p. Hence, the fact $state(v_2) = state(v_2')$ implies that $\mathcal{M}(\mathcal{G}[v_2']) = \pi(\mathcal{M}(\mathcal{G}[v_2]))$ for some permutation π which means that $\mathcal{M}(\mathcal{G}[v_2'])$ and $\mathcal{M}(\mathcal{G}[v_2])$ are *isomorphic*. Note that $v_1' = \pi(v_1)$ and $v_2' = \pi(v_2)$. Since $U \in \text{SPAN}(\mathcal{M}(\mathcal{G}[v_2]))$ then $\pi(U) \in \text{SPAN}(\pi(\mathcal{M}(\mathcal{G}[v_2])))$. Hence $\exists x \in U : \exists x' \in \pi(U)$ such that $id(x) = id(x')$. Let $U' = \pi(U)$. Note that v_2' U'-validates v_1'.

Let us prove the following four properties:

- (C1): $(v_1, x) \in E(\mathcal{G})$
- (C2): $(v_1', x') \in E(\mathcal{G})$
- (C3): $(x, v_2) \in E(\mathcal{G})$
- (C4): $(x', v_2') \in E(\mathcal{G})$

Proof. Since v_2 U-validates v_1 and $x \in U$, it follows that $(v_1, x) \in E(\mathcal{M}(\mathcal{G}[v_2]))$ and $(x, v_2) \in E(\mathcal{M}(\mathcal{G}[v_2]))$. Hence, $(v_1, x), (x, v_2) \in E(\mathcal{G})$. This proves both (C1) and (C3). Properties (C2) and (C4) can be proved using a similar argument.

Let $x = V(p_x, k_x)$ and $x' = V(p_x, k_x')$. We prove that $(x, x') \in E(\mathcal{G})$. For this it suffices to show that $k_x' > k_x$ (Property (d2) of precedence graphs). Assume towards contradiction that $k_x \geq k_x'$. This means that either $(x = x')$ or $(x', x) \in E(\mathcal{G})$. Combining this with properties (C2) and (C3), it follows by the transitive closure property of $\mathcal{G}$ that $(v_1', v_2) \in E(\mathcal{G})$. But we assumed that $(v_1', v_2) \notin E(\mathcal{G})$, contradiction. This proves that $k_x' > k_x$. Thus $(x, x') \in E(\mathcal{G})$.

To finish the proof, we divide the analysis into two sub-cases depending on whether $(p_x = p)$ or not and we show that both of them lead to a contradiction.

1. $(p_x = p)$. In this case, $x = V(p, k_x)$ and $x' = V(p, k'_x)$. We have two cases: either (i) $k'_x \geq k_2$ or (ii) $k'_x < k_2$. (i) Consider the first case ($k'_x \geq k_2$). Since $v_2 = V(p, k_2)$ it follows that either $v_2 = x'$ or $(v_2, x') \in E(G)$ (by property (d2) of precedence graphs). Combining this with (C4) we conclude in both cases, according to the transitive closure property of $\mathcal{G}$, that $(v_2, v'_2) \in E(\mathcal{G})$. But this contradicts the fact that $\mathcal{M}(\mathcal{G}[v_2])$ and $\mathcal{M}(\mathcal{G}[v'_2])$ are isomorphic. (ii) In the second case ($k'_x < k_2$), since $v_2 = V(p, k_2)$ it follows that $(x', v_2) \in E(G)$ (by property (d2) of precedence graphs). Combining this with (C2) we conclude that $(v'_1, v_2) \in E(G)$. Contradiction as we assumed above that $(v'_1, v_2) \notin E(G)$. Hence, both cases (i) and (ii) lead to a contradiction.

2. $(p_x \neq p)$. In this case we have $id(v_1) \neq id(x)$ as $id(x) = p_x$ and $id(v_1) = p$. Since $x \in U$ and v_2 U-validates v_1, it follows that $(v_1, x) \in E(\mathcal{M}(\mathcal{G}[v_2]))$. But $id(v_1) \neq id(x)$. Thus, according to rule (r3) of the definition of minimal graphs (Definition D.4) we have $\forall w \in S(v_1, \mathcal{G}[v_2]) : (w, x) \in E(\mathcal{G}[v_2])$. Combining this with the fact that $(x, x') \in E(\mathcal{G})$ and (C4) we conclude, according to the transitive closure of $\mathcal{G}$, that $\forall w \in S(v_1, \mathcal{G}[v_2]) : (w, v'_2) \in E(\mathcal{G})$. Therefore, $S(v_1, \mathcal{G}[v_2]) \subseteq V(\mathcal{G}[v'_2])$. But $V(\mathcal{G}[v_2]) = V(\mathcal{M}(\mathcal{G}[v_2]))$ and $V(\mathcal{G}[v'_2]) = V(m(\mathcal{G}[v'_2]))$ (rule (r1) of Definition D.4). Hence, $S(v_1, \mathcal{M}(\mathcal{G}[v_2])) \subseteq V(\mathcal{M}(\mathcal{G}[v'_2]))$. It follows that $S(v_1, \mathcal{M}(\mathcal{G}[v_2])) \subseteq S(v_1, \mathcal{M}(\mathcal{G}[v'_2]))$. But the fact that $\mathcal{M}(\mathcal{G}[v_2])$ and $\mathcal{M}(\mathcal{G}[v'_2])$ are isomorphic implies that $|S(v, \mathcal{M}(\mathcal{G}[v_2]))| = |S(v, \mathcal{M}(\mathcal{G}[v'_2]))|$. Hence $S(v_1, \mathcal{M}(\mathcal{G}[v_2])) = S(v_1, \mathcal{M}(\mathcal{G}[v'_2]))$.

   Since $v'_1 = \pi(v_1)$ and $v'_1 \in V(\mathcal{G}[v'_2]) = V(\mathcal{M}(\mathcal{G}[v'_2]))$, it follows that $v'_1 \in S(v_1, \mathcal{M}(\mathcal{G}[v'_2]))$. But we showed that $S(v_1, \mathcal{M}(\mathcal{G}[v_2])) = S(v_1, \mathcal{M}(\mathcal{G}[v'_2]))$. Hence $v'_1 \in S(v_1, \mathcal{M}(\mathcal{G}[v_2]))$ also. But $V(\mathcal{M}(\mathcal{G}[v_2])) = V(\mathcal{G}[v_2])$. Therefore $v'_1 \in S(v_1, \mathcal{G}[v_2])$. Consequently, $(v'_1, v_2) \in E(\mathcal{G})$. But we assumed that $(v'_1, v_2) \notin E(\mathcal{G})$, contradiction.

□

Lemma F.3. *Let $X \subseteq \mathcal{G}$ a (closed) subgraph of $\mathcal{G}$. For every $v_2 \in V(\mathcal{M}(X))$:*

$$(v_2 \textit{ is stable in } \mathcal{M}(X)) \Rightarrow (S(v_2, \mathcal{M}(X)) = S(v_2, \mathcal{G}))$$

Proof. Fix $v_2 \in V(\mathcal{M}(X))$. Assume that v_2 is U-stable in $\mathcal{M}(X)$ for some $U \in \text{SPAN}(\mathcal{M}(\mathcal{G}[v_2]))$, *i.e.* $\exists v_1 \in V(\mathcal{M}(X))$ with $id(v_2) = id(v_1)$ such that v_2 U-validates v_1 in $\mathcal{M}(X)$.

Since X is closed then for every vertex $u \in V(\mathcal{M}(X))$, all its predecessors in $\mathcal{G}$ that where created by $id(u)$ belong also to $V(\mathcal{M}(X))$. It follows that:

$$\forall u \in S(v_2, \mathcal{M}(X)) : \exists u' \in S(v_1, \mathcal{M}(X)) \text{ with } (id(u') = id(u)) \tag{3}$$

But v_2 is stable which means that $|S(v_2, \mathcal{M}(X))| = |S(v_1, \mathcal{M}(X))|$. This combined with Property (3) gives:

$$\forall u' \in S(v_1, \mathcal{M}(X)) : \exists u \in S(v_2, \mathcal{M}(X)) \text{ with } (id(u') = id(u)) \tag{4}$$

Since v_2 validates v_1, it follows from Lemma F.2 that:

$$\forall u \in S(v_2, \mathcal{G}) : \exists u' \in S(v_1, \mathcal{M}(X)) \text{ with } (id(u') = id(u)) \tag{5}$$

From Properties (5) and (4) we conclude that $S(v_2, \mathcal{G}) \subseteq S(v_2, \mathcal{M}(X))$. But $S(v_2, \mathcal{M}(X)) \subseteq S(v_2, \mathcal{G})$ as $\mathcal{M}(X) \subseteq \mathcal{G}$. Hence $S(v_2, \mathcal{G}) = S(v_2, \mathcal{M}(X))$. □

G $A\Sigma'$ is Necessary for Consensus

Let $\mathcal{D}$ be any failure detector that allow processes to solve consensus in ξ using algorithm $\mathcal{A}$. We show in this section that $\mathcal{D} \succeq A\Sigma'$. We start by giving some definitions.

Definition G.1. *Let $\mathbb{I}_0$ (resp. $\mathbb{I}_1$) denote the initial configuration in which all processes propose v_0 (resp. v_1). Let S be a schedule. We say that S is 0-**valent** if there exists at least one process that decides (v_0) in the run $\langle \mathcal{F}, \mathcal{H}, \mathbb{I}_0, S\rangle$. The notion of 1-valence is defined similarly using $\mathbb{I}_1$ instead of $\mathbb{I}_0$. S is bivalent if it is both 0-valent and 1-valent. Given a bivalent schedule S, let $\mathtt{B}(S)$ denote the smallest prefix of S that is bivalent and let $\mathtt{P}(S) \subseteq \Pi$ denote the set of processes that take at least one step in $\mathtt{B}(S)$.*

A path T in $\mathcal{G}$ is i-valent ($i \in \{0,1\}$) iff the schedule to which it corresponds is i-valent. $\mathtt{B}(T)$ and $\mathtt{P}(T)$ are defined in the natural way.

Lemma G.2. *For every T, T' bivalent paths of $\mathcal{G}$, it holds that $\mathtt{P}(T) \cap \mathtt{P}(T') \neq \emptyset$.*

Proof. Denote by S, S' the (bivalent) schedules to which correspond T and T' respectively. To prove our lemma it suffices to show that $\mathtt{P}(S) \cap \mathtt{P}(S') \neq \emptyset$. Since $\mathtt{B}(S)$ is bivalent, there exists a process in $\mathtt{P}(S)$ that decides v_0 in the run $R_0 = \langle \mathcal{F}, \mathcal{H}, \mathbb{I}_0, \mathtt{B}(S)\rangle$. Similarly, there exists a process among $\mathtt{P}(S')$ that decides v_1 in the run $R_1 = \langle \mathcal{F}, \mathcal{H}, \mathbb{I}_1, \mathtt{B}(S')\rangle$.

Assume for contradiction that $\mathtt{P}(S) \cap \mathtt{P}(S') = \emptyset$. Let $\mathbb{I}$ be the initial configuration in which the processes of $\mathtt{P}(S)$ propose v_0 and those of $\mathtt{P}(S')$ propose v_1. $\mathbb{I}$ is well defined since we assumed that $\mathtt{P}(S) \cap \mathtt{P}(S') \neq \emptyset$.

Consider the run $R = \langle \mathcal{F}, \mathcal{H}, \mathbb{I}, \mathtt{B}(S).\mathtt{B}(S')\rangle$. The processes of $\mathtt{P}(S)$ cannot distinguish between R and R_0 and those of $\mathtt{P}(S')$ are not able to distinguish between R and R_1. Hence, there exists a process in $\mathtt{P}(S)$ that decides v_0 in R and another process in $\mathtt{P}(S')$ that decides v_1 in R. Contradiction. □

Definition G.3. *Given a bivalent path T, let $\mathtt{V}_1(T)$ denote the set of the last vertices of processes of $\mathtt{P}(T)$ in $\mathtt{B}(T)$. Formally, $\mathtt{V}_1(T) = \underset{k}{\text{argmax}}\ \{V(p_i,k) \in \mathtt{B}(T) \mid p_i \in \mathtt{P}(T)\}$. Note that $|\mathtt{V}_1(T)| = |\mathtt{P}(T)|$. Let $\mathtt{S}_1(T)$ be the following multiset $\{state(v) \mid v \in \mathtt{V}_1(T)\}$.*

Given a graph $X \subseteq \mathcal{G}$ and T a bivalent path in $\mathcal{M}(X)$, let $\mathtt{V}_1^+(\mathcal{M}(X),T) = \{V(p_i,k) \in V(\mathcal{M}(X)) \mid \exists k' \leq k : V(p_i,k') \in \mathtt{V}_1(T)\}$.

Let $\mathtt{W}(\mathcal{M}(X),T)$ be the set of subsets of $|\mathtt{P}(T)|$ vertices of $\mathtt{V}_1^+(\mathcal{M}(X),T)$ which belong to distinct processes. Formally $\mathtt{W}(\mathcal{M}(X),T) = \{U \subseteq \mathtt{V}_1^+(\mathcal{M}(X),T) \mid (|U| = |\mathtt{P}(T)|) \wedge (\forall p \in \mathtt{P}(T) : \exists v \in U : id(v) = p)\}$.

*T is said to be **certified** in $\mathcal{M}(X)$ iff $\forall v_1 \in \mathtt{V}_1(T) : \exists v_2 \in V(\mathcal{M}(X)) : \exists U \in \mathtt{W}(\mathcal{M}(X),T)$ such that v_2 U-validates v_1. Remember that $(id(v_1) = id(v_2))$ in this case.*

If T is certified in $\mathcal{M}(X)$, for each $v_1 \in \mathtt{V}_1(T)$, let $\sigma(v_1, \mathcal{M}(X))$ denote the first vertex in $V(\mathcal{M}(X))$ that U-validates v_1 for $U \in \mathtt{W}(\mathcal{M}(X),T)$. That is, $\sigma(v_1,\mathcal{M}(X)) = \underset{k}{\text{argmin}}\ \{V(id(v_1),k) \mid \exists U \in \mathtt{W}(\mathcal{M}(X),T) : V(id(v_1),k)\ U-validates\ v_1\}$. Let $\mathtt{V}_2(\mathcal{M}(X),T) = \{\sigma(v_1,\mathcal{M}(X)) \mid v_1 \in \mathtt{V}_1(T)\}$ and let $\mathtt{S}_2(\mathcal{M}(X),T)$ be the following multiset $\{state(v) \mid v \in \mathtt{V}_2(\mathcal{M}(X),T)\}$.

Given $X \subseteq \mathcal{G}$, let $\text{QUORA}(X) = \{\mathtt{S}_2(\mathcal{M}(X'),T) \mid (X' \subseteq X) \wedge (T$ is a certified path in $\mathcal{M}(X'))\}$. Note that $\forall X' \subseteq X : \text{QUORA}(X') \subseteq \text{QUORA}(X)$.

Given $x \in \mathbb{S}$, let $T(x) = \{p_i \in \Pi \mid \exists k > 0 : \text{STATE}(p_i,k) \sqsupseteq x\}$. Given $Q = \{x_1, \ldots, x_l\}$ be a multiset of $\mathbb{S}$, let $I(Q) = \{\{p_1, \ldots, p_l\} \mid (p_1 \neq \ldots \neq p_l) \wedge (p_1 \in T(x_1)) \ldots \wedge (p_l \in T(x_l))\}$.

Lemma G.4. *For each graph $X \subseteq \mathcal{G}$, for every bivalent path $T \in \mathcal{M}(X)$, every set in $\mathtt{W}(\mathcal{M}(X),T)$ is a span.*

Proof. Fix $X \subseteq \mathcal{G}$ and T a bivalent path $\mathcal{M}(X)$. Let $U \in \mathtt{W}(\mathcal{M}(X), T)$. We prove that U is a span. Note that $U \subseteq V(\mathcal{M}(X))$.

Let X' be any subgraph of $\mathcal{G}$ such that $\mathcal{M}(X') = \pi(\mathcal{M}(X)) = \mathcal{M}(\pi(X))$ for some permutation π. $U \subseteq V(\mathcal{M}(X))$ implies that $\pi(U) \subseteq V(\pi(\mathcal{M}(X)))$.

To prove that U is a span we have to show that $\exists x \in U : \exists x' \in \pi(U)$ with $(id(x) = id(x'))$.

Since $T \in \mathcal{M}(X)$, it follows that $\pi(T) \in \pi(\mathcal{M}(X))$. Moreover, $\pi(T)$ is bivalent like T as the protocol $\mathcal{A}$ is anonymous and does not rely on processes identities. This implies according to Lemma G.2 that $\mathtt{P}(T) \cap \mathtt{P}(\pi(T)) \neq \emptyset$. Assume that $q \in \mathtt{P}(T) \cap \mathtt{P}(\pi(T))$.

Since $q \in \mathtt{P}(T)$ and $U \in \mathtt{W}(\mathcal{M}(X), T)$, it follows by definition of $\mathtt{W}(\mathcal{M}(X), T)$ that there exists a vertex $x \in U$ with $id(x) = q$. Similarly, since $q \in \mathtt{P}(\pi(T))$ and $\pi(U) \in \mathtt{W}(\pi(\mathcal{M}(X)), \pi(T))$, there exists a vertex $x' \in \pi(U)$ such that $id(x') = q$. Therefore, $id(x) = id(x')$ which proves that U is a span.

□

Lemma G.5. *For every graph $X \subseteq \mathcal{G}$, for every certified path $T \in \mathcal{M}(X)$, for every $P' \in I(\mathtt{S_2}(\mathcal{M}(X), T))$, there exists a bivalent path $T' \in \mathcal{M}(X)$ with $\mathtt{P}(T') = P'$.*

Proof. Fix $X \subseteq \mathcal{G}$ and T a certified path of $\mathcal{M}(X)$. Let $P = \mathtt{P}(T)$. Assume that $P = \{p_1, \ldots, p_l\}$, $\mathtt{V_1}(T) = \{v_1^1, \ldots, v_l^1\}$, $\mathtt{V_2}(\mathcal{M}(X), T) = \{v_1^2, \ldots, v_l^2\}$, $\mathtt{S_1}(T) = \{s_1, \ldots, s_l\}$ and $\mathtt{S_2}(\mathcal{M}(X), T) = \{x_1, \ldots, x_l\}$. These sets and multisets are well defined since T is certified. The reader can check that $\forall i \in [1, l] : state(v_i^2) = x_i$ and $p_i \in T(x_i)$.

Fix $P' \in I(\mathtt{S_2}(\mathcal{M}(X), T))$. Assume that $P' = \{p'_1, \ldots, p'_l\}$ where for every $i \in [1, l] : p'_i \in T(x_i)$. Note that $P' = \pi(P)$ for some permutation π. That is, each p'_i correspond to $\pi(p_i)$.

In the following we prove the existence of a path $T' = \pi(\mathtt{B}(T))$ that is bivalent in $\mathcal{M}(X)$ with $\mathtt{P}(T') = P'$. For this we show that:

1. For each $v \in \mathtt{B}(T)$, the vertex $\pi(v)$ is well defined and belong to $V(\mathcal{M}(X))$:

   For each $i \in [1, l]$, v_i^2 U-validates v_i^1 with $U \in \mathtt{W}(\mathcal{M}(X), T)$. According to Lemma G.4, U is a span. Hence, by Lemma F.2, since $p'_i \in T(x_i)$, it follows that $(p'_i, s_i) \in V(\mathcal{M}(\mathcal{G}[v_2^i]))$. But $V(\mathcal{M}(\mathcal{G}[v_i^2])) = V(\mathcal{G}[v_i^2])$. Thus $(p'_i, s_i) \in V(\mathcal{G}[v_i^2])$. Note that $v_i^2 \in V(X)$ and X is closed; thus $\mathcal{G}[v_i^2] \subseteq X$. Consequently $(p'_i, s_i) \in V(X)$. Since X is closed it follows that:

   $$\forall i \in [1, l] : \forall t \in [1, size(s_i)] : (p'_i, s_i^t) \in V(X) = V(\mathcal{M}(X)) \tag{6}$$

   Moreover, by definition of $\mathtt{S_1}(T)$, s_i is the greatest state of p_i in $\mathtt{B}(T)$. Thus,

   $$\forall p_i \in \mathtt{P}(T) : \forall (p_i, s) \in \mathtt{B}(T) : \exists t \in [1, size(s_i)] : s = s_i^t \tag{7}$$

   Combining equations (6) and (7), we conclude that for every vertex $v = (p_i, s)$ of T, the vertex $\pi(v) = (p'_i, s)$ is well defined and belongs to $V(\mathcal{M}(X))$.

2. For each edge (u, v) of $\mathtt{B}(T)$, $(\pi(u), \pi(v)) \in E(\mathcal{M}(X))$:

   Let (u, v) be any edge of $\mathtt{B}(T)$. Since $T \in \mathcal{M}(X)$, then $\mathtt{B}(T) \in \mathcal{M}(X)$ also. Thus, $(u, v) \in E(\mathcal{M}(X))$. As showed above, $\pi(u)$ and $\pi(v)$ are well defined and belong to $V(\mathcal{M}(X))$. It follows according to Lemma E.8 that $(\pi(u), \pi(v)) \in E(\mathcal{M}(X))$.

Therefore, the path $T' = \pi(\mathtt{B}(T))$ is well defined and belong to $\mathcal{M}(X)$. Moreover, it is bivalent like T and $\mathtt{P}(T') = \pi(\mathtt{P}(T)) = \pi(P) = P'$. This finishes the proof of the lemma. □

The following Theorem will be used in the proof of the intersection property of $A\Sigma'$:

Theorem G.6. *It holds that for every* $Q, Q' \in \text{QUORA}(\mathcal{G}) : \forall P \in I(Q) : \forall P' \in I(Q') : P \cap P' \neq \emptyset$.

Proof. $Q, Q' \in \text{QUORA}(\mathcal{G})$ means that there exists $X, X' \subseteq \mathcal{G}$ and T, T' certified paths in $\mathcal{M}(X)$ and $\mathcal{M}(X')$ respectively such that $Q = \mathtt{S}_2(\mathcal{M}(X), T)$ and $Q' = \mathtt{S}_2(\mathcal{M}(X'), T')$.

Let $P \in I(Q)$ and $P' \in I(Q')$. According to Lemma G.5, there exists two bivalent paths $\overline{T} \in \mathcal{M}(X)$ and $\overline{T'} \in \mathcal{M}(X')$ with $\mathtt{P}(\overline{T}) = P$ and $\mathtt{P}(\overline{T'}) = P'$. But $\mathcal{M}(X) \subseteq \mathcal{G}$ and $\mathcal{M}(X') \subseteq \mathcal{G}$. Hence $\overline{T} \in \mathcal{G}$ and $\overline{T'} \in \mathcal{G}$. By applying Lemma G.2 to $\overline{T}$ and $\overline{T'}$, we conclude that $P \cap P' \neq \emptyset$. □

Lemma G.7. *For every correct process* $p_i \in Correct$*, there exists* $k_i > 0$ *such that* $\mathcal{M}(\mathcal{G}[p_i, k_i])$ *contains a certified path* T *such that* $\mathtt{P}(T) \subseteq Correct$.

Proof. A schedule is fair if every correct process takes infinitely many steps in it. Consider the paths in $\mathcal{M}(\mathcal{G})$ in which only correct processes take steps. Since $\mathcal{D}$ allows processes to solve consensus, it follows that all these paths are bivalent. Take T to be any path among them. Note that $\mathtt{P}(T) \subseteq Correct$. Hence T is certified in $\mathcal{M}(\mathcal{G})$. Let $p_i \in Correct$. We have to show the existence of $k_i > 0$ such that T is certified in $\mathcal{M}(\mathcal{G}[p_i, k_i])$.

Since T is certified in $\mathcal{M}(\mathcal{G})$, the vertices $\mathtt{V}_1(T)$ and $\mathtt{V}_2(\mathcal{M}(\mathcal{G}), T)$ are well defined. Moreover, according to property (m4) of Definition D.4, there exists $k_i > 0$ such that $\mathcal{M}(\mathcal{G})[p_i, k_i]$ contains T, $\mathtt{V}_1(T)$ and $\mathtt{V}_2(T)$. According to property (m5) of Definition D.4, $\mathcal{M}(\mathcal{G})[p_i, k_i] \subseteq \mathcal{M}(\mathcal{G}[p_i, k_i])$. This means that $\mathcal{M}(\mathcal{G}[p_i, k_i])$ contains T, $\mathtt{V}_1(T)$ and $\mathtt{V}_2(\mathcal{M}(\mathcal{G}), T)$. Therefore T is certified in $\mathcal{M}(\mathcal{G}[p_i, k_i])$.

□

Theorem G.8. $A\Sigma'$ *is necessary for consensus in anonymous message passing systems.*

Proof. The extraction of $A\Sigma'$ is done as follows. Each process maintain a variable QUORUM in which it stores the quorums it generates. Let QUORUMS_i^k denotes the value of QUORUM at process p_i at the end of its k-th step. Initially $\text{QUORUMS}_i^0 = \emptyset$. When p_i executes its k-th step, it generates $\mathcal{M}(\pi_p^k(\mathcal{G}[p, k]))$ where $\pi_p^k : \Pi \mapsto \Pi$ is a permutation of processes identities (Theorem E.1). Then it computes $\text{QUORA}(\mathcal{M}(\pi_p^k(\mathcal{G}[p, k])))$. Note that $\text{QUORA}(\mathcal{M}(\mathcal{G}[p, k])) = \text{QUORA}(\mathcal{M}(\pi_p^k(\mathcal{G}[p, k])))$. After that, p_i updates its variable QUORUMS as follows: $\text{QUORUMS} \leftarrow \text{QUORUMS} \cup \text{QUORA}(\mathcal{M}(\pi_p^k(\mathcal{G}[p, k])))$. The output of $A\Sigma'$ at p_i at this step corresponds to the pair $(\text{STATE}(p_i, k), \text{QUORUMS}_i^k)$. The formal properties of $A\Sigma'$ are satisfied:

Monotony: Follows from the fact that $\forall k > 1 : st(p_i, k) \sqsupset st(p_i, k-1)$ and $\text{QUORUMS}_i^k = \text{QUORUMS}_i^{k-1} \cup \text{QUORA}(\mathcal{M}(\mathcal{G}[p, k]))$.

Liveness: Let p_i be any correct process. According to Lemma G.7, there exists $k_i > 0$ such that $\mathcal{M}(\mathcal{G}[p_i, k_i])$ contains a certified path T such that $\mathtt{P}(T) \subseteq Correct$. Let $Q = \mathtt{S}_2(\mathcal{M}(\mathcal{G}[p_i, k_i]), T)$. Since T is certified in $\mathcal{M}(\mathcal{G}[p_i, k_i])$ It follows that $Q \in \text{QUORA}(\mathcal{M}(\mathcal{G}[p_i, k_i]))$. But $\text{QUORA}(\mathcal{M}(\mathcal{G}[p, k])) = \text{QUORA}(\mathcal{M}(\pi_p^k(\mathcal{G}[p, k])))$. It follows that $Q \in \text{QUORA}(\mathcal{M}(\pi_p^k(\mathcal{G}[p, k])))$. Therefore, $Q \in \text{QUORUMS}_i^{k_i}$.

Note that for every path T' that is certified in $\mathcal{M}(X)$, it holds that $\mathtt{P}(T') \in I(\mathtt{S}_2(\mathcal{M}(X), T'))$. Hence $\mathtt{P}(T) \in I(Q) = I(\mathtt{S}_2(\mathcal{M}(\mathcal{G}[p_i, k_i]), T))$. To summarize: $\exists Q \in \text{QUORUMS}_i^{k_i} : \exists P = \mathtt{P}(T) \in I(Q) : P \subseteq Correct$.

Intersection: Below we prove that for each $\forall p_i \in \Pi : \forall k > 0 : \forall Q \in \text{QUORUMS}_i^k : Q \in \text{QUORA}(\mathcal{G})$. Hence, the intersection property follows directly from Theorem G.6. Fix QUORUMS_i^k and let $Q \in \text{QUORUMS}_i^k$. Thus, $\exists k' < k : Q \in \text{QUORA}(\mathcal{M}(\pi_p^{k'}(\mathcal{G}[p, k'])))$. But $\text{QUORA}(\mathcal{M}(\mathcal{G}[p, k'])) =$

QUORA($\mathcal{M}(\pi_p^{k'}(\mathcal{G}[p,k']))$). Hence $Q \in$ QUORA($\mathcal{M}(\mathcal{G}[p,k'])$). Since $m(\mathcal{G}[p,k']) \subseteq \mathcal{G}$, it follows that QUORA($\mathcal{M}(\mathcal{G}[p,k'])) \subseteq$ QUORA($\mathcal{G}$). Therefore, $Q \in$ QUORA($\mathcal{G}$).

□

H $\mathcal{AL}$ is Necessary for Consenus

In this section we prove that $\mathcal{AL}$ is necessary for consensus. We start by some technical preliminaries, then we present the necessity proof. Our strategy is to reduce the necessity proof to CHT [12]. To this end, our precedence graphs must have the same properties as those used in [12]. Hence, we have to fix the problems already observed in Property D.9. Moreover, we prove that eventually, all the permutations π_i^k converge to the same unique permutation χ.

H.1 Preliminaries

Let $\mathcal{D}$ be any failure detector that allows processes to solve consensus using some algorithm $\mathcal{A}$. Let $\mathcal{H} \in \mathcal{D}(\mathcal{F})$. In Section G we showed that $\mathcal{D} \succeq A\Sigma'$. Hence, there is no loss of generality to assume that for every $p_i \in \Pi$, for every time $\tau \in \mathbb{T}$, $\mathcal{H}(p_i, \tau)$.QUORUMS and $\mathcal{H}(p_i, \tau)$.LABEL are well defined and satisfy the properties of $A\Sigma'$.

Denote by τ_i^k the time at which p_i executes its k-th step. Let $R_i^k = \mathcal{H}(p_i, \tau_i^k)$.QUORUMS. We suppose that processes periodically exchange the values of their R_i^k. Hence, there is no loss of generality to assume that $\exists \overline{R} : \forall p_i \in \Pi : \lim_{k\to\infty} R_i^k = \overline{R}$.

Observe that each member of $\overline{R}$ is a multiset of elements of $\mathbb{L}$. Given $x \in \mathbb{L}$, let $S(x) = \{v \in V(\mathcal{G}) : fd(v).\text{LABEL} \sqsupseteq x\}$. Given $Q = \{x_1, \ldots, x_m\} \in \overline{R}$, let $J(Q)$ denotes the set $\{\{v_1, \ldots, v_m\} \mid (v_1 \in S(x_1)) \wedge \ldots \wedge (v_m \in S(x_m)) \wedge (id(v_1) \neq \ldots \neq id(v_m)\}$. Given a set of vertices U, let $P(U) = \{p_i \in \Pi \mid \exists v \in U : id(v) = p_i\}$.

Lemma H.1. $\forall Q \in \overline{R} : \forall U \in J(Q) : \forall X \subseteq \mathcal{G}$: *if* $U \subseteq V(X)$ *then* U *is a span in* $\mathcal{M}(X)$.

Proof. Fix $Q \in \overline{R}$ and $U \in J(Q)$. Let $X \subseteq \mathcal{G}$ such that $U \subseteq V(X)$. Let $X' \subseteq \mathcal{G}$ with $X' = \pi(X)$ for some permutation π. Note that $\pi(U) \in J(Q)$. Observe that $P(U) \in I(Q)$ and $P(\pi(U)) \in I(Q)$. Hence, according to the specification of $A\Sigma'$, $P(U) \cap P(\pi(U)) \neq \emptyset$. Let $q = P(U) \cap P(\pi(U))$. Hence, $\exists x \in U : \exists x' \in \pi(U) : id(x) = id(x') = q$. Thus, U is a span. □

Let $R \subseteq \overline{R}$ and let $X \subseteq \mathcal{G}$. We say that $v_2 \in V(X)$ is R-**stable in** $\mathcal{M}(X)$ iff $\exists Q \in R : \exists U \in J(Q)$ such that v_2 is U-stable in $\mathcal{M}(X)$.

Definition H.2. *Let* $R \subseteq \overline{R}$. *Let* $X \subseteq \mathcal{G}$ *a closed subgraph of* $\mathcal{G}$. *Denote by* $\mathcal{VL}(X, R)$ *the set of* R*-stable vertices of* $\mathcal{M}(X)$. *Formally,* $\mathcal{VL}(X, R) = \{v \in V(X) : v$ *is* R*-stable in* $\mathcal{M}(X)\}$. *The graph* $\mathcal{L}(X, R)$ *is the subgraph of* X *induced by the vertices of* $\mathcal{VL}(X, R)$. *That is* $\mathcal{L}(X, R) = X[\mathcal{VL}(X, R)]$. *We say that* $\mathcal{L}(X, R)$ *is* stable.

The following lemma proves some properties about stable graphs:

Lemma H.3. *Let* $R, R' \subseteq \overline{R}$ *such that* $R \subseteq R'$. *Let* $X \subseteq \mathcal{G}$ *a closed subgraph of* $\mathcal{G}$. *The following properties hold:*

(1) $\mathcal{M}(\mathcal{L}(X, R)) = \mathcal{M}(\mathcal{G})[\mathcal{VL}(X, R)]$

(2) $\mathcal{M}(\mathcal{L}(X, R)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G}, R))$

(3) $\mathcal{M}(\mathcal{L}(X, R)) \subseteq \mathcal{M}(\mathcal{L}(X, R'))$

(4) $\mathcal{M}(\mathcal{L}(X, R)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G}, R'))$

Proof. (1) By definition, all the vertices of $\mathcal{VL}(X,R)$ are R-stable. It follows according to Lemma F.3 that $\forall v \in \mathcal{VL}(X,R) : S(v,\mathcal{VL}(X,R)) = S(v,V(\mathcal{G}))$. Note that $\mathcal{VL}(X,R) \subseteq V(X) \subseteq V(\mathcal{G})$. Hence, according to Corollary E.10, we have $\mathcal{M}(\mathcal{G}[\mathcal{VL}(X,R)]) = \mathcal{M}(\mathcal{G})[\mathcal{VL}(X,R)]$. But $\mathcal{L}(X,R) = \mathcal{G}[\mathcal{VL}(X,R)]$. Therefore $\mathcal{M}(\mathcal{L}(X,R)) = \mathcal{M}(\mathcal{G})[\mathcal{VL}(X,R)]$.

(2) Since $X \subseteq \mathcal{G}$, it holds that $V(X) \subseteq V(\mathcal{G})$. Hence $\mathcal{VL}(X,R) \subseteq \mathcal{VL}(\mathcal{G},R)$ which implies that $\mathcal{M}(\mathcal{G})[\mathcal{VL}(X,R)] \subseteq \mathcal{M}(\mathcal{G})[\mathcal{VL}(\mathcal{G},R)]$. By applying (1) we conclude that: $\mathcal{M}(\mathcal{L}(X,R)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G},R))$.

(3) $R \subseteq R'$ implies that $\mathcal{VL}(X,R) \subseteq \mathcal{VL}(X,R')$. Hence $\mathcal{M}(\mathcal{G})[\mathcal{VL}(X,R)] \subseteq \mathcal{M}(\mathcal{G})[\mathcal{VL}(X,R')]$ which means according to (1) that $\mathcal{M}(\mathcal{L}(X,R)) \subseteq \mathcal{M}(\mathcal{L}(X,R'))$.

(4) Let $R' \supseteq R$. According to (3) $\mathcal{M}(\mathcal{L}(X,R)) \subseteq \mathcal{M}(\mathcal{L}(X,R'))$ and according to (2) $\mathcal{M}(\mathcal{L}(X,R')) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G},R'))$. Hence by transitivity we have: $\mathcal{M}(\mathcal{L}(X,R)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G},R'))$.

□

The following lemma proves that each correct process creates an infinity of $\overline{R}$-stable vertices in $\mathcal{G}$.

Lemma H.4. *For every correct process p_i, there exists $K_i > 0$ such that $\forall k \geq K_i : V(p_i,k)$ is $\overline{R}$-stable in $\mathcal{G}$.*

Proof. Fix $p_i \in \Pi$. To prove the lemma it suffices to show that $\exists K_i > 0 : \forall k \geq K_i : V(p_i,k)$ $\overline{R}$-validates $V(p_i,1)$. Let v_1 denotes $V(p_i,K_i)$. According to the specification of $A\Sigma'$, there exists a quorum output by the failure detector that has an instance containing only correct processes. Formally, $\exists Q = \{l_1,\ldots,l_x\} \in \overline{R} : \exists S : S \subseteq Correct \wedge S \in I(Q)$. Let $S = \{q_1,\ldots,q_x\}$ such that $\forall j \in [1,x] : \exists \tau : \forall \tau' \geq \tau : \mathcal{H}(q_j,\tau').\text{LABEL} \sqsupseteq l_j$. Hence,

$$\forall j \in [1,x] : \exists v_j \in V(\mathcal{G}) : (id(v_j) = q_j) \wedge (fd(v_j) \sqsupseteq l_j) \tag{8}$$

Moreover, according to the property (m4) of $\mathcal{M}(\mathcal{G})$ (Theorem D.8), it holds that:

$$\forall q_j \in S : \exists k_j : \forall k \geq k_j : (v_1, V(q_j,k) \in E(\mathcal{M}(\mathcal{G})) \tag{9}$$

Combining Equations (8) and (9) we conclude that:

$$\forall j \in [1,x] : \exists u_j \in V(\mathcal{G}) : (id(u_j) = q_j) \wedge (fd(u_j) \sqsupseteq l_j) \wedge (v_1,u_j) \in E(\mathcal{M}(\mathcal{G})) \tag{10}$$

Let $U = \{u_1,\ldots,u_x\}$. Note that $U \in J(Q)$. According to property (m4) of $\mathcal{M}(\mathcal{G})$ (Theorem D.8) we have:

$$\exists K_i > 0 : \forall k \geq K_i : \forall u_j \in U : (u_j, V(p_i,k)) \in E(\mathcal{M}(\mathcal{G})) \tag{11}$$

By applying property (m5) of Theorem D.8 to equations (10) and (11) we conclude that:

$$\exists K_i > 0 : \forall k \geq K_i : \forall u_j \in U : (v_1,u_j),(u_j V(p_i,k)) \in E(\mathcal{M}(\mathcal{G}[p_i,k]))$$

This means that $\forall k \geq K_i$, $V(p_i,k)$ U-validates v_1 which implies that $V(p_i,k)$ is U-stable in $\mathcal{G}$. But $U \in J(Q)$ and $Q \in \overline{R}$. Consequently $V(p_i,k)$ is $\overline{R}$-stable in $\mathcal{G}$. □

In the following thorem, we prove some properties of $\mathcal{M}(\mathcal{L}(\mathcal{G},\overline{R}))$:

Theorem H.5. *Let $X \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$. $\mathcal{M}(X)$ and $\mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$ satisfy the properties (m1), (m2), (m4) and (m5) of Theorem D.8. Moreover:*

(m6) $\mathcal{M}(\mathcal{L}(\mathcal{G}[p_i, k], R_i^k)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$ *and* $\lim_{k\to\infty} \mathcal{M}(\mathcal{L}(\mathcal{G}[p_i, k], R_i^k)) = \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$.

Proof. Properties (m1), (m2) and (m5) follows directly by application of Theorem D.8. In the following we prove (m4) and (m6):

- (m4) Fix $V \subseteq V(\mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R})))$. Note that $V(\mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))) = \mathcal{VL}(\mathcal{G}, \overline{R}) \subseteq V(\mathcal{G})$. Fix p_i a correct process. According to property (m4) of Theorem D.8, it follows that:

$$\exists k_i^1 : \forall k \geq k_i^1 : \forall v \in V : (v, V(p_i, k)) \in E(\mathcal{M}(\mathcal{G})) \tag{12}$$

  Moreover, according to Lemma H.4, it holds that:

$$\exists k_i^2 > 0 : \forall k \geq k_i^2 : V(p_i, k) \in \mathcal{VL}(\mathcal{G}, \overline{R}) \tag{13}$$

  Let $k_i = \max(k_i^1, k_i^2)$. Note that $V \subseteq \mathcal{VL}(\mathcal{G}, \overline{R})$ and $\forall k \geq k_i : V(p_i, k) \in \mathcal{VL}(\mathcal{G}, \overline{R})$. Hence, equation (12) implies that:

$$\forall k \geq k_i : \forall v \in V : (v, V(p_i, k)) \in E(\mathcal{M}(\mathcal{G})[\mathcal{VL}(\mathcal{G}, \overline{R})])$$

  But according to Property (m5) of Theorem D.8 we have: $\mathcal{M}(\mathcal{G})[\mathcal{VL}(\mathcal{G}, \overline{R})] \subseteq \mathcal{M}(\mathcal{G}[\mathcal{VL}(\mathcal{G}, \overline{R})]) = \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$

  Hence:

$$\forall k \geq k_i : \forall v \in V : (v, V(p_i, k)) \in E(\mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R})))$$

  This proves the claim.

- (m6) Since $\mathcal{G}[p_i, k] \subseteq \mathcal{G}$ and $R_i^k \subseteq \overline{R}$, it follows from Lemma H.3 (4) that $\mathcal{M}(\mathcal{L}(\mathcal{G}[p_i, k], R_i^k)) \subseteq \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$.

  Moreover, $\lim_{k\to\infty} \mathcal{G}[p_i, k] = \mathcal{G}$ and $\lim_{k\to\infty} R_i^k = \overline{R}$. Hence $\lim_{k\to\infty} \mathcal{M}(\mathcal{L}(\mathcal{G}[p_i, k], R_i^k)) = \mathcal{M}(\mathcal{L}(\mathcal{G}, \overline{R}))$.

□

Theorem H.6. *There exists an algorithm that allows each process p_i executing $S^k(p_i)$ to compute $\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i, k]), R_i^k))$ where $\pi_i^k : \Pi \mapsto \Pi$ is a permutation of processes identities.*

Proof. Let $X = \pi_i^k(\mathcal{G}[p_i, k])$. Let $Y = \mathcal{L}(X, R_i^k)$. Note that $V(Y) = \mathcal{VL}(X, R_i^k)$ and $Y = X[V(Y)]$. To prove the lemma, it suffices to show how p_i can compute $\mathcal{M}(Y)$. But according to Lemma E.9 $\mathcal{M}(Y) = \mathcal{M}(X)[V(Y)]$. It suffices then to show how to construct $\mathcal{M}(X)$ and $V(Y)$. $\mathcal{M}(X) = \mathcal{M}(\pi_i^k(\mathcal{G}[p_i, k]))$ is computed as indicated in Theorem H.6 and $V(Y) = \mathcal{VL}(X, R_i^k) = \mathcal{VL}(\mathcal{M}(X), R_i^k)$.

□

Before proceeding to the $\mathcal{AL}$-extraction procedure, we need to prove one more property about the constructed graphs (remember that $\chi = \chi[\mathcal{G}]$):

Theorem H.7. *There exists $\mathcal{N} > 0$ such that for every correct process p_i, for every $k \geq \mathcal{N}$, $\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i, k]), R_i^k)) = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i, k]), R_i^k))$.*

The remainder of this subsection is devoted to the proof of the Theorem.

Definition H.8. *For each $p_i \in \Pi$, let $state_i$ denote $maxState(p_i, \mathcal{G})$. Note that if p_i is faulty, then $size(state_i)$ is finite. We define the following equivalence relation between processes. We say that processes p_i and p_j are homonyms, denote $p_i \approx p_j$ iff $state_i = state_j$, otherwise we write $p_i \not\approx p_j$.*

For each process $p_i \in \Pi$, we associate a variable N_i defined as follows:

1. *If all the processes in Π are homonyms, then $\forall i \in [1, n] : K_i = 1$.*
2. *Otherwise:*
   (a) *If p_i is faulty, then $N_i = size(state_i)$.*
   (b) *If p_i is correct, then for every $p_j \in \Pi$ with $p_i \not\approx p_j$, define N_{ij} as follows: (1) If p_j is faulty, then $N_{ij} = size(state_j) + 1$; otherwise N_{ij} is the smallest k such that $state_i^k \neq state_j^k$. Let $N_i = max_{p_j \not\approx p_i} N_{ij}$.*

We prove the following property:

Lemma H.9. *$\exists \mathcal{N} > 0$, such that for every correct process $p_i : \forall k \geq \mathcal{N} : \forall p_j \in \Pi : V(p_j, N_j) \in \mathcal{G}[p_i, k]$.*

Proof. According to Properties (d4), (d3) and (d2), it follows that for every correct process $\exists k_i : \forall k \geq k_i : \forall p_j \in \Pi : V(p_j, N_j) \in \mathcal{G}[p_i, k]$. The lemma follows by setting $\mathcal{N}$ to be the maximum of the different k_i. □

Lemma H.10. *For every correct process p_i, $\forall k > N_j : \forall p_j, p_l \in \Pi$, it holds that: $(\pi_i^k(p_j) = \chi(p_l)) \Rightarrow (p_j \approx p_l))$.*

Proof. Fix p_i, p_j, p_l three (not necessarily distinct) processes. Remember that $p_i^k = \chi[\mathcal{G}[p_i, k]]$ and $\chi = \chi[\mathcal{G}]$. The lemma thus says:

$$(\mathcal{O}(p_j, \mathcal{G}[p_i, k]) = \mathcal{O}(p_l, \mathcal{G})) \Rightarrow (p_j \approx p_l))$$

Proof by contrapositive. Suppose that $p_j \not\approx p_l$. Hence, either $(state_j < state_l)$ or $(state_l > state_j)$. Assume without loss of generality that $(state_j < state_l)$. Note that according to Lemma H.9, $V(p_j, N_j), V(p_l, N_l) \in \mathcal{G}[p_i, k]$. Therefore $maxState(p_j, \mathcal{G}[p_i, k]) = \text{STATE}(p_j, k)$ for some $k \geq N_j$ and $maxState(p_l, \mathcal{G}[p_i, k]) > \text{STATE}(p_l, k)$ for some $k \geq N_l$. We distinguish between the following cases and we prove that each in case $(\mathcal{O}(p_j, \mathcal{G}[p_i, k]) < \mathcal{O}(p_l, \mathcal{G}))$ which proves our claim.

Case 1: Both p_j and p_l are faulty. In this case, $maxState(p_j, \mathcal{G}[p_i, k]) = \text{STATE}(p_j, N_j) = state_j$ and $maxState(p_l, \mathcal{G}[p_i, k]) = \text{STATE}(p_l, N_l) = state_l$. But we assumed that $(state_j < state_l)$. Hence $maxState(p_j, \mathcal{G}[p_i, k]) < maxState(p_l, \mathcal{G}[p_i, k])$, implying that $(\mathcal{O}(p_j, \mathcal{G}[p_i, k]) < \mathcal{O}(p_l, \mathcal{G}))$.

Case 2: Only p_j is correct. Since p_l is faulty, we have $maxState(p_l, \mathcal{G}[p_i, k]) = \text{STATE}(p_l, N_l) = state_l$. Moreover, by definition of N_j we have $N_j \geq N_{jl}$ and since p_l is faulty $N_{jl} = N_l + 1$. Thus, $N_j > N_l$.

Since we assumed that $(state_j < state_l)$ and $(size(state_l) = N_l)$, it follows that $\forall t > N_l : state_j^t < state_l$. Hence $\forall t \geq N_j : state_j^t < state_l$. But $\forall t \geq 1 : \text{STATE}(p_j, t) = state_j^t$. Consequently $\forall t \geq N_j : \text{STATE}(p_j, t) < state_l$. This implies that $(maxState(p_j, \mathcal{G}[p_i, k]) < state_l)$ because $(maxState(p_j, \mathcal{G}[p_i, k]) = \text{STATE}(p_j, t))$ for some $t \geq N_j$. Since $state_l = maxState(p_l, \mathcal{G}[p_i, k])$ this becomes $maxState(p_j, \mathcal{G}[p_i, k]) < maxState(p_l, \mathcal{G}[p_i, k])$. Therefore, $(\mathcal{O}(p_j, \mathcal{G}[p_i, k]) < \mathcal{O}(p_l, \mathcal{G}))$.

Case 3: Only p_l is correct. Symmetric to Case 2.

Case 4: Both p_j and p_l are correct. In this case, since $(state_j < state_l)$, it follows that $\forall t_j \geq N_{jl} : \forall t_l \geq N_{jl} : state_j^{t_j} < state_l^{t_l}$. But $maxState(p_j, \mathcal{G}[p_i, k]) = \text{STATE}(p_j, t_j) = state_j^{t_j}$ for some $t_j \geq N_j \geq N_{jl}$ and $maxState(p_l, \mathcal{G}[p_i, k]) = \text{STATE}(p_l, t_l) = state_l^{t_l}$ for some $t_l \geq N_l \geq N_{jl}$. Consequently $maxState(p_j, \mathcal{G}[p_i, k]) < maxState(p_l, \mathcal{G}[p_i, k])$ implying that $(\mathcal{O}(p_j, \mathcal{G}[p_i, k]) < \mathcal{O}(p_l, \mathcal{G}))$.

□

Proof of Theorem H.7. To avoid clutter, we refer to π_i^k by π. Let $X = \mathcal{G}[p_i, k]$, $X_1 = \pi(X), X_2 = \chi(X), Y = \mathcal{VL}(X, R_i^k), Y_1 = \mathcal{VL}(X_1, R_i^k)$ and $Y_2 = \mathcal{VL}(X_2, R_i^k)$. We need to prove that $\mathcal{M}(X_1[Y_1]) = \mathcal{M}(X_2[Y_2])$. The result follows from the following two claims:

C1: $Y_1 = Y_2$

Proof. We prove that $Y_1 \subseteq Y_2$; the proof of $Y_2 \subseteq Y_1$ is symmetric. Let $v \in Y_1$. We have to show that $v \in Y_2$. Since $v \in Y_1 = \mathcal{VL}(\pi(X), R_i^k)$, then $v = \pi(u)$ where $u \in Y$. Let $p_v = id(v)$. Note that $p_v = \pi(p)$ with $p \in \Pi$. There exists $p' \in \Pi$ such that $\chi(p') = \pi(p)$. According to Lemma H.10, this implies that $p' \approx p_x$. Since $u \in V(\mathcal{G})$ with $id(u) = p$ and $p \approx p'$, there must exists a vertex $u' \in V(\mathcal{G})$ such that $id(u') = p'$ and $state(u') = state(u)$. Note that since $\chi(p') = \pi(p) = p_v$, it follows that $\chi(u) = \pi(u') = v$. But $u \in Y = \mathcal{VL}(X, R_i^k)$ meaning that u is stable in $\mathcal{M}(X)$. Hence, according to Lemma F.3, since $state(u') = state(u)$, it must be the case that $u' \in V(X)$ also. Moreover, u' is also stable in $\mathcal{M}(X)$, hence $u' \in \mathcal{VL}(X, R_i^k)$. This implies that $\chi(u') \in \mathcal{VL}(\chi(X), R_i^k)$. But $v = \chi(u')$ and $Y_2 = \mathcal{VL}(\chi(X), R_i^k)$. Therefore $v \in Y_2$.

C2: $\forall u, v \in Y_1 = Y_2$, it holds that $((u, v) \in E(\mathcal{M}(X_1))) \Leftrightarrow (u, v) \in E(\mathcal{M}(X_2)))$.

Proof. Since $u, v \in Y_1$, it holds that $u = \pi(x)$ and $v = \pi(y)$ for $x, y \in V(\mathcal{G}[p_i, k])$. Similarly, since u and v belong to Y_2 also, it holds that $u = \chi(x')$ and $v = \chi(y')$ for $x', y' \in V(\mathcal{G}[p_i, k])$. Hence $x' = \psi(x)$ and $y' = \chi(y)$ for some permutation ψ. It follows according to Lemma E.8 that $(x, y) \in E(\mathcal{M}(\mathcal{G}[p_i, k]))$ iff $(x', y') \in E(\mathcal{M}(\mathcal{G}[p_i, k]))$.

Note that

$$\begin{aligned}
(x, y) \in E(\mathcal{M}(\mathcal{G}[p_i, k])) &\Leftrightarrow (\pi(x), \pi(y)) \in E(\pi(\mathcal{M}(\mathcal{G}[p_i, k]))) \\
&\Leftrightarrow (u, v) \in E(\pi(\mathcal{M}(\mathcal{G}[p_i, k]))) \\
&\Leftrightarrow (u, v) \in E(\mathcal{M}(\pi(\mathcal{G}[p_i, k]))) \\
&\Leftrightarrow (u, v) \in E(\mathcal{M}(X_1)))
\end{aligned}$$

Similarly,

$$((x', y') \in E(\mathcal{M}(\mathcal{G}[p_i, k]))) \Leftrightarrow ((u, v) \in E(\mathcal{M}(X_2)))$$

Consequently, $(u, v) \in \mathcal{M}(X_1)$ iff $(u, v) \in \mathcal{M}(X_2)$.

□

H.2 Extraction Procedure

The extraction procedure is shown in Figure 6. When p_i executes its k-th step, it generates the graph $\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i,k]),R_i^k))$. Eventually, as proved in Theorem H.7, any constructed graph the constructed graph will be equal to $\mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$. Hence, its transitive closure has all the necessary properties needed for CHT [12] to eventually elect a correct unique $leader$ (line 4). If the executing process is not an homonym of the currently elected leader, it outputs $(0,0)$ (line 10), otherwise it outputs $(1,|Leaders|)$ where $Leader$ is the estimated set of homonyms of $leader$ (line 8). We show that eventually, this estimation is always correct thanks to the stability property of the generated local graphs. Hence, hence if a vertex of the leader is present (thus stable) in a graph G, all the homonyms of the leader will have also vertices in G having the same state. Since the states of the processes that are not homonyms eventually diverge, the estimation is eventually correct.

Proof. Let us define:

- τ_1 : The time at which all faulty processes have crashed.
- $\tau_2 \geq \tau_1$: The time starting at which every correct process p_i executing $S^k(p_i)$ generates $\mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$. The existence of τ_2 is guaranteed by Theorem H.7.
- $\tau_3 \geq \tau_2$: The time after which all processes elect the same leader p_l. That is $\exists p_l \in \Pi : \forall \tau \geq \tau_3 : \forall p_i \in \Pi : leader_i^\tau = \chi(p_l)$. Note that $leader_i^\tau$ is the value of variable $leader$ at process p_i at time τ. τ_3 is well defined according to CHT [12] since the precedence graphs generated after τ_2 have the same properties as those used in [12]. Let LEADERS denote the set $\{p_i \in \Pi \mid p_i \approx p_l\}$ and NBLEADERS denote its cardinality.
- $\tau_4 \geq \tau_3$: Let $V(p_l,k_l)$ the first $\overline{R}$-stable vertex of the leader in $\mathcal{M}(\mathcal{G})$ with $k_l \geq N_l$. This vertex is well defined by Lemma H.4 and the fact that p_l is correct. Therefore, since $\lim_{k\to\infty}\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i,k]),R_i^k)) = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),\overline{R}))$, there exists $\mathcal{Z} \geq \mathcal{N}$, such that $\forall k \geq \mathcal{Z} : \forall p_i \in \Pi$ a correct process, it holds that $\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i,k]),R_i^k)) = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$ and $V(p_l,k_l)$ belongs to this graph. Let τ_4 be the first time after which if p_i executes a step $S^k(p_i)$ then $k \geq \mathcal{Z}$.

The following properties hold:

C1: $\forall p_i \in \Pi : \forall k \geq \mathcal{Z} : \text{STATE}(p_i,k) \sqsupseteq \text{STATE}(p_l,k_l) \Rightarrow (p_l \approx p_i)$

Proof. Assume towards contradiction that $p_i \not\approx p_l$. Since both p_i and p_l are correct, it holds that $\text{STATE}(p_i,N_{il}) \neq \text{STATE}(p_l,N_{il})$. Hence any extension of $\text{STATE}(p_l,N_{il})$ cannot be a substring of $\text{STATE}(p_i,N_{il})$. Since $k \geq \mathcal{Z} \geq \mathcal{N} \geq N_{il}$ and $k_l \geq N_l \geq N_{il}$, it follows that $\text{STATE}(p_i,k) \not\sqsupseteq \text{STATE}(p_l,k_l)$, contradiction !

C2: $\forall p_i \in \Pi : \forall k \geq \mathcal{Z} : maxState(p_l,G) \sqsupseteq \text{STATE}(p_l,k_l)$ with $G = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$

Proof. Since $k \geq \mathcal{Z}$, then $V(p_l,k_l)$ belongs to $\mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$ by definition of $\mathcal{Z}$.

C3: $\forall p_i \in \Pi : \forall k \geq \mathcal{Z} : \text{LEADERS} \subseteq \{p_i \in \Pi \mid maxState(p_i,G) = maxState(p_l,G)\}$ with $G = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i,k]),R_i^k))$

Proof. Because the leaders are homonyms, and G is stable. According to Lemma F.3, if a process p have a vertex that is stable in G and whose state is st, then all its homonyms must have a vertex in G with the same state.

C4: $\forall p_i \in \Pi : \forall k \geq \mathcal{Z} : \{p_i \in \Pi \mid maxState(p_i, G) = maxState(p_l, G)\} \subseteq$ LEADERS with $G = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i, k]), R_i^k))$

Proof. Follows from C1 and C2.

C5: $\forall p_i \in \Pi : \forall k \geq \mathcal{Z} : \{p_i \in \Pi \mid maxState(p_i, G) = maxState(p_l, G)\} =$ LEADERS with $G = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i, k]), R_i^k))$

Proof. Follows from C3 and C4.

Property C1 says that starting from step $\mathcal{Z}$, only processes in LEADERS can have a state that is an extension of STATE(p_l, k_l). Hence, starting from time τ_4, the processes not in LEADERS cannot consider themselves as leaders:

- $\forall \tau \geq \tau_4 : \forall p_i \notin$ LEADERS : RANK$_i^\tau = 0$

Let $p_i \in$ LEADERS executing $S^k(p_i)$ after τ_4. According to C3: $p_i \in \{p_i \in \Pi \mid maxState(p_i, G) = maxState(p_l, G)\}$ with $G = \mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i, k]), R_i^k))$. Clearly, STATE$(p_i, k) \sqsupseteq maxState(p_i, G)$. Therefore, the condition of line 6 is true for p_i. Hence it follows:

- $\forall \tau \geq \tau_4 : \forall p_i \in$ LEADERS : RANK$_i^\tau = 1$

Then, combining this with C5:

- $\forall \tau \geq \tau_4 : \forall p_i \in$ LEADERS : MULTIPLICITY$_i^\tau = |$LEADERS$|$

□

Upon $S^k(p_i)$**:**

(1) $myState \leftarrow$ GETSTATE()
(2) COMPUTE $\mathcal{M}(\mathcal{L}(\pi_i^k(\mathcal{G}[p_i, k]), R_i^k))$ (Theorem H.6)
(3) $G \leftarrow$ The transitive closure of $\mathcal{M}(\mathcal{L}(\chi(\mathcal{G}[p_i, k]), R_i^k))$
(4) $leader \leftarrow CHT(G)$
(5) $leaderState \leftarrow maxState(leader, G)$
(6) **if** $(myState \sqsupseteq leaderState)$
(7) $\quad Leaders \leftarrow \{p_i \in \Pi \mid maxState(p_i, G) = leaderState\}$
(8) $\quad$ RETURN (RANK, MULTIPLICITY) $= (1, |Leaders|)$
(9) **else**
(10) $\quad$ RETURN (RANK, MULTIPLICITY) $= (0, 0)$

Figure 6: Extraction of $\mathcal{AL}$