Round-based Synchrony Weakened by Message Adversaries vs Asynchrony Enriched with Failure Detectors

Michel Raynal, Julien Stainer

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 2002 – February 2013

UMR IRISA

Round-based Synchrony Weakened by Message Adversaries

vs

Asynchrony Enriched with Failure Detectors

Michel Raynal[*] [**] Julien Stainer[**]

Abstract: A message adversary is a daemon that suppresses messages in round-based message-passing synchronous systems in which no process crashes. A property imposed on a message adversary defines a subset of messages that cannot be eliminated by the adversary. It has recently been shown that when a message adversary is constrained by a property denoted TOUR (for tournament), the corresponding synchronous system and the asynchronous crash-prone read/write system have the same computability power for task solvability.

This paper introduces new message adversary properties (denoted SOURCE and QUORUM), and shows that the synchronous round-based systems whose adversaries are constrained by these properties are characterizations of classical asynchronous crash-prone systems (1) whose communication is through atomic read/write registers or point-to-point message-passing, and (2) enriched with failure detectors such as Ω and Σ. Hence these properties characterize maximal adversaries, in the sense that they define strongest message adversaries equating classical asynchronous crash-prone systems. They consequently provide strong relations linking round-based synchrony weakened by message adversaries with asynchrony enriched with failure detectors. This not only enriches our understanding of the synchrony/asynchrony duality, but also allows for the establishment of a meaningful hierarchy of property-constrained message adversaries.

Key-words: Asynchronous system, Distributed computability, Failure detector, Fair link, Message adversary, Message-passing model, Model equivalence, Ω, Process crash, Quorum, Read/write model, Round, Σ, Simulation, Source, Synchronous system, Task, Tournament, Wait-freedom

---

Systèmes synchrones affaiblis par des suppresseurs de messages

vs

systèmes asynchrones renforcés par des détecteurs de fautes

Résumé : *Un suppresseur de messages est une entité qui retire des messages dans un système synchrone à passage de messages dans lequel aucune défaillance ne survient. Les propriétés contraignant les suppresseurs de messages définissent les sous-ensembles de messages pouvant être retirés. Il a été récemment prouvé qu'un système synchrone dans lequel le suppresseur de messages est contraint par une propriété notée TOUR (pour tournoi) a la même puissance de calcul vis-à-vis des tâches qu'un système asynchrone sujet à des défaillances dans lequel les processus partagent de la mémoire.*

Ce rapport introduit de nouvelles propriétés pour contraindre les suppresseurs de messages (notées SOURCE et QUORUM), et montre que les systèmes asynchrones dans lesquels les suppresseurs de messages suivent ces propriétés sont des caractérisations des systèmes asynchrones (1) communicant par mémoire partagée ou par passage de message, (2) enrichis avec des detecteurs de fautes tels que Σ ou Ω. Ces propriétés enrichissent notre compréhension de la dualité synchrone/asynchrone mais permettent également l'établissement d'une hiérarchie au sein des propriétés caractérisant les suppresseurs de messages.

Mots clés : *systèmes asynchrones, calculabilité distribuée, détecteur de fautes, lien équitable, suppresseur de messages, modèle à passage de messages, équivalence de modèles, Ω, défaillances, quorum, mémoire partagée, ronde, Σ, simulation, source, système synchrone, tâche, tournoi, sans attente*

---

[*] Institut Universitaire de France
[**] ASAP : équipe commune avec l'Université de Rennes 1 et Inria

1 Introduction

Message adversaries for synchronous message-passing systems In a round-based message-passing synchronous system, processes communicate by exchanging messages at every round, and the synchrony assumption provided by the model guarantees that the messages sent at the beginning a round are received by their destination processes by the end of the corresponding round. Assuming that no process is faulty, the notion of a *message adversary* has been introduced in [21] (where it is called *mobile transmission failures*) to model messages losses and study their impact on the computability power of synchronous systems [21, 22].

Interestingly, the notion of constraining message deliveries has also been investigated in asynchronous systems, under distinct names, and in different contexts. As an example, asynchronous message patterns which allow failure detectors to be implemented despite asynchrony have been investigated in [9, 15]. The view of failure detectors as being schedulers which encapsulate fairness assumptions can also be related to this approach [6, 16]. Recently, assumptions on message deliveries and message exchange patterns have been used to define new asynchronous computation models and study their computability power [5, 12, 17, 23]. The general idea, which underlies these works, consists in capturing the "weakest pattern of information exchange" that allows a family of problems to be solved despite failures.

Notation The notation $\mathcal{SMP}_n[adv : \text{AD}]$ is used to denote a round-based synchronous system made up of n reliable sequential processes whose communications are under the control of the adversary AD. While, in every round, each process sends a message to each other process, the power of the adversary AD consists in suppressing some of these messages (which are consequently never received).

According to their power, several classes of adversaries can be defined. $\mathcal{SMP}_n[adv : \emptyset]$ denotes a synchronous system in which the adversary has no power (it can suppress no message), while $\mathcal{SMP}_n[adv : \infty]$ denotes the synchronous system in which the adversary can suppress all messages. It is easy to see that, from a message adversary and computability point of view, $\mathcal{SMP}_n[adv : \emptyset]$ is the most powerful crash-free synchronous system, while $\mathcal{SMP}_n[adv : \infty]$ is the weakest. More generally, the weaker the message adversary AD, the more powerful the system.

Asynchrony from synchrony Informally, a *task* is a one-shot distributed computing problem where each process has a private input, and each process has to compute a local output such that each output may depend on the whole vector of input values (this vector – initially unknown to each process – contains the input values of all the processes). The most famous and studied task is the consensus task.

Afek and Gafni addressed recently task solvability in synchronous message-passing systems weakened by message adversaries [1]. Let $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ denote the asynchronous read/write model where up to $(n-1)$ processes may crash ("$fd : \emptyset$" stands for "no failure detector", see below; this is the classical read/write wait-free model [10]). Afek and Gafni's main results are the following ones.

- Their first result concerns the adversary TOUR (for tournament) whose behavior is the following one. For each pair of processes p_i and p_j, and in each synchronous round, TOUR is allowed to suppress either the message sent by p_i to p_j or the message sent by p_j to p_i, but not both. The important result attached to TOUR is that $\mathcal{SMP}_n[adv : \text{TOUR}]$ and $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ have the same computability power for read/write wait-free solvable tasks.
- In addition to TOUR, two more adversaries, denoted TP and PAIRS, are described and it is shown that the three adversary-based synchronous models $\mathcal{SMP}_n[adv : \text{TOUR}]$, $\mathcal{SMP}_n[adv : \text{TP}]$, and $\mathcal{SMP}_n[adv : \text{PAIRS}]$ are equivalent for task solvability. Moreover, $\mathcal{SMP}_n[adv : \text{PAIRS}]$ is used to show that, from a topology point of view, the protocol complex of PAIRS is a subdivided complex. This means that the message adversary PAIRS (and consequently also TOUR and TP) captures, in a very simple way, Herlihy and Shavit's condition equating the read/write wait-free model with a complex subdivision [11].

Failure detectors for asynchronous crash-prone systems Informally, a failure detector is a device that provides each process p_i with information on failures [3]. According to the quality and the type of information they provide, several classes of failure detectors can be defined (see [19] for an introductory survey).

The failure detectors denoted Ω and Σ are among the most important failure detectors. This is due to the following reasons: (1) Ω is the weakest failure detector that allows consensus to be solved in $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ [4, 14]; (2) Σ is the weakest failure detector that allows an atomic register to be implemented on top of $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ [7], where $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ denotes the classical asynchronous message-passing system where up to $(n-1)$ processes may crash and every message is eventually received. "Weakest" means that any failure detector that allows to solve consensus (resp., implement a register) provides at least as much information on failures as the one provided by Ω (resp., Σ). Finally, the pair (Σ, Ω) is the weakest failure detector that allows consensus to be solved in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ [4, 7].

Let FD denote a failure detector. $\mathcal{ARW}_{n,n-1}[fd : \text{FD}]$ denotes the asynchronous read/write model where up to $(n-1)$ processes may crash, enriched with FD. Similarly, $\mathcal{AMP}_{n,n-1}[fd : \text{FD}]$ denotes $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ enriched with FD.

Content of the paper Following Afek and Gafni's seminal approach, the aim of this paper is to better understand and extend the message adversary approach, and capture its relations with asynchrony restricted by failure detectors. Considering the round-based synchronous message-passing model with reliable processes ($\mathcal{SMP}_n[adv : \emptyset]$) as core model, it has the following contributions, which concern (1) the crash-prone asynchronous read/write model, and (2) the crash-prone message-passing model, both enriched with failure detectors. Its contributions are described in the hierarchy depicted in Figure 1. $A \simeq_M B$ means that the computing model A can be simulated in the model B and vice-versa. $A \simeq_T B$ means that any task that can be solved in the model A, can be solved in the model B and vice-versa. An arrow from A to B means that the model A is stronger than the model B, but not vice-versa. These arrows follow from known results (e.g., $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ is stronger than both $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ and $\mathcal{AMP}_{n,n-1}[fd : \Omega]$). Let us observe that, as they are failure-free, the system models $\mathcal{SMP}_n[adv : \emptyset]$, $\mathcal{ARW}_{n,0}[fd : \emptyset]$, and $\mathcal{AMP}_{n,0}[fd : \emptyset]$, are computationally equivalent (first line of the figure).

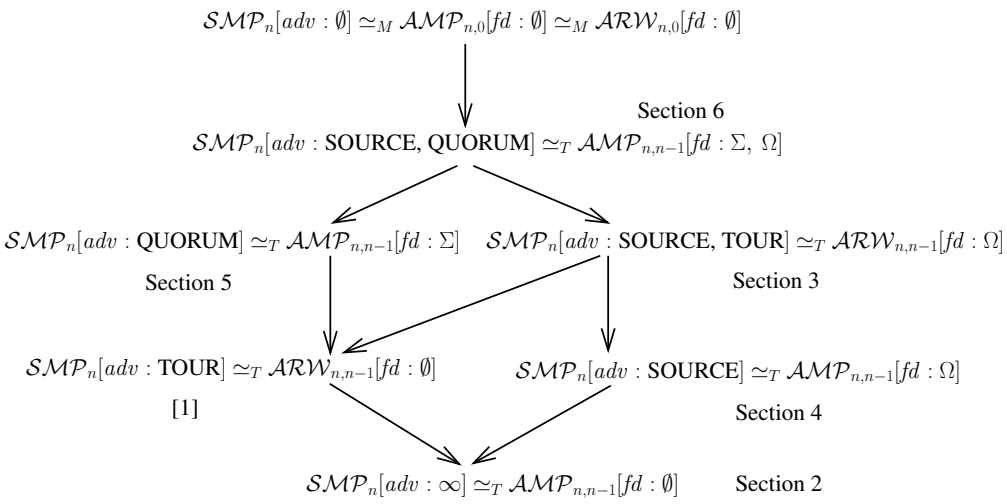

Figure 1: A message adversary hierarchy based on task equivalence and failure detectors

- Starting from the fact that the property TOUR (for tournament) captures the constraint on message delivery such that $\mathcal{SMP}_n[adv :$ TOUR$] \simeq_T \mathcal{ARW}_{n,n-1}[fd : \emptyset]$, Section 3 focuses on the properties of a message adversary which allow to enrich $\mathcal{SMP}_n[adv :$ TOUR$]$ to obtain $\mathcal{ARW}_{n,n-1}[fd : \Omega]$. To that end (1) it presents a new message delivery property, denoted SOURCE, and (2) shows that $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$ and $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ are equivalent for task solvability. It follows that SOURCE is a minimal requirement that has to be added to $\mathcal{SMP}_n[adv :$ TOUR$]$ in order to proceed from $\mathcal{SMP}_n[adv :$ TOUR$]$ to $\mathcal{ARW}_{n,n-1}[fd : \Omega]$.
- Then Section 4 shows that, by weakening $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$ into $\mathcal{SMP}_n[adv :$ SOURCE$]$, the resulting synchronous message-passing system is such that $\mathcal{SMP}_n[adv :$ SOURCE$] \simeq_T \mathcal{AMP}_{n,n-1}[fd : \Omega]$. It follows that SOURCE captures the weakest property on message delivery that an adversary AD has to satisfy so that any task solvable in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ can be solved in $\mathcal{SMP}_n[adv :$ AD$]$.[1] Said differently, when considering tasks solvability in crash-prone asynchronous systems enriched with Ω, what allows going from "message-passing" communication to "read/write" communication is characterized by the property TOUR from a message adversary point of view in a synchronous system (vertical arrow on the right of Figure 1).
- Then Section 5 focuses on a new message delivery property denoted QUORUM, and shows that a message adversary constrained by this property captures in $\mathcal{SMP}_n[adv : \emptyset]$ the same computability power (from a task point of view) as the one added by the failure detector Σ to $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$. To that end, it shows that $\mathcal{SMP}_n[adv :$ QUORUM$] \simeq_T \mathcal{AMP}_{n,n-1}[fd : \Sigma]$.
- Finally, as a consequence of the previous results, Section 6 shows that the properties SOURCE + QUORUM characterize the pair of failure detectors Σ + Ω, i.e., $\mathcal{SMP}_n[adv :$ SOURCE, QUORUM$] \simeq_T \mathcal{AMP}_{n,n-1}[fd : \Sigma, \Omega]$.

As just indicated, the paper provides message adversary-based characterizations of failure detectors for both read/write and message-passing crash-prone asynchronous systems. The aim of these results is to enrich our understanding of both message adversaries used to weaken communication in synchronous systems and failure detectors used to enrich asynchronous crash-prone

[1] As shown in [1] with the properties TOUR, TP, and PAIRS, several properties can be equivalent (i.e., each one can be implemented in $\mathcal{SMP}_n[adv : \emptyset]$ under the control of an adversary constrained by any other one). Hence, if a property P is the "weakest", so are the properties equivalent to P.

systems. They complement the results of [1] and exhibit strong intimate relations linking synchrony, message losses, and round-based model, on the one side, with asynchrony, process crashes, and failure detectors, on the other side. Interestingly, this seems to show that $\mathcal{SMP}_n[adv : \emptyset]$ (the base reliable synchronous round-based model) and the notion of a message adversary are central in the quest for a *Grand Unified* model of distributed computing.

Roadmap The paper is composed of 7 sections. Section 2 presents base models, message adversaries, and failure detectors. Section 3 introduces the property SOURCE on message deliveries, and show that it characterizes the failure detector Ω in read/write systems. Section 4 shows that, taken alone, the property SOURCE characterizes the failure detector Ω in asynchronous message-passing systems. Section 5 and Section 6 introduce the property QUORUM and show that it characterizes the failure detector Σ. Finally, Section 7 concludes the paper.

2 Models, Adversaries, Failure Detectors, Tasks

2.1 Base computation models

The base computation models relevant to this paper have been presented in the introduction. They are (1) the reliable round-based synchronous model $\mathcal{SMP}_n[adv : \emptyset]$ possibly weakened with a message adversary AD, and (b) the crash-prone asynchronous models $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ (*read/write wait-free* model [10]) and $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$, both possibly enriched with a failure detector FD.

2.2 Message Adversary, Message Graphs, and Dynamic Graphs in Synchronous Systems

Message adversary Given a run of a synchronous system, a message adversary suppresses messages sent by processes. A property associated with a message adversary restricts its power by specifying messages which cannot be suppressed. A message adversary is consequently defined by a set of properties which constrain its behavior.

Message graphs associated with each round of a synchronous system Given a message adversary AD, and a round r of a run of a synchronous system, let $\mathcal{G}^r$ be the directed graph (as defined in [1]), whose vertices are the process identities, and such that there is an edge from i to j iff the adversary AD does not suppress the message sent by p_i to p_j at round r. We consider the following definition associated with each graph $\mathcal{G}^r$.

- $i \stackrel{r}{\longrightarrow} j$ means that the directed edge (i, j) belongs to $\mathcal{G}^r$ (at round r, the message from p_i to p_j is not removed by the adversary).

The property TOUR As indicated in the introduction, the property TOUR [1] restricts the behavior of a message adversary as follows. For any r, and any pair of processes (p_i, p_j), $\mathcal{G}^r$ contains the directed edge (i, j) or the directed edge (j, i) or both. This means that, at every round, the adversary cannot suppress both the messages sent to each other by two processes. Hence, the graphs $\mathcal{G}^r$ associated with the rounds r of a run in $\mathcal{SMP}_n[adv : \text{TOUR}]$ are such that:

$$\forall r \geq 1 : \forall\, (i, j) : (i \stackrel{r}{\longrightarrow} j) \ \vee \ (j \stackrel{r}{\longrightarrow} i).$$

Strongly/weakly correct processes in a synchronous run The aim of this section is to introduce the notion of a *strongly correct* process which captures the processes whose an infinite number of messages are received (directly or indirectly) by any other process [20]. Such a notion is defined as follows.

- $i \stackrel{\geq r}{\rightsquigarrow} j$ means that there is a directed path starting from p_i and leading to p_j in a dynamically defined sequence of message graphs starting at a round $\geq r$. More formally,

$$\exists k \geq 0, \exists r_1 < \cdots < r_k, \ \exists \lambda_0, \lambda_1, \ldots, \lambda_k \in \{1, \ldots, n\} \ : \\ (r_1 \geq r) \wedge (\lambda_0 = i \wedge \lambda_k = j) \wedge (\forall m \in \{1, \ldots, k\} \ : \ \lambda_{m-1} \stackrel{r_m}{\longrightarrow} \lambda_m).$$

- $i \stackrel{\infty}{\rightsquigarrow} j \stackrel{def}{=} \big(\forall r > 0 \ : i \stackrel{\geq r}{\rightsquigarrow} j\big)$. Hence, $i \stackrel{\infty}{\rightsquigarrow} j$ means that, whatever r, there is eventually a directed path starting at p_i at a round $\geq r$ and finishing at p_j in the dynamically defined sequence of message graphs.
- $(i \stackrel{\infty}{\leftrightsquigarrow} j) \Leftrightarrow (i \stackrel{\infty}{\rightsquigarrow} j \wedge j \stackrel{\infty}{\rightsquigarrow} i)$. Assuming each process always receive its own messages, this relation is reflexive, symmetric, and transitive. Hence, it is an equivalence relation.

- Let G be the graph whose vertices are $\{1, ..., n\}$ and directed edges are defined by the relation $\stackrel{\infty}{\rightsquigarrow}$; let $SC(G)$ be the graph of its strongly connected components. If $SC(G)$ has a single vertex X with no input edge, the processes in X are called *strongly correct* processes, while the processes in $\{1, ..., n\} \setminus X$ are called *weakly correct*. If X is not unique, all processes are weakly correct.

Let $\mathcal{SC}$ denote the (possibly empty) set of strongly correct processes in a synchronous round-based system under the control of a message adversary.

2.3 Failure Detectors in Asynchronous Systems

While a message adversary weakens a synchronous round-based system (made up of reliable processes) by suppressing messages, a failure detector enriches an asynchronous system where no message is lost but where processes may suffer crash failures. Informally, a failure detector is a device that provides each process p_i with a read-only local variable xx_i containing (possibly unreliable) information on process crashes [3]. This paper considers two failure detectors. Let τ denote any time instant; xx_i^τ denotes the value of xx_i at time τ. This time notion, which is used in the definition of a failure detector, is not accessible to the processes. The identity of a process p_i is i. Given a run, a process that crashes is said to be *faulty* in that run, otherwise it is *correct*. Let $\mathcal{C}$ denote the sets of identities of the correct processes.

- Ω is called an *eventual leader* failure detector [4]. In the system models $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ or $\mathcal{AMP}_{n,n-1}[fd : \Omega]$, each process p_i is endowed with a local variable $xx_i = leader_i$ that always contains a (possibly changing) process identity. Moreover, there is an unknown but finite time τ and a process identity $\ell \in \mathcal{C}$ such that $\forall \tau' \geq \tau : (i \in \mathcal{C}) \Rightarrow (leader_i^{\tau'} = \ell)$.
- Σ is called a *quorum* failure detector [7]. In the system model $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$, each process p_i is endowed with a local variable $xx_i = qr_i$ that always contains a non-empty set of process identities and is such that (1) $\forall \tau, \tau', \forall i, j$: $qr_i^\tau \cap qr_j^{\tau'} \neq \emptyset$ (intersection property), and (2) $\forall i \in \mathcal{C} : \exists \tau : \forall \tau' \geq \tau : qr_i^{\tau'} \subseteq \mathcal{C}$ (liveness property).

2.4 Tasks

A task is a one-shot computation problem specified in terms of an input/output relation Δ. Each process starts with a private input value and must eventually computes a private output value. From an external observer point of view, an input vector $I[1..n]$ specifies the input value $I[i] = v_i$ of each process p_i. Similarly, an output vector $O[1..n]$ specifies a result value $O[j]$ for each process p_j.

A task is defined by a set of input vectors and a relation Δ which describes which output vectors are correct for each input vector I. More precisely, for each valid input vector I, the values computed by the processes must be such that there is an output vector $O \in \Delta(I)$ such that, for each j, $O[j]$ is the value computed by p_j; moreover, if no value is computed by p_j, it is because p_j has crashed during the computation. (A formal introduction to tasks can be found in [11].)

Theorem 1 *A task T can be solved in $\mathcal{SMP}_n[adv : \infty]$ iff it can be solved in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$.*

Proof Let us observe that if T can be solved in $\mathcal{SMP}_n[adv : \infty]$, it is a purely *local* task (each process can compute its result from its own input only [8][2]). It is possible to simulate on $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ an algorithm A solving a task T in $\mathcal{SMP}_n[adv : \infty]$, by sending no message. It follows that each process in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ which computes a result behaves as in $\mathcal{SMP}_n[adv : \infty]$ when the adversary removes all messages. As at least the correct processes of $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ computes a result, it follows that the task T is solved in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$.

If T can be solved in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$, it can be solved if all except one process crashed initially. From a process point of view, this cannot be distinguished from the case where messages are arbitrarily delayed. It follows from this observation and the fact that a process has to eventually computes a result that T can be solved without communication. Hence, it can be solved in $\mathcal{SMP}_n[adv : \infty]$. $\square_{Theorem\ 1}$

3 SOURCE + TOUR is a Characterization of Ω in $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$

This section shows that the computing models $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$ and $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ have the same computational power for tasks.

[2]This corresponds to a weakening of the system model $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ such that, from an operational point of view, a process computes its own result from its input and what it read from the memory before writing its input in the shared memory.

```
initialization:
(1)  r_i ← 0;
(2)  ls_state_i ← initial state of the local simulated algorithm;
(3)  msgs_to_send_i[1..n] ← initial messages to send to each process;
(4)  ∀r > 0 : MEM[i][r][1..n] init to [⊥, ..., ⊥].

repeat forever
(5)  r_i ← r_i + 1;
(6)  repeat leader_val_i ← MEM[leader_i][r_i][i]
(7)       until (leader_val_i ≠ ⊥) ∨ (leader_i = i)
(8)  end repeat;
(9)  MEM[i][r_i] ← msgs_to_send_i;
(10) rec_msgs_i[1..n] ← MEM[1..n][r_i][i];
(11) (msgs_to_send_i, ls_state_i) ← simulate(ls_state_i, rec_msgs_i)
end repeat.
```

Figure 2: From $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ to $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$

3.1 The Property SOURCE

This property is defined as follows:

$$\exists s \in \{1,\ldots,n\} : \exists r_0 \geq 1 : \ \forall r \geq r_0 : \ \forall\, i \in \{1,\ldots,n\} : \ (s \xrightarrow{r} i).$$

This statement means that, in each run of $\mathcal{SMP}_n[adv : \text{SOURCE}]$, there are a process p_s and a round r_0, such that, at every round $r \geq r_0$, the adversary does not suppress the message sent by p_s to the other processes.

3.2 From $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ to $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$

This section presents a simulation of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ on top of $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ such that, any task that can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$.

Global and local variables of the simulation The simulation uses a shared variable $MEM[1..n][1..][1..n]$ where $MEM[i][r][j]$ is an atomic read/write register written by p_i and read by p_j. This register contains the message sent by p_i to p_j in round r of the simulation of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$; $\perp$ is a default value used to indicate that no message has yet been written or the corresponding message has been suppressed by the adversary.

The local variable r_i simulates the current round number of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, while ls_state_i represents the local simulation state. The local variable $msgs\_to\_send_i[1..n]$ contains the messages that p_i will send to each other process during the next simulated round ($msgs\_to\_send_i[j]$ contains the message for p_j). $leader_i$ is the read-only local variable containing the current local output of Ω.

The simulation is locally defined by the function simulate() which takes as input parameters the current local state of the simulation and the messages received from the other processes at the current round. It modifies accordingly the local simulation state and computes the messages that will be sent to the other processes during the next round.

The simulation algorithm The local simulation algorithm is described in Figure 2. The local simulator of process p_i first proceeds to the next round (line 5) and waits until its current leader has sent it a message ($MEM[leader_i][r_i][i] \neq \perp$) or it is its own leader (lines 6-8). When this occurs, the simulator writes in $MEM[i][r_i]$ the messages sent by p_i at the current round (line 9). Then, p_i consumes messages (line 10), and uses them to modify its local simulation state and compute the message it will send during the next round (line 11).

Lemma 1 *If a task can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, it can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$.*

Proof Let us consider a simulated process p_i (i.e., p_i executes in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$). When there is no ambiguity, we use the same identifier p_i, for a simulated process and its simulator.

Let us first observe that no correct simulator p_i can block forever in the loop of lines 6-8. This is an immediate consequence of the eventual leadership property of Ω (the eventually elected process p_ℓ cannot block forever), and the fact that it writes its messages in $MEM[\ell][r][1..n]$.

Let us show that the tournament property TOUR is satisfied at every round. Let us consider two processes that terminates a round $r \geq 0$. It follows from lines 9-10 that (1) p_i has written a message into $MEM[i][r][j]$ and then read $MEM[j][r][i]$, while (2) p_j has

written a message in $MEM[j][r][i]$ and then read $MEM[i][r][j]$. As registers are atomic, it follows that either p_i has written into $MEM[i][r][j]$ before p_j has written into $MEM[j][r][i]$, or the opposite. Whatever the case, as each process writes before reading, at least one of them reads the message from the other, and consequently $\mathcal{G}^r$ contains (i,j) or (j,i). Let us now consider a process p_j whose simulator crashes during the execution. From the point of view of any process p_i whose simulator is correct, everything appears as if, after the simulator of p_j has crashed, the simulated adversary removes all the messages sent by p_j to p_i and keeps the messages sent by p_i to p_j. Hence, if p_j crashes after round r, we have $(i,j) \in \mathcal{G}^{r'}$ at any round $r' > r$.

Finally, let us consider a time after which all the correct simulators have forever the same correct leader p_ℓ and no more simulator crashes. It follows from lines 6-8, that there is a round r such that, at any round $r' \geq r$, any correct process p_i receives the message sent by p_ℓ. Moreover, crashed processes receive implicitly all messages sent by p_ℓ. It follows that we have $(\ell,i) \in \mathcal{G}^{r'}$ which establishes the SOURCE property of the adversary.

It follows from the previous arguments that, if the task can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, it can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$. A process with a correct simulator behaves the same way in both models, and a process with a faulty simulator either computes a correct output value or crashes before it has computed an output value (in this case, its entry in the output vector contains $\perp$). $\square_{Lemma\ 1}$

3.3 From $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ to $\mathcal{ARW}_{n,n-1}[fd : \Omega]$

This section presents a simulation of $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ on top of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ such that, any task that can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$. This simulation has the same structure as the simulation of $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ on top of $\mathcal{SMP}_n[adv : \emptyset]$ described in [1]. Basically, it adds to it the management of the local variables $missed_i$ (defined below) from which Ω is extracted.

Global and local variables of the simulation The shared memory of $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ is made up of an array of single-writer/multi-reader atomic registers $MEM[1..n]$ such that only p_i can write $MEM[i]$. The simulation associates a sequence number with each read or write operation of a simulated process p_i. To simplify notations, a read of $MEM[\ell]$ by p_i is denoted $\mathsf{read}_i(\ell)$ and a write of v into $MEM[i]$ is denoted $\mathsf{write}_i(v)$.

As in the previous simulation, the procedure $\mathsf{simulate}()$ is used to locally simulate the behavior of p_i from its current step until its next invocation of a communication operation (i.e., a read or a write of the simulated shared memory). The simulation stops just before this invocation. It takes as input parameters the current local state of p_i (ls_state_i) and the last value read from the shared memory by p_i. This value, saved in $read\_value_i$ (and initialized to $\perp$), is meaningless if the operation is a write. The local variable $next\_op_i$ contains p_i's next read or write operation to be simulated.

The local variable $view_i$ contains all the read/write operations issued by the processes and known by p_i. Such an operation is represented by a triple $(j, seq\_nb, next\_op)$. The simulation algorithm is a full information algorithm and consequently the set $view_i$ increases forever.

The local variable $informed_i$ contains the set of processes which, to p_i knowledge, know the last read/write operation it is currently simulating. Finally, the set $missed_i$ (from which Ω is built) contains pairs (k,r) whose meaning is the following: $((k,r) \in missed_i) \Rightarrow$ there is at least one process that, during round r of the simulation, has not received and delivered the message sent by (the simulator of) p_k during that round.

The simulation algorithm The simulation algorithm is described in Figure 3. When it starts a new round, the simulator of p_i sends its control local state, i.e., the triple $(i, view_i, missed_i)$ to each other process (line 5). Then (lines 6-10), it considers all the messages it has received during the current round r, and updates accordingly rec_msg_i and $missed_i$.

Lines 11-12 locally implement Ω (see below). The variable $informed_i$ is then updated to take into account what has been learned from the messages just received. Let us notice (line 13) that it follows from TOUR that $(j \notin rec\_from_i) \Rightarrow p_j$ has received p_i's round r message.

Then (the simulator of) p_i executes rounds in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ until it learns that (the simulators of) all the processes know its last read/write operation (line 15). Then, (line 16) it invokes $\mathsf{simulate}(ls\_state_i, read\_value_i)$. If its (simulated) shared memory operation is a read, the value $read\_value_i$ is the value obtained by this read operation. Otherwise (the simulated operation is a write of p_i), $read\_value_i$ is useless. As already indicated, the invocation of $\mathsf{simulate}(ls\_state_i, read\_value_i)$ simulates then the behavior of p_i until its next read/write operation.

If the operation at which the local simulation stopped is a read of $MEM[\ell]$ (line 17), the local simulator computes, and deposits in $read\_value_i$, the value that will be associated with this read (line 18-22). If p_ℓ has not issued a write, $read\_value_i$ is set to the default value $\perp$ (line 19). Otherwise, $read\_value_i$ is set to the last value written by p_ℓ (line 18-21). Then, whatever the next operation (read or write) of p_i, the local simulator associates a sequence number with it and adds the triple $(i, seq\_nb_i, next\_op_i)$ to

```
initialization:
(1)  ls_state_i ← initial state of the local simulated algorithm; read_value_i ← ⊥;
(2)  (next_op_i, ls_state_i) ← simulate(local_sim_state_i, read_value_i);
(3)  seq_nb_i ← 1; informed_i ← {i}; missed_i ← ∅;
(4)  view_i ← {(i, seq_nb_i, next_op_i)}.

round r = 1, 2, ··· do:
(5)  send(i, view_i, missed_i) to each other process;
(6)  rec_msgs_i ← set of triples (j, view_j, missed_j) received during this round;
(7)  view_i ← view_i ∪ (⋃_{(j,view_j,missed_j)∈rec_msgs_i} view_j);
(8)  missed_i ← missed_i ∪ (⋃_{(j,view_j,missed_j)∈rec_msgs_i} missed_j);
(9)  rec_from_i ← {j ∈ {1,...,n} : ∃(j, view_j, missed_j) ∈ rec_msgs_i} ∪ {i};
(10) missed_i ← missed_i ∪ {(k, r) : k ∈ {1,...,n} \ rec_from_i};
(11) min_missed_i ← min{|{r : (j, r) ∈ missed_i}|, j ∈ {1,...,n}};
(12) ℓd_i ← min{j : |{r : (j, r) ∈ missed_i}| = min_missed_i};
(13) informed_i ← informed_i ∪ ({1,...,n} \ rec_from_i)
(14)                          ∪{j ∈ rec_from_i : (i, seq_nq_i, next_op_i) ∈ view_j};
(15) if (informed_i = {1,...,n}) then
(16)   (next_op_i, ls_state_i) ← simulate(ls_state_i, read_value_i);
(17)   if (next_op_i = read_i(ℓ)) then
(18)     if (∄(ℓ, −, write_ℓ(−)) ∈ view_i)
(19)       then read_value_i ← ⊥
(20)       else max_snℓ_i ← max{sn_ℓ, (ℓ, sn_ℓ, write_ℓ(−)) ∈ view_i};
(21)            read_value_i ← v_ℓ : (ℓ, max_snℓ_i, write_ℓ(v_ℓ)) ∈ view_i
(22)     end if
(23)   end if;
(24)   seq_nb_i ← seq_nb_i + 1; informed_i ← {i};
(25)   view_i ← view_i ∪ {(i, seq_nb_i, next_op_i)}
(26) end if.

when leader_i is read: return (ℓd_i).
```

Figure 3: Simulation of $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ in $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$

$view_i$ (line 24-25). Moreover, as its scope is the simulation of $next\_op_i$, the set $informed_i$ is reset to $\{i\}$.

As previously indicated, the current value (kept in ℓd_i) of the read-only variable $leader_i$, which locally implements Ω, is computed from the set $missed_i$ at lines 11-12. The simulator of p_i (1) computes, for each p_j, the set of rounds at which at least one simulator has not received the round r message sent by p_j's simulator (these are messages suppressed by the adversary); then (2) it associates with each p_j the cardinality of the previous set; and finally, (3) it considers the process p_ℓ for which the adversary has suppressed the less messages (if there are several such processes, ties are solved by using the total order on process identities).

Lemma 2 *If a task can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$, it can be solved in $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$.*

Proof The proof consists of four parts: (1) the simulation is non-blocking; (2) the definition of which are the correct/faulty processes in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$; (3) the definition of the linearization of the read and write operations; and (4) the fact that local variables $leader_i$ implement Ω.

Part 1: the simulation is non-blocking.
Given that the simulation algorithm is a full information algorithm, the "*king in two tournaments*" Theorem [2][3] states that at least one read/write operation issued by a simulated process p_i is known by all simulators in two simulation rounds (i.e., there is a simulator p_i such that, for any j, we have $next\_op_i \in view_j$ in at most two rounds).

Let us consider three consecutive simulation rounds r, $r+1$ and $r+2$, and p_i a simulator whose message $(i, view_i, missed_i)$ sent at line 5 has reached (directly or indirectly) all the process simulators by the end of round $r+1$. As (due to TOUR), $\mathcal{G}^{r+2}$ contains a tournament, we have one of the following for each $j \neq i$: (a) $j \xrightarrow{r+2} i$ and in that case, p_i receives its own triple $(i, seq\_nb_i, next\_op_i)$ from p_j and consequently it knows that p_j knows its triple $(i, seq\_nq_i, next\_op_i)$ (line 14); or (b) $\neg(j \xrightarrow{r+2} i)$ and in that case, we necessarily have $i \xrightarrow{r+2} j$ and consequently p_i knows that p_j knows its triple. It follows that, at the end of the simulation round $r+2$, p_i terminates the simulation of its read or write operation $next\_op_i$ (line 15-line 23). As there is a bounded number of processes,

[3]This theorem extends a theorem on graph tournament by Landau [13] to the case where consecutive tournaments can be different.

there is at least one process that eventually executes until it computes its local result. It follows that the simulation is non-blocking.

Part 2: correct and faulty processes in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$.
As each message graph $\mathcal{G}^r$ contains a tournament, it follows that the relation $\overset{\infty}{\rightsquigarrow}$ (introduced in Section 2.2) defines a total order on the equivalence classes of the relation $\overset{\infty}{\leftrightsquigarrow}$. Hence, there is a single set X of strongly correct process simulators (i.e., X has no input edge in $SC(G)$). This set X contains exactly all the process simulators whose messages sent at line 5 are always eventually received (at some round, directly or indirectly) by all other process simulators.

It follows from the previous reasoning (Part 1) on the fact that the simulation is non-blocking, and the condition $informed_i = \{1, ...n\}$ (line 15), that each process in X simulates any number of its operations. Hence, those processes are correct in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$. Differently, for each process p_j such that $j \in \{1, ..., n\} \setminus X$, there is a round from which the predicate $informed_j = \{1, ...n\}$ is never satisfied, and consequently the simulation of the process p_j stops progressing. Hence, each weakly correct simulator p_j such that $j \in \{1, ..., n\} \setminus X$ simulates a faulty process in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$.

Part 3: definition of the linearization of the read and write operations.
A write operation is linearized at the first of the two following time instants: (1) τ_1 the end of the simulation of this write operation (i.e., when the condition line 15 is satisfied by the associated simulator), and (2) τ_2 the time instant just before the linearization point of the simulation of the first read operation returning the written value. If none of these two instants ever happen, then the write is never linearized and the corresponding simulated process appears as crashed.

The linearization of a read operation is close to but different from the one of a write operation. A read operation is linearized at the first of these two time instants: (1) τ_1 the end of this read operation simulation (when the condition line 15 is satisfied by the associated simulator), and (2) τ_2 the time instant just before the linearization point of the simulation of the first write operation which overwrites the read value. However, if the instant τ_1 never happen, then the read is never linearized (even if τ_2 exists) and the simulated process appears as crashed. (Remark that, thanks to the fact that a simulator selects the freshest value it knows to prepare the value returned by a simulated read operation, and to the fact that, at the instant of the linearization of a write (or read) operation, all processes have the corresponding written (or read) value in their views, the read value cannot have been effectively overwritten before the beginning of the read operation returning it.)

As the reading (resp. overwriting) of a written (read) value cannot occur neither before the start of the corresponding write (read) operation, the linearization point of this operation occurs after its the start of the operation. Moreover, the selection of the first among τ_1 and τ_2 for both types of operation implies that the linearization point occurs at the latest at end of its simulation, and that (a) a read is always linearized after the writing of the value it returns, (b) the overwriting of a value is always linearized after all read operations that return it.

Part 4: the local variables $leader_i$ implement Ω.
It follows from the property SOURCE that there is a process p_s and a round r_0 such that, from r_0, no message from p_s is removed by the adversary. In particular, after some round $r_s \geq r_0$, all the messages sent by (the simulator of) p_s are received by all the strongly correct processes. Let $S \subseteq X$ be the set of processes p_s satisfying this property (eventually all their messages are –directly– received by all the strongly correct processes). Let $r_S = \max\{r_s : s \in S\}$. (Let us notice that an arbitrary number of messages from the processes in S to processes which are not strongly correct can be suppressed by the adversary.)

As, at any round $r \geq r_S$, no message from (the simulator of) a process in S to (the simulator of) a strongly correct process is suppressed, no simulator of a strongly correct process p_i adds a pair (s, r), $s \in S$, to its set $missed_i$ (line 10).

Let us observe that: (a) the simulator of any process p_i adds all the set $missed$ it receives to its own set $missed_i$ (line 8); (b) due to the definition of "strongly correct" simulator, messages are eventually propagated (directly or indirectly) in each direction between each pair of strongly correct simulators; and (c) after some finite time, no strongly correct simulator receive message from a weakly correct simulator.

It follows from the previous observations that, for each $s \in S$, the values $|\{r : (r, s) \in missed_i\}|$ eventually stabilize at the same finite value at each strongly correct simulator p_i, while, for each $j \in \{1, ..., n\} \setminus S$, the value $|r : (r, j) \in missed_i\}|$ never stops increasing. Hence, the same eventual leader is elected at all strongly correct processes by the code of lines 11-12, and consequently the correct simulated processes inherits the same eventual leader. Moreover, as the process simulators in S are strongly correct, the elected leader is a process which is correct in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$. $\square_{Lemma\ 2}$

3.4 SOURCE + TOUR is a Characterization of Ω in $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$

Theorem 2 *A task can be solved in $\mathcal{SMP}_n[adv :$ SOURCE, TOUR$]$ iff it can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$.*

Proof The proof follows immediately from Lemma 1 and Lemma 2. $\square_{Theorem\ 2}$

Remark Let us remark that it is not possible to conclude from the previous theorem and the fact that Ω is the weakest failure detector to solve consensus in $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$, that the property SOURCE + TOUR defines a weakest message adversary AD allowing consensus to be solved in $\mathcal{SMP}_n[adv : \text{AD}]$. It remains possible that a property AD weaker than SOURCE + TOUR allows consensus to be solved in $\mathcal{SMP}_n[adv : \text{AD}]$. Said differently nothing allows us to claim that the "granularity on the properties which can be defined to constrain message adversaries" is the same as the "granularity on the information on failures" provided by failure detectors.

Let CONS be the minimal message adversary property that allows consensus to be solved in $\mathcal{SMP}_n[adv : \text{CONS}]$. As consensus is solvable in $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, it follows that $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ is at least as powerful as $\mathcal{SMP}_n[adv : \text{CONS}]$. On another side, as (1) a read/write register can be implemented from consensus, and (2) consensus cannot be solved in $\mathcal{SMP}_n[adv : \text{TOUR}]$, it follows that $\mathcal{SMP}_n[adv : \text{CONS}]$ is strictly more powerful than $\mathcal{SMP}_n[adv : \text{TOUR}]$.

4 SOURCE is a Characterization of Ω in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$

4.1 From $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ to $\mathcal{SMP}_n[adv : \text{SOURCE}]$

The algorithm described in Figure 4 presents a simulation (for tasks) of $\mathcal{SMP}_n[adv : \text{SOURCE}]$ on top of $\mathcal{AMP}_{n,n-1}[fd : \Omega]$. Its principles are close to the ones of the simulation of Figure 2. The algorithm ensures that the eventual leader p_ℓ satisfies the property SOURCE. Hence, there are strongly correct processes and the eventual leader is one of them. The aim of the simulation algorithm is then to eventually withdraw all the messages except the ones from the leader.

Local variables of the simulation As in the previous simulations, r_i is the locally simulated round number; $msgs\_to\_send_i[j]$ (initialized to $\perp$) contains the next simulated message to be sent to p_j; $rec\_msg_i[r]$ contains the simulated messages received at round r; $sim\_rec\_msgs_i[x]$ contains the message received from the process p_x currently considered as the leader by p_i; $leader_i$ is the read-only variable provided by Ω.

```
(1)  r_i ← 0; sim_rec_msgs_i[1,...,n] ← [⊥,...,⊥];
(2)  (msgs_to_send_i[1,...,n], ls_state_i) ← simulate(sim_rec_msgs_i);
(3)  for each r > 0 do rec_msgs_i[r][1,...,n] ← [⊥,...,⊥] end for;
(4)  repeat forever
(5)    r←r_i + 1;
(6)    for each j ∈ {1,...,n} do send(r_i, msgs_to_send_i[j]) to p_j end for;
(7)    repeat cur_ℓd_i ← leader_i
(8)      until (cur_ℓd_i = i ∨ rec_msgs_i[r_i][cur_ℓd_i] ≠ ⊥)
(9)    end repeat;
(10)   sim_rec_msgs_i[cur_ℓd_i] ← rec_msgs_i[r_i][cur_ℓd_i];
(11)   (msgs_to_send_i[1,...,n], ls_state_i) ← simulate(sim_rec_msgs_i);
(12)   sim_rec_msgs_i[1,...,n] ← [⊥,...,⊥]
(13) end repeat.

when (r, m) received from p_j:  rec_msgs_i[r][j] ← m.
```

Figure 4: From $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ to $\mathcal{SMP}_n[adv : \text{SOURCE}]$

The simulation algorithm The procedure simulate() takes as input parameter the simulated messages received by p_i at the current round, and simulates the local algorithm until the next sending of messages by p_i. This procedure returns the simulated messages to be sent at the beginning of the next round.

After the initialization stage (lines 1-3), the local simulator of p_i enters a loop whose each body execution simulates a round of the synchronous system. It first sends the messages that p_i has to send at the current round (line 6). Then it waits until it has received a message from its current leader or it is its own leader (lines 7-9). When this occurs, it retrieves the message sent by its current leader (line 10) and invokes the procedure simulate() with this message as input parameter, before proceeding to the simulation of the next synchronous round.

Lemma 3 *If a task can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE}]$, it can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$.*

Proof Let us first show that no correct process remains blocked forever in the loop lines of 7-9. Indeed, there is a finite time τ after which an eventual leader (say p_ℓ) is elected by Ω at each process. It then follows from the first part of the predicate of line 8 that p_ℓ cannot remain blocked at line 8, and consequently executes rounds forever. Moreover, as its messages are eventually received at

each round by all correct processes, it follows that there is a time after which the second part of the predicate of line 8 is always satisfied by these processes. Consequently, none of them can remain blocked forever at line 8.

The previous reasoning shows also that the eventual leader elected by Ω behaves as a source, and consequently the property SOURCE is satisfied in the simulated synchronous system. $\square_{Lemma\ 3}$

4.2 From $\mathcal{SMP}_n[adv : \text{SOURCE}]$ to $\mathcal{AMP}_{n,n-1}[fd : \Omega]$

The simulation algorithm is described in Figure 5. It is similar to the algorithm of Figure 3 (which simulates $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ on top of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$).

Local variables of the simulation The local variables ls_state_i, $view_i$, rec_from_i, and $missed_i$ have the same meaning as in Figure 3. The local variable $msgs\_to\_rec_i$ contains messages to be consumed by the simulated process (it corresponds to $read\_value_i$ in Figure 3). The variable $msgs\_to\_send_i$ contains the messages to be sent in the next simulation round (it corresponds to $next\_op_i$ in Figure 3). The variable $msgs\_received_i$ is a new variable containing the messages already received by the simulated process p_i. Finally, ℓd_i is the local variable containing the current local value of Ω built by the algorithm.

The simulation algorithm As in the simulation of Figure 3, lines 1-4 are an initialization stage. Similarly to previous simulations, the procedure simulate() locally simulates the process p_i. It takes messages to be consumed by p_i as input parameter and returns the next set of messages to be sent.

The simulation algorithm is a full information algorithm. During each simulation round r, the simulator of p_i first sends its control local state to each other process, and waits for the same information from them (lines 5-6). Then, according to the messages it has received during the current round, it updates $view_i$, $missed_i$, and rec_from_i (lines 7-10). As in Figure 3, it also computes the identity ℓd_i of its current candidate to be the eventual leader (lines 11-12).

If a simulation message has been received from the process $p_{\ell d_i}$, the simulator of p_i strives to make p_i progress. It considers the last message sent by $p_{\ell d_i}$ to p_i (triple $(\ell d_i, i, m)$), and adds it to the set $msgs\_to\_rec_i$ (lines 14-15). Then, if the messages p_i has to send are known by its current leader $p_{\ell d_i}$ (line 16), the procedure simulate() is invoked to make p_i progress (line 17), and the local control variables $msgs\_received_i$ and $view_i$ are updated accordingly (line 18).

initialization:
(1) $ls\_state_i \leftarrow$ initial state of the local simulated algorithm;
(2) $msgs\_to\_rec_i \leftarrow \emptyset$; $msgs\_received_i \leftarrow \emptyset$;
(3) $(msgs\_to\_send_i, ls\_state_i) \leftarrow \mathsf{simulate}(ls\_state_i, msgs\_to\_rec_i)$;
(4) $view_i \leftarrow msgs\_to\_send_i$; $missed_i \leftarrow \emptyset$; $\ell d_i \leftarrow i$.

round $r = 1, 2, \cdots$ **do**:
(5) send$(i, view_i, missed_i)$ to each other process;
(6) $rec\_msgs_i \leftarrow$ set of triples $(j, view_j, missed_j)$ received during this round;
(7) $view_i \leftarrow view_i \cup \Big(\bigcup_{(j,view_j,missed_j)\in rec\_msgs_i} view_j\Big)$;
(8) $missed_i \leftarrow missed_i \cup \Big(\bigcup_{(j,view_j,missed_j)\in rec\_msgs_i} missed_j\Big)$;
(9) $rec\_from_i \leftarrow \{j \in \{1,\ldots,n\} : \exists (j, view_j, missed_j) \in rec\_msgs_i\} \cup \{i\}$;
(10) $missed_i \leftarrow missed_i \cup \{(k, r) : k \in \{1,\ldots,n\} \setminus rec\_from_i\}$;
(11) $min\_missed_i \leftarrow \mathsf{min}\{|\{r : (j, r) \in missed_i\}|, j \in \{1,\ldots,n\}\}$;
(12) $\ell d_i \leftarrow \mathsf{min}\{j : |\{r : (j, r) \in missed_i\}| = min\_missed_i\}$;
(13) **if** $(\ell d_i \in rec\_from_i)$ **then**
(14) **let** $view_{\ell d_i}$ **be such that** $(\ell d_i, view_{\ell d_i}, missed_{\ell d_i}) \in rec\_msgs_i$;
(15) $msgs\_to\_rec_i \leftarrow msgs\_to\_rec_i \cup \{(\ell d_i, i, m) : (\ell d_i, i, m) \in view_{\ell d_i}\}$;
(16) **if** $(msgs\_to\_send_i \subseteq view_{\ell d_i})$ **then**
(17) $(msgs\_to\_send_i, ls\_state_i) \leftarrow \mathsf{simulate}(ls\_state_i, msgs\_to\_rec_i \setminus msgs\_received_i)$;
(18) $msgs\_received_i \leftarrow msgs\_to\_rec_i$; $view_i \leftarrow view_i \cup msgs\_to\_send_i$
(19) **end if**
(20) **end if**.

when $leader_i$ **is read**: return (ℓd_i).

Figure 5: Simulation of $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ in $\mathcal{SMP}_n[adv : \text{SOURCE}]$

Lemma 4 *If a task can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$, it can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE}]$.*

Proof Preliminary definition on simulators in $\mathcal{SMP}_n[adv : \text{SOURCE}]$.
Let S be the set of processes which satisfy the property SOURCE. As, by assumption there at least one source, we have $S \neq \emptyset$. Moreover, due the definition of the set $\mathcal{SC}$ of strongly correct simulators we have $S \subseteq \mathcal{SC}$. Let S' be the set of processes which, albeit they are not necessarily source, appear as sources to all processes of $\mathcal{SC}$. Hence we have $S \subseteq S' \subseteq \mathcal{SC}$, and $S' \neq \emptyset$.

The variables $leader_i$ implement Ω.
According to the definition of $\mathcal{SC}$, there is a round r_0 after which no more message from a weakly correct simulator is received (directly or indirectly) by a strongly correct simulator. Let $r_1 = \max\{r_s, s \in S'\}$ where r_s is the first round after which no message sent by p_s to a strongly correct simulator is eliminated. As, after r_1, each strongly correct simulator receives at every round a message from each simulator in S', it follows that none of them adds a pair $(s, r), r \geq r_1, s \in S'$ in its variable $missed_i$ at line 10. After $r_2 = \max\{r_0, r_1\}$, the only pairs $(s, r), s \in S'$ $(r < r_1)$ that are added by a strongly correct simulator in its variable $missed_i$ are those that have been added by other strongly correct simulators at line 8 or line 11 before r_2. Since strongly correct simulators are infinitely often able to transmit (directly or not) messages to each other, there is a round $r_3 \geq r_2$ such that any strongly correct simulator p_i has received (directly or not) during a round $rj \geq r_2$ the information contained in the variable $missed_j$ from each other strongly correct simulator p_j. After r_3, for any $s \in S'$, the number of pairs (s, r) in the variables $missed_i$ of all strongly correct simulators p_i is the same and does not increase anymore.

For each simulator p_i, $i \notin S'$, there is an infinite number of rounds r such that p_i's message is not received during round r by at least one of the strongly correct simulator p_j, and accordingly, this simulator adds a pair (i, r) to its variable $missed_j$ during round r at line 10. As the strongly correct simulators communicate (directly or not) infinitely often with each other, all of them eventually add this pair to their variable $missed$ during r (at line 10) or later (at line 8). Consequently, for each such simulator p_i, $i \notin S'$, the number of pairs (i, r) in the variable $missed_j$ of every strongly simulator p_j increases forever.

It follows from the previous discussion that the minimal number of rounds missed by a simulator (as calculated at line 12, and using simulator identity to do tie-breaking) eventually becomes and remains the same at each strongly correct simulator. Let ℓd denote this simulator identity. As it is the identity that is eventually always returned when $leader_i$ is read by any simulated process p_i whose simulator is strongly correct, the unicity eventual property of Ω is ensured for these processes. The next paragraph shows that the set of strongly correct simulators corresponds exactly to the set of correct simulated processes. As $p_{\ell d}$ is strongly correct, the elected process is a correct process, which concludes the proof of Ω.

Correct and faulty (simulated) processes in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$.
It follows from the previous paragraphs that each strongly correct simulator p_i is always eventually able to transmit (directly or not) a new message m to $p_{\ell d}$, and then eventually receive (directly) a message from $p_{\ell d}$ containing m. Hence the conditions of line 13 and line 16 are fulfilled an infinite number of times and, consequently, the corresponding simulator can always issue enough steps (line 17) to progress in the simulated code.

Hence, the correct simulated processes and the faulty simulated processes are the ones simulated by the strongly correct and weakly correct simulators, respectively. It follows that

Linearization of communication operations.
Let us consider a simulated process p_i that sends a message m to a simulated process p_j. This operation is disseminated to each simulator by p_i's simulator at line 5. Then a simulator considers this simulated message m only at line 17 when the second input parameter of its invocation of simulate() contains the message m. (Let us observe, that this message m arrives at a simulator p_k from its current leader ℓd_k, lines 13 and 16).

Let τ_1 be the time of the first invocation of simulate() by a simulator such that m belongs to the second input parameter of this invocation, where $\tau_1 = \infty$ if there is no such invocation. Let τ_2 be the time at which the simulator of p_i starts the execution of simulate() (line 17) after it has disseminated m, where $\tau_2 = \infty$ if there is no such invocation.

The send of m is linearized at time $\min(\tau_1, \tau_2)$ (let us notice that the simulation of p_i does not progress between the sending of m by p_i and its linearization point). If $\min(\tau_1, \tau_2) = \infty$, the send of m is linearized after the receiver p_j has computed its result.

The reception of m is linearized at the time of the invocation by p_j of simulate() whose second input parameter contains the message m, or after p_j has computed its result if there is no such invocation. $\square_{Lemma\ 4}$

4.3 SOURCE is a characterization of Ω in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$

Theorem 3 *A task can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ iff it can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE}, \text{FAIR}]$.*

Proof The proof follows directly from Lemma 3 and Lemma 4. $\square_{Theorem\ 3}$

5 QUORUM is a Characterization of Σ in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$

This section shows that the computing models $\mathcal{SMP}_n[adv : \text{QUORUM}]$ and $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ have the same computational power for tasks.

5.1 The Property QUORUM

Let us remember that $\mathcal{SC}$ is the set of strongly correct processes in the considered synchronous message-passing system (processes whose an infinite number of messages are received by each other process). The property QUORUM is defined as follows:

$$\big[\forall\, i,j : \forall\, r_i, r_j : (\{k : k \xrightarrow{r_i} i\} \cap \{k : k \xrightarrow{r_j} j\} \neq \emptyset)\big] \ \wedge\ (\mathcal{SC} \neq \emptyset).$$

This property is a statement of Σ suited to the context of round-based synchronous message-passing systems prone to message adversaries. Given any pair of processes p_i and p_j, its first part states that, whatever the synchronous rounds r_i and r_j executed by p_i and p_j, respectively, there is a process p_k whose messages to p_i at round r_i and to p_j at round r_j are not eliminated by the adversary (intersection property). The second part states that there is at least one process whose messages are infinitely often received by each other process (liveness property). Theorem 4 will show that this formulation of Σ is correct for the equivalence of $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ and $\mathcal{SMP}_n[adv : \text{QUORUM}]$ for task solvability.

5.2 From $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ to $\mathcal{SMP}_n[adv : \text{QUORUM}]$

The simulation algorithm described in Figure 6. It has the same local variables as, and is very close to, the one of Figure 4. In addition to the local output of the failure detector Σ, which is denoted qr_i, the only modifications are the lines 7-10 which differ in both algorithms.

The simulator of p_i waits until it has received a message from each process that appears in its current quorum qr_i (lines 7-9). It then invokes the procedure simulate() with these messages as input (line 10).

The principle of this simulation is the following: after some time, the simulated message adversary suppresses all the messages sent by processes that do not belong to a quorum, but is prevented from suppressing the messages sent by processes belonging to quorums.

```
(1)  r_i ← 0; sim_rec_msgs_i[1,...,n] ← [⊥,...,⊥];
(2)  (msgs_to_send_i[1,...,n], ls_state_i) ← simulate(sim_rec_msgs_i);
(3)  for each r > 0 do rec_msgs_i[r][1,...,n] ← [⊥,...,⊥] end for;
(4)  repeat forever
(5)    r_i ← r_i + 1;
(6)    for each j ∈ {1,...,n} do send(r_i, msgs_to_send_i[j]) to p_j end for;
(7)    repeat cur_qr_i ← qr_i
(8)      until (∀j ∈ cur_qr_i \ {i} : rec_msgs_i[r_i][j] ≠ ⊥)
(9)    end repeat;
(10)   for each j ∈ cur_qr_i do sim_rec_msgs_i[j] ← rec_msgs_i[r_i][j] end for;
(11)   (msgs_to_send_i[1,...,n], ls_state_i) ← simulate(sim_rec_msgs_i);
(12)   sim_rec_msgs_i[1,...,n] ← [⊥,...,⊥]
(13) end repeat.

when (r, m) received from p_j:   rec_msgs_i[r][j] ← m.
```

Figure 6: From $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ to $\mathcal{SMP}_n[adv : \text{QUORUM}]$

Lemma 5 *If a task can be solved in $\mathcal{SMP}_n[adv : \text{QUORUM}]$, it can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$.*

Proof The proof that no simulator of a process p_i remains forever blocked in a round r_i follows directly from the fact that (1) each process simulator send a message to each other process simulator at every round (line 6), and (2) each quorum qr_i eventually contains only correct simulators (liveness of Σ).

Let qr_i^r be the value of qr_i that allows p_i to exit the repeat loop during the simulation of round r (lines 7-9). It follows from line 8 and line 10 that $qr_i^r = \{k : k \xrightarrow{r_i} i\}$. Moreover, it follows from the intersection property provided by Σ that $\forall\, i,j,r_i,r_j : qr_i^{r_i} \cap qr_j^{r_j} \neq \emptyset$. The first part of the property QUORUM, namely, $\forall\, i,j,r_i,r_j : (\{k : k \xrightarrow{r_i} i\} \cap \{k : k \xrightarrow{r_j} j\} \neq \emptyset)$, is consequently satisfied.

Let us now show that $\mathcal{SC} \neq \emptyset$. To that end, let us first observe that it follows from the intersection property of Σ that $\forall\, i, j, \forall\, r$, $\exists k(i,j,r)$ such that $k(i,j,r) \xrightarrow{r} i \ \wedge\ k(i,j,r) \xrightarrow{r} j$. As $\{k(i,j,r)\}_{r>0} \subseteq \{1,...,n\}$, it follows that there is some $k'(i,j)$ which appears infinitely often in the sequence $k(i,j,1),\ k(i,j,2)$, ... Hence, we have $k'(i,j) \overset{\infty}{\rightsquigarrow} i \wedge k'(i,j) \overset{\infty}{\rightsquigarrow} j$. As this is true for any pair (i,j), it follows that the graph G, whose set of vertices is $\{1,...,n\}$ and edges are defined by the relation $\overset{\infty}{\rightsquigarrow}$, has a single strongly connected component without input edges. As this strongly connected component defines the set of strongly correct processes, this set is not empty, which concludes the proof of the lemma. $\square_{Lemma\ 5}$

5.3 From $\mathcal{SMP}_n[adv : \text{QUORUM}]$ to $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$

The simulation algorithm is described in Figure 7. It is very close to the simulation of $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ on top of $\mathcal{SMP}_n[adv :$ SOURCE] presented in Figure 5. It has the same local variables, except the variable $missed_i$ which is now useless. The value returned when qr_i is read by a simulated process p_i is now the current value of the set rec_from_i.

The only other difference appears at lines 9-10. The simulation of the simulated process p_i (invocation of the procedure simulate() at lines 11) is now constrained by the predicate of line 9 which states that the messages that p_i wants to send (the messages saved in $msg\_to\_send_i$) must be known by at all the simulators defining the current quorum of p_i (set rec_from_i). When this is satisfied, the set of messages to be received by p_i in the next invocation of simulate() is redefined (line 11) to include the last simulated messages sent to p_i by processes p_j such that $j \in rec\_from_i$.

initialization:
(1) $ls\_state_i \leftarrow$ initial state of the local simulated algorithm;
(2) $msgs\_to\_rec_i \leftarrow \emptyset$; $msgs\_received_i \leftarrow \emptyset$;
(3) $(msgs\_to\_send_i, ls\_state_i) \leftarrow \mathsf{simulate}(ls\_state_i, msgs\_to\_rec_i)$;
(4) $view_i \leftarrow msgs\_to\_send_i$; $rec\_from_i \leftarrow \{1, \ldots, n\}$.

round $r = 1, 2, \cdots$ **do**:
(5) $\mathsf{send}(i, view_i)$ to each other process;
(6) $rec\_msgs_i \leftarrow$ set of pairs $(j, view_j)$ received during this round;
(7) $view_i \leftarrow view_i \cup \left(\bigcup_{(j,view_j) \in rec\_msgs_i} view_j\right)$;
(8) $rec\_from_i \leftarrow \{j \in \{1, \ldots, n\} \ :\ \exists (j, view_j) \in rec\_msgs_i\} \cup \{i\}$;
(9) **if** $(msgs\_to\_send_i \in \bigcap_{(j,view_j) \in rec\_msgs_i} view_j)$ **then**
(10) $\quad msgs\_to\_rec_i \leftarrow msgs\_to\_rec_i \cup \{(j,i,m) \ :\ (j, view_j) \in rec\_msgs_i \wedge (j,i,m) \in view_j\}$;
(11) $\quad (msgs\_to\_send_i, ls\_state_i) \leftarrow \mathsf{simulate}(ls\_state_i, msgs\_to\_rec_i \setminus msgs\_received_i)$;
(12) $\quad msgs\_received_i \leftarrow msgs\_to\_rec_i$; $view_i \leftarrow view_i \cup msgs\_to\_send_i$
(13) **end if**.

when qr_i **is read**: $\mathsf{return}(rec\_from_i)$.

Figure 7: Simulation of $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ in $\mathcal{SMP}_n[adv :$ QUORUM]

Lemma 6 *If a task can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$, it can be solved in $\mathcal{SMP}_n[adv :$ FAIR, QUORUM].*

Proof Part 1: Correct and faulty simulated processes.
According to the definition of $\mathcal{SC}$ and to the second part of QUORUM property, we have $\mathcal{SC} \neq \emptyset$ and $\forall i \in \mathcal{SC}$, $\forall j \in \{1, \ldots, n\} \ :\ i \overset{\infty}{\rightsquigarrow} j$. Let p_i be a simulated process p_i whose simulator is strongly correct. As its messages are always received by every process, they are received by the processes in its local set rec_from_i. Moreover, as the simulation algorithm is a full information algorithm, it eventually receives from each process in rec_from_i the messages whose it is simulating the sending. The condition of line 9 becomes then satisfied, and the simulator of p_i is allowed to progress in the simulation of p_i (line 11). Hence, no strongly correct simulator can block forever in the simulation of its simulated process p_i.

According to (1) the intersection property of QUORUM, and (2) the fact that $\mathcal{SC} \neq \emptyset$ implies that $\exists r \ :\ \forall r' \geq r,\ \forall i \in \mathcal{SC} \ :\ \{k \ :\ k \overset{r'}{\rightsquigarrow} i\} \subseteq \mathcal{SC}$, it follows that $\forall r > 0,\ \forall i \in \{1, \ldots, n\} \ :\ \{k \ :\ k \overset{r}{\rightsquigarrow} i\} \cap \mathcal{SC} \neq \emptyset$ (A). Moreover, there is a round after which no message from a weakly correct process reaches a strongly correct process (B). Finally, (predicate at line 9) to progress in the simulation, a simulator has to receive from each simulator in its current set rec_from_i a copy of the messages $msgs\_to\_send_i$ whose it is simulating the sending (C). It follows from A, B, and C that, eventually, the predicate at line 9 of any weakly correct process remains false forever (because its set $msgs\_to\_send_i$ is never received by a strongly correct process). Consequently, there is a finite time after which, all weakly correct simulators stop progressing in the simulation (while they forever execute rounds, they never execute lines 10-12).

According to the previous discussion, a correct (resp., faulty) process in $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ is a process whose simulator is strongly (resp., weakly) correct.

Part 2: the local variables rec_from_i implement Σ.
The intersection property of Σ comes directly from the first predicate defining the property QUORUM. The liveness property of Σ is a consequence of Item (2) noticed above, and the fact that the correct processes in the simulated system $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ are exactly those whose simulators are strongly correct in $\mathcal{SMP}_n[adv : \text{QUORUM}]$.

Part 3: The linearization points of the communication operations are defined the same way as in the proof of Lemma 4. $\square_{Lemma\ 6}$

5.4 QUORUM is a Characterization of Σ in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$

Theorem 4 *A task can be solved in $\mathcal{SMP}_n[adv : \text{QUORUM}]$ iff it can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$.*

Proof The proof follows immediately from Lemma 5 and Lemma 6. $\square_{Theorem\ 4}$

6 SOURCE + QUORUM Characterizes $\Sigma + \Omega$ in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$

Let us notice that the properties SOURCE and QUORUM are independent of one another in the sense that none of them can be obtained from the other. It follows that the power provided by SOURCE and the power provided by QUORUM can be added. More specifically, we have the following:

- A merge of the simulation of $\mathcal{SMP}_n[adv : \text{SOURCE}]$ in $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ (Figure 4) with the simulation of $\mathcal{SMP}_n[adv : \text{QUORUM}]$ in $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ (Figure 6) provides a simulation $\mathcal{SMP}_n[adv : \text{SOURCE, QUORUM}]$ in $\mathcal{AMP}_{n,n-1}[fd : \Sigma,\ \Omega]$. The difference between this simulation and the one of Figure 4 (or Figure 6) is at lines 7-10 which becomes

  (7) **repeat** $cur\_\ell d_i \leftarrow leader_i$; $cur\_qr_i \leftarrow qr_i$
  (8) **until** $[(\forall j \in cur\_qr_i \setminus \{i\} : rec\_msgs_i[r_i][j] \neq \bot) \wedge (cur\_\ell d_i = i \vee rec\_msgs_i[r_i][cur\_\ell d_i] \neq \bot)]$
  (9) **end repeat**;
  (10) **for each** $j \in cur\_qr_i \cup cur\_ld_i$ **do** $sim\_rec\_msgs_i[j] \leftarrow rec\_msgs_i[r_i][j]$ **end for**.

  The proof is the same as in Lemma 5 augmented by the fact that the eventual leader elected by Ω verifies the property SOURCE as shown in Lemma 3.
- Similarly, adding the management of $missed_i$ and the procedure to query Ω (as done at lines 8-11 of Figure 5) to the simulation of $\mathcal{AMP}_{n,n-1}[fd : \Sigma]$ in $\mathcal{SMP}_n[adv : \text{QUORUM}]$ (Figure 7) provides a simulation of $\mathcal{AMP}_{n,n-1}[fd : \Sigma,\ \Omega]$ in $\mathcal{SMP}_n[adv : \text{SOURCE, QUORUM}]$.

  The linearization points and the proof of the properties of Σ are the same as in Lemma 6, while the proof of the properties of Ω follows the one of Lemma 4. Let us finally notice that it follows directly from the properties SOURCE and QUORUM that a process verifying the SOURCE property appears eventually in all the simulated quorums.

Theorem 5 then follows:

Theorem 5 *A task can be solved in $\mathcal{SMP}_n[adv : \text{SOURCE, QUORUM}]$ iff it can be solved in $\mathcal{AMP}_{n,n-1}[fd : \Sigma,\ \Omega]$.*

7 Conclusion

Considering crash-free synchronous round-based systems, message adversaries have been designed as daemons that suppress messages. Failure detectors have been introduced to enrich crash-prone asynchronous (read/write or message-passing) systems. A previous work [1] has shown that, from a task solvability point of view, the message adversaries constrained by a property denoted TOUR (for tournament) characterizes the well-known wait-free read/write model.

Considering task solvability, this paper has introduced relations linking failures detectors and message adversaries. More precisely, it has introduced two new properties, denoted SOURCE and QUORUM, which are restrictions on message adversaries, and has shown that

- SOURCE + TOUR characterizes the wait-free read/write model enriched with Ω,

- SOURCE characterizes the crash-prone asynchronous message-passing model enriched with Ω,
- QUORUM characterizes the crash-prone asynchronous message-passing model enriched with Σ,
- SOURCE + QUORUM characterizes the crash-prone asynchronous message-passing model enriched with $\Sigma + \Omega$.

Hence, when considering task solvability, these characterizations state "minimal properties" defining the strongest message adversaries for synchronous round-based message-passing systems equating classical asynchronous crash-prone systems. Interestingly, this allows for the establishment of a hierarchy on message adversaries (e.g., $\mathcal{SMP}_n[adv : \text{QUORUM}]$ is stronger than $\mathcal{SMP}_n[adv : \text{TOUR}]$ as shown on the left of Figure 1; this follows from the fact that the computability power of Σ is strictly stronger than read/write registers).

In our understanding of the foundations of distributed computing, a lot of issues remain still open. As examples, here are two interesting message adversary-related problems. Which is the weakest message adversary AD such that consensus can be solved in $\mathcal{SMP}_n[adv : \text{AD}]$? (The only thing we know is that $\mathcal{SMP}_n[adv : \text{CONS}]$ is weaker than or equivalent to $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ and strictly stronger than $\mathcal{SMP}_n[adv : \text{TOUR}]$). Is the addition of the constraint $|\mathcal{SC}| \geq n - t$ to an adversary sufficient to characterize t-resilient asynchronous crash-prone read/write or message-passing systems?

Acknowledgments

This work has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing. Special thanks to D. Imbs and S. Rajsbaum for interesting discussions on distributed computing models.

References

[1] Afek Y. and Gafni E., Asynchrony from synchrony. *Proc. Int'l Conference on Distributed Computing and Networking (ICDCN'13)*, Springer LNCS 7730, pp. 225-239, 2013.

[2] Afek Y., Gafni E, and Linial N., A king in two tournaments. *Unpublished report*, 3 pages, March 2012. http://www.cs.huji.ac.il/ nati/PAPERS/king_tournaments.pdf.

[3] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996.

[4] Chandra T., Hadzilacos V. and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722, 1996.

[5] Charron-Bost B. and Schiper A., The *heard-of* model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49-71, 2009.

[6] Cornejo A., Rajsbaum S., Raynal M., Travers C., Failure Detectors as Schedulers (Brief Announcement). *Proc. 26th ACM Symposium on Principles of Distributed Computing (PODC)*, ACM Press, pp. 308-309, 2007.

[7] Delporte-Gallet C., Fauconnier H., and Guerraoui R., Tight failure detection bounds on atomic object implementations. *Journal of the ACM*, 57(4), Article 22, 2010.

[8] Delporte-Gallet C., Fauconnier H., and Toueg S., The minimum information about failures for solving non-local tasks in message-passing systems. *Distributed Computing*, 24(5):255-269, 2011.

[9] Fernández Anta A. and Raynal M., From an asynchronous intermittent rotating star to an eventual leader. *IEEE Transactions on Parallel Distributed Systems*, 21(9):1290-1303, 2010.

[10] Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991.

[11] Herlihy M.P. and Shavit N., The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858-923, 1999.

[12] Kuhn F., Lynch N.A., and Oshman R., Distributed computation in dynamic networks. *Proc. 42nd ACM Symposium on Theory of Computing (STOC'10)*, ACM press, pp. 513-522, 2010.

[13] Landau H.G., On dominance relations and the structure of animal societies, III: The condition for score structure. *Bulletin of Mathematical Biophysics*, 15(2):143-148, 1953.

[14] Lo W.-K. and Hadzilacos V., Using failure detectors to solve consensus in asynchronous shared-memory systems. *Proc. 8th Int'l Workshop on Distributed Algorithms (WDAG'94)*, Springer LNCS 857, pp. 280-295, 1994.

[15] Mourgaya E., Mostéfaoui A., and Raynal M., Asynchronous implementation of failure detectors. *Proc. Int'l IEEE Conference on Dependable Systems and Networks (DSN 2003)*, IEEE Press, pp. 351-360, 2003.

[16] Pike S.M., Sastry S. and Welch J.L., Failure detectors encapsulate fairness. *Distributed Computing*, 25(4): 313-333, 2012.

[17] Rajsbaum S., Raynal M., Travers C., The iterated restricted immediate snapshot model. *Proc. 14th Annual Int'l Conference on Computing and Combinatorics (COCOON 2008)*, Springer LNCS 5092, pp. 487-497, 2008.

[18] Raynal M., K-anti-Omega. *Rump session*, PODC 2007.

[19] Raynal M., Failure detectors for asynchronous distributed systems: an introduction. *Wiley Encyclopedia of Computer Science and Engineering*, 2:1181-1191, 2009.

[20] Raynal M. and Stainer J., Increasing the power of the iterated immediate snapshot model with failure detectors. *Proc. 19th Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'12)*, Springer LNCS 7355, pp. 231-242, 2012.

[21] Santoro N. and Widmayer P., Time is not a healer. *Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89)*, Springer LNCS 349, pp. 304-316, 1989.

[22] Santoro N. and Widmayer P., Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, 384(2-3): 232-249, 2007.

[23] Schmid U., Weiss B., and Keidar I., Impossibility results and lower bounds for consensus under link failures. *SIAM Journal of Computing*, 38(5): 1912-1951, 2009.

[24] Zieliński P., Anti-Omega: the weakest failure detector for set agreement. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC)*, ACM Press, pp. 55-64, 2008.

A The Properties TP and PAIRS and their Source Extensions

A.1 Definition

In addition to TOUR, Afek and Gafni have defined two more properties, denoted TP (for traversal path) and PAIRS [1], defined as follows from the graphs $\mathcal{G}^r$ introduced in Section 2.2.

- The property TP is similar to TOUR. It differs in the replacement of the directed edge constraint $i \stackrel{r}{\longrightarrow} j$ by the directed path constraint $i \stackrel{r}{\dashrightarrow} j$ meaning that there is a directed path from p_i to p_j in $\mathcal{G}^r$ (i.e., at round r there is a path from p_i to p_j on which no message is removed by the adversary). Formally, the graphs $\mathcal{G}^r$ defined by TP are such that:
$$\forall r \geq 1 : \forall\, (i,j) : (i \stackrel{r}{\dashrightarrow} j) \ \vee \ (j \stackrel{r}{\dashrightarrow} i).$$

- Let $Cn2$ be the number of combinations of 2 elements in a set of n elements (binomial coefficient), and $\sigma()$ be a bijection from $\{0, \ldots, Cn2-1\}$ into $\{(i,j) \in \{1,\ldots,n\}^2 : i < j)\}$. PAIRS is similar to TOUR except that a round of TOUR is decomposed into $C(n,2)$ rounds of PAIRS[4]. During a round of PAIRS, the adversary suppresses all messages but one or both of the messages sent by the pair of processes associated by $\sigma()$ with the current round number. The messages graphs $\mathcal{G}^r$ associated with PAIRS are formally defined as follows:
$$\forall r : \big(\sigma(r \bmod Cn2) = (i,j)\big) \Leftrightarrow \big((i \stackrel{r}{\longrightarrow} j) \ \vee \ (j \stackrel{r}{\longrightarrow} i)\big).$$

A main result of [1] concerns computability power of the synchronous models $\mathcal{SMP}_n[adv : \text{TOUR}]$, $\mathcal{SMP}_n[adv : \text{TP}]$, and $\mathcal{SMP}_n[adv : \text{PAIRS}]$, namely, they are all equivalent to $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$.

A.2 Adding a source to TP and PAIRS

While the property TOUR is, at each round, on each pair of processes, this is not the case of the properties TP and PAIRS. Hence, the property SOURCE has to be customized to be appropriately associated with TP or PAIRS. More specifically we have the following.

- The eventual source property associated with TP is denoted SOURCE$_{tp}$. It considers a path instead of an edge. Formally, it is defined a s follows:
$$\exists s \in \{1,\ldots,n\} : \exists r_0 \geq 1 : \ \forall r \geq r_0 : \ \forall\, i \in \{1,\ldots,n\} : \ (s \stackrel{r}{\dashrightarrow} i).$$

- Similarly the intermittent source property associated with PAIRS is denoted SOURCE$_{pairs}$. It preserves messages sent by a source s in every round r such that $\sigma(r)$ involved s. Formally:
$$\exists s \in \{1,\ldots,n\} : \exists r_0 \geq 1 : \ \forall r \geq r_0 : \ \big(\sigma(r \bmod Cn2) \in \{(s,i),(i,s)\}\big) \ \Rightarrow (s \stackrel{r}{\longrightarrow} i).$$

[4] This definition of PAIRS is slightly different but equivalent to the one given in [1].

A.3 SOURCE + TOUR, SOURCE$_{tp}$ +TP, and SOURCE$_{pairs}$ + PAIRS are Equivalent

The following theorem extends to the pairs SOURCE + TOUR, SOURCE$_{tp}$ + TP, and SOURCE$_{pairs}$ + PAIRS, the equivalence of the properties TOUR, TP, and PAIRS stated in [1]. While the structure of these simulations are based on the ones described in [1], some of their technical developments are different. This is due to the introduction of the *source* process which requires message patterns in consecutive rounds that are no longer fully independent.

Theorem 6 *The the system model $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$, the system model $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$, and the system model $\mathcal{SMP}_n[adv$: SOURCE$_{pairs}$, PAIRS$]$ have the same computational power.*

Proof To show that two system models are equivalent, the proof consists in simulating each of them on top of the other one. The proof first shows that $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$ and $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$ are equivalent. It then shows that $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$ and $\mathcal{SMP}_n[adv$: SOURCE$_{pairs}$, PAIRS$]$ are equivalent. It then follows by transitivity that $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$ and $\mathcal{SMP}_n[adv$: SOURCE$_{pairs}$, PAIRS$]$ are equivalent.

Part 1 of the proof: SOURCE + TOUR is a weaker adversary than SOURCE$_{tp}$ + TP.
Since $\forall i, j,\ \forall r > 0\ :\ i \xrightarrow{r} j \implies i \overset{r}{\dashrightarrow} j$, it follows directly from the property definitions that an adversary that respects the properties SOURCE and TOUR respects also the properties SOURCE$_{tp}$ and TP.

Part 2 of the proof: simulation of $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$ on top of $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$.
Simulation protocol. Let A be an algorithm designed for $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$. The simulation consists in executing $2n-3$ simulation rounds of a full information protocol in $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$ in order to simulate each round of A. For each of these simulated rounds, each process p_i does the following. It first sends a set M_i containing all the messages it knows (initially it knows only the pair (i, msg_i) where msg_i is the set of messages sent by p_i in A during the currently simulated round in $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$). Then, it gathers all the messages it receives during a simulation round, and adds the union of all the sets they contain to its set of known messages M_i. At the end of the simulation round $2n-3$, the set of messages known by p_i and addressed to itself is returned as the set of messages it receives during the simulated round of A in $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$.

The simulated message graphs verify TOUR. Let M_i^r denote the value of the set M_i of the messages known by a process p_i at the end of the simulation round r (with $M_i^0 = \{(i, msg_i)\}$). Let K_i^r denote the set of processes which received msg_i before the end of the simulation round r, namely, $K_i^r = \{j \in \{1, \ldots, n\}\ :\ (i, msg_i) \in M_j^r\}$. Let i and j be two process identities and $r \in \{1, \ldots, 2n-3\}$ a simulation round number such that, at the beginning of round r, we have $(j, msg_j) \notin M_i^{r-1}$ and $(i, msg_i) \notin M_j^{r-1}$ (i.e., p_i does not know msg_j and p_j does not know msg_i). According to property TP, $\mathcal{G}^r$ preserves a path from p_i to p_j or from p_j to p_i:

$$\begin{array}{l}\exists k \geq 1,\ \exists(\pi_1, \ldots, \pi_k) \in \{1, \ldots, n\}^k\ :\\ (\forall l \in \{1, \ldots, k-1\}\ :\ (\pi_l, \pi_{l+1}) \in \mathcal{G}^r) \wedge (\{\pi_1, \pi_k\} = \{i, j\})\,.\end{array}$$

If $\pi_1 = i$, then, as $\pi_1 = i \in K_i^{r-1}$ and $\pi_k = j \notin K_i^{r-1}$, it exists $m \in \{1, \ldots, k-1\}$ such that $\pi_m \in K_i^{r-1} \wedge \pi_{m+1} \notin K_i^{r-1}$. Since a message goes from π_m to π_{m+1} during round r, $|K_i^r| > |K_i^{r-1}|$. If $\pi_1 = j$, the same reasoning applies and we have then $|K_j^r| > |K_j^{r-1}|$. As initially $|K_i^0| = |K_j^0| = 1$, it follows that, for all $r \in \{1, \ldots, 2n-3\}$, if $i \notin K_j^{r-1}$ and $j \notin K_i^{r-1}$, then $|K_i^r| + |K_j^r| \geq 2 + r$. Consequently:

$$\begin{aligned}|K_i^{2n-3} \cap K_j^{2n-3}| &= |K_i^{2n-3}| + |K_j^{2n-3}| - |K_i^{2n-3} \cup K_j^{2n-3}|,\\ &\geq 2 + (2n-3) - |K_i^{2n-3} \cup K_j^{2n-3}|,\\ &\geq 2n - 1 - n \geq n - 1,\end{aligned}$$

and it follows that $\{i, j\} \cap K_i^{2n-3} \cap K_j^{2n-3} \neq \emptyset$, which proves that at least one simulated message between p_i and p_j is received during the $2n-3$ simulation rounds. As the proof holds for any pair of processes, the proposed protocol simulates a round of $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$ in $2n-3$ rounds of $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$ and this simulation satisfies the property TOUR.

The source of $\mathcal{SMP}_n[adv$: SOURCE$_{pairs}$, PAIRS$]$ simulates a source of $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$. Considering the previous protocol which simulates a round of $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$ in $2n-3$ rounds of $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$, it is sufficient to show that, if $\mathcal{SMP}_n[adv$: SOURCE$_{tp}$, TP$]$ eventually always preserves paths starting from a process p_s to any other process, then eventually all messages originating from s are preserved in the simulated rounds of $\mathcal{SMP}_n[adv$: SOURCE, TOUR$]$.

Let us consider a simulation in $\mathcal{SMP}_n[adv : \text{SOURCE}_{tp}, \text{TP}]$ of a sequence of rounds of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$. Each round R of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ is simulated using $2n-3$ rounds (denoted with small r letters) of protocol described above. Let p_s be a process such that, after the simulation round r_0, for all $r' \geq r_0$, $\mathcal{G}^{r'}$ is such that there are (message) paths from p_s to any other process. Let $r_1 = 1 + \lceil \frac{r_0-1}{2n-3} \rceil \times (2n-3)$ be the first simulation round after r_0 such that the simulation of a new simulated round $R_0 = 1 + \lceil \frac{r_0-1}{2n-3} \rceil$ of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ begins. Hence, for all $r' \geq r_0$, the graphs $\mathcal{G}^{r'}$ are such that there are paths (possibly changing at every round) from p_s to any other process. It follows that the arguments used above hold with $\pi_1 = s$ and $\pi_k = j$.

Let r be a round such that $r \geq r_1$, and $R = 1 + \lfloor \frac{r-1}{2n-3} \rfloor$ be the round of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ that is simulated during r ($r \geq r_1 \geq r_0$ hence, $R \geq R_0$). Moreover, let $r' = r - (R-1) \times (2n-3) = ((r-1) \mod (2n-3)) + 1$ denote one of the $2n-3$ simulation rounds simulating the round R. With the same notations as above, as $s \overset{r}{\dashrightarrow} j$ for any p_j, the previous reasoning can be applied with $\pi_1 = s$ and $\pi_k = j$, namely, if $j \notin |K_s^{r'-1}|$ we have $|K_s^{r'}| > |K_s^{r'-1}|$. As this is true for any $r \geq r_1$, it is possible, similarly as before, to infer that if $j \notin K^{r'-1}$, then $|K_s^{r'}| \geq 1 + r'$ (if $j \notin K_s^{r'-1}$, then, for all $1 \leq r'' \leq r'$, $j \notin K_s^{r''-1}$, consequently $|K_s^{r''}| > |K_s^{r''-1}|$, moreover $|K_s^0| = 1$ then a simple induction allows to conclude). It follows that if $r' \geq n-1$ then $j \in K_s^{r'}$, in particular $j \in K_s^{2n-3}$. That implies that, in any simulated round $R \geq R_0$, the sets of received messages are such that all processes receive the messages sent by p_s. The properties SOURCE + TOUR are consequently verified by the adversary (which is simulated by sequences of $2n-3$ simulation rounds), which concludes the proof of the computational equivalence of the models $\mathcal{SMP}_n[adv : \text{SOURCE}_{tp}, \text{TP}]$ and $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$.

Part 3 of the proof: simulation of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ on top of $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$.
The definitions of SOURCE + TOUR and SOURCE_{pairs} + PAIRS imply directly that $Cn2$ consecutive rounds of $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$ simulate a round of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$. Indeed, to send a message m to a process p_j in the currently simulated round R of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, a process p_i just sends m during the simulation round r associated with the pair $\{p_i, p_j\}$.

The simulated message graphs verify TOUR. As at least one of the two messages exchanged by p_i and p_j is preserved by $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$ at this round r, and as each pair of processes is associated with one of the $Cn2$ rounds, it follows that, in the simulated round R of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, at least one message is preserved on each link, thus $\mathcal{G}^R$ satisfies TOUR.

The source of $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$ *simulates a source of* $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$. According to definition of SOURCE, it exists a process p_s and a round r_0 after which no message from p_s is removed by the adversary in all the rounds associated with a pair containing p_s. Let $r_1 = 1 + \lceil \frac{r_0-1}{Cn2} \rceil \times Cn2$ be the first round after r_0 at which the simulation of a new round $R_0 = 1 + \lceil \frac{r_0-1}{Cn2} \rceil$ of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ begins. In all the simulated rounds starting from R_0, all processes receive p_s's message. The *source* property of TOUR is then verified.

Part 4 of the proof: simulation of $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$ on top of $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$.
Here again, it follows directly from the definitions of PAIRS and SOURCE + TOUR that a protocol which, for each simulated round R, suppresses all the messages except the ones sent by the processes of the pair associated (by PAIRS) with R, simulates a round of $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$ (moreover, a simulated round R requires exactly one simulation round r). Moreover, as at least one message is preserved in each round on each link by TOUR, it follows that one of the two messages sent by the processes that belong to the pair associated with the round R is not removed. Thus the PAIRS property is verified by the simulated system. Beside that, by definition of SOURCE, it exists a process p_s and a simulation round number r_0 such that all the messages sent by p_s after r_0 are preserved. It follows that, after r_0, all the messages sent by p_s (during the simulation rounds associated with a pair to which it belongs) are preserved. Consequently, the property SOURCE_{pairs} is verified, which concludes the proof of the computational equivalence between $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$ and $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$. $\square_{Theorem\ 6}$

The next corollary follows from Theorem 2 and the previous theorem.

Corollary 1 *A task T can be solved in $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ iff it can be solved in any of the three following synchronous models: $\mathcal{SMP}_n[adv : \text{SOURCE, TOUR}]$, or $\mathcal{SMP}_n[adv : \text{SOURCE}_{tp}, \text{TP}]$, or $\mathcal{SMP}_n[adv : \text{SOURCE}_{pairs}, \text{PAIRS}]$.*

B The Failure Detector $\overline{\Omega}_k$

Definition $\overline{\Omega}_k$ ($1 \leq k \leq n$) is called an *anti-omega-k* failure detector [18, 24]. It is a simple generalization of Ω ($\overline{\Omega}_1 = \Omega$). In the system model $\mathcal{AMP}_{n,n-1}[fd : \Omega_k]$, each process p_i is endowed with a local variable $xx_i = leaders_i$ that always contains a (possibly changing) set of k process identities. Moreover, there is a time τ and a process identity $\ell \in \mathcal{C}$ such that $\forall \tau' \geq \tau$: $(i \in \mathcal{C}) \Rightarrow (leaders_i^{\tau'} = \ell)$.

$\mathcal{ARW}_{n,n-1}[fd : \overline{\Omega}_k] \simeq_T \mathcal{SMP}_n[adv : k - \text{SOURCE, TOUR}]$ It is possible to extend the Theorem 2 to the system model $\mathcal{ARW}_{n,n-1}[fd : \overline{\Omega}_k]$. The corresponding adversary, denoted k-S-TOUR is defined as follows:

$$\exists S \subseteq \{1, \ldots, n\},\ \exists r_0 > 0 \ :\ (|S| = k) \wedge (\forall r \geq r_0 \ :\ \exists s_r \in S \ :\ \forall\, i :\ s_r \xrightarrow{r} i).$$

The proof that a task is solvable in $\mathcal{ARW}_{n,n-1}[fd : \overline{\Omega}_k]$ iff it is solvable in $\mathcal{SMP}_n[adv : k-\text{SOURCE, TOUR}]$ is a straightforward generalization of the previous one. It is left to the reader.