Noname manuscript No.
(will be inserted by the editor)

Fault-Tolerant Simulation of Population Protocols

Giuseppe A. Di Luna · Paola Flocchini · Taisuke Izumi · Tomoko Izumi · Nicola Santoro · Giovanni Viglietta

Abstract In this paper we investigate the computational power of population protocols under some unreliable or weaker interaction models. More precisely, we focus on two features related to the power of interactions: omission failures and one-way communications. An omission failure, a notion that this paper introduces for the first time in the context of population protocols, is the loss by one or both parties of the information transmitted in an interaction. The failure may or may not be detected by either party. In one-way models, on the other hand, communication happens only in one direction: only one of the two agents can change its state depending on both agents' states, and the other agent may or may not be aware of the interaction. These notions can be combined, obtaining one-way protocols with (possibly detectable) omission failures.

We start our investigation by providing a complete classification of all the possible models arising from the aforementioned weaknesses, and establishing the computational hierarchy of these models. We then address for each model the fundamental question of what additional power is necessary and sufficient to completely overcome the model's weakness and make it able to simulate faultless two-way protocols; by "simulator" we mean a wrapper protocol that, implementing an atomic communication of states between two agents, converts any standard two-way protocol into one that works in a weaker model. We answer this question by presenting simulators that work under certain assumptions (e.g., additional memory, unique IDs, etc.) and by proving that simulation is impossible without such assumptions.

A preliminary version of this article has appeared in [25].

G. Di Luna
Aix-Marseille University, LiS Laboratory,
Campus de Luminy, 163 Avenue de Luminy, Marseille, France.
E-mail: giuseppe.diluna@lis-lab.fr

P. Flocchini
School of Electrical Engineering and Computer Science, University of Ottawa,
800 King Edward Avenue, Ottawa, ON, Canada.
E-mail: paola.flocchini@uottawa.ca

T. Izumi
Department of Computer Science and Engineering, Nagoya Institute of Technology
Gokisocho, Showa Ward, Nagoya 466-8555, Japan
E-mail: t-izumi@nitech.ac.jp

T. Izumi
College of Information Science and Engineering, Ritsumeikan University
56-1 Tojiin Kitamachi, Kita-ku, Kyoto, Japan.
E-mail: izumi-t@fc.ritsumei.ac.jp

N. Santoro
School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, ON, Canada.
E-mail: santoro@cs.carleton.ca

G. Viglietta
Jaist,
1-1 Asahidai, 923-1211 Nomi City,
Ishikawa Prefecture, Japan.
E-mail: johnny@jaist.ac.jp

1 Introduction

1.1 Framework

Population protocols [4] are a mathematical model that describes systems of simple mobile computational entities, called *agents*. Two agents can interact (i.e., exchange information) only when their movement brings them into communication range of each other. However, the movements of the agents, and thus the occurrences of their interactions, are completely unpredictable, a condition called "passive mobility". Such would be, for example, the case of a flock of birds, each provided with

a sensor; the resulting passively mobile sensor network can then be used for monitoring the activities of the flock and for individual intervention, such as a sensor inoculating the bird with a drug, should a certain condition be detected.

In population protocols, when an interaction occurs, the states of the two agents involved change according to a set of deterministic rules, or "protocol". Interactions are asymmetric: one agent is the "starter", and the other is the "reactor". The execution of the protocol, through the interactions originating from the movements of the entities, generates a non-deterministic sequence of changes in the states of the entities themselves, and thus in the global state of the system. The requirements and goals of a protocol depend on the particular application. In some applications, the goal of the protocol might be to ensure that, in every execution, the system converges to a predefined *final global state*; for example, converging to an "epidemic-alert" state if the number of sensors detecting avian influenza is above a threshold. In other applications, the *sequence of state changes* of each agent must obey precise constraints; for example, if entering some state causes a bird to be inoculated, that state should be entered at most once.

In an interaction, communication is generally assumed to be bidirectional or *two-way*: each agent of a pair receives the state of the other agent and applies the protocol's transition function to update its own state, based on the received information and its current state. From an engineering standpoint, this round-trip communication between two interacting agents may be difficult to implement. Moreover, the standard population protocol model does not include faults [4,23].

In this paper we investigate the computational power of population protocols under some unreliable and/or weaker interaction models. More precisely, we focus on two features related to the power of interactions: *omission failures* and *one-way* communications. An omission failure, a notion that this paper introduces for the first time in the context of population protocols, is the loss by one or both parties of the information transmitted in an interaction. The failure may or may not be *detected* by either party. On the other hand, in one-way models (originally introduced in [5,9]), communication occurs only in one direction: only one of the two agents can change its state depending on both states, and the other agent may or may not be aware of the interaction. In our paper we assume that the "receiver" is the agent that receives both states. These notions can be combined, obtaining one-way protocols with (possibly detectable) omission failures.

A general question is what additional power is necessary and sufficient to fill the gap between the standard two-way model and the weaker models stated above. In this paper, we start addressing this question using as a main investigation tool the concept of a *simulator*: a wrapper protocol converting any protocol for the standard two-way model into one running under some weaker model. A simulator provides an interface between the simulated protocol and the physical communication layer, giving the system the illusion of being in a two-way environment. As a basic feature, a simulator has to implement an atomic communication of states between two agents, always guaranteeing both safety and liveness of any problem specification. Indeed, the simulator should not allow agents to change states in illegal ways (safety), and it should also guarantee that eventually a communication between two agents is carried out successfully (liveness). This task is further complicated by the anonymity of the agents, their lack of knowledge of the system, and the limited amount of memory that they may have.

In a companion paper [26], we have investigated the scenario where the system contains a unique "leader" agent. In this context, we fully characterized the models where a two-way simulator exists, with or without additional assumptions, such as an unbounded amount of memory on all agents or the knowledge of an upper bound on the number of omissive interactions involving the leader.

We stress that the importance of focusing on the existence of simulators (as opposed to, say, studying the set of predicates that are computable in a given model) arises from the fact that the decisions taken by some of the agents may be irrevocable by their very nature (e.g., set the value of a write-once register, inoculate a bird, fire distress flares as an SOS signal, etc.).

1.2 Main Contributions

As a first step, we define several omissive models considering both the capability to detect the proximity of another agent (i.e., interacting with another agent without viewing its state) and to detect an omission (i.e., being aware that an interaction failed). Note that these two features are independent. Proximity detection is common in many Medium Access Control protocols, such as CSMA-MPS and STEM. The models are shown in Figure 1.

In the two-way models with omissions, indicated with T_i, detecting the proximity of an agent and not receiving its state is an implicit detection of an omission. For this reason, in these models, we only look at omission detection when: no detection is possible (model T_1), only one side detects it (model T_2), or both sides can detect the omission (model T_3).

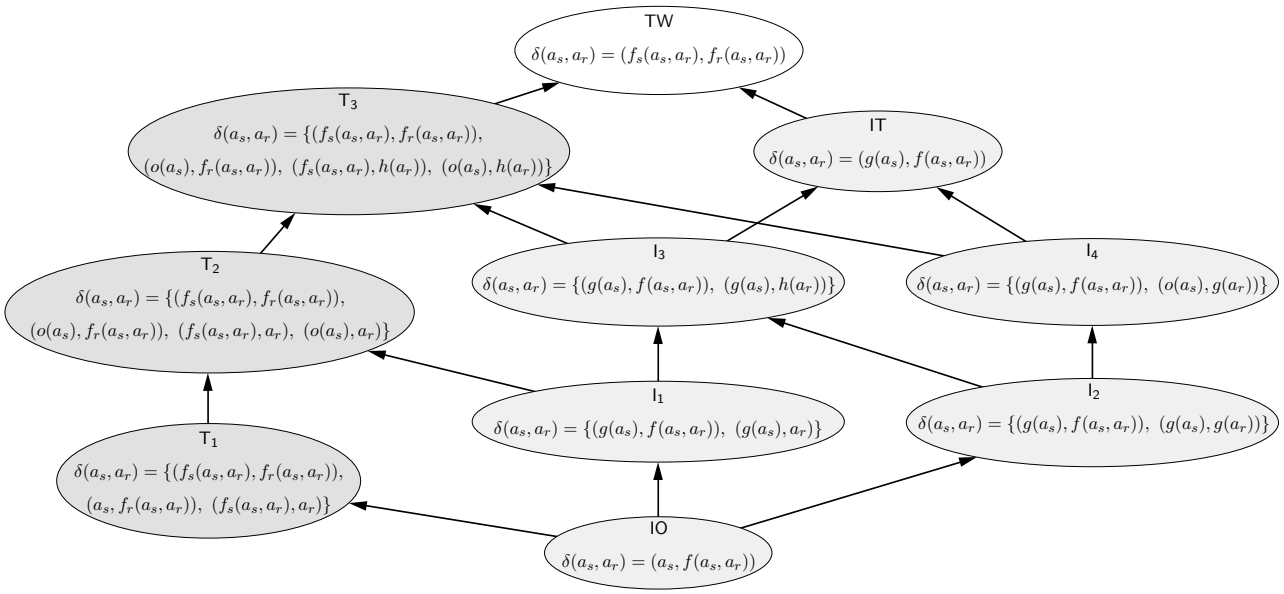


Fig. 1: Computational relationships between models. An arrow between two blobs indicates that the class of problems solvable in the source blob is included in that of the destination blob. The models on the left, T_1, T_2, T_3, are the two-way models with omissions. The models on the right, I_1, I_2, I_3, I_4, are the one-way models with omissions.

In the one-way models, indicated with I_i, detecting the proximity of another agent but not receiving its state could imply either the presence of an omission or that the first agent is the starter of the interaction. We consider and analyze all possible combinations of these two capabilities. As a result, we obtain four distinct models: where no detection of omissions is present but the proximity can be detected by the starter (model I_1) or by both parties (model I_2), and where there is detection of proximity by both parties but omissions are detected only by the starter (model I_3) or by the reactor (model I_4). For completeness we also consider the known models without omissions, the original two-way model (TW) and the one-way models introduced in [5]: the *Immediate Transmission* model (IT), where the starter detects the proximity of the reactor, and the *Immediate Observation* model (IO), where there is no detection of proximity.

The hierarchy of these models is shown in Figure 1; more details can be found in Section 2.

We consider two main types of *omission adversaries*: a "malignant" one, called *Unfair Omissive* (**UO**), which can insert omissions at any point in the execution, and a "benign" one, called *Eventually Non-Omissive* ($\diamond$**NO**), which must eventually stop inserting omissions. Interactions are otherwise "globally fair". Interestingly, all our main simulators work even under the malignant **UO** adversary, while all our main impossibility results hold even under the benign $\diamond$**NO** adversary.

We start by analyzing the negative impact that omissions have on computability. We show that, in the absence of additional assumptions, the simulation of TW protocols in the presence of omissions is impossible even if the agents have infinite memory (Theorem 2). Among other results, we also show that, in the two weak omission models I_1 and I_2, simulation is impossible even under an extremely limited omission adversary, called $\diamond\mathbf{NO_1}$, which can only insert at most one omission in the entire execution.

On the other hand we prove that, in the weakest one-way model, IO, simulation is possible if the agents have unique IDs or the total number of agents, n, is known (Theorems 6 and 7).

In the two strong omission models I_3 and I_4, simulation is possible when an upper bound on the number of omissions is known (Theorem 5). This result in turn implies that, in the non-omissive IT model, TW simulation is possible with a memory overhead of $\Theta(\log n)$ bits for each state of the simulated protocol (Corollary 1). In light of the fact that with constant memory, in absence of additional capabilities, IT protocols are strictly less powerful than two-way protocols [5], our results show that this computational gap can be overcome by using additional memory.

Our main results are summarized in Figure 4 on page 13, where white blobs represent possibilities and gray blobs impossibilities. As a consequence of these results, we have a complete characterization of the feasibility of simulation when agents have infinite memory,

unique IDs, or knowledge of the size of the system. If an upper bound on the number of omissions is known, we also have a complete characterization, except in model T_2, where the problem is open.

1.3 Related Work

Since their introduction, there have been extensive investigations on Population Protocols (e.g., see [6,10, 13,15,16,32,20,21,28,30]), and the basic assumptions of the original model have been expanded in several directions, typically to overcome inherent computability restrictions. For example, allowing each agent to have non-constant memory [1–3,17]; assuming the presence of a leader [8]; allowing a certain amount of information to be stored on the edges [18,19,22,31] of the interaction graph.

The issue of dependable computations in population protocols, first raised in [23], has been considered and studied only with respect to processors' faults, and the basic model has necessarily been expanded. In [24] it has been shown how to compute functions tolerating $\mathcal{O}(1)$ crash-stops and transient failures, assuming that the number of failures is bounded and known. In [7] the specific majority problem under $\mathcal{O}(\sqrt{n})$ Byzantine failures, assuming a fair probabilistic scheduler, has been studied. In [29] unique IDs are assumed, and it is shown how to compute functions tolerating a bounded number of Byzantine faults, under the assumption that Byzantine agents cannot forge IDs. Self-stabilizing solutions have been devised for specific problems such as leader election (assuming knowledge of the system's size and a non-constant number of states [14], or assuming a leader detection oracle [27]) and counting (assuming the presence of a leader [11]). Moreover, in [12] a self-stabilizing transformer for general protocols has been studied in a slightly different model and under the assumption of unbounded memory and a leader.

Finally, to the best of our knowledge, the one-way model without omissions has been studied only in [5], where it was shown that IT and IO, when equipped with constant memory, can compute a set of functions that is strictly included in that of TW. Combined with our results in Figure 4, this implies that, without using extra resources (e.g., infinite memory, leader, etc.), simulations are impossible in all the one-way and omissive models.

2 Models and Terminology

2.1 Population Protocols

We consider a system consisting of a set $A = \{a_1, \dots, a_n\}$ of mobile agents. The mobility is passive, in the sense that it is decided by an external entity. When two agents meet, they interact with each other and perform some local computation. We always assume that interactions are instantaneous. Each interaction is asymmetric, that is, an interaction between a_s and a_r is indicated by the ordered pair $i = (a_s, a_r)$, where a_s and a_r are called *starter* and *reactor*, respectively. A protocol $\mathcal{P}$ is defined by the following three elements: a set of local states $Q_{\mathcal{P}}$, a set of initial states $Q'_{\mathcal{P}} \subseteq Q_{\mathcal{P}}$, and a transition function $\delta_{\mathcal{P}} \colon Q_{\mathcal{P}} \times Q_{\mathcal{P}} \to Q_{\mathcal{P}} \times Q_{\mathcal{P}}$. The function $\delta_{\mathcal{P}}$ defines the states of the two interacting agents at the end of their local computation. With a small abuse of notation, and when no ambiguity arises, we will use the same literal (e.g., a_i) to indicate both an agent and its internal state. Since the structure of the system is uniquely determined by $\mathcal{P}$ and n, we refer to it as the *system* $(\mathcal{P}, n)$. A *configuration* C of a system $(\mathcal{P}, n)$ is the n-tuple of local states from $Q_{\mathcal{P}}$ (i.e., $C \in Q^n_{\mathcal{P}}$).

Given an k-tuple $t = (x_0, x_1, \dots, x_{k-1})$ we denote the element x_j by $t[j]$.

Initial Knowledge. To empower the agents, we sometimes assume that each agent has some additional knowledge, such as an ID and/or the value of n. We model this information by encoding it as a set of initial states of the agents (i.e., in $Q'_{\mathcal{P}}$).

Executions and Fairness. Whenever an interaction $i = (a_j, a_k)$ turns a configuration of the form

$$C = (a_1, \dots, a_j, \dots, a_k, \dots, a_n)$$

into one of the form

$$C' = (a_1, \dots, \delta(a_j, a_k)[0], \dots, \delta(a_j, a_k)[1], \dots, a_n),$$

we use the notation $C \xrightarrow{i} C'$. A *run* of $\mathcal{P}$ is an infinite sequence of interactions $I = (i_0, i_1, \dots)$. Given an initial configuration $C_0 \in Q'^n_{\mathcal{P}}$, each run I induces an infinite sequence of configurations, $\Gamma_I(C_0) = (C_0, C_1, \dots)$ such that $C_j \xrightarrow{i_j} C_{j+1}$ for every $j \geq 0$, which is called an *execution* of $\mathcal{P}$.

We say that a set of configurations $\mathcal{C} \subseteq Q^n_{\mathcal{P}}$ is *closed under permutations* if, for every $C \in \mathcal{C}$, and for every configuration $\widehat{C}$ obtained by permuting the states of the agents of C, also $\widehat{C} \in \mathcal{C}$.

An execution Γ is *globally fair* if it satisfies the following condition: for every two (possibly infinite) sets of configurations $\mathcal{C}, \mathcal{C}' \subseteq Q^n_{\mathcal{P}}$, closed under permutations,

such that for every $C \in \mathcal{C}$ there exists an interaction i and some $C' \in \mathcal{C}'$ such that $C \xrightarrow{i} C'$, if infinitely many configurations of Γ belong to $\mathcal{C}$, then infinitely many configurations of Γ belong to $\mathcal{C}'$ (although not necessarily appearing in Γ as immediate successors of configurations of $\mathcal{C}$).

Note that our definition of global fairness extends the standard one, which only deals with single configurations, as opposed to sets (see [10]). The two definitions are equivalent when applied to protocols that use only finitely many states, but our extension also works with infinitely many states, where the standard one is ineffective.

2.2 Non-Omissive Interaction Models

In this paper we consider three main models of interactions: the standard *Two-Way* one, and two one-way models presented in [5], i.e., the *Immediate-Transmission* model and the *Immediate-Observation* model.

Two-Way Model (TW). In this model, any protocol $\mathcal{P}$ has a state transition function consisting of two functions $f_s \colon Q_\mathcal{P} \times Q_\mathcal{P} \to Q_\mathcal{P}$ and $f_r \colon Q_\mathcal{P} \times Q_\mathcal{P} \to Q_\mathcal{P}$ satisfying $\delta_\mathcal{P}(a_s, a_r) = (f_s(a_s, a_r), f_r(a_s, a_r))$ for any $a_s, a_r \in Q_\mathcal{P}$.

Immediate-Transmission Model (IT). Any protocol $\mathcal{P}$ has a state transition function consisting of two functions $g \colon Q_\mathcal{P} \to Q_\mathcal{P}$ and $f \colon Q_\mathcal{P} \times Q_\mathcal{P} \to Q_\mathcal{P}$ satisfying $\delta_\mathcal{P}(a_s, a_r) = (g(a_s), f(a_s, a_r))$ for any $a_s, a_r \in Q_\mathcal{P}$.

Immediate-Observation Model (IO). Any protocol $\mathcal{P}$ has a state transition function of the form $\delta_\mathcal{P}(a_s, a_r) = (a_s, f(a_s, a_r))$ for any $a_s, a_r \in Q_\mathcal{P}$.

Note that, in the IT model, the starter explicitly detects the interaction, as it applies function g to its own state. In other terms, even if the starter cannot read the state of the reactor, it can still detect its "proximity". In the IO model, on the other hand, there is no such detection of an interaction (or proximity) by the starter.

Proximity Detection (PD). When we consider one-way models we may assume that agents have the capability to detect the proximity of their interaction partners. This means that an agent knows that it is involved in an interaction even if it does not receive the state of the other agent. When an agent is equipped with proximity detection and it interacts with some other agent, it applies the function g to its own state. This is the case of the IT model, that is a one-way model without omission where the starter is equipped with proximity detection. Later we will also introduce omission detection, and we will show that the detection of an omission is a property that is not always equivalent to the detection of proximity.

2.3 Omissions and Model Hierarchy

An omission is a fault affecting a single interaction. In an omissive interaction an agent does not receive any information about the state of its counterpart. Omissions are introduced by an adversarial entity. We consider:

Definition 1 (Unfair Omissive (UO) Adversary) The **UO** adversary takes a run I and outputs a new sequence I', which is obtained by inserting a (possibly empty) finite sequence of omissive interactions between each pair of consecutive interactions of I. (Note that, since I is infinite, the total number of omissive interactions in I' may be infinite.)

Definition 2 (Eventually Non-Omissive ($\diamond$NO/$\diamond\mathbf{NO_1}$) Adversary) The $\diamond$**NO** adversary takes a run I and outputs a new sequence I', which is obtained by inserting any finite sequence of omissive interactions between finitely many pairs of consecutive interactions of I. (Note that the total number of omissive interactions in I' is finite.) The $\diamond\mathbf{NO_1}$ adversary is even weaker, and can only output interaction sequences with at most one omission.

If we incorporate omissions in our runs, then transition functions become more general relations.

2.3.1 TW *Omissive Models*

In the two-way omissive model, we have the transition relation

$$\delta(a_s, a_r) = \{(f_s(a_s, a_r), f_r(a_s, a_r)), (o(a_s), f_r(a_s, a_r)), \\ (f_s(a_s, a_r), h(a_r)),\ (o(a_s), h(a_r))\}$$

(model T_3). The first pair is the outcome of an interaction when no omission is present; the other three pairs represent all possible outcomes when there is an omission: respectively, an omission on the starter's side, on the reactor's side, and on both sides. The functions o and h represent the detection capabilities of each agent: in TW, if one of these is the identity, then omissions are *undetectable* on the respective side. If omissions are undetectable on both sides, we have the model T_1:

$$\delta(a_s, a_r) = \{(f_s(a_s, a_r), f_r(a_s, a_r)), (a_s, f_r(a_s, a_r)), \\ (f_s(a_s, a_r), a_r))\}.$$

Omission Detection (OD). In some models we assume that agents have the capability to detect an omission. This means that an agent knows that it is involved

in an interaction that contains an omission. We assume that the agent detecting the omission is the one that does not receive the state of the other agent. This has a practical reason: in case of an omission, an agent could receive corrupted information, allowing it to detect the fault but not the state of the other agent.

Moreover, if the agent that receives the state of the other also detects the omission, then omission can be easily removed by ignoring the omissive interaction. Therefore, this would lead to models that are equivalent to one-way interaction models without omissions.

If an agent is equipped with omission detection and it initiates an interaction that happens to be omissive, then the agent does not receive the state of the other agent, and it applies the function o to its own state only.

If omissions are detected only on the starter's side, we have the model T_2:

$$\delta(a_s, a_r) = \{(f_s(a_s, a_r), f_r(a_s, a_r)), (o(a_s), f_r(a_s, a_r)), \\ (f_s(a_s, a_r), a_r),\ (o(a_s), a_r)\}.$$

If omissions are detected only on the reactor's side, we have a symmetric model, equivalent to T_2.

2.3.2 One-Way Omissive Models

In the case of one-way interactions, we have the transition relation

$$\delta(a_s, a_r) = \{(g(a_s), f(a_s, a_r)),\ (o(a_s), h(a_r))\}.$$

The first pair is the outcome of an interaction when no omission is present, and the second pair when there is an omission. (Note that the IO model corresponds to the case in which g is the identity function.) Once again, omissions are undetectable starter-side (respectively, reactor-side) if o (respectively, h) is the identity function.

Model Investigation for One-Way Interactions All possible one-way interaction models are detailed in Figure 2. The figure contains all possible combinations of detection of proximity and omission, starter-side or reactor-side.

We identify four models in Figure 2 for which there is no immediate equivalence with any other model in the table. For convenience these models will be named I_1, I_2, I_3, I_4:

– Model I_1 is the one where only the starter detects the proximity, Str(PD). The transition relation is

$$\delta(a_s, a_r) = \{(g(a_s), f(a_s, a_r)),\ (g(a_s), a_r)\}.$$

– In model I_2, the transition relation is

$$\delta(a_s, a_r) = \{(g(a_s), f(a_s, a_r)),\ (g(a_s), g(a_r))\}.$$

Proximity is detected by both starter and reactor: Str(PD) & Rct(PD). That is, in case of a correct transition, the starter detects the presence of another agent, and its transition is $g(a_s)$. The other agent detects the presence of an interacting agent and it also receives its state: therefore its transition function is $f(a_s, a_r)$. If there is a faulty interaction, then both agents will detect the presence of each other, but no one will receive the other's state. Therefore, they will both apply the same function g to their internal state, in this case the transition function is $(g(a_s), g(a_r))$. In this model, in case of omission, we have a symmetric interaction, because both agents apply g to their states. Notice that, if the starter and the receiver applied two different functions, say g, t, in case of omission, then this would have been equivalent to having receiver-side omission detection. Indeed, in this case the receiver would know that its role is the receiver, and if it did not receive the state of the starter, then it would infer that an omission has occurred.

– In model I_3, the starter detects proximity and the reactor detects omissions:

$$\delta(a_s, a_r) = \{(g(a_s), f(a_s, a_r)),\ (g(a_s), h(a_r))\}.$$

This is model Str(PD) & Rct(OD) of Figure 2. In this case, the starter detects the proximity of an agent with which it is interacting both in the case of a correct transition and in the case of an omission. Thus, the starter always transitions to $g(a_s)$. On the contrary, the reactor detects the presence of an omission in case of a faulty interaction, outcome $h(a_r)$, or it receives the state of the starter in case of a correct interaction, outcome $f(a_s, a_r)$.

– Finally, I_4 is the model in which the starter detects both proximity and omission and the reactor can only detect proximity: Str(PD+OD) & Rct(PD) in Figure 2. The transition relation is

$$\delta(a_s, a_r) = \{(g(a_s), f(a_s, a_r)),\ (o(a_s), g(a_r))\}.$$

In case of a correct transition, the starter detects the presence of another agent, transitioning to $g(a_s)$, and the other agent detects the presence of the starter as well as its state: therefore its transition is $f(a_s, a_r)$. In case of a faulty interaction, then both agents will detect each other's presence, none of them will receive the state of the other, and the starter will detect an omission. Therefore the starters applies

Detection Capabilities	Transition Relation	Relationship with Models of Figure 1
Str(PD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(g(a_s),a_r)\}$	I_1 by definition
Str(OD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(o(a_s),a_r)\}$	equivalent to IO
Str(PD+OD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(o(a_s),a_r)\}$	equivalent to IT
Rct(PD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(a_s,g(a_r))\}$	equivalent to IO
Rct(OD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(a_s,h(a_r))\}$	equivalent to IO
Rct(PD+OD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(a_s,h(a_r))\}$	equivalent to IO
Str(PD) & Rct(PD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(g(a_s),g(a_r))\}$	I_2 by definition
Str(PD) & Rct(OD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(g(a_s),h(a_r))\}$	I_3 by definition
Str(PD) & Rct(PD+OD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(g(a_s),h(a_r))\}$	equivalent to I_3
Str(OD) & Rct(PD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(o(a_s),g(a_r))\}$	equivalent to IO
Str(OD) & Rct(OD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(o(a_s),h(a_r))\}$	equivalent to IO
Str(OD) & Rct(PD+OD)	$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),(o(a_s),h(a_r))\}$	equivalent to IO
Str(PD+OD) & Rct(PD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(o(a_s),g(a_r))\}$	I_4 by definition
Str(PD+OD) & Rct(OD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(o(a_s),h(a_r))\}$	equivalent to IT
Str(PD+OD) & Rct(PD+OD)	$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),(o(a_s),h(a_r))\}$	equivalent to IT

Fig. 2: Various models of one-way interactions in the presence of proximity detection (PD) and omission detection (OD) for starter (Str) and reactor (Rct), and their relationship with the models of Figure 1.

function o to its own state, and the receiver cannot distinguish this faulty interaction from a correct interaction in which it had the role of starter: therefore, it will apply function g to its own state.

There are some models that have the same transition relation, such as model Str(PD+OD) & Rct(OD) and Str(PD+OD) & Rct(PD+OD). The difference between these two models is that in the second one the reactor detects also the proximity of the starter and not only the omission. It is trivial to see that the transition relation is the same:

$$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),\ (o(a_s),h(a_r))\}.$$

Indeed, if there is an omission in the second model, the reactor detects the proximity of the starter, but detecting the omission gives more information than just detecting the interaction. Therefore, in both models, when an omission occurs the reactor uses the function h instead of g.

In the following theorem we will show the immediate relationships between the models in the table, the four models I_1, I_2, I_3, I_4 and the two one-way models without omissions, IT and IO.

Theorem 1 *Considering the models of Figure 2, the following hold.*

(1) Models Str(OD), Rct(PD), Rct(OD), Rct(PD+OD), Str(OD) & Rct(PD), Str(OD) & Rct(OD), and Str(OD) & Rct(PD+OD) are equivalent to IO.
(2) Models Str(PD+OD), Str(PD+OD) & Rct(OD), and Str(PD+OD) & Rct(PD+OD) are equivalent to IT.
(3) Model Str(PD) & Rct(PD+OD) is equivalent to I_3.

Proof We prove each statement of the theorem:

(1) All these models share a transition relation of the form

$$\delta(a_s,a_r)=\{(a_s,f(a_s,a_r)),\ (a_s/o(a_s),a_r/g(a_r)/h(a_r))\}.$$

When no omission occurs the transition relation is the same as in IO: $\delta(a_s,a_r)=(a_s,f(a_s,a_r))$. Omissions are introduced by an adversary. This means that if omissions improve the computational power of the model with respect to IO, then the adversary will just create executions without omissions. On the other hand, if the presence of omissions leads to a model that is weaker than IO, than the agents can ignore each omission by setting the functions o, g and h to the identity. Therefore, all these models are equivalent to IO.

(2) All these models share a transition relation in the form

$$\delta(a_s,a_r)=\{(g(a_s),f(a_s,a_r)),\ (o(a_s),a_r/h(a_r))\}.$$

In this case, the agents can simulate a model without omissions by using the identity function when an omission is detected (i.e., setting o and h to the identity). This leads to the transition function $\delta(a_s,a_r)=(g(a_s),f(a_s,a_r))$. Thus, all these models are equivalent to IT.

(3) The equivalence is immediate by observing that model Str(PD+OD) & Rct(PD+OD) has the same transition relation as model Str(PD+OD) & Rct(OD).

□

Hierarchy of Models In the previous sections we identified the models obtained by all possible combinations

of omission and detection, and we pruned out the trivially equivalent ones. The significant models and their relationships have been reported in Figure 1.

For TW omissive models, in T_2 we have the models where there is no detection of omission either on the starter's or the reactor's side. Since these two models are symmetric, only the one without reactor-side detection is reported, i.e., function h is forced to be the identity. In T_1 we have the weaker model where no detection is available, i.e., both o and h are the identity. In one-way models, function g is applied when an agent detects the *proximity* of another agent. However, detecting the proximity does not imply the detection of an omission: in I_2, no agent detects an omission, but both detect the proximity of the other agent. Finally, models IT and IO are the non-omissive models introduced in [5].

Each arrow in Figure 1 indicates either the obvious inclusion, that is, the transition relation of the source is a subset of the transition relation of the destination, or the adversary can force the inclusion by avoiding omissions: this is the case with T_3 and TW, for instance.

Another case is give by model I_4, which gets the same transition relation as I_2 upon setting $o = g$. This means that any protocol for I_2 can be used also for I_4, which implies that any problem solvable in I_2 can also be solved I_4. Moreover, if a problem cannot be solved in I_4, that is, any protocol implementable in I_4 cannot solve the problem, then it cannot be solved in I_2, since the protocols implementable in I_2 are a subset of the protocols implementable in I_4. The same can be said for models T_3 and I_3: the transition relation of T_3 becomes that of I_3 if we set $f_s = o = g$ and $f_r = f$.

2.4 Simulation of Two-Way Protocols

In this section we define the *two-way protocol simulator* (or "simulator" for short) and other related concepts. Given a two-way protocol $\mathcal{P}$, consider a protocol $\mathcal{S}(\mathcal{P})$, whose set of local states is $Q_{\mathcal{P}} \times Q_{\mathcal{S}}$, where $Q_{\mathcal{P}}$ is the set of local states of $\mathcal{P}$ (the "simulated states"), and $Q_{\mathcal{S}}$ is additional memory space used in the simulation. Let $\pi_{\mathcal{P}} \colon Q_{\mathcal{P}} \times Q_{\mathcal{S}} \to Q_{\mathcal{P}}$ be the projection function onto the set of local states of $\mathcal{P}$. By extension, if C is a configuration of $\mathcal{S}(\mathcal{P})$, we write $\pi_{\mathcal{P}}(C)$ to indicate the configuration of $\mathcal{P}$ consisting of the projections of the states of the agents of C.

Next we define the concept of *event*: intuitively, an event represents an update of the simulated state of an agent. Given an execution $\Gamma_I(C_0)$ of $\mathcal{S}(\mathcal{P})$, where $I = (i_0, i_1, \dots)$, we say that $E(\Gamma) = (e_0, e_1, \dots)$ is a sequence of events for Γ if it is a weakly increasing sequence of indices of interactions of I, such that no three indices are the same, and containing at least the indices of the interactions that determine the update of the simulated state of some agent in the execution Γ (if an interaction updates the simulated states of two agents, then its index must appear twice in $E(\Gamma)$). So, with each event e_j in $E(\Gamma)$, we can associate a unique agent involved in the interaction i_{e_j}; preferably, this agent is one that effectively changes simulated state as a result of i_{e_j}. We also allow extra events in $E(\Gamma)$, associated with agents that do not change simulated state, because we want to take into account simulations of two-way protocols that occasionally leave the state of an agent unchanged.

If $\Gamma_I(C_0) = (C_0, C_1, \dots)$, we let $C_j^- = C_{e_j}$ and $C_j^+ = C_{e_j+1}$. In other words, C_j^- and C_j^+ are the configurations before and after the j-th update of the simulated state, respectively.

Definition 3 (Perfect matching of events) Given an execution $\Gamma_I(C_0)$ of a run I and a sequence of events $E(\Gamma)$, a *perfect matching* $M(E)$ is a partition of $\mathbb{N}$ into ordered pairs (viewed as indices of events of $E(\Gamma)$) such that, if $(j,k) \in M(E)$, where e_j is associated with agent a_x and e_k with agent a_y, then $x \neq y$ and

$$\delta_{\mathcal{P}}(\pi_{\mathcal{P}}(C_j^-[x]), \pi_{\mathcal{P}}(C_k^-[y])) = (\pi_{\mathcal{P}}(C_j^+[x]), \pi_{\mathcal{P}}(C_k^+[y])).$$

Intuitively, a pair (j,k) in a perfect matching corresponds to the pair of events (e_j, e_k) representing the two state changes given by a two-way interaction of agents under the simulated protocol $\mathcal{P}$. The events e_j and e_k correspond to the updates of the simulated states of the starter and the reactor, respectively. A matching $M(E)$ induces a *derived run* D of $\mathcal{P}$ as follows. Sort the pairs (j,k) of $M(E)$ by increasing $\min\{e_j, e_k\}$, and let M' be the sorted sequence. Now, if (j,k) is the m-th element of M', agent a_x is associated with event e_j and agent a_y is associated with event e_k, then the m-th element of D is (x,y). Now, the *derived execution* induced by $M(E)$ is simply the execution of $\mathcal{P}$ induced by D, i.e., $\Gamma_D(\pi_{\mathcal{P}}(C_0))$.

Definition 4 (Simulation) A protocol $\mathcal{S}(\mathcal{P})$ *simulates* the two-way protocol $\mathcal{P}$ if, for any initial configuration C_0 of n agents of $\mathcal{S}(\mathcal{P})$, and any run I whose execution $\Gamma_I(C_0)$ satisfies the global fairness condition, there exists a sequence of events $E(\Gamma)$ with a perfect matching $M(E)$ whose derived execution is an execution of n agents of $\mathcal{P}$ starting from the initial configuration $\pi_{\mathcal{P}}(C_0)$ and satisfying the global fairness condition. We further require that, for each initial configuration C_0, every finite initial sequence of interactions of $\mathcal{S}(\mathcal{P})$ (possibly with omissions) can be extended to an infinite

one I, having no additional omissions, whose execution $\Gamma_I(C_0)$ satisfies the global fairness condition.

The last clause of the definition is a bland consistency condition that has been added because, with infinite-memory protocols, the existence of globally fair executions cannot be taken for granted. It essentially states that a protocol should not be considered a simulator simply because it is so "pathological" that it does not admit globally fair executions. This clause is redundant in practice, as all the protocols presented in this paper obviously have globally fair executions in abundance. However, we will need to assume this consistency condition in order to prove that simulators for certain protocols do not exist (cf. Lemma 1).

3 Impossibilities for Simulation in Presence of Omissions

In this section, we derive several impossibility results in the presence of omissions.

3.1 Additional Definitions

All our impossibility proofs rely on the existence of a two-way protocol that cannot be simulated.

Definition 5 (Pairing Problem) A set of agents A is given, partitioned into *consumer* agents A_c, starting in state c, and *producer* agents A_p, starting in state p. We say that a protocol $\mathcal{P}$ solves the *Pairing Problem (Pair)* if it enforces the following properties:

- **Irrevocability.** $\mathcal{P}$ has a state s that only agents in state c can get; once an agent has state s, its state cannot change any more.
- **Safety.** At any time, the number of agents in state s is at most $|A_p|$.
- **Liveness.** In all globally fair executions of $\mathcal{P}$, eventually the number of agents in state s is stably equal to $\min\{|A_c|, |A_p|\}$.

It is easy to see that Pair can be solved by the simple protocol below in the standard two-way model.

> Pairing Protocol $\mathcal{P}_{IP}$. $Q_{\mathcal{P}_{IP}} = \{s, c, p, \perp\}$. The only non-trivial transition rules are $(c, p) \mapsto (s, \perp)$ and $(p, c) \mapsto (\perp, s)$.

Let us now define a property on the behavior of a generic simulator $\mathcal{S}(\mathcal{P})$ over a sequence of interactions I. We will later show how this property is related to the omission resilience of $\mathcal{S}(\mathcal{P})$.

Definition 6 (Transition Time (TT)) Given a TW protocol $\mathcal{P}$, a simulator $\mathcal{S}(\mathcal{P})$, and an execution $\Gamma = (C_0, C_1, \ldots)$ of $\mathcal{S}(\mathcal{P})$ on a system of two agents, the *Transition Time (TT)* of the triplet $(\mathcal{S}, \mathcal{P}, \Gamma)$ is the smallest t such that

$$\pi_{\mathcal{P}}(C_t[0]) = \delta_{\mathcal{P}}(\pi_{\mathcal{P}}(C_0[0]), \pi_{\mathcal{P}}(C_0[1]))[0]$$

and

$$\pi_{\mathcal{P}}(C_t[1]) = \delta_{\mathcal{P}}(\pi_{\mathcal{P}}(C_0[0]), \pi_{\mathcal{P}}(C_0[1]))[1]$$

(or ∞, if no such t exists).

Let $O(I)$ be the number of omissions in a sequence of interactions I.

Definition 7 (Fastest Transition Time (FTT)) Given a TW protocol $\mathcal{P}$, a simulator $\mathcal{S}(\mathcal{P})$, and a configuration C_0 for a system of two agents of $\mathcal{S}(\mathcal{P})$, the *Fastest Transition Time (FTT)* of the triplet $(\mathcal{S}, \mathcal{P}, C_0)$ is the smallest TT of all the triplets of the form $(\mathcal{S}, \mathcal{P}, \Gamma_I)$, where I ranges over all runs with $O(I) = 0$ and $\Gamma_I[0] = C_0$.

Intuitively, FTT is the minimum number of (non-omissive) interactions needed by a specific simulator $\mathcal{S}$ to simulate one step of protocol $\mathcal{P}$ in a system of two agents. Thus it can be seen as the "maximum speed" of a simulator. We will show in the following that such a metric is intrinsically related with the omission resilience of $\mathcal{S}$.

3.2 Impossibilities in Spite of Infinite Memory

In this section we show that simulations of TW models are impossible when omissions are present, even if the system is endowed with infinite memory. We start presenting a key indistinguishability argument.

Lemma 1 *Let $\mathcal{S}(\mathcal{P})$ be a simulator working in the omission model T_3. Let $t > 0$ be the FTT of the triplet $(\mathcal{S}, \mathcal{P}, C_0)$, where one agent in C_0 has simulated state q_0, the other agent has q_1 with $q_0 \neq q_1$, and $\delta_{\mathcal{P}}(q_0, q_1) = (q_0', q_1')$ and $\delta_{\mathcal{P}}(q_1, q_0) = (q_1', q_0')$. Let A be a system of $2t + 2$ agents of $\mathcal{S}(\mathcal{P})$, and let B_0 be an initial configuration of A in which t agents have simulated state q_0 and $t + 2$ agents have q_1. Then, there exists a sequence of interactions I^* of A such that $\Gamma_{I^*}(B_0)$ is globally fair and $O(I^*) = t$, with a sequence of events $E(\Gamma_{I^*}(B_0))$ in which at least $t + 1$ events represent a transition of some agent from simulated state q_1 to q_1'.*

Proof Intuitively, we construct a system with t pairs of agents which, thanks to omissive interactions, we "fool" into believing that they are operating in a system of only two agents, until one agent per pair transitions

from simulated state q_1 to q_1'. Then we have an extra agent that interacts once with one member of each of the t pairs, also "believing" that the system consists of only two agents, which finally transitions from simulated state q_1 to q_1'. One last auxiliary agent serves as a "generator" of omissive interactions.

Let I be any run of a system of two agents achieving FTT for $(\mathcal{S}, \mathcal{P}, C_0)$; let d_0 be the agent whose initial state is q_0 and let d_1 be the other one. For every $0 \leq k < t$, we construct a sequence of interactions I_k for two agents as follows: copy the first k interactions from I; append an omissive interaction with the same starter as $I[k]$, and with omission (and detection) on d_1's side; extend the resulting sequence to an infinite one whose execution from C_0 satisfies the global fairness condition, without adding extra omissions (such an extension exists by Definition 4, since $\mathcal{S}$ is a simulator). Note that I_k has exactly one omissive interaction.

Because the execution of I_k is globally fair, the derived execution must also be globally fair by definition of simulator, and in particular it makes the simulated states of the two agents transition according to $\delta_{\mathcal{P}}$ infinitely many times. Hence, the agent whose initial simulated state is q_1 will eventually transition to q_1', say after the execution of the first t_k interactions of I_k. Note that this happens regardless of which agent is the starter of the two-way simulated interaction, because by assumption $\delta_{\mathcal{P}}$ is symmetric on (q_0, q_1).

Now let $a_0, a_1, \ldots, a_{2t+1}$ be the agents of A, with indices chosen in such a way that, for all $0 \leq k < t$, the agents of the form a_{2k} have simulated state q_0 in B_0, while all other agents of A have q_1. For every $0 \leq k < t$, we construct a sequence J_k, consisting of $t_k + 1$ interactions, involving only agents a_{2k}, a_{2k+1}, a_{2t}, and a_{2t+1}. We make a_{2k} and a_{2k+1} interact with each other as in I_k, but we "redirect" the omissive interaction $I_k[k]$ to a_{2t} and a_{2t+1}. Specifically, we replicate the first k interactions of I_k (where d_0 becomes a_{2k} and d_1 becomes a_{2k+1}); then we add an interaction between a_{2k} and a_{2t}, where the role of a_{2k} (i.e., starter or reactor) is the same as that of d_0 in $I_k[k]$; then we insert an omissive interaction between a_{2k+1} and a_{2t+1}, where the role of a_{2k+1} is the same as that of b_1 in $I_k[k]$, and the omission (and detection) is on a_{2k+1}'s side; finally, we replicate the $t_k - k - 1$ interactions of I_k from $I_k[k+1]$ to $I_k[t_k - 1]$. Observe that J_k contains exactly one omissive interaction, $J_k[k+1]$.

The final sequence I^* is now simply the concatenation of all the sequences J_k (where k goes from 0 to $t-1$, in increasing order), extended to an infinite sequence of interactions whose execution is globally fair, and having exactly t omissions in total (again, this extension exists by Definition 4).

Let us examine the execution of I^* from the initial configuration B_0. Each of the t pairs of the form (a_{2k}, a_{2k+1}), with $0 \leq k < t$, has an initial execution that is the same as that of (d_0, d_1) interacting for k turns as in I_k. Hence, the execution of a_{2t} is that of d_1 interacting as in I for the first t turns. It follows that a_{2t} transitions from simulated state q_1 to q_1' by the end of the sub-run J_{t-1}. Also, for each pair (a_{2k}, a_{2k+1}), the execution is as in I_k for the first t_k turns; hence, a_{2k+1} transitions from simulated state q_1 to q_1' at the end of the sub-run J_k. Thus, in total, we have at least $t + 1$ agents that transition from q_1 to q_1'. □

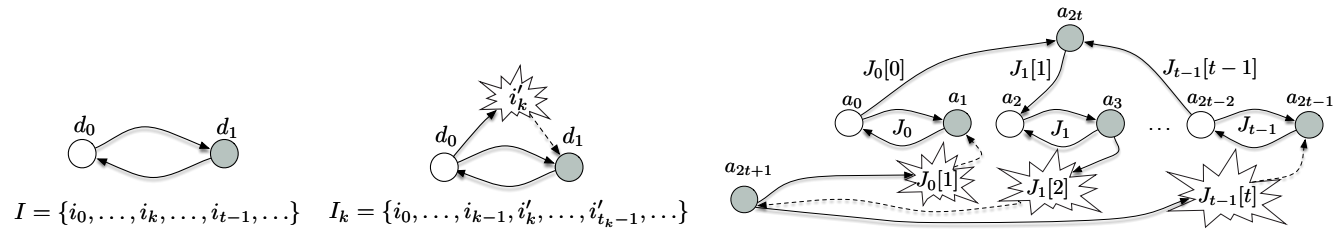
Theorem 2 *Given an infinite amount of memory on each agent, it is impossible to simulate every* TW *protocol in the* T_3 *model (hence in all the* omissive *models of Figure 1), even under the* $\diamond$**NO** *adversary.*

Proof We show that the protocol $\mathcal{P}_{IP}$ for Pair cannot be simulated if any type of omissive interaction is allowed. Assume by contradiction that there is a simulator $\mathcal{S}$ for $\mathcal{P}_{IP}$, i.e., $\mathcal{S}$ solves Pair under some omissive model. Let us now apply Lemma 1 to $\mathcal{S}$ and $\mathcal{P}_{IP}$, where q_0 is the initial state of the providers (hence there are t providers), q_1 is the initial state of the consumers (hence there are $t + 2$ consumers), and q_1' is the irrevocable state.

Because $\mathcal{P}_{IP}$ is symmetric with respect to starter and reactor, the hypotheses of Lemma 1 are satisfied, and hence there is a sequence of interactions I^* whose execution is globally fair, which causes $t+1$ transitions into state s. Since the execution is globally fair, the derived execution of I^* must be an execution of $\mathcal{P}_{IP}$, due to Definition 4. In particular, it satisfies the irrevocability property of Pair. Therefore, no agent entering state s can ever change it. It follows that, eventually, there are at least $t + 1$ agents in state s, which contradicts the safety property of Pair.

Since I^* contains just finitely many omissive interactions, it can be generated by the $\diamond$**NO** adversary. □

Theorem 2 uses as counterexample the construction of Lemma 1, implying that a simulator S fails to simulate protocol $\mathcal{P}_{IP}$ in a run where the number of failures is exactly the FTT of $(\mathcal{S}, \mathcal{P}_{IP}, (c, p))$. This is even more interesting if we consider simulators that are unaware of the protocol they are simulating, where by "unaware" we mean that the sequence of simulated two-way interactions is not influenced by the protocol that is being simulated or by the initial configuration (i.e., general-purpose and not ad-hoc simulators). We have shown that each of these simulators fails as soon as the number of omissions is above some constant threshold, which is independent of the simulated protocol and the initial



(a) Run I achieving FTT. (b) Partial run I_k. (c) Final run I^*.

Fig. 3: Construction of the run I^*

configuration. Such a threshold is precisely the minimum number of non-omissive interactions needed to simulate a single two-way transition.

For models T_1, I_1, and I_2, we can strengthen Theorem 2.

Theorem 3 *Given an infinite amount of memory on each agent, it is impossible to simulate every* TW *protocol in the interaction models* T_1, I_1, *and* I_2, *even under the* $\diamond\mathbf{NO_1}$ *adversary.*

Proof The proof uses a construction analogous to the one used in Lemma 1. We consider a system A of $2t+2$ agents $a_0, a_1, \ldots, a_{2t+1}$, and we build t sequences of interactions I_k between two agents d_0 and d_1, exactly as in Lemma 1. Recall that the run I_k contains only one omission. Hence, if a simulator is resilient to the $\diamond\mathbf{NO_1}$ adversary, it eventually succeeds in making d_0 and d_1 simulate a full two-way interaction, say after t_k one-way interactions. Since t_k is well defined, we can go on and construct the sequence J_k. However, the J_k that we will use in this proof differs from its counterpart used in Lemma 1 by two elements: $J_k[k]$ and $J_k[k+1]$. In particular, our new J_k's will contain no omissions.

If the model is T_1, we replace the old interactions $J_k[k]$ and $J_k[k+1]$ by a single non-omissive interaction between a_k and a_{2t} (in which a_k is the starter if and only if d_0 is the starter in $I[k]$).

Let the model be I_1. If the interaction $I[k]$ is (d_0, d_1), then we replace the old interactions $J_k[k]$ and $J_k[k+1]$ by the single interaction (a_k, a_{2t}). Otherwise, if $I[k] = (d_1, d_0)$, we set $J_k[k] = (a_{2t}, a_{2t+1})$ and $J_k[k+1] = (a_{k+1}, a_{2t+1})$.

Consider now model I_2. If the interaction $I[k]$ is (d_0, d_1), then we set $J_k[k] = (a_k, a_{2t})$ and $J_k[k+1] = (a_{k+1}, a_{2t+1})$. Otherwise, if $I[k] = (d_1, d_0)$, we replace the old interactions $J_k[k]$ and $J_k[k+1]$ by the three interactions (a_{2t}, a_{2t+1}), (a_k, a_{2t+1}), and (a_{k+1}, a_{2t+1}).

Finally, we concatenate the t finite sequences J_k to obtain the new run I^*, which contains no omissions. Let us now examine the execution of I^* from the initial configuration B_0 defined in Lemma 1. Once again, each of the t pairs (a_{2k}, a_{2k+1}), with $0 \leq k < t$, has an initial execution that is the same as that of (d_0, d_1) interacting for k turns as in I_k. Then, the new interactions that we added in lieu of $J_k[k]$ and $J_k[k+1]$ make a_{2k} and a_{2k+1} change state in the same way as in the omissive interaction $I_k[k]$. But as a side effect, also a_{2t} changes state as it would in a non-omissive interaction with a_{2k}. As a consequence, by the end of I^*, all the agents of the form a_{2k+1} with $0 \leq k < t$, as well as a_{2t}, have transitioned from simulated state q_1 to q'_1. Thus, in total, at least $t+1$ agents transition from q_1 to q'_1.

Now the proof can be completed exactly as in Theorem 2, by showing that the protocol $\mathcal{P}_{IP}$ cannot be simulated. □

One may wonder what would happen if we wanted to construct simulators that "gracefully degrade" when omissions reach a certain threshold t_O. More precisely, for a sequence of interactions I with $O(I) < t_O$, the simulator has to perform a full simulation of $\mathcal{P}$; if $O(I) \geq t_O$, the simulator has to start a simulation, but then it is allowed to stop forever in a "consistent" simulated state. Essentially, in the second case, we allow the sequence of events $E(\Gamma)$ defined in Section 2.4 to be finite (in other terms, we drop the simulator's "liveness" requirement).

Theorem 4 *Given an infinite amount of memory on each agent, in the* T_3 *model (and hence in all the* omissive *models of Figure 1), any gracefully degrading simulator that simulates all* TW *protocols must have a threshold* $t_O \leq 1$.

Proof Recall that in Lemma 1 we constructed a sequence of interactions I^* for a set of agents A, which was then applied to the protocol $\mathcal{P}_{IP}$ in order to prove Theorem 2. Suppose now that a simulator has threshold $t_O > 1$. If such a simulator executes a run with at most one omission, it must effectively simulate infinitely many two-way interactions. In particular, it is able to simulate the first two-way interaction in a system of two agents, and therefore the sequence I mentioned in

Lemma 1 is well defined for this simulator, as well as the sequences I_k and the numbers t_k. But then, as the agents of A execute the same simulator according to the sequence I^*, they violate the safety property of Pair, because $t+1$ of them change their simulated state from c to s. Since they reach a non-consistent simulated state, this means that a gracefully degrading simulator with threshold $t_O > 1$ cannot simulate $\mathcal{P}_{IP}$. □

4 Simulation in Omissive Models

In this section we focus on designing simulators of two-way protocols. In light of the impossibilities presented in the previous section, additional assumptions are necessary. Section 4.1 assumes some knowledge on the maximum number of omissions, Section 4.2 assumes the presence of unique IDs, and finally in Section 4.3 we assume to know the number of agents.

4.1 Knowledge on Omissions

Here we assume to know an upper bound o on the number of omissions, i.e., for any sequence of interactions I on which the simulator runs we have $O(I) \leq o$. We will show that under this assumption there exists a simulator for models I_3 and I_4. These contrast with models I_1 and I_2, in which it is impossible to simulate even when $O(I) \leq 1$ (see Theorem 3).

We explain the simulator $\mathcal{S}_{KO}$ under model I_3; the version for model I_4 is only slightly different, and its correctness follows from symmetry considerations. The simulator is reminiscent of the card game Rummy, and is based on the exchange of "tokens". Each simulated state $q \in Q_{\mathcal{P}}$ is represented as a sequence of numbered tokens: $\langle q, 1\rangle, \ldots, \langle q, o+1\rangle$. Intuitively, an agent tries to transmit its state to others by sending one token at a time, for $o+1$ consecutive interactions, each time incrementing the counter. When a reactor detects an omission, it generates a *joker* token $\langle J\rangle$, which will also start circulating. Note that there are never more than o jokers in the system. Every time an agent gets a new token, it checks if it owns the complete set of $o+1$ tokens representing some state q and, if so, it simulates (part of) an interaction with a hypothetical partner in state q. If the complete set of tokens is not available, the agent is allowed to replace the missing tokens with the jokers that it currently owns. After the $o+1$ tokens have been used, they are discarded and withdrawn from circulation. However, if an agent uses some joker tokens, it "takes note" of what tokens these jokers are replacing. If later on the same agent obtains one of the tokens in this list, say $\langle q, i\rangle$, it turns $\langle q, i\rangle$ into a joker and removes $\langle q, i\rangle$ from the list.

Simulator Variables. Each agent has a multiset of tokens to be sent, called *sending*, initially empty. It also has a variable $state_{sim}$ (initially set to available) for the state of the simulator protocol, a variable $state_{\mathcal{P}}$ for the state of the simulated protocol (initialized according to its initial simulated state), and a multiset of tokens called *graveyard*, initially empty.

Simulator Protocol. Suppose that an agent interacts as a starter. If $state_{sim} =$ available and *sending* is empty, the agent performs a *production operation*: it switches to $state_{sim} =$ pending and inserts the complete string of tokens

$$\langle state_{\mathcal{P}}, 1\rangle, \langle state_{\mathcal{P}}, 2\rangle, \ldots, \langle state_{\mathcal{P}}, o+1\rangle$$

into *sending*. Subsequently, regardless of $state_{sim}$, the starter removes a token from the *sending* multiset, chosen deterministically (unless of course the multiset is empty), and the reactor reads it, as detailed below. (It is important that the token is chosen deterministically from the multiset, so the starter and the reactor will implicitly choose the same.)

Suppose now that an agent interacts as a reactor. To begin with, it reads a token from the *sending* multiset of the starter (unless of course it is empty), and inserts it into its own *sending* multiset. If it detects an omission, it inserts a joker token $\langle J\rangle$ instead. Then it runs a preliminary check: if $state_{sim} =$ pending and the agent can find a complete string of tokens for its own state (i.e., $state_{\mathcal{P}}$) in its own *sending* multiset (possibly using some joker tokens as wild cards), then it performs a *deletion operation*: it switches to $state_{sim} =$ available and removes the set of $o+1$ used tokens from the multiset, one copy of each. After this preliminary check, the core protocol starts: if $state_{sim} =$ available and the agent has a complete string of tokens for some state q in its own *sending* multiset (possibly using some joker tokens), it performs a *right-side transition operation*: it removes the set of $o+1$ used tokens from the multiset, it simulates its part of the two-way transition with an agent in state q (i.e., it sets $state_{\mathcal{P}} = \delta(q, state_{\mathcal{P}})[1]$), and it inserts a complete string of "state-change" tokens into *sending*:

$$\langle (q, state_{\mathcal{P}}), 1\rangle, \langle (q, state_{\mathcal{P}}), 2\rangle, \ldots, \langle (q, state_{\mathcal{P}}), o+1\rangle.$$

On the other hand, if $state_{sim} =$ pending and the agent has a complete string of state-change tokens of the form $\langle (state_{\mathcal{P}}, q'), i\rangle$ in its own *sending* multiset (possibly using some joker tokens), it performs a *left-side transition operation*: it removes the set of $o+1$ used tokens from the multiset, it sets $state_{\mathcal{P}} = \delta(state_{\mathcal{P}}, q')[0]$, and switches to $state_{sim} =$ available.

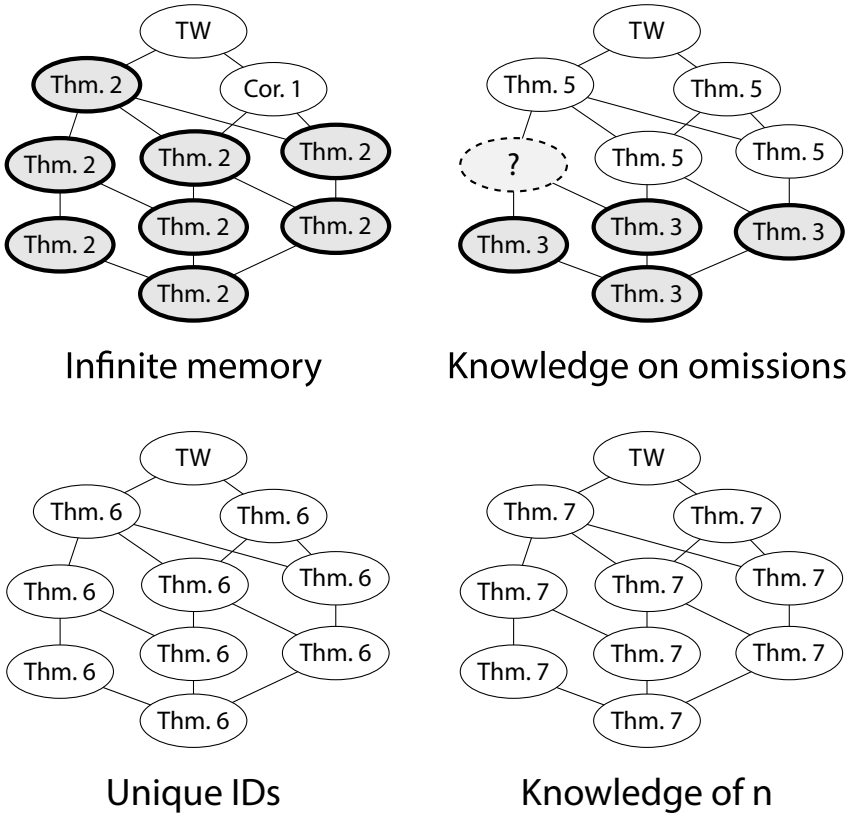


Fig. 4: Map of results (cf. Figure 1). The models where it is possible to simulate two-way protocols are represented with a white blob; the models where simulation is impossible are represented with a gray blob.

Also, whenever a reactor uses a joker token as a substitute for some token $\langle q, i\rangle$, it adds $\langle q, i\rangle$ to the multiset *graveyard*. Symmetrically, when it receives a new token $\langle q, i\rangle$ from a starter and that token is in the *graveyard*, it removes one copy of $\langle q, i\rangle$ from the *graveyard* and it replaces the last copy of $\langle q, i\rangle$ in *sending* with a joker token.

We say that a token is *circulating* if it is in the *sending* multiset of some agent. A token is *dead* if it is in a *graveyard*.

Lemma 2 *In an execution of $\mathcal{S}_{KO}$ with at most o omissive interactions, the number of circulating joker tokens plus the number of dead tokens never decreases and never exceeds o.*

Proof Since we are assuming that the execution contains no more than o omissive interactions, at most o joker tokens are generated in total. These joker tokens circulate among agents through their *sending* multisets, and may be consumed when a deletion or a transition operation is performed. When a joker token is consumed, a token is created in some agent's *graveyard*; when a copy of the same token reaches the *sending* multiset of that agent, it becomes a joker token, eliminating the copy in the *graveyard*. This means that, throughout the execution, the sum of the circulating jokers and the sizes of all *graveyard* multisets never decreases, and is incremented only when an omissive interaction occurs: therefore, this sum never exceeds o.

Let us fix a $q \in Q_{\mathcal{P}}$ and an $i \in \{1, \ldots, o+1\}$: we call (q, i)-*tokens* the tokens of the form $\langle q, i\rangle$ and $\langle (q, q'), i\rangle$, for some $q' \in Q_{\mathcal{P}}$. Observe that there are exactly $|Q_{\mathcal{P}}| + 1$ possible (q, i)-tokens.

Lemma 3 *Throughout an execution of $\mathcal{S}_{KO}$, for every q and i, the number of circulating (q, i)-tokens never exceeds the number of* pending *agents with simulated state q plus the number of dead (q, i)-tokens.*

Proof We can prove our claim by induction: it is true at the beginning of the execution, when there are no tokens and no agent is pending, and we can show that the property is preserved after every interaction. A (q, i)-token $\langle q, i\rangle$ is created if and only if an agent with simulated state q becomes pending: this operation preserves our property. Then, during a deletion operation or a left-side transition operation by a pending agent with simulated state q, the agent becomes available, and either a (q, i)-token is consumed or a joker token is consumed (as a wild card substituting a missing (q, i)-

token), resulting in the creation of a dead (q,i)-token: again, this preserves the property. Moreover, a right-side deletion operation creates a state-change (q,i)-token and either consumes a (q,i)-token $\langle q,i\rangle$ or it consumes a joker token and creates a dead (q,i)-token $\langle q,i\rangle$: this preserves the property. Finally, when an omissive interaction occurs, a circulating (q,i)-token may be lost and replaced with a joker token, which of course preserves our property.

Theorem 5 *Given an upper bound o on the number of omissions and $\Theta(|Q_{\mathcal{P}}|^2(o+1)\log(n+o))$ bits of memory on each agent, every* TW *protocol can be simulated in* I_3 *and* I_4.

Proof We will first prove that the protocol $\mathcal{S}_{KO}$ realizes the simulation in I_3. Let us prove that $\Theta(|Q_{\mathcal{P}}|^2(o+1)\log(n+o))$ bits of memory on each agent are sufficient to execute the protocol. Recall that the possible types of tokens are the ones of the form $\langle q,i\rangle$, the state-change tokens of the form $\langle (q,q'),i\rangle$, and the joker token. This yields a total of $|Q_{\mathcal{P}}|(o+1)+|Q_{\mathcal{P}}|^2(o+1)+1 = \Theta(|Q_{\mathcal{P}}|^2(o+1))$ possible token types. By Lemma 3, the number of circulating non-joker tokens of each type never exceeds $n+o$: indeed, at most n agents can be pending at the same time, and by Lemma 2 there are at most o dead tokens. On the other hand, the number of circulating joker tokens never exceeds o, again by Lemma 2. So, only $\Theta(\log(n+o))$ bits of memory are needed by an agent to remember how many copies of each token its *sending* multiset contains: this yields an upper bound of $\Theta(|Q_{\mathcal{P}}|^2(o+1)\log(n+o))$ bits on the size of the *sending* multiset. The other variables are all smaller: in particular, by Lemma 2, the *graveyard* multiset can contain at most o tokens in total.

Let us show that, in a globally fair execution of $\mathcal{S}_{KO}$, infinitely many production, right-side transition, and left-side transition operations are performed by the agents. When an agent with simulated state q turns pending and performs a production operation, it creates a complete string of tokens bearing state q, which then circulate (possibly turning into jokers) until they are consumed, either in a deletion operation or in a right-side transition operation. The latter occurrence is bound to happen, since the execution is globally fair, unless all agents remain pending forever. Indeed, if there exist infinitely many configurations in the execution where at least one agent is available, then eventually enough tokens will reach an available agent, which will then perform a right-side transition. For the same reason, again because the execution is globally fair, unless all agents remain pending forever, an available agent will eventually have no tokens in its *sending* multiset, and it will therefore perform a production operation. However, it is impossible for all agents to remain pending forever: since the execution is globally fair, eventually an agent would find itself with all the circulating tokens in its *sending* multiset, which would make it consume a string of tokens and become available. With a similar argument, we can show that infinitely many left-side transitions will occur, as well.

Note that in the previous paragraph we did not take into account the presence of joker tokens, and the fact that an agent may replace a missing token in a string with a joker, even if that token is actually circulating. That could prevent any other agent from ever getting a complete string of tokens, because some tokens could be missing, and there may not be enough circulating jokers to replace them. However, this cannot actually happen: recall that the tokens that have been replaced by jokers become dead tokens in some *graveyard*. So, for every joker token that has "wrongfully" been used to replace a token t that is actually circulating, a dead copy of t is created. The missing joker may be needed to replace a token that was actually lost during an omissive interaction, but in that case the joker will be generated again: since the execution is globally fair, eventually the circulating copy of t will reach the agent that has a copy of t in its *graveyard*, and a new joker token will be created. Then, because the execution is globally fair, that joker will eventually be found in the same *sending* multiset with enough tokens to form a complete string and enable a deletion or a transition operation.

Let us assume that exactly k production operations are performed by agents with simulated state q: as a result, k complete strings of tokens bearing state q are produced and become circulating tokens. We claim that these $k(o+1)$ tokens cannot contribute to more than k deletion or right-side transition operations, no matter how many joker tokens are used. In order to perform $k+1$ deletion or right-side transition operations using these tokens, the system would need to collect $k+1$ complete strings of tokens bearing state q, possibly using jokers. Let $i \in \{1,\dots,o+1\}$, and let us focus on the j jokers that are used to replace missing copies of the token $\langle q,i\rangle$ during these $k+1$ operations. These j jokers are turned into j dead $\langle q,i\rangle$ tokens. Later, j' of these dead tokens are removed from their *graveyard* when circulating copies of $\langle q,i\rangle$ appear, but these circulating copies are consumed during the process. It follows that $k+1-j$ copies of $\langle q,i\rangle$ must be consumed in the $k+1$ operations, and j' copies must be consumed by dead tokens. In total, $k+1-j+j'$ unique circulating copies of $\langle q,i\rangle$ are needed, but k copies are only ever generated. This yields the inequality $k \geq k+1-j+j'$, which reduces to $j \geq j'+1$: in other words, there is at least one copy of $\langle q,i\rangle$ that is never removed from a

graveyard. Since this is true of all $i \in \{1, \ldots, o+1\}$, there must be at least $o+1$ tokens that remain dead forever, which contradicts Lemma 2. We conclude that any set of tokens generated by k production operations can only be used in at most k deletion or right-side transition operations. Moreover, since the execution is globally fair, not only these operations will be at most k, but they will be exactly k: indeed, the relevant circulating tokens will eventually be found in the *sending* multiset of the same agent, which will consume them to perform a deletion or right-side transition operation, and this will happen as many times as possible, i.e., k. By a similar argument, the same can be said about left-side transition operations: if k right-side transition operations bearing the same states are performed, the resulting state-change tokens will be used in exactly k left-side transition operations.

So far we have proved that infinitely many production, right-side transition, and left-side transition operations must occur, and we can regard each right-side or left-side transition operation as an event. We now have to show that there exists a perfect matching between these events that yields a consistent derived execution of the simulated two-way protocol. The previous paragraph suggests how to match them: given any execution with only a finite set of k production operations, in the same execution there must be exactly k deletion or right-side transition operations with corresponding states. If the right-side transition operations are k', there will be exactly k' left-side transition operations with corresponding states: we can match each of these left-side transition operations with any right-side transition operation with corresponding state that occurred in a previous interaction, and this will yield an initial segment of a consistent derived execution with k' interactions. Now a standard compactness argument can be applied: since the above holds for any finite k', then in an execution with infinitely many left-side transition operations there exists a perfect matching of events that yields a consistent derived execution.

It remains to prove that such a derived execution is globally fair. Suppose that $\mathcal{C}$ and $\mathcal{C}'$ are sets of configurations (closed under permutations) of $\mathcal{P}$ such that every configuration of $\mathcal{C}$ can become a configuration of $\mathcal{C}'$ after a two-way interaction. Assume that $\mathcal{C}$ is reached infinitely often in the derived execution, and let us prove that $\mathcal{C}'$ is reached infinitely often, as well. Let $\widetilde{\mathcal{C}}$ be the set of configurations of the simulator protocol whose simulated states are in $\mathcal{C}$, and let $\widetilde{\mathcal{C}}'$ be constructed similarly from $\mathcal{C}'$. (Note: if a configuration of the simulator protocol contains a pending agent a_s whose partner a_r has already performed its side of the simulated interaction, i.e., a right-side transition operation, then the simulated state of a_s is assumed to be the state it would reach after the corresponding left-side transition operation. This agrees with the definition of derived run given in Section 2.4.) By assumption, the simulation passes through $\widetilde{\mathcal{C}}$ infinitely often; we claim that it must go through $\widetilde{\mathcal{C}}'$ infinitely many times, as well. By definition of $\mathcal{C}$, for every $C_j \in \widetilde{\mathcal{C}}$, there is an interaction in $\mathcal{P}$ between two agents a_s and a_r that maps $\pi_{\mathcal{P}}(C_j)$ into $\pi_{\mathcal{P}}(C'_j)$, where $C'_j \in \widetilde{\mathcal{C}}'$. We will prove that such a C'_j can be reached from C_j after a bounded number of interactions.

- If a_s and a_r are both available in C_j, then a suitable C'_j can be obtained by first letting a_s interact as the starter to deplete its own *sending* multiset: as already noted, this takes at most $\Theta(|Q_{\mathcal{P}}|^2(o+1)(n+o))$ interactions, because such is the maximum size of the *sending* multiset. Then, we let a_s interact with a_r for $o+1$ times in order to transfer a complete string of tokens to it (including perhaps some jokers if some interactions are omissive) and enable a right-side transition operation. So the simulated state of a_r changes according to $\delta_{\mathcal{P}}$, and a_r generates a complete string of state-change tokens. We let a_r interact as a starter with a_s for $o+1$ times, so that eventually a_s will get enough tokens to perform a left-side transition operation, changing its simulated state according to $\delta_{\mathcal{P}}$.
- Let a_s be pending and a_r be available in C_j. Then a_s must already have produced its string of tokens, and these tokens must bear the correct simulated state that is needed for the two-way transition with a_r. Note that these or equivalent tokens must be able to eventually reach a_r in order to enable a transition to a suitable C'_j. In order for them to reach a_r, at most $o+1$ interactions between a_r and the agents currently holding the relevant tokens are sufficient. Then, the construction proceeds as in the previous case.
- If a_r is pending in C_j, we have to make it become available and then apply one of the two previous constructions. If the string of tokens that a_r generated when it became pending (or a set of equivalent tokens) is still circulating, then we can make these tokens reach a_r in at most $o+1$ interactions, and then force a_r to perform a deletion operation, which will make it become available. Otherwise, a_r must be able to perform a left-side transition operation that does not change its simulated state, or else it would not be possible to map $\pi_{\mathcal{P}}(C_j)$ into $\pi_{\mathcal{P}}(C'_j)$ by a single interaction. Making a_r perform such an operation is again a matter of letting a_r interact with the proper agents a bounded amount of times,

which will eventually make it become available without changing its simulated state.

So, a suitable configuration C'_j can be reached from C_j after a number of interaction bounded by c (which is a function of o and n), and this holds for every j. By applying the definition of global fairness to the simulator's execution c times, we have that a suitable C'_j is indeed reached for infinitely many j's. Therefore $\widetilde{\mathcal{C}'}$ is reached infinitely many times, and thus so is $\mathcal{C}'$ by the derived execution. This proves that the derived execution is globally fair.

The simulator for I_4 is obtained by slightly modifying $\mathcal{S}_{KO}$: in an omissive interaction, the roles of the two agents are reversed, and the starter generates a joker token, instead of the reactor. Referring to Figure 1, the function $o(a_s)$ of I_4 is the same as the function $h(a_r)$ in I_3. The proof of correctness of this protocol is essentially the same as the above.

By applying this theorem to a system without omissions (i.e., plugging $o = 0$), we have:

Corollary 1 *Given $\Theta(|Q_{\mathcal{P}}|^2 \log n)$ bits of memory on each agent, every* TW *protocol can be simulated in* IT. □

4.2 Unique IDs

Now we assume that the agents have unique IDs as part of their initial state, and we give a TW simulator for the IO model, named $\mathcal{S}_{ID}$, which is reported in Figure 5. The idea is to use the uniqueness of the IDs to implement a locking mechanism that ensures the consistent matching of simulated state changes. Essentially, at a certain point an agent commits itself to executing a transition only with another agent with a specific ID. The locking scheme contains a rollback procedure to avoid deadlocks.

Simulator Variables. Each agent has the following variables: my_id for its own ID, $state_{sim} =$ avaible for the state of the simulator protocol, and $state_{\mathcal{P}}$ for the state of the simulated protocol. Moreover, it keeps two variables, id_{other} and $state_{other}$, which are the ID and the state of the other agent in the simulated two-way interaction.

Simulator Protocol. When an available reactor a_r, with ID r and simulated state $state_{\mathcal{P}} = q_r$, observes a starter a_s with ID s and $state^s_{sim} =$ available, it enters a pairing state. Moreover, it saves the ID s in id_{other} and the simulated state $state^s_{\mathcal{P}} = q_s$ of a_s in $state_{other}$ (see the details at Lines 3–5). The pairing state could be seen as a soft commitment in which a reactor picks a specific agent as a possible partner for a two-way interaction. In some specific conditions, an agent in the pairing state can roll back to the available state without completing a simulated two-way interaction; this will be covered later.

The simulation proceeds as soon as a_s, which is available, receives the information that some other agent a_r is in the pairing state and wants to pair up with an agent that has $my\_id = s$ and simulated state q_s. In this case a_s sets its simulator state to locked, stores a_r's simulated state and ID, and executes the transition $\delta_{\mathcal{P}}(state_{\mathcal{P}}, state_{other} = q_r)[0] = f_s(state_{\mathcal{P}}, state_{other})$. We remark that this happens only if the current simulated state of a_s is equal to the variable $state_{other}$ of a_r (see Line 6).

Suppose that a_s is locked; if a_r observes a_s, it executes the transition $\delta_{\mathcal{P}}(state^s_{\mathcal{P}}, state_{\mathcal{P}})[1] = f_r(state^s_{\mathcal{P}}, state_{\mathcal{P}})$, becomes available, and resets the variable id_{other} (see Lines 10–13). Now, if a_s is locked and observes that a_r's variable id_{other} is not s, then it resets its own state to available (see Lines 14–16).

It may happen, due to the IO model's nature, that a_s, with variable $state_{\mathcal{P}} = q_s$, induces an agent a_r to enter state pairing, but then a_s starts a two-way simulation with a different agent. In order to prevent a_r from waiting forever, we make it reset the pending transition if it encounters a_s again with $id^s_{other} \neq my\_id$ (this is incorporated in Lines 14–16).

Theorem 6 *Assuming* IO *and unique IDs, $\mathcal{S}_{ID}$ is a* TW *simulator.*

Proof Let us consider the simulation of a generic two-way protocol $\mathcal{P}$. Assume that an agent a_0 becomes pairing upon observing an agent a_1. Later, a_1 can either become locked with a_0, or pairing as well, upon observing some other agent a_2. It is clear that, if such a "chain" of pairing agents is formed, it must stop eventually. The last agent in the chain, say a_k, will then have to become locked upon observing some pairing agent with id_{other} equal to a_k's ID (which will eventually happen due to the global fairness condition).

Now, whenever an agent a_s enters state locked after observing an agent a_r in state pairing, it changes its simulated state according to $\delta_{\mathcal{P}}$, say at time t_s, and sooner or later also a_r will do the same, say at time t_r, with $t_r > t_s$. This is because a_r cannot start a new interaction with a_s between times t_s and t_r (since a_s would have to be in state available), and hence it will necessarily be seen by a_s with $id_{other} \neq s$, due to the global fairness condition. Moreover, a_s cannot change its own simulated state after t_s and before t_r, because it is locked.

1: $my\_id = unique\_ID;\ state_{\mathcal{P}} = initial\_state_{\mathcal{P}};\ id_{other} = \perp;\ state_{other} = \perp;\ state_{sim} = available$ ▷ (Agent's variables)
2: Upon Event Reactor delivers $(id^s, state^s_{\mathcal{P}}, id^s_{other}, state^s_{other}, state^s_{sim})$
3: **if** $(state_{sim} = available \wedge state^s_{sim} = available)$ **then**
4: $state_{sim} = pairing$
5: $id_{other} = id^s;\ state_{other} = state^s_{\mathcal{P}}$
6: **else if** $(state_{sim} = available \wedge state^s_{sim} = pairing \wedge id^s_{other} = my\_id \wedge state^s_{other} = state_{\mathcal{P}})$ **then**
7: $state_{sim} = locked$
8: $id_{other} = id^s;\ state_{other} = state^s_{\mathcal{P}}$
9: $state_{\mathcal{P}} = \delta_{\mathcal{P}}(state_{\mathcal{P}}, state_{other})[0]$
10: **else if** $(state_{sim} = pairing \wedge id_{other} = id^s \wedge id^s_{other} = my\_id \wedge state^s_{sim} = locked\ )$ **then**
11: $state_{sim} = available$
12: $id_{other} = state_{other} = \perp$
13: $state_{\mathcal{P}} = \delta_{\mathcal{P}}(state^s_{\mathcal{P}}, state_{\mathcal{P}})[1]$
14: **else if** $(id_{other} = id^s \wedge id^s_{other} \neq my\_id)$ **then**
15: $state_{sim} = available$
16: $id_{other} = state_{other} = \perp$

Fig. 5: Simulation protocol $\mathcal{S}_{ID}$

We have proved that infinitely many simulated state transitions must occur; these events can easily be paired up into a consistent perfect matching. We only have to prove that the derived execution satisfies the global fairness condition. We will do it in the case in which the system consists of $n \geq 3$ agents; the proof for the case $n = 2$ is simpler, and we omit it. Let $\mathcal{C}$ and $\mathcal{C}'$ be sets of configurations (closed under permutations) of $\mathcal{P}$, such that every configuration of $\mathcal{C}$ can become one of $\mathcal{C}'$ after a two-way interaction, and suppose that the derived execution passes through $\mathcal{C}$ infinitely many times. Let $\widetilde{\mathcal{C}}$ be the set of configurations of the simulator protocol whose simulated states are in $\mathcal{C}$ and let $\widetilde{\mathcal{C}}'$ be constructed similarly from $\mathcal{C}'$. (Note: if a configuration of the simulator protocol contains a locked agent a_s, the simulated state of its partner a_r is assumed to be the state it would reach after the interaction with a_s. This agrees with the definition of derived run given in Section 2.4.) By assumption, the simulation passes through $\widetilde{\mathcal{C}}$ infinitely often; we claim that it must go through $\widetilde{\mathcal{C}}'$ infinitely many times, as well. By definition of $\mathcal{C}$, for every $C_j \in \widetilde{\mathcal{C}}$, there is an interaction in $\mathcal{P}$ between two agents a_s and a_r that maps $\pi_{\mathcal{P}}(C_j)$ into $\pi_{\mathcal{P}}(C'_j)$, where $C'_j \in \widetilde{\mathcal{C}}'$. We will prove that such a C'_j can be reached from C_j after at most a constant number of interactions.

- If a_s is available in C_j and a_r is either available or pairing with a_s, then C'_j can be obtained by simply letting a_s and a_r interact together multiple times until they perform a full simulated interaction, and their states transition according to $\delta_{\mathcal{P}}$.
- If a_s or a_r (perhaps both) is locked in C_j, we let it interact with its current partner until the simulated interaction is completed and its internal state is again available. Then we proceed as in the other cases.
- If a_s is pairing in C_j or a_r is pairing with an agent that is not a_s, we have to make it become available without performing a full two-way interaction, and then we can proceed as in the other cases. Suppose that a_s is pairing (the case with a_r is handled similarly), and let a_q be the agent with which a_s is paired (perhaps $a_q = a_r$).
  - If a_q is pairing in C_j (of course not with a_s), then we let a_s observe a_q and roll back to the available state.
  - If a_q is available in C_j, we let it pair up with some other available agent (possibly a_r), and the we proceed as in the previous case. If an available agent does not exist, we can create one by letting some pairing agent roll back or some locked agent complete its current interaction, as explained in the first paragraph of the proof.
  - If a_q is locked in C_j, we let it finish the simulated interaction and become available. If it was locked with a_s, we are finished because now a_s is available too. Otherwise, we proceed as in the previous case.

As already observed, C'_j can be reached from C_j after at most a constant number c of interactions, and this holds for every j. By applying the definition of global fairness to the simulator's execution c times, we have that C'_j is indeed reached for infinitely many j's. Therefore $\widetilde{\mathcal{C}'}$ is reached infinitely many times, and thus so is $\mathcal{C}'$ by the derived execution. □

4.3 Knowledge of n

We give another result on simulating when additional knowledge is available to the agents. The proof uses a naming algorithm $\mathcal{N}_n$ in conjunction with $\mathcal{S}_{ID}$. This result complements the impossibility of Section 3.2, showing that a minimal amount of global knowledge, the size of the system, is enough to build a simulator. Moreover, the simulation is possible also under the stronger UO Adversary.

The following naming protocol, $\mathcal{N}_n$, uses the knowledge of n to give each agent a unique ID. This naming protocol is similar to the threshold protocol for IO presented in [5].

Simulator Variables. Each agent has a variables *my_id*, initially set to 1, and a flag *seen_n*, which is initially false. Moreover, it has all the variables used by protocol $\mathcal{S}_{ID}$ of Section 4.2.

Simulator Protocol. When an agent a_r is the reactor of an interaction and the starter a_s has its same value for *my_id*, then a_r increments its own variable *my_id*. Then, the flag *seen_n* of a_r becomes true if a_r's or a_s's current value of *my_id* is n. If the flag *seen_n* of a_r is true, then a_r executes protocol $\mathcal{S}_{ID}$ with ID the value of its own *my_id* variable.

Lemma 4 *In every globally fair execution of protocol $\mathcal{N}_n$ by a system of n agents in* IO*, eventually one agent will get my_id $= n$. As soon as this happens, all agents have unique IDs from 1 to n, and no agent's ID will ever change afterwards.*

Proof We can prove by induction that, at any time in the execution, there is a number M such that all agents have IDs in $\{1, \ldots, M\}$, and for every $m \in \{1, \ldots, M\}$ there is at least one agent with *my_id* $= m$. Indeed, this is true in the initial configuration, with $M = 1$. For the inductive step, assume that the proposition is true in some configuration $\mathcal{C}$, and let us prove that it remains true after two agents a_s and a_r interact, which results in the next configuration $\mathcal{C}'$. If the IDs of the two agents in $\mathcal{C}$ are different, they do not change: in this case, the proposition is still true in $\mathcal{C}'$. Otherwise, if the IDs have the same value, say m, then the ID of the reactor a_r is incremented. So, in $\mathcal{C}'$, the agent a_s still has the ID m, while a_r has $m + 1$: hence, no "gap" is introduced in the set of IDs. On the other hand, the value of M either remains the same or increases by 1 (if $m = M$ in $\mathcal{C}$). In both cases, the proposition is true in $\mathcal{C}'$.

We can also prove that, as long as $M < n$, the sum of the IDs of all the n agents keeps increasing. Indeed, if $M < n$, by the pigeonhole principle there must be at least two agents with the same ID. Since the execution is globally fair, eventually two such agents will interact, and one of them will increment its own ID, which increases the sum of all IDs. Eventually the largest ID will become $M = n$: when this happens, the n agents will have all the IDs from 1 to n. Therefore, necessarily all IDs will be distinct, and no ID will change again as a result of an interaction. □

The correctness of this protocol now immediately follows.

Theorem 7 *With the knowledge of $|A| = n$ and $\Theta(\log n)$ bits of memory on each agent, every* TW *protocol can be simulated in* IO*.*

Proof Let the agents execute protocol $\mathcal{N}_n$. By Lemma 4, eventually some agent a_r will get ID n, and at that point all IDs will be distinct. In a globally fair execution, eventually all agents will interact with a_r, set their flags *seen_n*, and start executing protocol $\mathcal{S}_{ID}$, whose correctness is given by Theorem 6. □

5 Conclusion

In this paper we have given a formal definition of two-way simulation in population protocols, and we identified several omission models. On top of this framework, we have given several impossibility results, as well as two-way simulators. Our results yield an almost comprehensive characterization, see Figure 4. The only gap left concerns the possibility of simulation in model T_2 when an upper bound on the number of omissions is known. As future work we are going to investigate this gap and study models where a unique leader agent is present. Our preliminary results, [26], in the latter direction show that the problem is far from trivial, and two-way simulation is still impossible in a wide set of models.

Acknowledgments. We would like to thank the anonymous reviewers for greatly improving the paper's readability. This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada under the Discovery Grant program and by Prof. Flocchini's University Research Chair.

References

1. D. Alistarh, J. Aspens and R. Gelashvili. "Space-optimal majority in population protocols", *29th Symposium on Discrete Algorithms, SODA*, 2018, pp. 2221–2239.
2. D. Alistarh and R. Gelashvili. "Polylogarithmic-time leader election in population protocols", *42th International Colloquium on Automata, Languages and Programming, ICALP*, 2015, pp. 479–491.
3. D. Alistarh, R. Gelashvili, and M. Vojnovic. "Fast and exact majority in population protocols", *34th Annual ACM Symposium on Principles of Distributed Computing, PODC*, 2015, pp. 47–56.
4. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. "Computation in networks of passively mobile finite-state sensors", *Distributed Computing*, Vol. 18(4), 2006, pp. 235–253.
5. D. Angluin, J. Aspnes, and D. Eisenstat. "On the power of anonymous one-way communication", *9th International Conference on Principles of Distributed Systems, OPODIS*, 2005, pp. 396–411.
6. D. Angluin, J. Aspnes, and D. Eisenstat. "Stably computable predicates are semilinear", *25th Annual ACM Symposium on Principles of Distributed Computing, PODC*, 2006, pp. 292–299.
7. D. Angluin, J. Aspnes, and D. Eisenstat. "A simple population protocol for fast robust approximate majority", *Distributed Computing*, Vol. 21(2), 2008, pp. 87–102.
8. D. Angluin, J. Aspnes, and D. Eisenstat. "Fast computation by population protocols with a leader", *Distributed Computing*, Vol. 21(3), 2008, pp. 61–75.
9. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. "The computational power of population protocols", *Distributed Computing*, Vol. 20(4), 2007, pp. 279–304.
10. J. Aspnes and E. Ruppert. "An introduction to population protocols", *Bulletin of the European Association for Theoretical Computer Science*, Vol. 93, 2007, pp. 98–117.
11. J. Beauquier, J. Burman, S. Clavière, and D. Sohier. "Space-optimal counting in population protocols", *29th International Symposium on Distributed Computing, DISC*, 2015, pp. 631–649.
12. J. Beauquier, J. Burman, and S. Kutten. "A self-stabilizing transformer for population protocols with covering", *Theoretical Computer Science*, Vol. 412(33), 2011, pp. 4247–4259.
13. O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. "On the convergence of population protocols when population goes to infinity", *Applied Mathematics and Computation*, Vol. 215(4), 2009, pp. 1340–1350.
14. S. Cai, T. Izumi, and K. Wada. "How to prove impossibility under global fairness: on space complexity of self-stabilizing leader election on a population protocol model", *Theory of Computing Systems*, Vol. 50(3), 2012, pp. 433–445.
15. D. Canepa, and M. Gradinariu Potop-Butucaru. "Self-stabilizing tiny interaction protocols", *3rd International Workshop on Reliability, Availability, and Security, WRAS*, pp. 1–6, 2010.
16. I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, and P. G. Spirakis. "Not all fair probabilistic schedulers are equivalent", *13th International Conference on Principles of Distributed Systems, OPODIS*, 2009, pp. 33–47.
17. I. Chatzigiannakis, O. Michail, S. Nikolaou, and A. Pavlogiannis. "Passively mobile communicating machines that use restricted space", *Theoretical Computer Science*, Vol. 412(46), 2011, pp. 6469–6483.
18. I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. "All symmetric predicates in NSPACE(n^2) are stably computable by the mediated population protocol model", *35th International Symposium on Mathematical Foundations of Computer Science, MFCS*, 2010, pp. 270–281.
19. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. "Stably decidable graph languages by mediated population protocols", *12th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS*, 2010, pp. 252–266.
20. I. Chatzigiannakis and P. G. Spirakis. "The dynamics of probabilistic population protocols", *22th International symposium on Distributed Computing, DISC*, 2008, pp. 498–499.
21. H.-L. Chen, R. Cummings, D. Doty, and D. Soloveichik. "Speed faults in computation by chemical reaction networks", *28th International Symposium on Distributed Computing, DISC*, 2014, pp. 16–30.
22. S. Das, G. A. Di Luna, P. Flocchini, N. Santoro, and G. Viglietta. "Mediated population protocols: Leader election and applications", *14th Annual Conference on Theory and Applications of Models of Computation, TAMC*, 2017, pp. 172–186.
23. C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. "What dependability for networks of mobile sensors?" *1st Workshop on Hot Topics in System Dependability, HotDep*, 2005, p. 8.
24. C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. "When birds die: making population protocols fault-tolerant", *2nd IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS*, 2006, pp. 51–66.
25. G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. "On the power of weaker pairwise interaction: fault-tolerant simulation of population protocols", *37th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2017, pp. 2472–2477.
26. G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. "Population protocols with faulty interactions: the impact of a leader", *Theoretical Computer Science*, Vol. 754, 2019, pp. 35–49.
27. M. Fischer and H. Jiang. "Self-stabilizing leader election in networks of finite-state anonymous agents", *10th International Conference on Principles of Distributed Systems, OPODIS*, 2006, pp. 395–409.
28. R. Guerraoui and E. Ruppert. "Even small birds are unique: population protocols with identifiers", *Technical Report CSE-2007-04, York University*, 2007.
29. R. Guerraoui and E. Ruppert. "Names trump malice: tiny mobile agents can tolerate byzantine failures", *36th International Colloquium on Automata, Languages and Programming, ICALP*, Vol. 16(2), 2009, pp. 484–495.
30. A. Kosowski, P. Uznanski. "Population Protocols are fast", *ArXiv e-prints,* 2018. https://arxiv.org/abs/1802.06872
31. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. "Mediated population protocols", *Theoretical Computer Science*, Vol. 412(22), 2011, pp. 2434–2450.
32. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. "New models for population protocols", *Synthesis Lectures on Distributed Computing Theory*, Morgan & Claypool, 2011.