The Heard-Of Model: Unifying all Benign Failures

Bernadette Charron-Bost
Ecole polytechnique, France

André Schiper
EPFL, Switzerland

Abstract

Problems in fault-tolerant distributed computing have been studied in a variety of models. These models are structured around two central ideas:

1. Degree of synchrony and failure model are two independent parameters that determine a particular type of system.
2. Failure and faulty component (i.e., the component responsible for the failure) are necessary and indissociable notions for the analysis of system behaviors.

In this work, we question these two dogmas and present a general computational model, suitable for describing any type of system with benign failures, that depends only on the notion of *transmission failure*.

In this model, computations evolve in rounds, and messages missed at a round are lost. Only information transmission is represented: for each round r and each process p, our model provides the set of processes that p "hears of" at round r (*heard-of set*) namely the processes from which p receives some message at round r. The features of a specific system are thus captured as a whole, just by a predicate over the collection of heard-of sets. We show that our model handles all types of benign failures, to be static or dynamic, permanent or transient, in a unified framework.

Using this new approach, we are able to give shorter and simpler proofs of important results (non-solvability, lower bounds). In particular, we prove that in general, Consensus cannot be solved without an implicit and permanent consensus on heard-of sets. We also examine Consensus algorithms in our model. In light of this specific agreement problem, we show how our approach allows us to devise new interesting solutions.

1 Introduction

Problems in fault-tolerant distributed computing have been studied in a variety of models. Such models are structured around two central ideas:

1. Degree of synchrony and failure model are two independent parameters that determine a particular type of system.
2. Failure and faulty component (i.e., the component responsible for the failure) are indissociable notions for the analysis of system behaviors.

In this paper we question these two dogmas and present a very general computational model, suitable for describing any type of system with benign failures, that is based only on the notion of *transmission failure.*

Computations in our model are composed of *rounds.* In each round, a process sends a message to the other processes, waits to receive messages from some processes, and then computes a new state. Every message received at some round has been sent at that round. Consequently any message missed at a round is definitely discarded. Using the terminology of Elrad and Francez [13], a round is a *communication-closed-layer.*

Most of the solutions to agreement problems that have been designed as well for synchronous message-passing systems as for partially synchronous or asynchronous ones are structured in rounds (e.g., [11, 1, 12, 10, 6]). However, concerning impossibility results (lower bounds, non-solvability, ...) in message-passing systems, round-based computational models have been considered almost always for synchronous systems. The reason for that lies in the fact that it is an open question whether round-based models are equivalent to the ones in which late messages are not discarded.

To the best of our knowledge, Dwork, Lynch, and Stockmeyer [12] are the first to define a round-based model for non synchronous computing. More precisely, they have generalized the classical round-based computational model for synchronous systems to a large class of partially synchronous systems. Then Gafni [15] extended the round-based model to any type of systems. The basic idea in his model is to study how a system evolves round-by-round and to abstract away the implementation of the communication between processes, be it shared-memory or message-passing. The properties of the communication mechanisms and system guarantees are captured as a whole by a single module that is called *Round-by-Round Failure Detector* (for short *RRDF*) module. More precisely, at each round r and each process p, the module provides a set of suspected processes from which p will not wait for a message (here, we call *messages* the pieces of information that are exchanged, whatever the medium of communication is). At this point, only non-transmission of information appears in the model: the reason why a process is suspected is not specified, whether it is due to the fact that the process is late or has crashed. In this way, *synchrony degree and failure model are encapsulated in the same abstract entity.*

The latter idea seems quite sound since separating synchrony degree and failure model breaks the continuum that naturally exists between them: for example, message asynchrony means that there is no bound on message delays, and message loss corresponds to infinite delays. Moreover, capturing synchrony degree and failures with the same abstraction gives us hope for relating different types of systems, in particular synchronous and asynchronous systems.

Unfortunately, this idea is not followed through to the end in [15] since the notion of failure model is underhandedly reintroduced *via* the one of *faulty component*. Indeed, the communication medium is implicitly assumed to be reliable (no anomalous delay, no loss) and when process p receives no message from q, the latter process is considered to be responsible for the transmission failure (q is late or has crashed). The so-called *Round-by-Round Failure Detector* modules only suspect processes, never links. Obviously, this impacts the design and correctness proofs of algorithms: for example, agreement problems are specified in [15] as usual, exempting faulty processes from making a decision.

The Round-by-Round Failure Detector model is here influenced by the whole literature — with one single exception, namely the work by Santoro and Widmayer [24] discussed in more details below — on fault-tolerant distributed computing that models transmission failures in terms of faulty components: for example, the loss of a message is attributed to a faulty behavior of either the sending process (send omission), the receiving process (receive omission), or the link. Unfortunately, the principle of *a priori* blaming some components for transmission failures yields several major problems. First, it may lead to undesirable conclusions: for example, in the send-omission failure model, the entire system will be considered faulty even if only one message from each process is lost. Second, it allows faulty processes to have deviant behaviors since problem specifications exempt faulty processes to satisfy some conditions (in decision problems, a faulty process is not obliged to make a decision). Finally, it appears that most of the time, the real causes of transmission failures, namely sender failure, receiver failure, or link failure, are actually unknown. Failure transmissions are often ascribed to some components in a totally arbitrary manner that may not correspond to reality.

Moreover, there is *no prima facie evidence* that the notion of faulty component is really helpful in the analysis of fault-tolerant systems. We show that our model leads to the development of new conditions guaranteeing the correctness of fault-tolerant algorithms, and to shorter and simpler proofs. This is due to the fact that the notion of faulty component unnecessarily overloads system analysis with non-operational details. In other words, it is sufficient that the model just notifies transmission failures (effects) without specifying faulty components (causes).

Santoro and Widmayer [24] pointed out this problem clearly. They introduce the *Transmission Faults Model* that locates failures without specifying their cause. A transmission failure can represent link failure as well as process failure. Contrary to classical models in which transmission failures involve only messages sent or received by an unknown but static set of processes (the so-called *faulty processes*), the transmission faults model is well-adapted to *dynamic* failures. The Transmission Faults Model is designed only for synchronous systems. Indeed, Santoro and Widmayer showed that dynamic transmission failures in synchronous systems have the same negative effect as asynchronicity. This *de facto* reintroduces synchrony degree and failure model as two separated parameters of systems.

Contribution

Our aim is to develop a computational model that combines the advantages of the models in [24, 15], but avoids their drawbacks. We propose a round-based model, called *Heard-Of* (*HO* for short) in which (1) synchrony degree and failure model are encapsulated in the same abstract structure, and (2) the notion of faulty component (process or link) has totally disappeared. The HO model merely notifies transmission failures without specifying by whom nor why such failures occur. More precisely, computations in the HO model evolves in rounds. In each round, a process sends a message to all the others, and then waits to receive messages from the other processes. Communication missed at a round is lost. For each round r and each process p, $HO(p,r)$ denotes the set of processes that p has "heard of" at round r, namely the processes from which p receives some message at round r. A transmission failure from q to p at round r is notified by the fact that q does not belong to $HO(p,r)$. The features of a specific system are captured in the HO model as a whole, just by a predicate over the collection of the $HO(p,r)$'s, called *communication predicate.*

The HO model handles all the types of benign failures, to be static or dynamic, permanent or transient, in a unified framework. In particular, the model can naturally represent link failures, contrary to models with failure detectors [6, 15]. Indeed, in such models, when the failure detector module indicates to some process p to stop waiting for a message from q, this is interpreted as "q is (suspected to be) faulty". Obviously, such an interpretation makes no sense if links may lose messages. This drastically limits the scope of positive results (design of fault-tolerant algorithms) in models with failure detectors, as well for theoretical as for practical viewpoint.

Another feature of the HO model is that contrary to the random model [23, 1] or the failure detector approach, there is no notion of "augmenting" asynchronous systems with external devices (oracles) that processes may query: the communication predicate corresponding to an HO system is an integral part of the model and should be rather seen as defining the *environment.* The weaker the predicate of an HO system is, the more freedom the environment allows the system, and the harder it is to solve problems. The HO abstraction (communication predicates) is supported only by the messages sent in the HO algorithm. In other words, we cannot decouple predicates from the underlying algorithms. This is the reason why we encapsulate algorithm and communication predicate in the same structure that we shall call *HO machine*.

Besides the very definition of the HO model, we present various results about the Consensus problem. Our first result concerns systems that never partition; it characterizes the minimal communication predicate needed to solve Consensus in such systems. To do so, we first introduce the concept of *translation* of communication predicates. Informally, a communication predicate $\mathcal{P}$ can be translated into another one $\mathcal{P}'$ if there is a distributed algorithm that transforms heard-of sets satisfying $\mathcal{P}$ into new ones satisfying $\mathcal{P}'$. Any problem that is solvable under $\mathcal{P}'$ is then solvable under $\mathcal{P}$ instead. The so-defined relation is transitive, and thus orders communication predicates with respect to their abilities to solve problems; so we shall say that $\mathcal{P}$ is *at least as strong as* $\mathcal{P}'$.

Of special interest is the communication predicate $\mathcal{P}^*_{sp\_unif}$ which guarantees that at each round, all processes hear of the same non-empty subset of processes. Such a permanent operational agreement on heard-of sets suffices to solve Consensus under the condition that there is no heard-of set partitioning. Conversely and more surprisingly, we show that if Consensus is solvable under the communication predicate $\mathcal{P}$, then $\mathcal{P}^*_{sp\_unif}$ can be emulated from $\mathcal{P}$ – that is, $\mathcal{P}$ is at least as strong as $\mathcal{P}^*_{sp\_unif}$. In other words, Consensus cannot be solved without an implicit permanent agreement on the heard-of sets.

Then we describe four basic translations. Using these translations, we prove several results related to the communication predicate that guarantees every round has a non-empty *kernel*, (i.e., at each round there is some process that is heard by all). We show that non-empty kernel rounds can be emulated by majority heard-of sets, and more generally, in any system that never partitions. By means of these basic translations, we also give a simple direct proof of the reduction of the worst-case synchronous lower bounds [19] to the general FLP asynchronous impossibility result [14] (this reduction has been previously established by Gafni for Atomic-Snapshot asynchronous systems [15]). This exemplifies how, by getting rid of the "first dogma" which artificially separates synchrony degree and failure model, we can relate synchronous and asynchronous systems, and take advantage of this relation to relate impossibility results that are traditionally considered as quite different in essence.

Finally we study how to solve Consensus in systems prone to partitioning. The HO formalism enables us to describe well-known Consensus algorithms in a clear and concise way, and also to design new solutions. For each Consensus algorithm, we determine a simple communication predicate which guarantees correctness. Interestingly, all the communication predicates that we display express conditions that have to hold just sporadically, contrary to the perpetual correctness conditions stated for classical models such as the partially synchronous models in [9, 12] or the Failure Detector models [6]. Consequently the HO model appears as a natural formalism in which we can express thin grained conditions with respect to time. Moreover, in many real systems, we observe series of "bad" and "good" periods in regards to both synchrony and failures. Since they just require sporadic conditions on heard-of sets, the algorithms that we examine are well-adapted to such systems, and so are quite realistic solutions to the Consensus problem.

This paper is structured as follows. In Section 2, we describe our model, and present many traditional systems in the HO framework. In Section 3, we define the notion of translation and propose a characterization of the communication predicates that make Consensus solvable (under certain transmission failure bound). In Section 4, we give four basic translations, and highlight the key role played by the "no partitioning" assumption. In Section 5, we describe several Consensus algorithms, and determine HO conditions for their correctness. Section 6 concludes the paper.

2 HO model

As explained in the Introduction, computations in our model are composed of rounds, which are communication-closed layers in the sense that any message sent in a round can be received only at that round. The technical description of computations is similar to the ones in [12] and [15], and so the model generalizes the classical notion of synchronized rounds developed for synchronous systems [18]. We introduce the notion of *kernel* at round r that represents what processes share during round r from the operational viewpoint. As we shall show further, this notion plays a key role to solve Consensus.

2.1 Heard-Of sets and communication predicates

We suppose that we have a non-empty set Π of cardinality n, a set of messages M, and a *null* placeholder indicating the empty message. To each p in Π, we associate a *process*, which consists of the following components: a set of states denoted by $states_p$, a subset $init_p$ of initial states, for each positive integer r called *round number*, a message-sending function S_p^r mapping $states_p \times \Pi$ to a unique (possibly *null*) message, and a state-transition function T_p^r mapping $states_p$ and partial vectors (indexed by Π) of elements of $M \cup \{null\}$ to $states_p$. In each round r, process p first applies S_p^r to the current state, emits the "messages" to be sent to each process, and then, for a subset $HO(p, r)$ of Π (indicating the processes which p *hears of*), applies T_p^r to its current state and the partial vector of incoming messages whose support is $HO(p, r)$. The collection of processes is called an *algorithm on* Π.

Computation evolves in an infinite sequence of rounds. For each computation, we determine its *heard-of collection* which is the collection of subsets of Π:

$$(HO(p, r))_{p\in\Pi,\ r>0}\,.$$

A *communication predicate* $\mathcal{P}$ is defined to be a predicate over collections of subsets of Π (representing heard-of collections) that is invariant under time translation, i.e., $\mathcal{P}$ has the same truth-value for all the heard-of collections $(HO(p, r + i))_{p\in\Pi,r>0}$, where $i \in \mathbb{N}$.[1] Note that if $\mathcal{C}$ is a condition over the heard-of sets at some round, then the natural communication predicate that guarantees $\mathcal{C}$ eventually holds at some round is the following:

$$\mathcal{P}_{(\mathcal{C})^\infty} \ :: \ \forall r > 0,\ \exists r_0 \geq r : \mathcal{C} \text{ holds at } r_0,$$

which expresses that $\mathcal{C}$ holds infinitely often.

For any round r, its *kernel* is defined as the set of processes

$$K(r) = \bigcap_{p\in\Pi} HO(p, r).$$

Intuitively, it consists of the processes which are heard by all at round r. More generally, we introduce the kernel $K(\phi)$ of any set ϕ of rounds as:

$$K(\phi) = \bigcap_{r\in\phi} K(r).$$

[1] The latter condition is due to the fact that we do not want correctness of algorithms depends on the time at which algorithms start to run (see Proposition 2.1).

When ϕ is the set of all the rounds in the computation, this defines the *(global) kernel* of the computation:

$$K = \bigcap_{r>0} K(r).$$

It will be convenient to introduce the *cokernel* of some round, or more generally of some collection of rounds, as the complement in Π of the above defined kernels. Thus, with the same notation as above, we let

$$coK(r) = \Pi \setminus K(r),\ coK(\phi) = \Pi \setminus K(\phi),\ \text{and } coK = \Pi \setminus K.$$

Round r is said to be a *nek* (for *non-empty kernel*) *round* if $K(r) \neq \emptyset$.

Round r is said to be *uniform* when, for any two processes p, q in Π,

$$HO(p,r) = HO(q,r),$$

and it is said to be *split* when there exist two processes p, q in Π such that

$$HO(p,r) \cap HO(q,r) = \emptyset.$$

Obviously, a nek round is not split.

A *nek computation* is a computation whose global kernel is non-empty. In such a computation, there is at least one process from which every process hears during the whole computation. A computation is said to be *space uniform* when each of its rounds is uniform. It is said to be *time uniform* when the sets $HO(p,r)$ do not vary according to time:

$$\forall r > 0, \forall p \in \Pi\ :\ HO(p,r) = HO(p,r+1).$$

Finally, a computation is said to be *regular* when a process that is not heard by some process at some round is not heard by any process later:

$$\forall r > 0, \forall p \in \Pi\ :\ HO(p,r+1) \subseteq K(r).$$

Note that regularity is a weak form of the combination of space and time uniformity.

Equivalently, we would rather consider *talked-to* sets, denoted $TT(p,r)$ and defined by

$$TT(p,r) = \{q \in \Pi\ :\ p \in HO(q,r)\},$$

which are the dual notion of heard-of sets. Contrary to $HO(p,r)$, process p cannot know $TT(p,R)$ at the end of round r, and this is the reason why we have preferred to express the communication properties of computations in terms of their heard-of collections instead of their talked-to collections.

2.2 HO machines

A *Heard-Of machine* (or *HO machine* for short) for Π is a pair $M = (A, \mathcal{P})$ where A is an algorithm on Π and $\mathcal{P}$ is a communication predicate. For example, we shall consider the HO machines with the communication predicate:

$$\mathcal{P}_{sp\_unif}\ ::\ \forall r > 0,\ \forall p, q \in \Pi^2\ :\ HO(p,r) = HO(q,r),$$

that is HO machines with space uniform computations, and those with regular computations:

$$\mathcal{P}_{reg}\ ::\ \forall r > 0,\ \forall p \in \Pi\ :\ HO(p,r+1) \subseteq K(r).$$

We shall also consider the class of HO machines that share the “no split” predicate:

$$\mathcal{P}_{nosplit} \ :: \ \forall r > 0, \ \forall p, q \in \Pi^2 \ : \ HO(p, r) \cap HO(q, r) \neq \emptyset,$$

the subclass of HO machines with the communication predicate:

$$\mathcal{P}_{nekrounds} \ :: \ \forall r > 0 \ : \ K(r) \neq \emptyset,$$

and the one with the stronger communication predicate:

$$\mathcal{P}_{nek} \ :: \ K \neq \emptyset.$$

More generally, we introduce the communication predicate:

$$\mathcal{P}_{K}^{f} \ :: \ |coK| \leq f$$

which is equivalent to

$$|K| \geq n - f.$$

We shall also consider the weaker predicate:

$$\mathcal{P}_{HO}^{f} \ :: \ \forall r > 0, \forall p \in \Pi \ : \ |HO(p, r)| \geq n - f,$$

and more specifically $\mathcal{P}_{HO}^{maj} = \mathcal{P}_{HO}^{[\frac{n-1}{2}]}$ that asserts every heard-of set is a majority set.

A *run* of $M = (A, \mathcal{P})$ is totally determined by a set of initial states (one per process) and a heard-of collection that satisfies $\mathcal{P}$. To each run corresponds the collection of the states (one per process and per round) reached by processes during the run. In all the sequel, given a run of M, for any variable X_p of process p, $X_p^{(r)}$ will denote the value of X_p after r rounds of this run.

A *problem* Σ *for* Π is a predicate over state collections. An HO machine $M = (A, \mathcal{P})$ solves a problem Σ if the state collection in each of its runs satisfies Σ; then we say that problem Σ is *solvable under* $\mathcal{P}$.

Since communication predicates are invariant under time translation, and round numbers are not part of processes states, correctness of algorithms does not depend on the time at which algorithms start to run. Formally, for any integer i and any HO algorithm A with the message-sending and state-transtion functions S_p^r and T_p^r respectively, let ${}^{i}A$ denote the algorithm defined as A is but with the message-sending functions ${}^{i}S_p^r = S_p^{r+i}$ and the state-transition functions ${}^{i}T_p^r = T_p^{r+i}$.

Proposition 2.1 *If the HO machine $M = (A, \mathcal{P})$ solves Σ, then for any integer i, the HO machine ${}^{i}M = ({}^{i}A, \mathcal{P})$ also solves Σ.*

In this paper, we concentrate on the *Consensus* problem, specified in our approach by the following conditions:

Integrity: Any decision value is the initial value of some process.

Agreement: No two processes decide differently.

Termination: All processes eventually decide.

Since there is no notion of faulty process in the HO model, a process is never exempted from making a decision. Such a strong liveness requirement may seem unreasonable in two respects. First, it may make Consensus needlessly unsolvable in the sense that the resulting Consensus specification might be unsolvable under some communication predicate $\mathcal{P}$ whereas the classical Consensus problem is solvable under $\mathcal{P}$. The paper shows that this objection does not hold. Secondly, one may wonder whether an algorithm in which *all* processes decide can be implemented in real systems with crash failures. The answer is yes. Of course, a process that has crashed takes no step, and so can make no decision. However, such a process is not heard of any more. Consequently, a crashed process has no impact on the rest of the computation. This is why there is no problem of transposing an HO machine solving the Consensus specification given above in a system with possible crash failures: processes that have crashed simply do not decide.

One major advantage of this approach is for models with transient failures: a faulty process that commits send- or receive-omissions but does not crash *is not* exempted from making a decision, a quite reasonable requirement. As a matter of fact, this specification of the Consensus problem requiring any process to make a decision already appears in several fundamental papers dealing with benign failures [20, 3, 16].

2.3 How to guarantee communication predicates

Obviously, an HO machine is implementable in a system as soon as the corresponding communication predicate can be guaranteed by the system. In Table 1, we go over various classical types of message-passing systems of interest — which only handle static failures as explained in the Introduction — and we examine the communication predicates that they can guarantee. For each type of system listed in Table 1 except asynchronous systems with initial crash failures, we use several results previously established in [9, 12, 15]. As for asynchronous systems with at most f initial crash failures, they clearly support the communication predicate $\mathcal{P}^f_{\diamond unif}$ defined by:

$$\mathcal{P}^f_{\diamond unif} \ :: \ \exists r_0 > 0, \exists \Pi_0 \in 2^\Pi \text{ s.t. } |\Pi_0| \geq n - f, \forall p \in \Pi, \forall r \geq r_0 \ : \ HO(p,r) = \Pi_0.$$

Moreover, the positive result by Fischer, Lynch, and Paterson [14] for initial crash failures shows that in the case of a majority of correct processes ($2f < n$), space-time uniformity of the HO's can be achieved from the beginning.[2] That is, HO machines with the predicate

$$\mathcal{P}^f_{unif} \ :: \ \exists \Pi_0 \in 2^\Pi \text{ s.t. } |\Pi_0| \geq n - f, \forall p \in \Pi, \forall r > 0 \ : \ HO(p,r) = HO$$

can be implemented in any asynchronous system provided a majority of processes is correct.

2.4 HO counterpart of a system

At this point, we are naturally led to examine the notion of *HO counterpart of a system* introduced by Gafni [15]. Roughly speaking, the HO counterpart of a system S is a communication predicate $\mathcal{P}$ which exactly captures all the properties (synchrony degree, failure model, ...) of S. With respect to the partial order that we shall formally define in Section 3.1, $\mathcal{P}$ corresponds to the strongest communication predicate that can be implemented from S. The notion of HO counterpart of a system rises two fundamental questions:

1. Does any system S have an HO counterpart $\mathcal{P}$?

[2] The algorithm in [14] ensures agreement on the membership of the initial clique.

2. If it exists, does $\mathcal{P}$ really capture all the properties of S? And what is more, is it possible to give a rigorous meaning to the latter question?

We leave these questions open,[3] and in the sequel we shall use the notion of HO counterpart informally. In Table 1, we give what we reasonably think to be the HO counterparts of various system types (i.e., the strongest implementability results).

As an example, $\mathcal{P}^f_{\diamond unif}$ is the HO counterpart of the partially synchronous systems with at most f crash failures described in [12]. Note that $\mathcal{P}^f_{\diamond unif}$ implies the HO counterpart of asynchronous systems with at most f initial crash failures when $f \geq n/2$. Interestingly, this explains why all these systems share the same "partitioning argument" to prove the impossibility of Consensus [14, 12, 6].

System Type	Communication Predicate
Synchronous, reliable links at most f faulty processes by send omission	$\lvert K\rvert \geq n - f$
Synchronous, reliable links at most f faulty processes by crash	$\lvert K\rvert \geq n - f$ $\wedge$ $\forall p \in \Pi, \forall r > 0 \,:\, HO(p, r+1) \subseteq K(r)$
Synchronous, reliable links asynchronous processes, atomic send to all at most f faulty processes by crash ([9])	$\forall p \in \Pi, \forall r > 0 \,:\, \lvert HO(p,r)\rvert \geq n - f$ $\wedge$ $\forall p, q \in \Pi^2, \forall r > 0 \,:\, HO(p,r) = HO(q,r)$
Synchronous, reliable links asynchronous processes, at most 1 faulty process by crash ([9])	$\forall p \in \Pi, \forall r > 0 \,:\, 1 \leq \lvert HO(p,r)\rvert \leq 2$ $\wedge$ $\forall p, q \in \Pi^2, \forall r > 0 \,:\, HO(p,r) = HO(q,r)$
Asynchronous, reliable links at most f faulty processes by crash	$\forall p \in \Pi, \forall r > 0 \,:\, \lvert HO(p,r)\rvert \geq n - f$
Asynchronous, reliable links at most f faulty processes by initial crash	$\forall p \in \Pi \,:\, \lvert HO(p,1)\rvert \geq n - f$ $\wedge\ (\forall p \in \Pi, \forall r > 0 \,:\, HO(p,r) \subseteq HO(p,r+1)) \wedge$ $\exists \Pi_0 \subseteq \Pi, \exists r_0 > 0, \forall p \in \Pi, \forall r > r_0 \,:\, HO(p,r) = \Pi_0$
Same with $f < n/2$	$\exists \Pi_0 \subseteq \Pi \text{ s.t. } \lvert\Pi_0\rvert \geq n - f, \forall p \in \Pi, \forall r > 0 \,:$ $HO(p,r) = \Pi_0$
Partially synchronous ([12]) Eventual reliable links at most f faulty processes by crash	$\exists \Pi_0 \subseteq \Pi \text{ s.t. } \lvert\Pi_0\rvert \geq n - f, \exists r_0 > 0, \forall p \in \Pi, \forall r > r_0 \,:$ $HO(p,r) = \Pi_0$

Table 1: Systems type and their HO counterparts

[3] The same question arises for models with failure detectors: for example, does the model with the *eventual perfect failure detector* $\diamond\mathcal{P}$ entirely capture the computational power of the partially synchronous systems in [12]? Except in [8], which provides a negative answer in the case of the perfect failure detector and synchronous systems, this question has not been addressed, and so is left open.

3 Communication predicates to solve Consensus

In this section, we address the fundamental question of determining the computational models in which Consensus is solvable. In terms of HO models, it consists in identifying the communication predicates of HO machines that solve Consensus.

We partially answer the question by limiting us to the class of communication predicates from which nek rounds can be emulated: in this class of HO models, we prove that permanent space uniformity is a necessary and sufficient condition for solving Consensus. In other words, *Consensus cannot be solved without an implicit permanent consensus on heard-of sets.* We then compare our result with the one established by Chandra, Hadzilacos, and Toueg [5] for asynchronous systems augmented with failure detector oracles.

We start by formalizing what it means for an HO machine $M = (A, \mathcal{P})$ to emulate a communication predicate $\mathcal{P}'$. To do that, we first define the notion of a k round translation from $\mathcal{P}$ to $\mathcal{P}'$, and then its generalization to translations that take a non constant number of rounds. Translations of the first type are called *uniform translations*.

3.1 Uniform translations

Let k be any positive integer, and let A be an algorithm that maintains a variable $NewHO_p$ at every process p, which contains a subset of Π. We call *macro-round* ρ the sequence of the k consecutive rounds $k(\rho - 1) + 1, \ldots, k\rho$. The value of $NewHO_p$ at the end of macro-round ρ is denoted $NewHO_p^{(\rho)}$. We say that the HO machine $M = (A, \mathcal{P})$ *emulates* the communication predicate $\mathcal{P}'$ in k rounds if for any run of M, the following holds:

E1: If process q belongs to $NewHO_p^{(\rho)}$, then there exist an integer l in $\{1, \ldots, k\}$, a chain of $l + 1$ processes $p_0, p_1, \ldots, p_l$ from $p_0 = q$ to $p_l = p$, and a subsequence of l rounds $r_1, \ldots, r_l$ in macro-round ρ such that for any index i, $1 \leq i \leq l$, we have

$$p_{i-1} \in HO(p_i, r_i).$$

E2: The collection $\left(NewHO_p^{(\rho)}\right)_{p \in \Pi, \rho > 0}$ satisfies predicate $\mathcal{P}'$.

Condition E1 states that if q is in $NewHO_p$ at macro-round ρ, then p has actually heard of q during this macro-round through some intermediate processes $p_1, \ldots, p_{l-1}$. Hence this condition excludes trivial emulations of $\mathcal{P}'$. Condition E2 states that the variables $NewHO_p$ simulate heard-of sets satisfying $\mathcal{P}'$. If there exists an algorithm A such that the HO machine $M = (A, \mathcal{P})$ emulates $\mathcal{P}'$ in k rounds, then we write $\mathcal{P} \succeq_k \mathcal{P}'$, and we say that A is a k *round translation* of $\mathcal{P}$ into $\mathcal{P}'$.

Note that if $\mathcal{P} \Rightarrow \mathcal{P}'$, the trivial algorithm in which each process p writes the value of $HO(p, r)$ into $NewHO_p$ at the end of each round r is a one round translation of $\mathcal{P}$ into $\mathcal{P}'$, and so $\mathcal{P} \succeq_1 P'$.

3.2 General translations

Now we generalize the previous definition to translations that take a non-constant number of rounds in time and space. For that, each process p maintains an additional variable $MacroRound_p$ initialized to 0. Upon updating $NewHO_p$, process p increments $MacroRound_p$ by 1. When p sends a basic message m, it tags m with the current value of $MacroRound_p$. Moreover, p ignores any message tagged by an integer different to the current value of $MacroRound_p$. Then rephrasing the condition E1 as follows

E1: If process q belongs to $NewHO_p^{(\rho)}$, then there exist a chain of processes $p_0, p_1, \ldots, p_l$ from $p_0 = q$ to $p_l = p$, and a subsequence of l rounds $r_1, \ldots, r_l$ such that for any index i, $1 \leq i \leq l$, we have

$$r_i < r_{i+1},\ MacroRound_{p_i}^{(r_i)} = \rho,\ \text{and } p_{i-1} \in HO(p_i, r_i)\,.$$

yields a general definition of translation.

If there exists an algorithm A such that the HO machine $M = (A, \mathcal{P})$ emulates $\mathcal{P}'$, we write $\mathcal{P} \succeq \mathcal{P}'$, and we say that A *translates* $\mathcal{P}$ into $\mathcal{P}'$. Obviously, the relation $\succeq$ contains all the relations $\succeq_k$.

Given an emulation of $\mathcal{P}'$ by an HO machine $(A, \mathcal{P})$, any problem that can be solved with $\mathcal{P}'$, can be solved with $\mathcal{P}$ instead. To see this, suppose the HO machine $(B, \mathcal{P}')$ solves a problem Σ. We compose A and B in the following way: each process p executes B with rounds that are "split" by A. More precisely, concurrently with B, every process p runs A, and so (locally) determines A macro-rounds and maintains the variable $A.NewHO_p$. The algorithm B at process p is then modified as follows: messages of A during A *macro-round* ρ piggyback messages sent by B at *round* ρ, and p computes its new state at the end of macro-round ρ by applying B's state-transition function at round ρ to (1) its state (with respect to B) at the beginning of ρ and (2) the partial vector of B's messages indexed by $A.NewHO_p^{(\rho)}$.

Proposition 3.1 *If Σ is solvable under $\mathcal{P}'$ and $\mathcal{P} \succeq \mathcal{P}'$, then Σ is solvable under $\mathcal{P}$.*

The relation $\succeq$ is clearly transitive; thus it orders communication predicates with respect to their ability to solve problems. If both $\mathcal{P} \succeq \mathcal{P}'$ and $\mathcal{P}' \succeq \mathcal{P}$ hold, then we say that $\mathcal{P}$ and $\mathcal{P}'$ are *equivalent*, and we denote $\mathcal{P} \simeq \mathcal{P}'$.

As we shall see below, an important class of translations are those that preserve kernels. More precisely, we introduce the notion of a *kernel preserving translation* from $\mathcal{P}$ to $\mathcal{P}'$ which is defined as an emulation of $\mathcal{P}'$ with an HO machine $M = (A, \mathcal{P})$ such that for any run of M, we have:

$$\bigcap_{p \in \Pi, r \in \rho_p} HO(p, r) \subseteq \bigcap_{p \in \Pi} NewHO_p^{(\rho)},$$

where ρ_p denotes the set of rounds that together form the macro-round ρ on p.

3.3 Consensus and nek rounds

Let $\mathcal{P}^*_{sp\_unif}$ denote the communication predicate that guarantees that any round s uniform and non-trivial, that is $\mathcal{P}^*_{sp\_unif} = \mathcal{P}_{sp\_unif} \wedge \mathcal{P}^*$ with

$$\mathcal{P}^* \ :: \ \forall r > 0, \forall p \in \Pi \ : \ HO(p, r) \neq \emptyset.$$

Proposition 3.2 *Let $\mathcal{P}$ be a communication predicate such that $\mathcal{P} \succeq \mathcal{P}^*_{sp\_unif}$. Then there exists an algorithm A such that the HO machine $M = (A, \mathcal{P})$ solves Consensus.*

Proof: Let A_0 be an algorithm on Π such that $(A_0, \mathcal{P})$ emulates $\mathcal{P}^*_{sp\_unif}$, and let us fix an arbitrary order $p_1, \ldots, p_n$ on Π. Let A be identical to A_0, except that

1. at each round, each process sends its knowledge about initial values to all;

2. at the end of the first macro-round, each process decides the initial value of the first process in $NewHO_p$, according to the order $p_1, \ldots, p_n$.

Note that thanks to E2, every $NewHO_p$ is non-empty at the end of macro-round 1. Moreover, from E1 it follows that the decision rule is well-defined since each process knows the initial values of all the processes in the set $NewHO_p$. Hence the decision rule is well-defined, and termination is satisfied. Integrity is a straightforward consequence of item 2. Agreement follows from E2. □

This result can be interestingly compared with the impossibility of Consensus with the communication predicate $\mathcal{P}^*_{\diamond unif}$ (cf. Section 2.3). This shows that *eventual* space-time uniformity is not sufficient for Consensus, whereas space uniformity alone makes Consensus solvable provided it holds *from the beginning*.

Conversely, the following proposition shows that in the class of HO machines with nek rounds, space uniformity can be achieved permanently if Consensus is solvable. In other words, *Consensus is solvable only if there is an implicit permanent agreement on the first heard-of sets.*

Proposition 3.3 *Let $\mathcal{P}$ be a communication predicate such that $\mathcal{P} \succeq \mathcal{P}_{nekrounds}$. If there is an HO machine $(A, \mathcal{P})$ that solves Consensus, then $\mathcal{P} \succeq \mathcal{P}^*_{sp\_unif}$.*

Proof:

Let B be an algorithm that emulates $\mathcal{P}_{nekrounds}$ from $\mathcal{P}$. From A and B, we design an algorithm C and prove that $(C, \mathcal{P})$ emulates $\mathcal{P}^*_{sp\_unif}$.

To simulate a macro-round with C, every process p first executes one macro-round of B and records the value of $B.NewHO_p$ at the end of the macro-round in some variable $Propose_p$. Then it executes n instances of A in parallel, where each solves Consensus (cf. Proposition 2.1). The initial value of p for the i-th instance of A is the truth-value of "$p_i \in Propose_p$". From the decision values, p sets

$$C.NewHO_p := \{p_i \in \Pi \,:\, p \text{ decides "}\texttt{true}\text{" for the } i\text{-th Consensus}\}.$$

By the agreement condition of Consensus, the emulated macro-round is uniform. Moreover, since B emulates $\mathcal{P}_{nekrounds}$, there is at least one process p_i that belongs to all the $B.NewHO_p$'s, and so all the initial values for the i-th Consensus are equal to "`true`". By the integrity condition of Consensus, the only possible decision value is "`true`". In other words, we have

$$p_i \in \bigcap_{p \in \Pi} C.NewHO_p.$$

This shows that the emulated uniform macro-round is non-trivial, and so E2 is satisfied.

We now argue E1. Let $q \in C.NewHO_p$; by the integrity condition of Consensus, there is some processes x such that $q \in Propose_x$. By the *Knowledge Transfer* theorem [7], for one of them, say x_1, there is a finite sequence of processes $x_2, \ldots, x_k = p$ such that during the execution of the i-th instance of A, x_1 sends a message m_1 to x_2, x_2 sends a message m_2 to x_3 after receiving m_1, …, x_{k-1} sends a message m_{k-1} to $x_k = p$ after receiving m_{k-2}. Hence any process in $C.NewHO_p$ is connected to p during the execution of A which is part of the macro-round. □

Propositions 3.2 and 3.3 provide a characterization of the HO machines with nek rounds that solves Consensus:

Theorem 3.4 *In the class of communication predicates which are at least as strong as $\mathcal{P}_{nekrounds}$, the following assertions are equivalent:*

1. *There is an algorithm A such that $M = (A, \mathcal{P})$ solves Consensus;*

2. $\mathcal{P} \succeq \mathcal{P}^*_{sp\_unif}$.

Note that $\mathcal{P}^*_{sp\_unif}$ is the HO counterpart of one of the system types described in [9], namely systems with asynchronous processes, synchronous and reliable links, atomic send-to-all primitive, and at most $n - 1$ crashes.

Remark: In [5], Chandra, Hadzilacos and Toueg characterize the failure detectors that make Consensus solvable in a system with reliable links and a minority of crash failures. Such a system can emulate the predicate $\mathcal{P}^{maj}_{HO}$. In Section 4.1, we prove that $\mathcal{P}^{maj}_{HO}$ can be translated into $\mathcal{P}_{nekrounds}$, for which Theorem 3.4 applies. Therefore our result, whose proof is quite simple and direct, has the same scope as the one in [5].

4 Basic communication predicate translations

In this section, our aim is to establish some relationships among communication predicates, and to outline a first (partial) map of various classes of these predicates that play a key role for solving Consensus. To do so, we describe several fundamental translations that are all uniform. Such translations simply handle union and intersection of heard-of sets. Interestingly, some of them allow us to amplify our characterization of nek round communication predicates that make Consensus solvable. We compare these translations with other ones that have been given in the literature in the context of the classical taxonomy of system types. Our main results are summarized in Figure 1 at the end of the section.

4.1 A two round translation for increasing kernels

First we present a two round translation and prove a lower bound on the membership of the (new) kernels of macro-rounds: The translation increases kernels in some significant cases. In particular, the translation transforms $\mathcal{P}^{maj}_{HO}$ into $\mathcal{P}_{nekrounds}$, that is $\mathcal{P}^{maj}_{HO} \succeq \mathcal{P}_{nekrounds}$. This translation also provides a direct proof of a very interesting result established by Gafni [15] relating synchronous and asynchronous models.

The translation computes $NewHO^{(\rho)}_p$ from the collection of heard-of sets at rounds $2\rho - 1$ and 2ρ as follows (see Algorithm 1):

$$NewHO^{(\rho)}_p := \bigcup_{q \in HO(p, 2\rho)} HO(q, 2\rho - 1).$$

Algorithm 1 Translation for increasing kernels

```
Initialization:
    $NewHO_p \in 2^V$, initially empty
Round $r$:
    $S^r_p$ :
        if $r = 2\rho$ then
            send $\langle\, HO(p, r-1) \,\rangle$ to all processes
$T^r_p$ :
    if $r = 2\rho$ then
        $NewHO_p := \bigcup_{q \in HO(p,r)} HO(q, r-1)$
```

In this way, we emulate a macro-round ρ whose kernel satisfies the following key property:

Proposition 4.1 *If all heard-of sets at rounds $2\rho - 1$ and 2ρ contain at least $n - f_1$ and $n - f_2$ processes respectively, then*

$$|\widetilde{K}^{(\rho)}| \geq n - f_1 \left(1 + \frac{f_2}{n - f_2}\right),$$

where $\widetilde{K}^{(\rho)} = NewHO_p^{(\rho)}$.

Proof: Consider the directed graph G_ρ whose vertices are the processes in Π, and there is an edge from p to q if and only if p belongs to $HO(q, 2\rho - 1)$. For any vertex x in G_ρ, let $nbIn(x)$ and $nbOut(x)$ be the numbers of in-neighbors and out-neighbors of x, respectively. The number of edges in G_ρ is equal to

$$E(G_\rho) = \sum_{x \in \Pi} nbIn(x) = \sum_{y \in \Pi} nbOut(y),$$

and since $nbIn(x) = |HO(x, 2\rho - 1)|$, we have

$$E(G_\rho) \geq n(n - f_1). \tag{1}$$

Let us separate the summation $\sum_{y \in \Pi} nbOut(y)$ into those y's in $\widetilde{K}^{(\rho)}$ and those not in $\widetilde{K}^{(\rho)}$, and let $\widetilde{k}_\rho$ denote the cardinality of $\widetilde{K}^{(\rho)}$. Clearly, we have

$$\sum_{y \in \widetilde{K}^{(\rho)}} nbOut(y) \leq n\widetilde{k}_\rho. \tag{2}$$

For the other term in the sum, we show that for any y that is not in $\widetilde{K}^{(\rho)}$, we have

$$nbOut(y) \leq f_2. \tag{3}$$

This is true because if y is not in $\widetilde{K}^{(\rho)}$, then there exists some process p such that

$$y \notin \bigcup_{q \in HO(p, 2\rho)} HO(q, 2\rho - 1),$$

that is for any q in $HO(p, 2\rho)$, y is not in $HO(q, 2\rho - 1)$. In other words, none of the out-neighbors of y in G_ρ belongs to $HO(p, 2\rho)$. Since any heard-of set $HO(p, 2\rho)$ has at least $n - f_2$ elements, $nbOut(y)$ is at most f_2. From (1), (2) and (3) it follows that

$$n(n - f_1) \leq n\widetilde{k}_\rho + (n - \widetilde{k}_\rho)f_2,$$

and so

$$\widetilde{k}_\rho \geq n - f_1 \left(1 + \frac{f_2}{n - f_2}\right).$$

□

With the communication predicate $\mathcal{P}_{HO}^{maj} = \mathcal{P}_{HO}^{[\frac{n-1}{2}]}$, Proposition 4.1 can be specialized as follows:

Corollary 4.2 *There is a two round translation of $\mathcal{P}_{HO}^{maj}$ into $\mathcal{P}_{nekrounds}$, and so*

$$\mathcal{P}_{HO}^{maj} \succeq \mathcal{P}_{nekrounds}.$$

Proof: Take $f_1 = f_2 = \lceil \frac{n-1}{2} \rceil$, which leads to $\widetilde{k}_\rho \geq 1$. □

Another interesting corollary of Proposition 4.1 is obtained with $f_2 = 1$: in this case, Proposition 4.1 gives

$$\widetilde{k}_\rho \geq n - f_1 - \frac{f_1}{n-1}.$$

Therefore, if $f_1 \leq n-1$, then we have $\widetilde{k}_\rho \geq n - f_1$. In particular, in a system with at least 3 processes and heard-of sets of cardinality $n-1$ ($f_1 = f_2 = 1$), we can emulate macro-rounds with kernels of size at least $n-1$, and so the global kernel of f macro-rounds has a membership of over $n-f$ processes. Considering that the communication predicates defined by:

$$\forall r > 0,\, \forall p \in \Pi \,:\, |HO(p,r)| \geq n-1$$

and

$$|K| \geq n - f$$

are the HO counterparts of asynchronous systems with at most one crash failure and synchronous systems with at most f send omission failures, respectively, we derive the following result relating synchronous and asynchronous systems:

Corollary 4.3 *Asynchronous message-passing systems with at most one crash failure can implement the first f rounds of a synchronous system with at most f send omission failures.*

A similar result is shown by Gafni [15] for asynchronous atomic-snapshot shared memory systems with at most one crash failure. Note that the very elegant reduction of the omission failure lower bound to the asynchronous impossibility result [14] that Gafni derives from his result can also be span off from Corollary 4.3.

4.2 Translating no split rounds into nek rounds

We now show that $\mathcal{P}_{nosplit}$ and $\mathcal{P}_{nekrounds}$ are actually equivalent. Clearly, $\mathcal{P}_{nekrounds}$ implies $\mathcal{P}_{nosplit}$, and so we have $\mathcal{P}_{nekrounds} \succeq \mathcal{P}_{nosplit}$. To prove that $\mathcal{P}_{nosplit} \succeq \mathcal{P}_{nekrounds}$, we present a $\lambda(n)$ round translation, where $\lambda(n)$ is the integer defined by

$$2^{\lambda(n)-1} < n \leq 2^{\lambda(n)},$$

which emulates nek macro-rounds from no split rounds. This translation, which appears in Algorithm 2, is an extension from 2 to $\lambda(n)$ of Algorithm 1.

Each macro-round consists of $\lambda(n)$ consecutive rounds. We fix such a macro-round ρ, and we denote $r_1, \cdots, r_{\lambda(n)}$ the sequence of rounds that form ρ [4] Each process p maintains a variable $Listen_p$, which is contained in Π and is equal to $HO(p, r_1)$ at the end of round r_1. At the following rounds, p sends the current value of $Listen_p$ to all and then computes the new $Listen_p$ as the union of the $Listen_q$'s it has just received. That is, at each round r, $r_2 \leq r \leq r_{\lambda(n)}$, p sets

$$Listen_p := \bigcup_{q \in HO(p,r)} Listen_q.$$

Theorem 4.4 *Algorithm 2 is a $\lambda(n)$ round translation of $\mathcal{P}_{nosplit}$ into $\mathcal{P}_{nekrounds}$, and so we have $\mathcal{P}_{nosplit} \simeq \mathcal{P}_{nekrounds}$.*

[4] Precisely, we have $r_1 = \lambda(n)(\rho - 1) + 1, \ldots, r_{\lambda(n)} = \lambda(n)\rho$.

Algorithm 2 Translating no split rounds into nek rounds

```
 1: Initialization:
 2:   $Listen_p \in 2^V$, initially empty
 3:   $NewHO_p \in 2^V$, initially empty
 4: Round $r$:
 5:   $S_p^r$ :
 6:     send $\langle\, Listen_p \,\rangle$ to all processes
 7:   $T_p^r$ :
 8:     if $r \equiv 1 \pmod{\lambda(n)}$ then
 9:       $Listen_p := HO(p, r)$
10:     else
11:       $Listen_p := \bigcup_{q \in HO(p,r)} Listen_q$
12:     if $r \equiv 0 \pmod{\lambda(n)}$ then
13:       $NewHO_p := Listen_p$
```

Proof: Condition E1 trivially follows from the code of Algorithm 2 (lines 9 and 11). We now prove E2. For that, consider the directed graphs G_i induced by the heard-of sets at round r_i. Let G_i^* denote the directed graph whose vertices are the processes in Π, and there is an edge from p to q iff there exists a chain of $i+1$ processes $x_1, \cdots, x_{i+1}$ from $x_1 = p$ to $x_{i+1} = q$ such that

$$x_2 \in HO(x_1, r_{\lambda(n)}),\ x_3 \in HO(x_2, r_{\lambda(n)-1}), \ldots, \text{ and } x_{i+1} \in HO(x_i, r_{\lambda(n)-i+1}).$$

Clearly, $G_1^* = G_{\lambda(n)}$, and $Listen_p$ at the end of round $r_{\lambda(n)}$ is the set of p's in-neighbours in $G_{\lambda(n)}^*$:

$$Listen_p^{(r_{\lambda(n)})} = \{q \in \Pi \ :\ (q, p) \text{ is an edge of } G_{\lambda(n)}^*\}.$$

Hence Condition E2 directly follows from the following lemma as the special case $i = \lambda(n)$ since $n \leq 2^{\lambda(n)}$.

Lemma 4.5 *For any index $i \in \{1, \ldots, \lambda(n)\}$, there is at least one common in-neighbour to any subset of 2^i processes in the graph G_i^*.*

Proof: By induction on i.
Basis: $i = 1$. We have $G_1^* = G_{n-1}$, and the lemma coincides with the no split predicate.
Inductive step: Suppose $i \geq 2$ and the lemma holds in G_{i-1}^*. Let $\{p_1, \ldots, p_{2^i}\}$ be any subset of 2^i processes. By inductive hypothesis, $p_1, \ldots, p_{2^{i-1}}$ have a common in-neighbour x_1 in G_{i-1}^*, and $p_{2^{i-1}+1}, \ldots, p_{2^i}$ have a common in-neighbour x_2 in G_{i-1}^*. Since the no split predicate holds at each round, x_1 and x_2 have a common in-neighbour in $G_{\lambda(n)-i+1}$, no matter $x_1 = x_2$ or not; let x denote this node. By definition of G_i^*, x is a common in-neighbour to $p_1, \ldots, p_{i+1}$ in this graph. $\Box_{\mathit{Lemma}\ 4.5}$

By definition of the $NewHO_p$'s, the translation preserves kernels, which completes the proof that Algorithm 2 translates $\mathcal{P}_{nosplit}$ into $\mathcal{P}_{nekrounds}$. $\Box$

Note that since $\mathcal{P}_{HO}^{maj}$ implies $\mathcal{P}_{nosplit}$, Algorithm 2 is a $\lambda(n)$ round translation of $\mathcal{P}_{HO}^{maj}$ into $\mathcal{P}_{nekrounds}$. Thus we get another proof of Corollary 4.2, but the translation requires $\lambda(n)$ rounds instead of two rounds in Algorithm 1.

4.3 A translation for achieving space uniformity

Our third translation achieves space uniformity under the condition of original non-empty global kernels. More precisely, we give a $f+1$ round translation of $\mathcal{P}_K^f$ into $\mathcal{P}_{sp\_unif} \wedge \mathcal{P}_K^f$.

Processes propagate and collect the heard-of sets that they have ever seen during $f+1$ consecutive rounds. At the end of the macro-round, the new p's heard-of set is the intersection of the sets of process names that p has just collected at the last round. Formally, each macro-round consists of $f+1$ consecutive rounds. Each process p maintains three variables $Listen_p$, $Known_p$, and $NewHO_p$, which are all contained in Π and are equal to Π, $\{p\}$, and $\emptyset$ at the beginning of each macro-round, respectively. At each round, p listens to process q only if it hears of q at the previous rounds of the macro-round, and so p sets:

$$Listen_p := Listen_p \cap HO(p,r).$$

Moreover, during the f first rounds of any macro-round, each process p collects the names of all the processes it hears of in its variable $Known_p$; for that, it sends $Known_p$ to all processes and then sets:

$$Known_p := Known_p \cup \left(\bigcup_{q \in Listen_p} Known_q \right).$$

At the last round of any macro-round, p computes the intersection (instead of the union as in the previous rounds of the macro-round) of the sets $Known_q$ it has just collected. The code of the translation is given below (see Algorithm 3).

Algorithm 3 Translation for space uniformity

```
1:  Initialization:
2:     $Listen_p \in 2^V$, initially $\Pi$
3:     $NewHO_p \in 2^V$, initially $\Pi$
4:     $Known_p \in 2^V$, initially $\{p\}$

5:  Round $r$:
6:     $S_p^r$ :
7:        send $\langle Known_p \rangle$ to all processes
8:  $T_p^r$ :
9:     $Listen_p := Listen_p \cap HO(p,r)$
10:    if $r \not\equiv 0 \pmod{f+1}$ then
11:       $Known_p := Known_p \cup \bigcup_{q \in Listen_p} Known_q$
12:    else
13:       $NewHO_p := \bigcap_{q \in Listen_p} Known_q$
14:       $Listen_p := \Pi$
15:       $Known_p := \{p\}$
```

We fix a macro-round ρ and introduce some piece of notation relative to ρ. Let $r_1, \cdots, r_{f+1}$ denote the sequence of the $f+1$ rounds that form ρ. Recall that $K(\rho)$ denotes the kernel of macro-round ρ, i.e.,

$$K(\rho) = \bigcap_{r=r_1}^{r_{f+1}} K(r).$$

We say that process p *knows process s at round r* if $s \in Known_p^{(r)}$. If $s \in Known_p^{(r)} \setminus Known_p^{(r-1)}$, $q \in Listen_p^r$, and $s \in Known_q^{(r-1)}$, then we say that *p hears of s from q at round r*. Finally, process s is said to be *good* (at macro-round ρ) if s is known by all processes at round r_f; otherwise s is *bad*. In other words, the set of good processes is defined by

$$Good = \bigcap_{p \in \Pi} Known_p^{(r_f)}.$$

Thus at line (13), every process computes a local approximation of the set of good processes.

We are going to prove that if $K(\rho)$ contains at least $n-f$ processes, then at the end of any macro-round, all the $NewHO$'s are equal and contain $K(\rho)$. For that, we start with some preliminary assertions.

Lemma 4.6

$$K(\rho) \subseteq \bigcap_{p\in\Pi, r\in\{r_1,\cdots,r_{f+1}\}} Listen_p^{(r)}.$$

Proof: It is immediate from the definition of $Listen_p$ that for any process p,

$$\cap_{r=1}^{r_f} Listen_p^{(r)} = \cap_{r=1}^{r_f} HO(p,r) \text{ and } HO(p, r_{f+1}) \subseteq Listen_p^{r_{f+1}}.$$

The result follows directly. $\square_{Lemma\ 4.7}$

Lemma 4.7 *Any process p in $K(\rho)$ is a good process.*

Proof: Let p be any process in $K(\rho)$. By lines (15) and (11), it follows that all processes know p at the end of round r_f. This shows that p is a good process. $\square_{Lemma\ 4.8}$

Lemma 4.8 *If process p hears of some process s at round r_k, then there exist $k-1$ processes $p_1, \cdots, p_{k-1}$, each different from p and s, such that p_1 hears of s from s at round r_1, p_2 hears of s from p_1 at round r_2, ..., p_{k-1} hears of s from p_{k-2} at round r_{k-1}, and p hears of s from p_{k-1} at round r_k. Moreover, processes $p_1, \cdots, p_{k-2}$, and s are all in the cokernel $coK(\rho)$.*

Proof: Since p hears of process s at round r_k, there exists some process p_{k-1} such that $p_{k-1} \in Listen_p^{(r_k)}$ and $s \in Known_{p_{k-1}}^{(r_{k-1})}$. Since $Listen_p$ is non-increasing, $p_{k-1} \in Listen_p^{(r_{k-1})}$. This implies that p_{k-1} hears of s at round r_{k-1} since p does not know s at this round. In turn, there exists some process p_{k-2} such that $p_{k-2} \in Listen_{p_{k-1}}^{(r_{k-1})}$, and $s \in Known_{p_{k-2}}^{(k-2)}$. From

$$s \in Known_{p_{k-2}}^{(r_{k-2})} \quad \text{and} \quad s \notin Known_p^{(r_{k-1})},$$

we deduce that $p_{k-2} \notin Listen_p^{(r_{k-1})}$. By Lemma 4.6, we have $p_{k-2} \in coK(\rho)$.

Step by step, we exhibit $k-1$ processes $p_1, \cdots, p_{k-1}$ such that for any index i, $1 \leq i \leq k-1$,

$$s \notin Known_{p_i}^{(r_{i-1})},\ s \in Known_{p_i}^{(r_i)}\ ,\ \text{and } p_{i-1} \in Listen_{p_i}^{(r_i)}.$$

For any index i such that $2 \leq i \leq k-2$, we have both

$$s \in Known_{p_{i-1}}^{(r_{i-1})} \text{ and } s \notin Known_{p_{i+1}}^{(r_i)}.$$

Therefore, $p_{i-1} \notin Listen_{p_{i+1}}^{(r_i)}$. By Lemma 4.6, we deduce that p_{i-1} belongs to $coK(\rho)$. Similarly, we have

$$s \notin Known_{p_2}^{(r_1)},$$

and so s belongs to $coK(\rho)$, too. From $p_{k-2} \in coK(\rho)$, it follows that all the processes $p_1, \cdots, p_{k-2}$, and s are in $coK(\rho)$. $\square_{Lemma\ 4.9}$

Lemma 4.9 *If process p knows some bad process s at the end of round r_{f+1}, then p has heard of s by the end of the round r_f, i.e,*

$$s \in Known_p^{(r_{f+1})}\ \wedge\ s \notin Good \Rightarrow s \in Known_p^{(r_f)}.$$

Proof: Let s be a bad process; so there exists some process q such that $s \notin Known_q^{(r_f)}$. Suppose for contradiction that p hears of s at round r_{f+1}. By Lemma 4.8, there are f processes $p_1, \cdots, p_f$ each different from both p and s such that p hears of s from p_f at round r_{f+1}, and processes $p_1, \cdots, p_{f-1}$, and s are all in $coK(\rho)$. Since $s \notin Known_q^{(r_f)}$, $Listen_q$ contains neither p nor p_f at this round. Therefore, p and p_f are also in $coK(\rho)$, which contradicts the fact that $coK(\rho)$ is of size at most f. $\Box_{Lemma\ 4.9}$

Lemma 4.10 *A process is good iff it is known by some process in the kernel, i.e,*

$$s \in Good \Leftrightarrow \exists p \in K(\rho) \ : \ s \in Known_p^{(r_f)}.$$

Proof: By definition, a good process is known by all processes at round r_f.

Conversely, let s be any process known by some process p in $K(\rho)$ at round r_f. Assume, for the sake of contradiction, that s is bad. Since p is in $K(\rho)$, every process q receives a message from p at round r_{f+1}, and so $Known_q$ contains s at the end of round r_{f+1}. By Lemma 4.9, we deduce that every process already knows s at round r_f. This contradicts that s is a bad process. $\Box_{Lemma\ 4.10}$

Lemma 4.11 *For any process p, at the end of round r_{f+1}, $NewHO_p$ is composed of all the good processes, i.e., $NewHO_p^{(r_{f+1})} = Good$.*

Proof: Obviously, we have $Good \subseteq NewHO_p^{(r_{f+1})}$.

Conversely, let s be any process in $NewHO_p^{(r_{f+1})}$; s is known at round r_f by all the processes in $Listen_p^{(r_{f+1})}$, and in particular by those in $K(\rho)$. By Lemma 4.10, it follows that s is a good process since $K(\rho)$ is non-empty. $\Box_{Lemma\ 4.11}$

Lemma 4.11 says that all the $NewHO_p$'s are equal after $f+1$ rounds, and so the collection of the $NewHO$'s satisfies $\mathcal{P}_{sp\_unif}$ at the end of each macro-round. Moreover, Lemma 4.7 implies that the translation preserves kernels. Since E1 is clearly guaranteed, we have proved the following theorem:

Theorem 4.12 *Algorithm 3 is a $f+1$ translation of $\mathcal{P}_K^f$ into $\mathcal{P}_K^f \wedge \mathcal{P}_{sp\_unif}$, and so we have*

$$\mathcal{P}_K^f \simeq \mathcal{P}_K^f \wedge \mathcal{P}_{sp\_unif}.$$

Combining the latter theorem with Proposition 3.2, we derive a $f+1$ round algorithm A such that the HO machine $(A, \mathcal{P}_K^f)$ solves Consensus. Thus we check that at least for nek machines, the strong termination requirement, namely "every process eventually decides", does not make the Consensus specification harder to solve. Note that the first f rounds of A are identical to those of the *FloodSet* algorithm [18] which is a well-known Consensus algorithm devised for synchronous systems with at most f crash failures. These two algorithms only differ in round $f+1$: Algorithm 3 computes the intersection of the $Known_p$'s instead of union in *FloodSet*. Hence, substituting intersection for union just at the last round is sufficient to guarantee a general agreement among *all* processes under the only communication predicate $\mathcal{P}_K^f$ (without regularity).

Interestingly, if we substitute

$$NewHO_p := NewHO_p \cap \left(\cap_{q \in Listen_p} Known_q\right)$$

for

$$NewHO_p := \cap_{q \in Listen_p} Known_q$$

at line (13) in Algorithm 3, the resulting algorithm translates $\mathcal{P}_K^f$ into $\mathcal{P}_K^f \wedge \mathcal{P}_{sp\_unif} \wedge \mathcal{P}_{reg}$. Considering that the communication predicates $\mathcal{P}_K^f$ and $\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$ are the HO counterparts of synchronous systems with at most f send omission failures and with at most f crash failures respectively, we get an automatic procedure which both guarantees space uniformity and masks send omissions into crash failures.

4.4 A two round translation for increasing time uniformity

We now describe a two round translation that increases time uniformity in the sense that it emulates regular runs. This translation can be viewed as a refinement of our first translation (Algorithm 1), with in addition a mechanism for the transition from a macro-round to the next one which guarantees regularity. The basic idea of this mechanism is at each macro-round and for each process to compute an approximation of the kernel of the previous macro-round.

More precisely, each process p maintains a variable $ApproxK_p$ whose initial value is Π. At the end of round 2ρ, p sets

$$ApproxK_p := \bigcap_{q \in HO(p,2\rho)} \Big(HO(q, 2\rho - 1) \cap ApproxK_q^{(\rho-1)} \Big) ,$$

where $ApproxK_q^{(\rho-1)}$ denotes the value of $ApproxK_q$ at the end of the macro-round $\rho - 1$. The heard-of set at macro-round ρ for process p is now defined by:

$$NewHO_p := \bigcup_{q \in HO(p,2\rho)} \Big(HO(q, 2\rho - 1) \cap ApproxK_q^{(\rho-1)} \Big) .$$

The resulting algorithm is called Algorithm 4.

Algorithm 4 Translating no split rounds into regular rounds

```
1: Initialization:
2:     $ApproxK_p \in 2^V$, initially equal to $\Pi$
3:     $NewHO_p \in 2^V$, initially empty

4: Round $r = 2\rho$:
5:     $S_p^r$ :
6:         send $\langle\, ApproxK_p \cap HO(p, r-1) \,\rangle$ to all processes

7: $T_p^r$ :
8:     $ApproxK_p := ApproxK_p \cap \ \bigcap_{q \in HO(p,r)} ApproxK_q \cap HO(q, r-1)$
9:     $NewHO_p := \bigcup_{q \in HO(p,r)} ApproxK_q \cap HO(q, r-1)$
```

Theorem 4.13 *Algorithm 4 translates $\mathcal{P}_{nosplit}$ into $\mathcal{P}_{reg}$, and so we have*

$$\mathcal{P}_{nosplit} \succeq \mathcal{P}_{reg}.$$

Moreover, it preserves global kernels.

Proof: Condition E1 immediately follows from the code of Algorithm 4, line 9. For the same reason, the algorithm preserves kernels. We now prove E2. According to the code of the algorithm, if $x \notin NewHO_q^{(\rho)}$, then for any s in $HO(q, 2\rho)$, we have

$$x \notin HO(s, 2\rho - 1) \cap ApproxK_s^{(\rho-1)}.$$

Because of the no split predicate, for any process y, $HO(y, 2\rho)$ intersects $HO(q, 2\rho)$, and so there exists $s \in HO(y, 2\rho)$ such that $x \notin HO(s, 2\rho - 1) \cap ApproxK_s^{(\rho-1)}$. Hence $x \notin ApproxK_y^{(\rho)}$. It follows that for any process p, $x \notin NewHO_p^{(\rho+1)}$. In other words, we have showed that

$$\forall p \in \Pi \, : \, NewHO_p^{(\rho+1)} \subseteq \bigcap_{q \in \Pi} NewHO_q^{(\rho)},$$

i.e., regularity holds.

Moreover, by definition of $ApproxK_p$, we easily get

$$\forall p \in \Pi \;\; : \;\; \bigcap_{r=1}^{\rho} K(2r-1) \subseteq ApproxK_p^{(\rho)}.$$

It follows that Algorithm 4 preserves global kernels. □

From the latter point, we derive the following corollary:

Corollary 4.14

$$\forall f \in \{1, \cdots, n-1\} \, : \, \mathcal{P}_K^f \simeq \mathcal{P}_K^f \wedge \mathcal{P}_{reg}.$$

Since the communication predicates $\mathcal{P}_K^f$ and $\mathcal{P}_K^f \wedge \mathcal{P}_{reg}$ are the HO counterparts of synchronous systems with at most f send omission failures, and synchronous systems with at most f crash failures, respectively (cf [15]), Algorithm 4 provides a general method to convert synchronous algorithms tolerant of f crash failures into ones tolerant of f omission senders.

Algorithm 4 is almost similar to the two round translation given by Neiger and Toueg [21] to mask send omission into crash failures. The only difference between both lies in the processes that are in charge of stopping information transmission: In Neiger and Toueg's algorithm, upon learning it is faulty at some point, process p self censors for the rest of the computation whereas in Algorithm 4, p sends the same messages but the other processes that detect p is faulty do not hear of p anymore.

Algorithms 1 and 4 can be combined into a three round translation that increases both space and time uniformity. In this new translation, each process p inductively computes $NewHO_p$ and $ApproxK_p$ as follows:

$$NewHO_p^{(\rho)} := \bigcup_{q \in HO(p, 3\rho)} \left(\cup_{s \in HO(q, 3\rho-1)} HO(s, 3\rho - 2) \cap ApproxK_s^{(\rho-1)} \right),$$

and

$$ApproxK_p := \bigcap_{q \in HO(p, 3\rho)} \left(\cup_{s \in HO(q, 3\rho-1)} HO(s, 3\rho - 2) \cap ApproxK_s^{(\rho-1)} \right).$$

In this way, the first f rounds of a synchronous system with at most f crash failures can be implemented from an asynchronous message-passing system with at most one crash failure. This three round simulation is identical to the one given by Gafni [15] from asynchronous atomic-snapshot shared memory systems: the last two rounds of the simulation actually correspond to the two round *adopt-commit* protocol in [15].

4.5 Comparing communication predicates

In the previous sections, we saw several interrelationships between some basic communication predicates. These relations and those that are directly derived from the implication

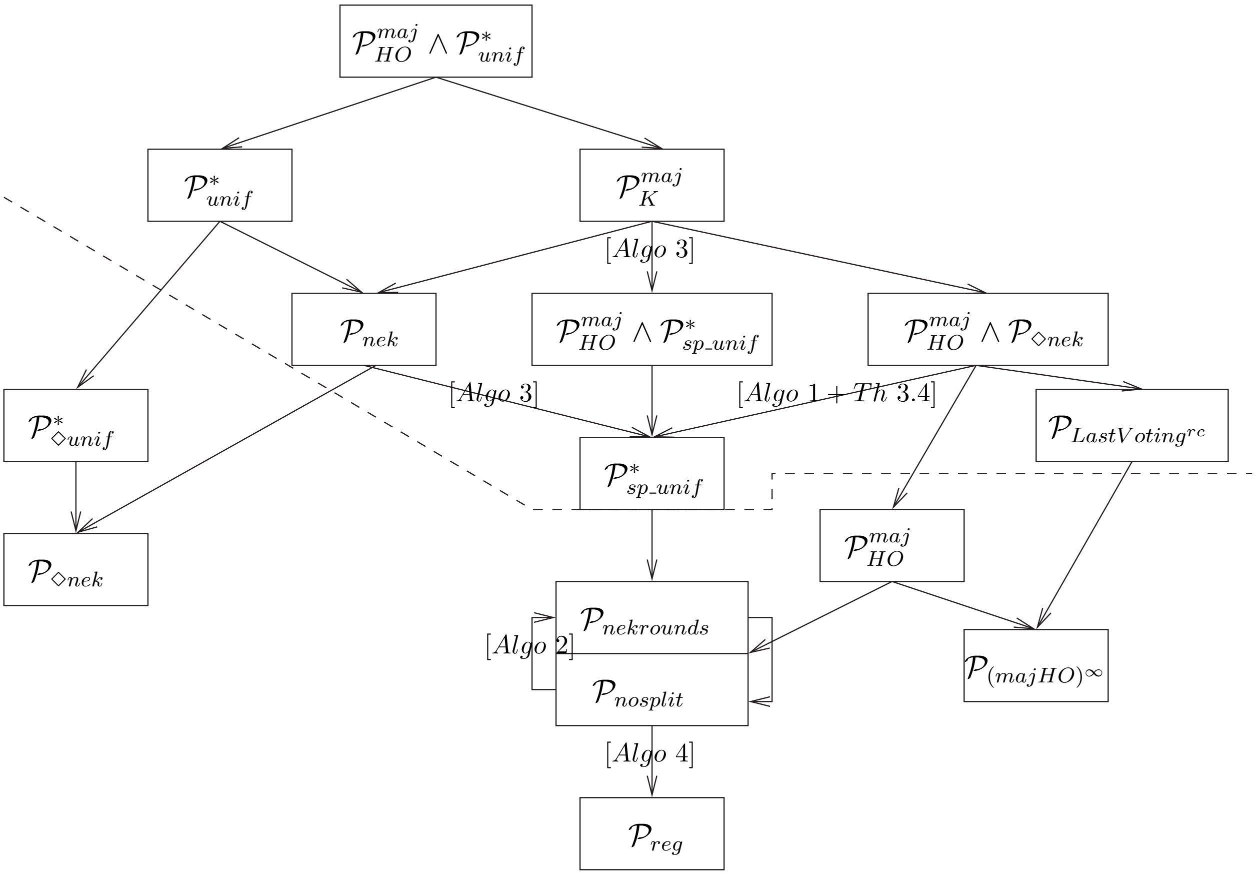
$\mathcal{P}_{HO}^{maj} \wedge \mathcal{P}_{unif}^{*}$
$\mathcal{P}_{unif}^{*}$
$\mathcal{P}_{K}^{maj}$
[Algo 3]
$\mathcal{P}_{nek}$
$\mathcal{P}_{HO}^{maj} \wedge \mathcal{P}_{sp\_unif}^{*}$
$\mathcal{P}_{HO}^{maj} \wedge \mathcal{P}_{\diamond nek}$
$\mathcal{P}_{\diamond unif}^{*}$
[Algo 3]
[Algo 1 + Th 3.4]
$\mathcal{P}_{LastVoting^{rc}}$
$\mathcal{P}_{sp\_unif}^{*}$
$\mathcal{P}_{\diamond nek}$
$\mathcal{P}_{HO}^{maj}$
$\mathcal{P}_{nekrounds}$
[Algo 2]
$\mathcal{P}_{nosplit}$
$\mathcal{P}_{(majHO)^\infty}$
[Algo 4]
$\mathcal{P}_{reg}$

Figure 1: Relationships among some basic communication predicates (we denote $\mathcal{P}_K^{[\frac{n-1}{2}]}$ by $\mathcal{P}_K^{maj}$ and we define $\mathcal{P}_{LastVoting^{rc}}$ in Section 5.5)

relations are illustrated in Figure 1 as follows: there is a direct edge from $\mathcal{P}$ to $\mathcal{P}'$ if $\mathcal{P} \succeq \mathcal{P}'$. We adopt the following notation: $\mathcal{P}_K^{maj} = \mathcal{P}_K^{[\frac{n-1}{2}]}$, and $\mathcal{P}_{(majHO)^\infty}$ denotes the predicate

$$\mathcal{P}_{(majHO)^\infty} \;::\; \forall r > 0, \exists r_0 \geq r, \forall p \in \Pi \;:\; |HO(p, r_0)| > n/2.$$

The dashed line represents the "Consensus line": a communication predicate $\mathcal{P}$ is above the line iff Consensus solvable under $\mathcal{P}$. The figure is completed with the predicate $\mathcal{P}_{LastVoting^{rc}}$ defined in Section 5.5.

At this point, it is natural to wonder whether the relations are strict, or whether some pair of predicates are actually equivalent. We do not address this issue here, but refer to paper [2] that examines other agreement problems in the HO model.

5 Consensus and general HO machines

We now examine general HO machines, some with empty kernel rounds, that solve Consensus. To do so, we first revisit various classical Consensus algorithms devised for asynchronous or partially synchronous systems. For coordinator-based algorithms, we introduce a generalization of HO machines, the *Coordinated HO machines* (or *CHO machines* for short).

HO and CHO machine formalisms enable us to express well-known Consensus algorithms in a quite concise and elegant way, and so to extract the algorithmic schemes on which they are based. This not only gives some new insights into these Consensus algorithms, but also allows us to design new ones that are quite interesting in practice since they are correct under very realistic conditions. Moreover, it is striking to see how easy it is to determine simple conditions that ensure the correctness of these algorithms from their HO or CHO counterparts.

5.1 A Consensus algorithm *à la* Ben-Or: the *UniformVoting* algorithm

First, we present a Consensus algorithm that to the best of our knowledge, has not yet been described in the litterature. It can be viewed as a deterministic version of the Ben-Or algorithm [1, 22]. We call it the *UniformVoting* algorithm, see Algorithm 5.

As for all the other algorithms described in Section 5, *UniformVoting* is organized into *phases*.[5] A *UniformVoting* phase consists of two rounds. Every process p maintains a variable x_p containing a value in V, initially equal to p's initial value. Process p broadcasts x_p at the first round of each phase, and then adopts the smallest value it has just received. Then, p votes for value v if it has not heard that some process has started the phase with another value; otherwise, p does not cast a vote. At the second round, p sends v or "?" to all, accordingly. In the same message, it sends again the current value of x_p. If each message that p receives at the second round contains a vote for v, then p decides v (this is why we call this algorithm *UniformVoting*). If p receives some values v different from "?", then it chooses one such value arbitrarily and adopts it for the next phase; otherwise, p adopts the smallest value of the x_q's it has just received.

Algorithm 5 The *UniformVoting* algorithm

```
Initialization:
  $x_p := v_p$      {$v_p$ is the initial value of $p$}
  $vote_p \in V \cup \{?\}$, initially ?

Round $r = 2\phi - 1$:
  $S_p^r$ :
    send $\langle x_p \rangle$ to all processes

  $T_p^r$ :
    $x_p :=$ smallest $v$ received
    if all the values received are equal to $v$ then
      $vote_p := v$

Round $r = 2\phi$:
  $S_p^r$ :
    send $\langle x_p , vote_p \rangle$ to all processes

  $T_p^r$ :
    if at least one $\langle * , v \rangle$ with $v \neq ?$ is received then
      $x_p := v$
    else
      $x_p :=$ smallest $w$ from $\langle w , ? \rangle$ received
    if all the messages received are
                        equal to $\langle * , v \rangle$ with $v \neq ?$ then
      DECIDE($v$)
    $vote_p := ?$
```

We now argue that if no round is split, then no process can make a bad decision (agreement). Then we prove that termination is enforced by just one uniform round, which is guaranteed by the communication predicate:

$$\mathcal{P}_{(unif)^\infty} \;::\; \forall r > 0, \exists r_0 \geq r,\ \forall p, q \in \Pi^2 \,:\, HO(p, r_0) = HO(q, r_0).$$

Theorem 5.1 *The HO machine consisting of the UniformVoting algorithm and the predicate $\mathcal{P}_{nosplit} \wedge \mathcal{P}_{(unif)^\infty}$ solves Consensus.*

Proof: Integrity is trivially satisfied.

[5] A phase consists of a fixed number of consecutive rounds. Basically, there is no difference between a phase and a macro-round. We have preferred the term "macro-round" for translations because it seems us more suggestive in this context, but here we use the classical terminology of "phase".

The proof of the agreement condition relies on the fact that if two processes p and q vote for v and v' at the same phase, then predicate $\mathcal{P}_{nosplit}$ ensures that $v = v'$. Moreover, predicate $\mathcal{P}_{nosplit}$ also guarantees that if some process decides v at round $r = 2\phi$, then all the x_p's remain equal to v from round r.

For termination, let r_0 be the first uniform round. There are two cases to consider.

1. Round r_0 is the first round of some phase ϕ_0, i.e., $r_0 = 2\phi_0 - 1$. Therefore at round r_0, either all processes vote for the same value v or no process votes.
2. Round r_0 is the second round of some phase ϕ_0, i.e., $r_0 = 2\phi_0$.

In both cases, all the x_p's are equal at the end of round $2\phi_0$, and every process has decided at the end of round $2\phi_0 + 2$. It follows that $\mathcal{P}_{(unif)^\infty}$, which is invariant by time translation and guarantees one uniform round, enforces termination of the *UniformVoting* algorithm. □

5.2 Coordinated HO machines

Numerous algorithms for Consensus are coordinator-based algorithms (eg. the Consensus algorithms proposed by Dwork, Lynch, and Stockmeyer [12], Chandra and Toueg's algorithm [6], Paxos [16]). The correctness of these algorithms is guaranteed by some properties on coordinators: for example, termination in Paxos requires that during some phase, all processes hear of the coordinator of the phase. For such algorithms, we introduce the *Coordinated HO machine* (or *CHO machine* for short) for which algorithms refer to the notion of coordinators, and predicates deal not only with heard-of sets, but also with coordinators.

A CHO machine is a pair $M^c = (A, P)$ much like the ordinary HO machine. Reflecting the fact that the messages sent by a process p in a round of a CHO machine do not uniquely depend on the current state, but also on the identity of a *coordinator*, the message-sending function S_p^r is no longer a function from $states_p \times \Pi$ to $M \cup \{null\}$ but instead a function

$$S_p^r \;:\; \Pi \times states_p \times \Pi \;\longrightarrow\; M \cup \{null\}.$$

Similarly, the state of process p at the end of a round does not only only depend on its current state and the collection of the messages it has just received, but also on the identity of its coordinator. So, the transition function T_p^r is a function

$$T_p^r \;:\; states_p \times \left((M \cup \{null\})^\Pi\right)^* \times \Pi \longrightarrow\; states_p$$

where $\left((M \cup \{null\})^\Pi\right)^*$ denotes the set of partial vectors, indexed by Π, of elements of $M \cup \{null\}$. The functions $(S_p^r)_{r>0}$ and $(T_p^r)_{r>0}$ define the *coordinated process* p, and the collection of coordinated processes is called a *coordinated algorithm.*

As for HO machines, at every round r, each process p (1) applies the message-sending function S_p^r to the current coordinator and the current state to generate the messages to be sent, and (2) applies the state-transition T_p^r to the current state and the incoming messages. The combination of the two steps is called a *coordinated round*, and p's coordinator at r is denoted $Coord(p, r)$. Process p *sets out to be the coordinator* of round r if $p = Coord(p, r)$.

We say that r is a *uniformly coordinated round* if

$$\forall p, q \in \Pi \;:\; Coord(p, r) = Coord(q, r)$$

and r is *well coordinated* if

$$\forall p \in \Pi \;:\; Coord(p, r) \in HO(p, r).$$

A computation of a CHO machine is *uniformly coordinated from round* r_0 if any round r such that $r \geq r_0$ is uniformly coordinated; a computation is *uniformly coordinated* if it is uniformly coordinated from the first round.

Usually, when algorithms are decomposed into phases, every process keeps the same coordinator during each whole phase. The coordinator of process p during phase ϕ is denoted by $Coord(p, \phi)$.

Each run of a CHO machine does not uniquely determine the heard-of set collection, but also the *coordinator collection* according to space and time, namely $(Coord(p,r))_{p \in \Pi, r>0}$. A *CHO machine for* Π consists of a coordinated algorithm A and a predicate over both heard-of sets and coordinator collections, called a *communication-coordinator predicate*, which is invariant by time translation. For example, we shall consider CHO machines with the predicate:

$$\forall r > 0, \exists r_0 \geq r, \forall p, q \in \Pi^2 \ : \ Coord(p, r_0) = Coord(q, r_0) \wedge Coord(p, r_0) \in HO(p, r_0)$$

or equivalently,

$$\forall r > 0, \exists r_0 \geq r, \forall p, q \in \Pi^2 \ : \ Coord(p, r_0) = Coord(q, r_0) \wedge Coord(p, r_0) \in K(r_0).$$

As we shall see in the next sections, uniformly and well coordinated rounds play a key role for guaranteeing correctness of coordinated Consensus algorithms.

Finally, the notion of what it means for a CHO machine to solve a problem is similar to the one for an HO machine.

With CHO machine formalism, the way processes determine the name of their coordinators is not determined: it may be the result of some computation (in other words, the CHO machine is emulated by an ordinary HO machine), or processes may use some external devices (physical devices or oracles) that are capable of reporting the name of coordinators to every processes. Most of the CHO machines that we shall consider can be simulated by ordinary HO machines, and so this generalization does not seem to lead to a more powerful computational model (as explained above, the basic motivation for introducing CHO machines is just to devise Consensus algorithms and to state conditions for their correctness in a more natural and elegant way). In particular, we can adopt an “off-line” strategy, usually called the *rotating coordinator* strategy, which consists in selecting for every process p in Π:

$$Coord(p, r) = p_{1+r \bmod n}$$

when $\Pi = \{p_1, \ldots, p_n\}$. Note that fixing the rotating coordinator strategy, any CHO machine reduces to an HO machine.

With the rotating coordinator strategy, agreement on the name of a coordinator is for free, that is every round is uniformly coordinated. On the other hand, the on-line strategy that consists in selecting p's coordinator in its heard-of set provides well coordinated rounds for free (in the case heard-of sets do not vary too much in time). A critical point is to achieve rounds which are both uniformly and well coordinated.

5.3 A first CHO machine for Consensus: the *CoordUniformVoting* machine

When looking closer at the *UniformVoting* machine, we may think to ensure uniformity of one round, and so termination, by the help of coordinators: at the beginning of each phase, coordinators are in charge to make the x_p's values uniform. More precisely, to each phase ϕ, we add a preliminary round in which every process p that sets out to be coordinator of phase

ϕ (i.e., $p = Coord(p, \phi)$) broadcasts the value of its variable x_p. Upon receiving a message with value v from $Coord(q, \phi)$, process q adopts this value for x_q. Actually, the additional round allows us to simplify the two rounds of *UniformVoting*: each process p just sends its vote instead of sending both its vote and the value of x_p. This yields an algorithm that we call *CoordUniformVoting* (see Algorithm 6).

Since the decision of one process at some round 2ϕ of *UniformVoting* entails all the x_p's to be equal, *CoordUniformVoting* still satisfies integrity and agreement. If at some phase ϕ_0, all processes agree on some coordinator's name c, and this coordinator is in the kernel of ϕ_0, then every process hears of and adopts x_c's value. In that case, all processes decide on this value at the end of phase ϕ_0. This proves the following theorem:

Theorem 5.2 *The CHO machine consisting of the* CoordUniformVoting *algorithm and the predicate that guarantees no split round and uniformly and well coordinated phases infinitely often solves Consensus.*

This exemplifies how the use of coordinators transforms the requirement of a non-trivial uniform round into the one of agreeing on the name of some process in the kernel. Note that agreement on the coordinator of a phase ϕ may be achieved by a leader election algorithm. At that point, the question is whether the elected process is actually in the kernel of round $3\phi - 2$.

Instead of using a leader election algorithm, we can adopt the rotating coordinator strategy. We denote by *CoordUniformVoting*^{rc} the resulting algorithm. With such a coordinator strategy, having the leader in the kernel at some point in computation is ensured by

$$\exists \phi_0 > 0 \; : \; \bigcap_{i=1}^{n} K(\phi_0 + i) \neq \emptyset.$$

Thus using the rotating coordinator strategy, we substitute uniformity of one round in the *UniformVoting* machine for some "temporal stability" guaranteeing a sufficiently long period with a non-empty kernel.

Algorithm 6 The *CoordUniformVoting* algorithm

```
Initialization:
    $x_p := v_p$    {$v_p$ is the initial value of $p$}
    $vote_p \in V \cup \{?\}$, initially ?

Round $r = 3\phi - 2$:
    $S_p^r$:
        if $p = coord_p(\phi)$ then
            send $\langle x_p \rangle$ to all processes

    $T_p^r$:
        if some message $\langle v \rangle$ is received
                                from $Coord(p, \phi)$ then
            $x_p := v$

Round $r = 3\phi - 1$:
    $S_p^r$:
        send $\langle x_p \rangle$ to all processes

    $T_p^r$:
        if all the values received are equal to $v$ then
            $vote_p := v$

Round $r = 3\phi$:
    $S_p^r$:
        send $\langle vote_p \rangle$ to all processes

    $T_p^r$:
        if at least one $\langle v \rangle$ with $v \neq ?$ is received then
            $x_p := v$
        if all the messages received are
                        equal to $\langle v \rangle$ with $v \neq ?$ then
            DECIDE($v$)
        $vote_p := ?$
```

5.4 The *DLS* algorithm

The algorithms described up to now work correctly only if some invariant properties for the *HO*'s are satisfied (e.g., $\mathcal{P}_{nekrounds}$ or $\mathcal{P}_{nosplit}$). When having a closer look at these algorithms, it turns out that the safety conditions of Consensus may be violated if there are some "bad" periods during which these predicates do not hold. Thereby, such algorithms cannot be used in systems with message losses (even very rare), which considerably limits the scope of these Consensus algorithms.

In a seminal paper [12], Dwork, Lynch, and Stockmeyer showed how to cope with such bad periods, and designed an algorithm, that we call *DLS*, which solves Consensus if a "sufficiently long" good period occurs. The basic idea of this algorithm is to satisfy safety conditions no matter how badly processes communicate, that is even if many failures occur in the system.

The *DLS* algorithm has been originally described in an HO-like style [12]. Rounds are grouped into phases, where each phase ϕ consists of four rounds. The algorithm includes the rotating coordinator strategy, and so each phase ϕ is led by a unique coordinator. We refer the reader to [12] for the complete description of DLS.

Dwork, Lynch, and Stockmeyer [12] proved that their algorithm never violates integrity and agreement. Moreover, they showed that it terminates if "there is a majority of correct processes, and there exists some round GST, such that all messages sent from correct processors at round GST or afterwards are delivered during the round at which they were sent." This corresponds to eventual space-time uniformity:

$$\exists \text{ GST } > 0,\ \exists \Pi_0 \text{ s.t. } |\Pi_0| > n/2\ :\ \forall p \in \Pi,\ \forall r \geq \text{ GST },\ HO(p,r) = \Pi_0$$

which in terms of failure detectors coincides with $\diamond\mathcal{P}$ [6].

Actually, this termination requirement can be drastically weakened: a single well coordinated phase without "too much" transmission failures entails termination. This is formalized in the following theorem:

Theorem 5.3 *The HO machine that consists of the DLS algorithm and the communication predicate:*

$$\begin{array}{l} \forall \phi > 0, \exists \phi_0 \geq \phi, \exists \Pi_0 \textit{ s.t. } |\Pi_0| > n/2\ : \\ \qquad (\forall p \in \Pi,\ \forall r \in \phi_0\ :\ HO(p,r) = \Pi_0) \ \wedge\ (p_{1+\phi_0 \bmod n} \in \Pi_0) \end{array}$$

solves Consensus.

Variant of DLS: Interestingly, the safety conditions of Consensus, namely integrity and agreement, still hold for any coordinator strategy, even when several processes lead the same phase. In other words, the CHO extension of *DLS*, that we denote *CoordDLS*, also satisfies integrity and agreement whatever the communication-coordinator predicate we consider. For termination, we just have to substitute the condition

$$(\forall p, q \in \Pi^2 :\ Coord(p, \phi_0) = Coord(q, \phi_0)) \ \wedge\ (\forall p \in \Pi :\ Coord(p, \phi_0) \in \Pi_0).$$

for the condition $p_{1+\phi_0 \bmod n} \in \Pi_0$ in Theorem 5.3. Thus this variant of *DLS* solves Consensus under the condition that there exists some uniform phase[6] ϕ_0 whose kernel $K(\phi_0)$ is a majority set, and which is led by a single process (coordinator) in $K(\phi_0)$.

5.5 A CHO algorithm *"à la Paxos"*: the *LastVoting* algorithm

The *DLS* algorithm is based on the rotating coordinator paradigm, which ensures permanent agreement on the coordinator, but as mentionned above, it supports a more flexible

[6] By *uniform phase*, we mean that each round of this phase is uniform.

Algorithm 7 The LastVoting algorithm

```
Initialization:
  $x_p \in V$, initially $v_p$   {$v_p$ is the initial value of $p$}
  $vote_p \in V \cup \{?\}$, initially ?
  $commit_p$ a Boolean, initially false
  $ready_p$ a Boolean, initially false
  $ts_p \in \mathbb{N}$, initially 0

Round $r = 4\phi - 3$:
  $S_p^r$ :
    send $\langle x_p , ts_p \rangle$ to $Coord(p, \phi)$

  $T_p^r$ :
    if $p = Coord(p, \phi)$ and
    number of $\langle \nu , \theta \rangle$ received $> n/2$ then
      let $\overline{\theta}$ be the largest $\theta$ from $\langle \nu , \theta \rangle$ received
      $vote_p :=$ one $\nu$ such that $\langle \nu , \overline{\theta} \rangle$ is received
      $commit_p :=$ true

Round $r = 4\phi - 2$:
  $S_p^r$ :
    if $p = Coord(p, \phi)$ and $commit_p$ then
      send $\langle vote_p \rangle$ to all processes

  $T_p^r$ :
    if received $\langle v \rangle$ from $Coord(p, \phi)$ then
      $x_p := v$ ; $ts_p := \phi$

Round $r = 4\phi - 1$:
  $S_p^r$ :
    if $ts_p = \phi$ then
      send $\langle ack \rangle$ to $Coord(p, \phi)$

  $T_p^r$ :
    if $p = Coord(p, \phi)$ and
    number of $\langle ack \rangle$ received $> n/2$ then
      $ready_p :=$ true

Round $r = 4\phi$:
  $S_p^r$ :
    if $p = Coord(p, \phi)$ and $ready_p$ then
      send $\langle vote_p \rangle$ to all processes

  $T_p^r$ :
    if received $\langle v \rangle$ from $Coord(p, \phi)$ then
      DECIDE($v$)
    if $p = Coord(p, \phi)$ then
      $ready_p :=$ false
      $commit_p :=$ false
```

coordinator strategy. The idea of using various policies for determining coordinators has been introduced by Lamport in the *Paxos* algorithm [16]. However, the idea is not followed through to the end in the latter algorithm: the first round of *Paxos* enforces the choice of a unique coordinator for the remaining rounds of the phase, and so the "Consensus core" in *Paxos* actually manages a single coordinator per phase.

Our main contribution here is to have observed that *Paxos* is still safe even in the presence of multiple coordinators. Thus we design a new CHO algorithm, called *LastVoting* (Algorithm 7), which follows the basic line of *Paxos*, but manages possible multiple coordinators per phase. More precisely, *LastVoting* is structured as *Paxos*, except the first round that is removed.[7] Agreement on a single coordinator (also called *leader*) is not achieved by the algorithm anymore, but is part of the conditions that guarantee termination.

The Consensus core in *Paxos* share many common features with *DLS*: as for *DLS*, the coordinator of a *Paxos* phase does not cast a vote and misses its turn if it does not receive conclusive information from enough (namely a majority) processes. This is the basic reason why *DLS* and *Paxos* both tolerate link failures. On the other hand, the two algorithms differ in the values of the coordinators' votes. In *DLS*, the coordinator of a phase votes for some value v if v is a *majority value* (i.e., it has heard that at least $n/2$ processes find v acceptable) whereas the coordinator of a *Paxos* phase votes for the *most recent value* it has heard of. This is why the coordinator of a *Paxos* phase can cast a vote (and so can make a decision) even if the preceding round is not uniform. In that respect, our *LastVoting* algorithm is similar to *Paxos*.

In the following theorem, we show that *LastVoting* is always safe (even in the presence of multiple coordinators at some phases), and we exhibit a very simple condition that enforces termination. Interestingly, the latter condition only involves *one* phase, and the corresponding communication-coordinator predicate is non-stable, contrary to the "Ω condition" – classically supposed for *Paxos* termination – that requires uniformly and well

[7] The first two rounds of every phase in *Paxos* are needed only when the coordinator changes. As said above, we always remove the first of these two rounds. The second of these two rounds is needed in *LastVoting* only when the coordinator changes.

coordinated phases *permanently* from some point in the computation.

Theorem 5.4 *The HO machine that consists of the LastVoting algorithm and the communication-coordinator predicate:*

$$\forall \phi > 0, \exists \phi_0 \geq \phi, \exists c_0 \in \Pi, \forall p \in \Pi \ : \left\{ \begin{array}{l} |HO(c_0, 4\phi_0 - 3)| > n/2 \wedge |HO(c_0, 4\phi_0 - 1)| > n/2 \\ (Coord(p, \phi_0) = c_0) \wedge (Coord(p, \phi_0) \in K(\phi_0)) \end{array} \right.$$

solves Consensus.

Proof: The above communication-coordinator predicate clearly enforces termination of the *LastVoting* algorithm.

Integrity is obvious. For agreement, let ϕ_1 be the first phase at which some process makes a decision. Let p be such a process and let v be its decision value. Lines (27) and (31) imply that p's coordinator at phase ϕ_1, denoted c, has received more than $n/2$ acknowledgements at round $4\phi_1 - 1$ and $vote_c^{(4\phi_1 - 1)} = v$. Moreover, c has received more than $n/2$ non-null messages at round $4\phi_1 - 3$.

For any phase $\phi \geq \phi_1$, let Π_ϕ denote the set of processes that have updated their timestamp variables at least once since phase ϕ_1:

$$\Pi_\phi = \{q \in \Pi \ : \ ts_q^{(\phi)} \geq \phi_1\}.$$

The heart of the proof is the following lemma, which says that from phase ϕ_1, each process q may barter the value of x_q only for v.

Lemma 5.5 *At any phase $\phi \geq \phi_1$, the following holds:*

1. *Π_ϕ is a majority set, i.e.,*
$$|\Pi_\phi| > n/2.$$

2. *For any process q in Π_ϕ, we have*
$$x_q^{(4\phi - 2)} = v.$$

Proof: By induction on $\phi - \phi_1$.
Basis: $\phi = \phi_1$. Any process q in Π_{ϕ_1} executes line (21), and so has received a vote from its coordinator $c' = Coord(q, \phi_1)$. Hence c' casts a vote at round $4\phi_1 - 3$, and so c' receives more than $n/2$ non-null messages at this round. Since each process sends at most one non-null message at round $4\phi_1 - 3$, we have $c' = c$. It follows that

$$x_q^{(4\phi_1 - 2)} = vote_c^{(4\phi_1 - 3)} = v.$$

Moreover, c receives more than $n/2$ acknowledgements, and so more than $n/2$ processes execute line (21) at phase ϕ_1. This shows that Π_{ϕ_1} is a majority set.
Inductive step: Suppose $\phi > \phi_1$, $|\Pi_{\phi - 1}| > n/2$, and for any $q \in \Pi_{\phi - 1}$, $x_q^{(4\phi - 5)} = v$. At phase ϕ, any process q in $\Pi_{\phi - 1}$ either lets ts_q unchanged or sets ts_q to ϕ. It follows that Π_ϕ contains $\Pi_{\phi - 1}$, and so Π_ϕ is a majority set.

Let q be any process in Π_ϕ. We consider two cases.

1. Process q does not execute line (21) at phase ϕ. Then $ts_q^{(\phi)} = ts_q^{(\phi - 1)}$ and $x_q^{(\phi)} = x_q^{(\phi - 1)}$. It follows that q belongs also to $\Pi_{\phi - 1}$. The inductive hypothesis implies that $x_q^{(\phi - 1)} = v$.

2. Process q updates ts_q and x_q at phase ϕ. Let $c' = Coord(q, \phi)$. From lines (17), (18), and (20), it follows that c' has casted a vote and $vote_{c'}^{(4\phi-3)} = x_q^{(4\phi-2)}$. Therefore c' has received more than $n/2$ non-null messages at round $4\phi - 3$. Since each process sends at most one non-null message at this round, one of them has been sent by a process in the majority set $\Pi_{\phi-1}$. Line (12) implies that the largest timestamp received by c' at round $4\phi - 3$ is at least equal to ϕ_1. From the inductive hypothesis we derive that $vote_{c'}^{(4\phi_1-3)} = v$. Hence $x_q^{(4\phi-2)} = v$, as needed.

$\Box_{Lemma\ 5.5}$

Let p' be a process that decides v' at phase ϕ, and let c' denote the coordinator of p' at phase ϕ. By definition of ϕ_1, we have $\phi \geq \phi_1$. We are going to prove that $v = v'$. For that, we proceed by induction on $\phi - \phi_1$.

Basis: $\phi = \phi_1$. Process c' has necessarily received more than $n/2$ acknowledgements at phase $\phi = \phi_1$. Since each process sends at most one such message per phase, we have $c = c'$, and so

$$v' = vote_{c'}^{(4\phi_1-3)} = vote_c^{(4\phi_1-3)} = v.$$

Inductive step: Let $\phi > \phi_1$ and assume that any decision value at phases $\phi_1, \cdots, \phi - 1$ is equal to v. Process c' has definitely casted a vote for v' at round $4\phi - 1$, and so more than $n/2$ processes q set x_q to v' and ts_q to ϕ at round $4\phi - 2$. Such processes share the same coordinator, namely c'. It follows that c' has received more than $n/2$ non-null messages at round $4\phi - 3$. By point (1) in Lemma 5.5, at least one of them has been sent by some process in Π_ϕ. From line (12) and point (2) in Lemma 5.5, it follows that $vote_{c'}^{(4\phi-3)} = v$. $\Box$

Rotating coordinator: Similarly to *CoordUniformVoting*, we can use the rotating coordinator strategy to determine coordinators in *LastVoting*: we denote by *LastVoting*^{rc} the resulting algorithm. With such a coordinator strategy, the existence of a uniformly and well coordinated phase is ensured by the following communication predicated $\mathcal{P}_{LastVoting^{rc}}$

$$\mathcal{P}_{LastVoting^{rc}} \ :: \ \exists \phi_0 > 0 \ : \ \bigcap_{i=1}^{n} K(\phi_0 + i) \neq \emptyset.$$

Observe that when (i) choosing the off-line strategy of the rotating coordinator in *LastVoting*, and (ii) requiring that the condition at line (11) always holds (which means that the coordinator sends a vote in every phase, see line 18), the resulting algorithm, denoted *RC*, corresponds to the *Rotating Coordinator* algorithm described in [6] for solving Consensus with the failure detector $\diamond\mathcal{S}$.

Because of point (ii), it turns out that *RC* is safe only under some non-trivial invariant property of the heard-of sets, namely the no split predicate $\mathcal{P}_{nosplit}$. More precisely, agreement may be violated if two coordinators receive messages from disjoint sets of processes (i.e., there is no process heard by both). This point is not discussed in [6] because the authors assume no link failure and a majority of correct processes, which guarantees $\mathcal{P}_{HO}^{maj}$, and so $\mathcal{P}_{nosplit}$. If this assumption does not hold, then the *Rotating Coordinator* algorithm forever, which is translated in the HO model by the fact *RC* is not safe.[8] The failure detector $\diamond\mathcal{S}$ or the predicate $\mathcal{P}_{\diamond nek}$ play a role only for the termination condition of Consensus.

[8] At two places in each phase, processes in the *Rotating Coordinator* algorithm wait for at least $n/2$ messages to advance to the next round. In the HO model, advancing from one round to the next is automatic, and so is not under the control of the processes. This is why executions of the *Rotating Coordinator* algorithm that block (because some process does not eventually hear of a majority of processes) are translated in the HO model into unsafe executions of the *RC* algorithm.

Obviously, the notion of failure detectors makes no sense in the context of link failures. However, our remark shows that basically, the *Rotating Coordinator* algorithm does not tolerate link failures, and more generally dynamic failures. To make it safe in the presence of such failures – which is a quite reasonable requirement –, it is sufficient just to add the test "number of $\langle \nu \, , \, \theta \rangle$ received $> n/2$" at line (11).

5.6 A non-coordinated algorithm without HO invariant

The *DLS* and the *LastVoting* algorithms have shown that Consensus can be solved without invariant predicate if we resort to coordinators. This naturally leads us to question whether Consensus is solvable without both invariant predicates and coordinators. As we shall show below, the answer is yes if there exist rounds in which heard-of sets have a membership larger than $2n/3$ (Algorithm 8), and we leave the question open in the case heard-of sets are only majority sets.

For that, we design an HO algorithm that we call *OneThirdRule* (Algorithm 8). A similar algorithmical schema is used in the first round of [4], in [22], and in the fast rounds of *Fast Paxos* [17]. Each phase of the *OneThirdRule* algorithm consists of one single round. Safety conditions of Consensus, namely integrity and agreement, are always satisfied: if some process decides v at line 13 of round r, then in any round $r' \geq r$, only v can be assigned to any x_p, and hence only v can be decided. Liveness is ensured by the following condition:

$$\exists r_0 > 0, \exists \Pi_0 \,\text{s.t.}\, |\Pi_0| > 2n/3, \forall p \in \Pi \,:\, (HO(p, r_0) = \Pi_0) \wedge (\exists r_p > r_0 \,:\, |HO(p, r_p)| > 2n/3).$$

The first part, namely the existence of some uniform round r_0 with an enough large kernel, makes the system "space uniform" in the sense that at the end of round r_0, all processes adopt the same value for x_p. The second part of the condition enforces every process p to make a decision at the end of round r_p. These observations establish the following result:

Theorem 5.6 *The HO machine consisting of the OneThirdRule algorithm and the communication predicate $\mathcal{P}_{(\mathcal{C}_0)^\infty}$, where $\mathcal{C}_0$ holds at round r_0 if*

$$\exists \, \Pi_0 \, s.t. \, |\Pi_0| > 2n/3, \forall p \in \Pi \,:\, HO(p, r_0) = \Pi_0,$$

solves Consensus.

Algorithm 8 The *OneThirdRule* algorithm

```
 1: Initialization:
 2:   $x_p := v_p$                                        { $v_p$ is the initial value of $p$ }
 3: Round $r$:
 4:   $S_p^r$ :
 5:     send $\langle x_p \rangle$ to all processes
 6:   $T_p^r$ :
 7:     if $|HO(p, r)| > 2n/3$ then
 8:       if the values received, except at most $[\frac{n-1}{3}]$, are equal to $\overline{x}$ then
 9:         $x_p := \overline{x}$
10:       else
11:         $x_p :=$ smallest $x$ received
12:       if more than $2n/3$ values received are equal to $\overline{x}$ then
13:         DECIDE($\overline{x}$)
```

Note that, contrary to all the algorithms we have described up to now, a decision is possible in just one round: if all the initial values are identical[9] and few transmission failures occur at the beginning (i.e., $\mathcal{C}_0$ holds at round 1), then every process decides at the end of the first round. Hence, the *OneThirdRule* algorithm which is a very simple algorithm that does not require any coordinator election procedure, may be quite efficient. Furthermore, the condition for its correctness is satisfied in many real systems. Indeed, one uniform round and "not too many" transmission failures (i.e., heard-of sets with a cardinality greater than $2n/3$) at some subsequent round is sufficient to entail decision. This may seem to be a strong condition in the classical context of permanent and static failures, such as crash failures, but it is a quite realistic assumption in systems with transient failures (eg., crash recovery). For all these reasons, we think that it would be very interesting to develop this solution in many real applications requiring Consensus among processes.

5.7 Summary

Table 2 summarizes the various HO and CHO Consensus algorithms described in this section, with their correctness requirements. Because of time invariance (cf. Section 2.1), the corresponding communication predicates guarantee conditions on rounds that must hold infinitely often, but not necessarily permanently (at least for the last four algorithms). Note that some of these requirements capture conditions that in the past have been expressed in terms of failure detectors, requiring conditions to eventually hold forever, and thus suitable for permanent failures only.

Algorithm	Predicate for safety	Predicate for liveness
UniformVoting	No split rounds	$\forall r > 0, \exists r_0 \geq r : \ r_0$ is uniform
CoordUniformVoting	No split rounds	$\forall \phi > 0, \exists \phi_0 \geq \phi : \phi_0$ is uniformly and well coordinated
$CoordUniformVoting^{rc}$	No split rounds	$\forall \phi > 0, \exists \phi_0 \geq \phi : \bigcap_{i=1}^{n} K(\phi_0 + i) \neq \emptyset$
CoordDLS	none	$\forall \phi > 0, \exists \phi_0 \geq \phi : \begin{cases} \phi_0 \text{ is uniform} \\ K(\phi_0) \text{ is a majority set} \\ \phi_0 \text{ is uniformly and well coordinated} \end{cases}$
LastVoting	none	$\forall \phi > 0, \exists \phi_0 \geq \phi : \begin{cases} \forall p \in \Pi : \lvert HO(p, \phi_0) \rvert > n/2 \\ \phi_0 \text{ is uniformly and well coordinated} \end{cases}$
$LastVoting^{rc}$	none	$\forall \phi > 0, \exists \phi_0 \geq \phi : \begin{cases} \forall p \in \Pi, \forall r, 4\phi_0 < r < 4(\phi_0 + n) : \lvert HO(p, r) \rvert > n/2 \\ \bigcap_{i=1}^{n} K(\phi_0 + i) \neq \emptyset \end{cases}$
OneThirdRule	none	$\forall r > 0, \exists r_0 \geq r : r_0$ is uniform and $\lvert K(r_0) \rvert > 2n/3$ $\wedge$ $\forall p \in \Pi, \exists r_p > r_0 : \lvert HO(p, r_p) \rvert > 2n/3$

Table 2: HO and CHO Consensus algorithms and correctness conditions

[9] In the context of Atomic Broadcast, messages can sometimes be spontaneously ordered, which translates into identical initial values for Consensus.

6 Conclusion

The paper proposes a new computational model for fault tolerant distributed systems, which is suitable for describing any type of system with benign failures. Apart from allowing elegant and concise expressions of Consensus algorithms, the model overcomes the limitations of the the classical approaches by getting rid of the dogmas around which all the existing models are structured.

In particular, our approach allows us to handle (1) transient failures and (2) failures that hit all the components of the system (links and processes) in a unified framework. By contrast, classical models are limited to static failures both in space and time. For example, there are few and non quite convincing results for the dynamic crash-recovery failure model. Furthermore, the HO model — contrary to classical approaches — can handle the (transient) omission failures of *all* processes.

As we have observed, a second key point of the HO model is to allow the expression of *sporadic* conditions, in contrast to the classical models where only *permanent* properties can be formulated. In other words, we are able to give a precise meaning to the statement "*the system works correctly for long enough*" in the HO model, whereas such a condition has only an informal meaning in the classical models. Indeed, Consensus algorithms in [12, 6] are proven correct with conditions of the type "*eventually and forever* the system behaves correctly", even though it is also claimed that the algorithms work correctly in practice, i.e., in real systems provided that a "good" period lasts long enough.

Besides, it is striking to see how, by removing the barrier between synchrony degree and failure model, the HO formalism enables us to give quite direct proofs of important results in fault tolerant distributed computing. In particular, we have unified results for synchronous and asynchronous systems, and have given a very simple proof of the weakest predicate that makes Consensus solvable under some failure bounds.

Acknowledgments. We would like to thank M. Biely, M. Huttle, U. Schmid and J. Widder for discussions related to the HO model. We also thank L. Lamport and J. Welch for their comments on an earlier version of the paper.

References

[1] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.

[2] M. Biely, B. Charron-Bost, M. Huttle, and J. Widder. Comparing failure detectors and the HO model. In preparation.

[3] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11(3):420–440, September 1990.

[4] F. Brasileiro, F. Greve, A. Mostéfaoui, and M. Raynal. Consensus in one communication step. In *6th International Conference Parallel Computing Technologies (PaCT)*, pages 42–50. Springer Verlag, LNCS 2127, 2001.

[5] T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of ACM*, 43(4):685–722, 1996.

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

[7] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[8] B. Charron-Bost, R. Guerraoui, and A. Schiper. Synchronous systems and perfect failure detectors: Solvability and efficiency issues. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 523–532. IEEE, June 2000.

[9] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchrony needed for distributed consensus. *Journal of ACM*, 34(1):77–97, January 1987.

[10] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in Byzantine agreement. *Journal of ACM*, 37(4):720–741, October 1990.

[11] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, November 1983.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–323, April 1988.

[13] T.E. Elrad and N. Francez. Decomposition of distributed programs into communication-closed-layers. *Science of Computer Programming*, 2(3), April 1982.

[14] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32:374–382, April 1985.

[15] Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *Proc of the 17th ACM Symp. Principles of Distributed Computing (PODC)*, pages 143–152, Puerto Vallarta, Mexico, June-July 1998.

[16] L. Lamport. The Part-Time Parliament. *ACM Trans. on Computer Systems*, 16(2):133–169, May 1998.

[17] L. Lamport. Fast Paxos. Technical Report MSR-TR-2005-12, Microsoft, 2005.

[18] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[19] M. J. Merritt. Unpublished Notes, 1985.

[20] S. Moran and Y. Wolfstahl. Extended impossibility results for asynchronous complete networks. *Information Processing Letters*, 26(3):145–151, 1987.

[21] G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, 1990.

[22] F. Pedone, A. Schiper, P. Urban, and D. Cavin. Solving Agreement Problems with Weak Ordering Oracles. In *Proceedings of the 4th European Dependable Computing Conference (EDCC-4)*, LNCS-2485, pages 44–61, Toulouse, France, October 2002. Springer-Verlag.

[23] M. Rabin. Randomized Byzantine Generals. In *Proc. 24th Annual ACM Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[24] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Symposium on Theor. Aspects of Computer Science*, pages 304–313, Paderborn, Germany, 1989.