THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *14 Décembre 2020* **par :**

Adam Shimi

On the Power of Rounds: Explorations of the Heard-Of Model

JURY

Bernadette Charron-Bost	Directrice de Recherche	Rapportrice
Rachid Guerraoui	Professeur	Rapporteur
Xavier Thirioux	Professeur	Président du Jury
Emmanuelle Anceaume	Directrice de Recherche	Examinatrice
Sébastien Tixeuil	Professeur des universités	Examinateur
Aurélie Hurault	Maître de conférences	Directrice de thèse
Philippe Quéinnec	Professeur des universités	Directeur de thèse

---

École doctorale et spécialité :

MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance

Unité de Recherche :

IRIT

Directeur(s) de Thèse :

Philippe Queinnec et *Aurélie Hurault*

Rapporteurs :

Bernadette Charron-Bost et *Rachid Guerraoui*

Remerciements

Une thèse comme la mienne, avec une grande marge de manoeuvre pour le doctorant, n'est pas possible sans une supervision adaptée. Je voudrais donc remercier mes encadrants Aurélie Hurault et Philippe Queinnec, qui ont accompli la lourde tâche de me canaliser. Je remercie aussi mes rapporteurs Bernadette Charron-Bost et Rachid Guerraoui pour leurs rapports, et les échanges sur des points techniques importants, notamment avec Bernadette Charron-Bost. Enfin, je remercie les membres de mon jury de thèse, Xavier Thirioux, Emmanuelle Anceaume et Sébastien Tixeuil, d'avoir accepter d'écouter et de juger mes travaux.

Bien sûr, une thèse se fait quelque part. Merci donc aux divers occupants du légendaire bureau F315 pour leur capacité toujours impressionnante à partir en vrille : Florent, Mathieu, Guillaume, Jonathan, et les autres. Tout le monde ne peut pas être élu au sain bureau; malgré tout, quelques braves méritent mes remerciements pour de nombreuses discusssions plus ou moins sensées : Crabe, Benoit et Ismail, entre autres. Merci aussi à mes collègues d'ACADIE d'avoir supporté mes questions étranges et mes sujets de recherche bizarres ces dernières années: Xavier, Yamine, Philippe, Marc, Xavier, Neeraj et les autres.

Je ne suis pas toujours resté dans ce bureau, contrairement à ce que la crise actuelle pourrait laisser penser. Merci donc à Armando et Karla pour leur accueil chaleureux à Mexico. Merci aussi à tous ceux qui font ou ont fait de l'Eurékafé ma deuxième (troisième? quatrième?) maison : Arnold, Sam, Agatha, Clémence, Seb, Maxime, Antoine, Lille, et pleins d'autres encore.

Enfin viennent les remerciements plus personnels. Merci à Jérémy pour avoir écouté mes plaintes et mes problèmes durant des années. Merci à Matthieu et Anaël pour avoir discuté encore et encore avec moi de la valeur de ma recherche, et de tout le reste. Merci à Romain de m'avoir changé les idées et encouragé sans jamais douter. Merci à mes parents de m'avoir soutenu, malgré l'incertitude de la recherche, pour faire quelque chose d'important à mes yeux. Merci à Amin, Raf et Skander de me changer les idées avec vos couilloneries. Merci à Flora d'être là.

À mes parents, pour m'avoir poussé à ne pas me contenter du minimum.
À Flora, pour le célébrer avec moi.

Contents

Chapter 0

Introduction

The only thing that makes life possible
is permanent, intolerable uncertainty;
not knowing what comes next.

The Left Hand of Darkness
Ursula K. Leguin

Sommaire

0.1 A Philosophy of Distributed Computing

0.1.1 Knowledge, Science and Computing

Distributed computing studies how information and uncertainty in part of a system constrains and generates information and uncertainty at the level of the whole system. Every algorithm, bound, impossibility result, and more, rely on each process' information and uncertainty about the system – its knowledge.

Even so, should we, researchers in distributed computing, care about such elusive philosophical details? Yes. The giants on which shoulders a field of research stands are not merely its technical results, but also the underlying intuitions about its objects of study. And none of these intuitions matters more than the type of knowledge produced by the field and how it is produced.

Take the natural sciences. Physics, Biology, Chemistry, all have concrete objects of study; things like lightning, mitosis and oxidation. The goal of these sciences is obvious: explaining these phenomena. To do so, models capture the simplest explanation accounting for all known facts; experiments test predictions made by those models, and falsify them. Such is the scientific method pushed by Karl Popper [1], and made into almost a definition of what Science is. Even accounting for variations, like the importance of synthesis in chemistry [2], this narrative captures most of the production of knowledge about natural phenomena.

On the other hand, what does Computer Science study? It's not computers, despite the name. It might be computation. But what's that? Computation is not, as far as human knowledge goes, a physical phenomenon like lightning, mitosis or oxidation. It is instead an abstraction, a process realized in many different ways, just like a short story can be written in ink, punched in braille, or spoken. The defining aspect of a computation lies in its mechanistic nature: a set of exact and non-ambiguous steps to answer a question or solve a problem. Theses steps might be probabilistic, complex or difficult to interpret (like the computation of a neural network), but they must still follow one from the other without holes or space for interpretation.

Computer Science thus studies procedures without need for creative interpretation. In doing so, it sheds light on what is knowable and doable with formal certainty – or at least formal guarantees.

But in distributed computing, these procedures interact with each other through the physical world, imprecise and uncertain. This adds a whole new layer to this analysis, captured by the many models of distributed computing.

0.1.2 Distributed Computing: in the Face of Uncertainty

Upon glancing at any textbook or proceedings on distributed computing, the abundance of models catches the eye. If not every paper, then every track at a conference or every chapter in a textbook likely studies a different model of distributed computation. [1] Sometimes the differences are subtle and easy to miss; sometimes the models have nothing in common, except their distributed nature.

Of course, models also abound in sequential computing. But they all end up equivalent to Turing Machines, or strictly weaker. Since concrete implementations of the strongest models exist – you might be reading this thesis on one –, the use of weakening is limited to self-referential issues in formal methods. Thus in sequential computability, a problem is considered computable if and only it is computable on Turing Machines. End of story. For distributed computing on the other hand, not all models are equivalent, some are incomparable, and no model is "better" than the other.

The heart of this difference comes from uncertainty. Sequential computing studies mechanic solvability; distributed computing studies mechanic solvability by interacting processes *despite uncertainty.* This uncertainty manifests itself in various ways:

- Capabilities of processes: polynomial time, exponential time, computable, more than computable,...
- Atomicity: which groups of events are considered to happen in a single step.
- Communication primitives: messages, shared-memory, one-to-one interactions,...

[1] The proceedings of PODC 2020 contains at least 15 different models!

- Degree of synchrony: how much the local clocks of processes are synchronized with each other, and the delays on communication.
- Kind and number of faults: issues or perturbations that might plague processes, communication primitive or exchanged information.
- ...

Because processes must take into account others, and how they might interact with them, knowledge plays a crucial role in distributed computing. Equivalently, every model of distributed computing provides a set of **axioms** about interactions between processes. The study of distributed computability and complexity simply unravels the consequences of these axioms for mechanical problem solving. Another way to say this is that the uncertainty needs to be tamed through assumptions; then it becomes non-determinism, which we can study formally.

But where do these axioms come from? In the case of sequential computing, there is at least the "one true notion of computation" as a guideline, thanks to the Church-Turing thesis. But as I've stated above, there is no "best" or "ideal" model of distributed computing. In this context, how to judge the worth of any such model?

To begin with, the value of these models lies not so much in capturing a concrete situation, but instead in the extraction of properties relevant to problems of distributed computation, like synchronization and symmetry-breaking. If a new model changes some long held hypothesis, or add new variants, and that alters computability and complexity, the model's in. It should shed light on what is useful and what is necessary for solving the problems the research community cares about.

Still, how to know if the property unearthed by a brand-new model occurs in any physical distributed system? For example, pure asynchrony – unbounded delays of communication – never occurs in a real system. It causes intricate and interesting behaviors; are they meaningful? Ultimately, as in Mathematics, a distributed model will be judged by the wealth of results it produces. Even if it makes too many assumptions or not enough to capture precisely a concrete system, such a model can clarify the fundamental questions of distributed computing. Overly strong models are enticing for impossibility results, because their inability to solve a problem propagates to their many weakenings. On the other hand, overly weak models serve to find algorithms: one working in this barren context will still work under any strengthening. In addition, for a solution to exist in a weak model, the latter must capture some necessary hypotheses for solving this problem.

We should thus recognize some value in the axioms unearthed by decades of research in distributed computing. These arguments only strengthens the need for many models, and thus the creation of new issues compared to sequential computing: keeping track of all these models in their own right, as well as the web of relations between them.

Managing its plethora of models therefore represents one of the fundamental questions of distributed computing.

0.2 The Monopoly of Rounds

Most of this section comes from Sections 1 and 2 of my paper "The Splendors and Miseries of Rounds" [3], published in SIGACT News.

The issue with having a lot of models stems not so much from their number, but from their organization: models in distributed computing form more of a menagerie than a taxonomy. What the field needs is a perspective through which some structure can be brought to this mess. Rounds offer one such promising approach, by constraining models enough to allow formalization. Just as importantly, rounds appear almost everywhere in distributed computing. So using them, instead of tacking on some ad-hoc concept from elsewhere, reveals underlying patterns in the field.

In this section, I explore in more details how and why rounds are used in distributed computing.

0.2.1 The Essence of Rounds

Rounds provide structure – a structure to build algorithms on. This aspect of rounds might be the most obvious, since numerous distributed algorithm designers leverage this structure, be it for design or analysis. But what is this structure? Usually, rounds involve sending messages tagged with the current round number, waiting for some messages with the same number, and computing the next state while incrementing the round number. Such a description helps to recognize rounds in the wild, but not in the unambiguous way of a mathematical definition. Worse, it implies the use of messages, whereas the structure of rounds seems independent of the means of communication.

Fortunately, Elrad and Francez [4] solved this problem almost 40 years ago. In studying the decomposition of concurrent composition of CSP programs into layers, they stumbled into a decomposition into chunks of time that was safe – equivalent to the initial program in terms of behavior. They called such chunks communication-closed layers: layers such that no communication ever happens between different layers.

Communication-closedness truly captures the essence of rounds. Operationally, it ensures that processes at round r don't use messages from other rounds in their local computation. Messages from rounds before r will never be used, and are thus thrown away; messages from rounds after r are buffered until their round. As long as some decomposition into communication-closed layers exists for an algorithm, the latter uses rounds as an underlying structure, whether implicitly or explicitly.

In general researchers and algorithm designers don't need such a formal understanding of rounds to use them. Yet there is a realm where it proves necessary: formal verification. There, knowing how communication-closedness works gives a handle into the verification of round-based distributed algorithms. Indeed, Chaouch-Saad et al. [5] proved that many properties one wants to check on distributed algorithms, such as the properties defining consensus, can be verified using communication-closed rounds as if they were synchronous. Such reductions were leveraged notably by Damien et al. [6] and Gleissenthal et al. [7], to verify asynchronous fault-tolerant algorithms.

Communication-closedness is also essential to ensure the usual atomicity of round-based models: the fact that at each process, for each round, the emission of messages for this round (or any form of outgoing communication like writing in shared-memory), the reception of messages for this round (or any form of incoming communication like reading in shared-memory) and the local computation for this round happen as if in one step. This is possible because the messages received at round r by p cannot depend on the messages p sent at round r, because if some q receives a message from p at round r, the next messages it will send are at round $r + 1$.

0.2.2 The Adventures of Rounds

One common mistake about rounds is assuming synchrony – the property of processes computing at the same speed, with a known upper bound on communication delay. Sure, almost every synchronous algorithm uses rounds, and synchrony provides a simple way to implement rounds: broadcast, wait for the upper bound on communication delay to receive all messages, do some computation, then change round. But communication-closedness is what really matters for rounds – and both synchrony and asynchrony allow it. It is therefore essential to disentangle rounds and synchrony, which I do next by exploring both synchronous and asynchronous uses of rounds.

In practice, synchrony generally costs too much to implement. But in theory, it gives an excuse for assuming rounds – without justifying how they are implemented. When the focus of a new model isn't the rounds themselves, but some other parameter, a synchronous version provides the simplest possible setting to study the new parameter. For example, many biological-inspired "message-passing" models assume rounds through synchrony, such as the beeping model of Cornejo and Kuhn [8], the task-allocation model of Cornejo et al. [9] or the Cellular Bioelectric model of Gilbert et al. [10]. Other more technological models also depend on synchrony for rounds, such as radio networks from Chlamtac and Kutten [11] or dynamic networks from Kuhn et al. [12].

In the asynchronous setting, rounds provide robustness instead. State-machine replication and failure detectors show that well. The former uses rounds where processes wait for a majority of messages before deciding – see Paxos from Lamport [13], Zab from Junqueira et al. [14] or Viewstamped Replication from Oki and Liskov [15]. Concerning failure detectors, rounds provide a natural way to use the information they give: which process is suspected of crashing. In most algorithms using failure detectors, like the ones of the original papers by Chandra and Toueg [16] and Chandra et al. [17], processes wait only for messages from the unsuspected processes, and then change round.

After seeing its value in both synchronous and asynchronous settings, you might wonder why this structure is so integral to distributed algorithms. The answer is because it allows a sequential analysis. As Raynal and Rasjbaum argued in a recent article [18], this reduction to sequential reasoning remains The Way for understanding concurrent and distributed computation.

The quintessential example of this is the combinatorial topology approach to distributed computing [19]. In it, problems and protocols are cast as manipulations of simplicial complexes, high-dimensional combinatorial objects. All possible inputs form a complex, all possible outputs another, and the problem imposes constraints on maps from the input complex to the output complex. The solvability of the problem then reduces to the existence of a map satisfying these constraints. For many standard models, the map from inputs to outputs splits naturally into communication-closed layers – and thus rounds. Then the existence of the sought-for map depends only on the properties of one layer, and on how they compose. This reduction of the combinatorial complexity usually yields a clean and powerful mathematical description, usable for many impossibility results and lower bounds. For example, wait-free shared-memory can be described as the iterated application of an elegant map called a barycentric subdivision [20]. This characterization proved integral to the proof of a general computability theorem for this model [20].

One must be careful, though: despite the benefits of thinking about distributed computation sequentially, not all distributed algorithms use rounds. Many asynchronous algorithms deliberately focus on point-to-point communication, usually to lower the message count or the waiting time. Sometimes the model itself ends up unsuited to communication-closed layers. Population protocols are an example: although rounds are mentioned for complexity issues, these models study one-to-one communication with minimal memory requirements – two reasons for not using rounds.

0.2.3 In Search of Distributed Time

Measuring time is another fundamental use of rounds, dating back to the dawn of distributed computing. Indeed, where sequential models of computation have no problem defining what a time step is, parallelism and interleavings complicate the story for distributed computing. What is a time step of a system with a lot of parts computing and communicating at different times and possibly different speeds?

The round provides an answer to this difficult question: it defines a step of a distributed algorithm cleanly, and in a way that focuses on the cost of communication. The time taken by the algorithm is then the number of rounds. This idea goes back at least to the early 80's, in a paper by Arjomandi et al. [21], although tracking its first use proves difficult. What's sure is that since then, round complexity has become a staple of distributed computing research, and the subject of most lower bounds in the literature.

The depth of this abstraction of time inspired a full-fledged distributed theory of complexity [22], analogous to the sequential one [23, 24]. This theory focuses on Peleg's $\mathcal{LOCAL}$ model [25], a standard synchronous model parameterized by a fixed topology. It defines and studies various complexity classes for distributed decision problems. Central to this study is the notion of locality: a decision problem is local if and only if some algorithm solves it in a constant number of rounds, whatever the number of processes. Local algorithms represent truly tractable solutions, in terms of scalability – the number of rounds stays the same when the number of processes increases. In that sense, the class of local decision problems plays an analogous role to the class $\mathcal{P}$ of tractable sequential decisions problems.

Following the analogy, many fascinating ideas from computational complexity translate to the distributed setting: verification of certificates [22, 26, 27], derandomization [28], hardness of approximation [29]... All these thanks to the underlying rounds providing a robust notion of time.

That being said, like any measure, round complexity is but a tool, not always useful or even meaningful. In asynchronous models for example, the "real" time taken by each round can vary wildly – eroding the relation between rounds and time. The repeated broadcasts inherent in rounds also put them at odds with other measures of performance focused on the communication cost, like message complexity. Nevertheless, the wealth of results generated by rounds as a complexity measure vindicates once more their place as a fundamental idea of distributed computing.

0.3 Nothing but Rounds: the Heard-Of Model

Most of this section comes from Section 3 of my paper "The Splendors and Miseries of Rounds" [3], published in SIGACT News.

As we saw, rounds crop up all around in the design and analysis of distributed algorithms. There is but a small step in concluding from this exploration that they could make a useful abstraction of distributed computing models.

As a matter of fact they do.

0.3.1 Equivalence with Round-Based Models

Most message-passing models consider faults at the level of components such as processes and links; Santoro and Widmayer [30] were the first to propose a model where only the consequences of faults are described. That is, the model only captures whether a message is received or not, and not why. Combined with the assumptions of rounds and synchrony, this gives a simple yet powerful round-based model parameterized by how many and which message losses are allowed at each round.

This model capture surprisingly well some classical asynchronous models. The first evidence came from Raynal and Roy [31], in a short note proving the equivalence of the asynchronous model with at most F permanent crashes, with the round-based model above where at each round, for each process, at most F messages sent to this process are lost. Such an equivalence follows from a simulation from one model to the other, and back. This entails that given an algorithm in one model, there is a systematic way to build an algorithm for the same problem in the other model; their computability is the same.

Afek and Gafni [32] then pushed further this line of inquiry. They generalized the previous model by defining "message adversaries", predicates constraining which messages can be lost at which round. With an adversary limited to remove a message from at most one direction for each communication link, they showed the equivalence with the wait-free single-reader multiple-writer shared-memory model. Then Raynal and Stainer [33] doubled down, by extending this approach for models using failure detectors, both in message-passing and shared-memory. They provided an adversary equivalent to wait-free shared-memory augmented with Ω, as well as one for asynchronous message-passing augmented with (Σ, Ω). Moreover, these results outlined the relations between various message-adversaries.

Therefore, many fundamental models in distributed computing reduce to round-based ones, at least for questions of computability. Yet I find an issue with the message adversary model: the assumption of synchrony. As I argued above, synchrony is not inherent in rounds – communication-closedness is. Moreover, the simulations don't use the additional power of synchrony over rounds: the knowledge that if a message is not received at its round, it will never be received. Thus synchrony serves only as a crutch to justify the use of rounds.

Fortunately, there are promising formalisms keeping all the power of message adversaries while abstracting away such implementation details.

0.3.2 Abstracting rounds

An approach to round-based models without synchrony is the Round-by-Round Fault Detector model (or RRFD) of Gafni [34]. Executions in this model proceed through rounds implemented by waiting for the messages of processes in a set given by an oracle, the RRFD. Different constraints on communication are then captured by predicates on the output sequences of RRFDs. Among others, Gafni found predicates for asynchronous message-passing with crashes, wait-free shared-memory and partially synchronous message-passing. This approach was also extended in the GIRAF model of Keidar and Shraer [35], which allows more predicates, and even some non-communication-closed rounds.

But these two abstractions suffer from the same issue than the message adversaries: they are too operational. They describe how rounds are implemented, not just how they unfold. Moreover, the use of fault detectors implicitly accuse some components for the failure. And such accusation forces the model to rely on actual crashes and failing links, going back to square one – many incomparable parameters. Last but not least, RRFDs don't ensure the correctness of their ouput, making processes wait forever in some cases. This behavior makes sense in the study of indulgent algorithms [36], where the point is to deal with untrustworthy oracles; but for general models of computation, stronger guarantees are required.

0.3.3 A Mathematical Abstraction: the Heard-Of model

Taking the best of message adversaries and RRFDs results in only considering which messages are received at each round. This yields the Heard-Of model of Charron-Bost and Schiper [37], which captures models as predicates on the messages actually received on time.

The building blocks of the Heard-Of model are the heard-of collections and the heard-of predicates. Since the latter are merely predicates on the former (or equivalently, sets of them), we can focus on the heard-of collections. A heard-of collection returns, for every process p and round r, the set of processes heard at round r by p.[2] From the viewpoint of an algorithm in the Heard-Of model, these processes are the senders of the messages given to p at round r for its local computation. From the viewpoint of a system-level implementation of the Heard-Of model, these processes are the senders of messages received before giving to the algorithm at p its set of messages for round r – before p went to round $r+1$, basically.

This deserves emphasis: a heard-of collection captures the messages that were received **on time**. In this context, on time means "before the end of the round it was sent in".

Fundamentally, the Heard-Of model is about the logical information available to processes during the rounds. It focuses on the flow of information within rounds, instead of on what happens to messages exactly. Hence the knowledge given is not of which messages will be delivered, but of which messages will be usable for local computation.

This entail an important aspect of the Heard-Of model: its atomicity. The succession of broadcast, receptions and local computation at round r for process p forms a single step – from the viewpoint of other processes, it is either over or has not happened yet. Such atomicity can be rephrased by saying that there is no distinct event that is causally between two of the three events of p at round r. This in turns follows from the fact that processes broadcast before receiving their messages, and the received message are from round r. So the receptions on p at round r cannot be causally linked with the broadcast of p at round r, because the receptions at another process q happen after q's own broadcast for r.

[2]A heard-of collection can also be seen as an infinite sequence of directed graphs over Π, where the r-th graph captures communication at round r. Then q being an incoming neighbor of p in the r-th graph means that p received the message of q at round r.

This atomicity is to be contrastred with the common atomicity in asynchronous message-passing models (like in Fischer, Lynch and Paterson [38])), where it's the succession of receptions, local computation and emissions that forms an atomic step. I explore the differences in the next chapter.

The main promise of the Heard-Of model lies in capturing multiple assumptions about communication – the axioms – in a formal mathematical framework. It considers synchrony and faults together; both are abstracted by which messages are received on time.

To see how this work, consider crashes in synchronous and asynchronous message-passing. If p doesn't receive a message from q at round r, what can it deduce about the state of q? In the synchronous case, that q crashed, and thus that no message from it will be received for future rounds, whether on time or not. In the asynchronous case on the other hand, maybe the message was just late – p cannot infer that it will never receive a message from q in the future. Notice how synchrony and asynchrony are used to predict the pattern of messages. The Heard-Of model cuts the middle man, and uses the patterns of messages from the start.

The benefits of this approach over the menagerie of classical models are numerous:

- **(Model comparison)** Predicates in the Heard-Of model come with their formal comparison relation: translation. One predicate HO_1 translates into another HO_2 if for each collection of HO_1, there is a way to compose successive rounds (and maybe throw away some messages) to get a collection of HO_2. Basically, HO_2 is simulated on top of HO_1 by taking multiple rounds of HO_1 to gather enough messages and then choosing the ones to keep according to HO_2. It has yet to be studied in depth, even if the original paper [37] gives some translations, as do a couple of papers about specific problems, such as the study of approximate consensus by Charron-Bost et al. [39]. Schmid et al. [40] also studied the limits of this comparison relation.

- **(Computability and complexity results)** A model's worth lies in the quality of the results proven in it. On this front, the Heard-Of model is already quite successful, both for computability (finding new algorithms for classic problems, on predicates never studied before) and for complexity (bounding below the number of rounds necessary to solve a problem on a predicate). The most notable results include new algorithms for consensus in the original paper by Charron-Bost and Schiper [37]; characterizations of the heard-of predicates on which consensus is solvable by Coulouma et al. [41] and Nowak et al. [42]; a syntactic characterization of the algorithms solving consensus for a subset of heard-of predicates by Balasubramanian and Walukiewicz [43]; a characterization of the heard-of predicates on which approximate consensus is solvable by Charron-Bost et al. [39]; and a study of k-set agreement by Biely et al. [44].

- **(Formal verification)** I mentioned previously that rounds, or more specifically communication-closedness, help with formal verification. It should thus be no surprise that the elegant mathematical abstraction of the Heard-Of model also works well with formal verification. Among others, there are: a proof-assistant based (Isabelle/HOL) verification of consensus algorithms in Charron-Bost et al. [45]; cutoff bounds for the model checking of consensus algorithms by Marić et al. [46]; the mentioned above syntactic characterization of algorithms solving consensus for a subset of heard-of predicates by Balasubramanian and Walukiewicz [43]; and a DSL to write code following the structure of the heard-of model and verify it with inductive invariants by Drăgoi et al. [47].

0.4 The Road Ahead

Despite the advantages of the Heard-Of model, it is not as widely used in the distributed computing literature as one could imagine. Which raises the question: what is missing? What we have is a clean mathematical model, that captures many interesting parameters of the uncertainty of distributed computation, and is amenable to formal verification. On the other hand, the Heard-Of model as it stands lacks at least three things: a mean to extract the predicate from a classical model, a comparison with classical models in terms of what is computable, and general techniques for proving computability and complexity results.

Let's start with the extraction of a predicate. An elegant model is great, but at the end of the day, a useful model is one that captures the object of study. Traditional models of distributed computing emerged from this tension; they arose through trial and error, back and forth between usable models and models describing the actual problems. But when confronted with the Heard-Of model, this hard-won legitimacy of the classical models vanishes. And no mean exists yet for knowing which predicate corresponds to what. Of course, there are examples and correspondence in the original paper [37]; but these are given, not proved. Even if I accept these examples, what about the ones not mentioned? How should I go about translating any given model into a heard-of predicate?

Next, even when knowing how to translate some model into a heard-of predicate, is something lost in translation? Does the heard-of predicate captures exactly the computability of the original model, or strictly less? The usual suspects in this line of reasoning are communication-closedness and the absence of failures. Intuitively, it seems harder to solve problems while unable to use late messages, and while forcing every process, even the silent ones, to decide. But no paper proved that a given model has no equivalent heard-of predicate; and even the literature on these equivalences is slim.

Finally, let's say that I have a model, its corresponding heard-of predicate, and both are as powerful in terms of computability. Then how do I prove lower bounds, impossibility results, properties of my algorithms? The other face of a clean mathematical model is an increased complexity of the reasoning. In addition, the Heard-Of model differs enough from previous models that the techniques established for the latter don't translate trivially to the former.

Addressing these three problems forms the core of my thesis. After a chapter defining formally the Heard-Of model and presenting its relation with the literature, the next three chapters each address one problem, instantiated to a more concrete form:

- **(Derivation)** Given a classical model of distributed computing, what is the "closest" heard-of predicate? This is answered in Chapter 2 for asynchronous models, by studying strategies that constrain when to change round.

- **(Expressivity)** Is there an equivalence between classical models and heard-of predicates? This question is addressed in Chapter 3, by showing the equivalence of failure detectors in the Chandra-Toueg hierarchy with specific heard-of predicates.

- **(Computability)** Given a heard-of predicate, what can be derived in terms of computability and complexity results? This is studied in Chapter 4, through impossibility results for k-set agreement.

Chapter 1

The Heard-Of Model: Definitions and Perspectives

All models are wrong, some are useful

George Box

Sommaire

1.1 Introduction

After setting the stage and outlining the results to come, let's introduce in full force the main character: the Heard-Of model. This chapter gets into the details of this model, and presents its place among the rest of distributed computing models.

1.2 Into the Weeds: Definitions and Details

1.2.1 Defining the Heard-Of Model

To start, we will only use the Heard-Of model in the context of a fixed set of processes Π and a complete topology. Although this is not in the original definition, many predicates and results do assume such hypotheses. See Section 1.2.3 for a discussion of some alternatives.

As mentioned in the introduction, the Heard-Of model constrains communication through heard-of predicates, which are themselves predicates on heard-of collections. These predicates play the role of assumptions about synchrony, faults, network topology, and more.

Usually, heard-of collections are represented as functions from a round r and a process p to a set of processes – the processes from which p heard the message sent at round r before or during its own round r.

Definition 1.1 (Heard-Of Collections and Predicates)**.** Let Π a set of processes. Then an element h of $(\mathbb{N}^* \times \Pi) \mapsto \mathcal{P}(\Pi)$ is a **heard-of collection**. The outputs of a heard-of collection are the **heard-of sets** of this collection.

In the same way, an element of $\mathcal{P}((\mathbb{N}^* \times \Pi) \mapsto \mathcal{P}(\Pi))$ is a **heard-of predicate** $\mathcal{HO}$ for Π.

From another perspective, heard-of collections are infinite sequences of communication graphs – directed graphs which capture who hears from whom on time, in that $q \in h(r,p) \iff (q,p)$ is an edge of the r-th communication graph.

Definition 1.2 (Collection as a sequence of directed graphs)**.** Let $Graphs_\Pi$ be the set of directed graphs whose nodes are the elements of Π. Then $gr \in (Graphs_\Pi)^\omega$ is a **heard-of collection**.

A function h and a sequence gr represent the same collection when $\forall r > 0, \forall p \in \Pi : h(r,p) = In_{gr[r]}(p)$, where $In(p)$ is the set of incoming neighbors of p.

In general, which perspective to use in a theorem or a proof follows naturally from the context. For example $h[r]$ only makes sense for a sequence of directed graphs, while $h(r,p)$ only makes sense for a function.

Although we won't be talking about algorithms in the Heard-Of model until Chapter 3, it is important to note that in this thesis, we assume that all algorithms in the Heard-Of model broadcast at each round. Then the heard-of collection used in some execution of such an algorithm really captures which messages are received on time. Another perspective would be to say that processes might send a message only to some processes, and thus that the heard-of collection captures the messages that would be received on time, assuming they are sent. This is an interesting approach, but for simplicity, in this thesis we priviledge the first one.

Now, remember that the Heard-Of model ensures communication-closedness. Thus in an algorithm, processes at a given round only interact with processes at the same round – they only consider messages from this round. To do so, the system-level implementation of the Heard-Of model has processes buffer messages received in advance, and discard ones received late. Say q, during its round r, sends a message to p. What happens at p splits into four possibilities (some of them only possible assuming asynchrony):

- p receives the message while it is at a round $< r$; it buffers it for the moment. Then it might be used during round r of p, and $q \in h(r,p)$.
- p receives the message while it is at round r; it might use it for its local computation during this round, and $q \in h(r,p)$.
- p receives the message while it is at a round $> r$; it discards the message, and $q \notin h(r,p)$.
- p never receives the message; it has nothing to do, and $q \notin h(r,p)$.

The fundamental intuition is that heard-of collections and predicates capture the logical information available at the end of a round. From this comes both their simplicity and their complex relations with classical models.

1.2.2 Defining Predicates

The original paper by Charron-Bost and Schiper [37] gives properties of collections useful in defining heard-of predicates.

The kernel of a round contains the processes heard by everyone at that round – that is, the processes that successfully broadcasted. As for the kernel of an execution, it contains processes that broadcast at every round. Then the cokernels are simply the complements of the corresponding kernels.

Definition 1.3 (Kernels and Cokernels of a heard-of collecton)**.** Let ho be a heard-of collection and let $r > 0$. Then the kernel of ho at r, $K_{ho}(r) \triangleq \bigcap\limits_{p \in \Pi} ho(r, p)$. The kernel of ho, $K_{ho} \triangleq \bigcap\limits_{r>0} K_{ho}(r)$.

Also, the cokernel of ho at r, $CoK_{ho}(r) \triangleq \Pi \setminus K_{ho}(r)$. And the cokernel of ho, $CoK_{ho} \triangleq \Pi \setminus K_{ho}$.

Let's now turn to the main heard-of predicates used in papers and in the next chapters. Table 1.1 lists their mathematical definition, while the following list unwraps them.

- $\mathcal{HO}^{sp\_unif}$ captures the **space-uniformity** property: at every round, processes either broadcasts or are not heard by anyone. Equivalently, every heard-of set at round r equals the kernel at this round. This predicate thus ensures a strong synchronization within each round.

- $\mathcal{HO}^{reg}$ captures the **regularity** property: at every round, the heard of sets are contained in the kernel of the previous round. If someone did not hear from p at round r, then no one will ever hear from it at rounds $> r$. This is a defining property of classical synchronous models.

- $\mathcal{HO}^{nosplit}$ captures the **no-split** property: at every round, every heard-of set intersects non trivially with every other. Such a property plays the role of quorums in asynchronous models.

- $\mathcal{HO}^{nekrounds}$ captures the **non-empty kernel rounds** property: as the name suggests, at every round, the kernel contains at least one process. So each round has at least one source (when considered as a communication graph). Note that the source might change between rounds.

- $\mathcal{HO}^{nek}$ captures the **non-empty kernel** property: the kernel of the whole collection contains at least one process. That is, a permanent source exists for each collection.

- $\mathcal{HO}^{K}_{F}$ captures the **kernel lower bound** property: the kernel of the whole collection contains at least $n - F$ processes. These processes act like correct processes in a synchronous setting: they don't crash and their messages are received on time.

- $\mathcal{HO}_{F}$ captures the **heard-of set lower bound** property: all heard-of set contains at least $n - F$ processes. So processes act like correct processes in an asynchronous setting with at most F crashes: they can always wait for $n - F$ processes, but not always for more.

Name	Definition
$\mathcal{HO}^{sp\_unif}$	$\{h$ a heard-of collection $\mid \forall r > 0, \forall p, q \in \Pi : h(r,p) = h(r,q)\}$
$\mathcal{HO}^{reg}$	$\{h$ a heard-of collection $\mid \forall r > 0, \forall p \in \Pi : h(r+1,p) \subseteq K_{ho}(r)\}$
$\mathcal{HO}^{nosplit}$	$\{h$ a heard-of collection $\mid \forall r > 0, \forall p, q \in \Pi : h(r,p) \cap h(r,q) \neq \emptyset\}$
$\mathcal{HO}^{nekrounds}$	$\{h$ a heard-of collection $\mid \forall r > 0 : K_{ho}(r) \neq \emptyset\}$
$\mathcal{HO}^{nek}$	$\{h$ a heard-of collection $\mid K_{ho} \neq \emptyset\}$
$\mathcal{HO}^{K}_{F}$	$\{h$ a heard-of collection $\mid \lvert coK_{ho}\rvert \leq n - F\}$
$\mathcal{HO}_{F}$	$\{h$ a heard-of collection $\mid \forall r > 0, \forall p \in \Pi : \lvert ho(r,p)\rvert \geq n - F\}$
$\mathcal{HO}^{count}_{L}$	$\{h$ a heard-of collection $\mid \sum\limits_{r>0, p\in\Pi} \lvert \Pi \setminus ho(r,p)\rvert \leq L\}$
$\mathcal{HO}^{countrounds}_{L}$	$\{h$ a heard-of collection $\mid \forall r > 0, \sum\limits_{p\in\Pi} \lvert \Pi \setminus ho(r,p)\rvert \leq L\}$

Table 1.1: A list of heard-of predicates.

- $\mathcal{HO}^{count}_{L}$ captures the **count** property: at most L messages in the entire execution might not arrive on time. This bounds the number of message loss, a form of constrained unreliable communication.
- $\mathcal{HO}^{countrounds}_{L}$ captures the **count by round** property: at most L messages per round might not arrive on time. Here the bound concerns each round, as if fixing the risk of message loss for specific periods of time.

1.2.3 Reaching the Limits

This thesis defends the Heard-Of model as a powerful lens for studying distributed computing. Nonetheless, like every model, it has limitations. I list the most important here to give an intuition of where the model can and can't be applied.

- **(Fixed number of processes)** Almost all heard-of predicates studied rely on the number of processes. This entails that for these predicates, enforcing dynamicity requires changes.

  An approach would be for Π to contain all processes that will eventually join, and make them silent until they join and after they leave. But such an approach would require a change in the known predicates – to deal with the join and removal of processes. For example, $\mathcal{HO}_F$ explicitly uses the (assumed fixed) number of processes to constrain the size of the heard-of sets. Another example comes from kernels: a process arriving in the middle of an execution that is heard by everyone after might be good enough for some predicates based around kernels; but this process wouldn't be in the kernel of the collection. Another alternative would be to define what was the heard-of collection for processes joining at round r.

- **(Assumed orthogonality of communication capability and algorithm)** In the Heard-Of model, whether processes can communicate at a round is assumed independent of the algorithm being used. This separates the heard-of collection (the communication behavior) and the algorithm. Such orthogonality comes from the fact that the Heard-Of model studies fault tolerance. On the other hand, as we will see in the state of the art below, some models break this independence. Models with interference are a prime example: two neighboring processes communicating at the same time cancel or alter their messages.

This assumption is so baked into the Heard-Of model that I don't see how to remove it while retaining the interesting results.

- **(No randomness)** As of now, the Heard-Of was never used for studying randomized algorithms, despite their importance in the field. The definition of a Heard-Of algorithm uses deterministic send and transition functions, but there is no issue in principle with replacing them by relations for non-determinism or probability distributions for randomness. That being said, the behavior of probabilistic algorithms running on infinite sequences of directed graphs with complex properties might prove more difficult to study than on the usual setting – where the topology is fixed or the properties are simpler.

- **(One shot tasks)** Finally, pretty much all known algorithms for the Heard-Of model solve one-shot tasks: problems like consensus that needs to be solved only once. But many distributed problems are not one-shot: implementation of shared objects, repeated consensus, or any problem used as a building block to solve another problem.

  A big difficulty here is that algorithms in the Heard-Of model assume that all processes start at round 1. To ensure this in the context of repeated tasks, termination of the previous task needs to be detected globally – a really difficult problem. A recent approach to this question comes from Damian et al. [48], which use synchronized Heard-Of rounds and calls to subprotocols to capture non-recursive distributed computations.

1.3 State of the Art: Distributed Computing Models

Since we want to use the Heard-Of model instead of other classical models of distributed computing, it makes sense to explore and describe the latter before moving on with the technical results.

1.3.1 I'll Do Anything for Synchrony

As mentioned above, synchrony eases the computation of a corresponding heard-of predicate by giving a single meaningful way to implement rounds: waiting for the communication upper-bound before changing rounds. Indeed, this is so obvious that most synchronous models in the literature never describe the implementation of rounds – they merely assume their existence. This also entails that all synchronous models ensure the same atomicity than the Heard-Of model: at each process and at each round, the broadcast, receptions and local computation happens as in one step.

Since the results in this thesis focus on the more difficult asynchronous case, this section offers an opportunity: detailing, for many synchronous models, the kind of heard-of predicates generated by the standard implementation of rounds in synchronous models, and whether seeing them through the Heard-Of model makes any sense at all.

$\mathcal{LOCAL}$ model The most popular synchronous model of distributed computing in recent times is assuredly the $\mathcal{LOCAL}$ model [25]: a synchronous model with a fixed topology, no faults, and reliable communication. Notice that every graph problem ends up computable in this model, as long as the underlying graph is connected: every process eventually receives the input of everyone else, and thus computes a solution locally.

Researchers using this model thus focus on complexity issues. Given a family of graphs of increasing size (the cycles of length n or the grid graphs with n^2 elements for example), they study the round complexity of solutions in function of the size of the graph. Just like in sequential complexity, what matters is the most efficient algorithms for the problem, not just any algorithm. For example, what problems are solvable by looking only at a constant size neighborhood of the node? These are the so-called local problems, whose solutions scales perfectly when the size of the graph increases. As an example, computing a maximal matching (a maximal set of edges which do not intersect) in a bicolored-graph (a graph with a 2-coloring of its vertices) is a local problem [49].

Returning to our original question, the Heard-Of model captures the $\mathcal{LOCAL}$ model in a very simple way: the predicate contains a single collection, where at each round the communication graph is the network topology in the $\mathcal{LOCAL}$ model. The Heard-Of model thus completely supersedes the $\mathcal{LOCAL}$ model in terms of expressivity. It makes sense, as the Heard-Of attempts to capture all meaningful models of distributed computing, whereas the $\mathcal{LOCAL}$ model functions as a crisp and simplified setting for studying distributed complexity theory.

$\mathcal{CONGEST}$ model The $\mathcal{CONGEST}$ model [25] is the version of the $\mathcal{LOCAL}$ model with bandwidth constraints: messages take at most $O(log(n))$ bits.

This changes entails two consequences: not all problems can be solved in $O(n)$ rounds, contrary to the $\mathcal{LOCAL}$ model (because forwarding the inputs might require messages of more than logarithmic size in the number of node); and local algorithms in the $\mathcal{CONGEST}$ model are in some sense even more scalable than ones in the $\mathcal{LOCAL}$. This is because a local algorithm can in theory send messages of a size $O(n)$, making it scalable only in terms of rounds, not message size.

Also in contrast with the $\mathcal{LOCAL}$ model, no heard-of predicates ensures the constraints on message size. It can be added on top of the model, but that is an ad-hoc change instead of a parameter tweak. This incompatibility between the two models probably stems from the orthogonal concerns behind their invention: studying distributed complexity with low-bandwidth for the $\mathcal{CONGEST}$ model, and studying distributed computability and (round) complexity in round-based models for the Heard-Of model.

Radio networks One aspect of distributed communication absent from the models above is interference. This stems from a focus on wired networks, where protocols for dealing with interference hide the issue. But for wireless communication, one must take into account these interference when designing algorithms.

A popular example of such a wireless model is the radio network model of Chlamtac and Kutter [11]: nodes lie on a fixed graph, either send a message or listen at each round, and receive a message if and only if they listened and only one of their neighbors sends a message. Here interference manifests itself in the most straightforward way – by cancelling out messages sent to the same process at the same round.

In this context, our hypothesis that algorithms in the Heard-Of model always broadcast is an issue: it will cause interferences all the time. But even when switching to an interpretation of heard-of predicates which capture which message could be received if they were sent, no heard-of predicates exists for these radio network models. This is because heard-of predicates capture the capability to communicate, which are assumed independent of the behavior of processes. A heard-of collection captures which process could hear which other at each round; it doesn't capture that p will hear q if and only if q sends a message, and no other process does.

Nonetheless, we could use a graph of actually received messages in place of the communication graphs of the Heard-Of model, and capture interference that way. This would define predicates parameterized by the algorithm used. Although it seems possible, this goes beyond the Heard-Of model, and thus the scope of this thesis.

Dynamic networks The common point of the $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models is their fixed topology. Dynamic networks aim to study what happens if this topology changes instead. The first paper presenting this model is by Kuhn et al. [50]. The model basically assumes synchronous rounds, with the connectivity between processes possibly changing at each round. Hence the model is parameterized by the possible sequence of communication graphs – that is to say by a heard-of predicate.

Therefore the Heard-Of model is really appropriate for capturing this kind of models. Even better, a heard-of predicate comes with less operational assumptions, and thus subsumes dynamic networks models.

That being said, since the Heard-Of model requires a fixed number of processes and synchronous starts, among other things, any variant of dynamic networks without these hypotheses might prove difficult to capture through a heard-of predicate.

Biological models Among models of biological distributed computing, many assume synchronous rounds. As a representative example, I'll focus on the beeping model [8]. It's a model with very minimal communication, possibly capturing computation between simple processes like cells.

The beeping model, first defined by Cornejo and Kuhn [8], consists of processes in a fixed network with synchronous rounds. What sets this model apart is that processes do not send messages; instead they beep. At each round, each process decides between beeping and listening for beeps. When some process listens, it only hears whether one of its incoming neighbors beeped. Depending on the variant of the model (see Casteigts et al. [51] for a survey), processes might also hear interferences (two neighbors beeping at the same time) while listening, while beeping, or never.

Is the Heard-Of model adequate for capturing such a model? The first problem comes from the nature of communication: the Heard-Of model considers messages, not beeps. One could argue that beeps are just a form of message, but they actually provide even less information than binary messages, and the "only one beep is heard even if multiple neighbors beep" property cannot be represented through messages alone. Nonetheless, it is probably straightforward to extend the Heard-Of model for this form of communication. On the other hand, a possibly insurmountable problem is the non orthogonality of communication capability with the algorithm. Interferences entail that whether or not a process can communicate with another process depends on who else tries to communicate in the neighborhood. This is the same sort of problem as for the radio network models.

1.3.2 Beyond Synchrony

Classical asynchronous models Maybe the most classical model of distributed computing is the asynchronous message-passing model with reliable communication and crashes. It's on this model that the most famous impossibility result of the field, the impossibility of consensus with one possible crash, was proven by Fischer et al. [38].

As mentioned in the previous chapter, this model (called FLP in the rest of this section) ensures another form of atomicity than the Heard-Of model: the atomicity at each process of receptions, local computation and emissions. The atomicity of FLP can be used to guarantee the atomicity of the Heard-Of model, by implementing communication-closed rounds. On the other hand, the atomicity of the Heard-Of model (and other round-based models) is not enough to guarantee the atomicity of FLP, because of communication-closedness. That being said, there is still a way to simulate the execution of algorithms in FLP-like models (for example the failure detector model) in the Heard-Of model with the right predicates. See Raynal and Stainer [33] and the chapter 3 of this thesis.

FLP can be captured by the Heard-Of model with the predicate $\mathcal{HO}_F$ in Table 1.1, where F is the maximum number of crashes. This characterization was done multiple time in the literature, notably in Charron-Bost and Schiper's original paper [37] and in a note by Raynal and Roy [31].

Another staple of asynchronous models is asynchronous shared memory. Instead of sending messages, processes in these models communicate by writing and reading shared objects, like Single-Writer-Multiple-Reader registers (which work as the name implies), or atomic snapshot (allowing atomic reading and writing an array of multiple registers). And although many intuitions about the Heard-Of model leverage messages, the shared-memory models can be captured by round-based models equivalent to the Heard-Of. These are the RRFDs of Gafni [34] and the message adversary model of Afek and Gafni [32] (although in this paper it is simply called "synchronous message passing with message delivery failures").

Failure detectors Although the asynchronous models mentioned above are indeed classical, they suffer from the impossibility results for many problems, such as consensus. One way to get around these comes from failure detectors. Failure detectors, as coined by Chandra and Toueg [16], are oracles providing for each process of the system, a local oracle that gives a set of suspected processes when called. The correlation between suspicion and the actual behavior of processes (like crashing) depends on the different properties of failure detectors: for example the perfect failure detector ensures that every process eventually detects every crash, and that no uncrashed process is ever suspected. Depending on these properties, using failure detectors allows one to solve consensus in asynchronous systems. Most of the uses of failure detectors are on top of a FLP-like model where at most $n - 1$ processes might crash, with a complete topology and reliable communication.

Failure detectors might appear incompatible with the Heard-Of model; yet exactly the opposite holds. One, if not the main, use of failure detectors is to implement rounds in asynchronous systems. It is then the properties of those rounds that allow or not the solving of problems like consensus.

This is the subject of Chapter 3 in this thesis. The results and characterizations there rely heavily on the work by Raynal and Stainer [33] on proving equivalences between asynchronous models augmented with failure detectors and synchronous models with message adversaries.

Partial Synchrony So-called partially synchronous models are really the FLP model with additional synchrony assumptions. They come from efforts to solve consensus on asynchronous systems. Instead of saying that no bounds on communication or processes speed exist (full asynchrony), partially synchronous models have additional assumptions about how periods of synchrony alternates with periods of asynchrony. Algorithms on this model aim at ensuring safety all the time, and advancing liveness when long enough periods of synchrony occur. Two of the biggest approaches to fault-tolerance, Paxos by Lamport [13] and Viewstamped Replication by Oki and Liskov [15], implicitly use such a model, ensuring correctness even with pure asynchrony, and liveness when synchrony is good enough.

The first instance of partially synchronous models probably comes from Dwork et al. [52]. They study additional synchrony assumptions on FLP, on the relative speed of processes and/or on communication delay. They define two kinds of additional synchrony assumptions: one with known upper bounds that hold only after some unknown time T; and another with unknown upper bounds that hold from the beginning.

Because such models lie between fully synchronous message-passing (which can be readily abstracted by a heard-of predicate capturing exactly the delivered messages) and fully asynchronous message-passing (which I show in the next chapter can be meaningfully captured by a heard-of predicate), partially synchronous models should be captured by the Heard-Of model too. Hutle and Schiper [53] take the first step in this direction, by studying the round properties that can be implemented on a weakened version of one of Dwork et al. [52] partially synchronous models: known upper bounds, which hold only for some unknown interval of time.

Nonetheless, a systematic approach to abstract partially synchronous models through the Heard-Of model is still missing for now.

1.4 Conclusion

The Heard-Of model provides a clean and powerful abstraction of distributed computing. It abstracts away many of the fundamental models of distributed computation. That being said, it still has limitations. These in turn explain why some models, notably the ones with interference, seem incompatible with the Heard-Of model, even if they use rounds.

Chapter 2

Making Heard-Of predicates

How did it get so late so soon?

Attributed to Dr. Seuss

Sommaire

2.1 Introduction

There are two levels at which we can consider the Heard-Of model: at the level of an algorithm, which only sees the messages for the current round, and at the system level, where the rounds are implemented concretely. This chapter focuses on the second level: no algorithm to solve a distributed problem like consensus or MIS will be considered. Instead, it explores how to implement rounds in ways that guarantee a certain heard-of predicate on a given operational model.

I thus define and explore an approach for formalizing the implementation of a heard-of predicate on top of an informal model of distributed computation. More specifically, I focus on asynchronous message-passing models, as implementing heard-of predicates on top of these models is more involved.

2.1.1 It's always asynchrony's fault

Finding the heard-of predicate corresponding to a given distributed computing model is difficult. Obviously, part of the problem comes with starting from a mishmash of English and Mathematics (the operational model) to build a formal predicate (the heard-of predicate). But the main issue is that the problem might not be well-defined: it's not clear what the "heard-of predicate corresponding to a distributed computing model" even means.

Let's start investigating with the easy mode: synchrony. By this I mean the traditional synchronous message-passing model, with synchronized local clocks, every computation step taking the same time, and upper bounds on communication delays. We'll start by assuming no crash. With such a model, rounds can only be defined in one meaningful way: broadcast, wait for the upper bound on communication delay to receive all the messages that will ever be received from this round, then change round. Thanks to this upper bound, every process eventually changes round. Implementing rounds in this way gives rise to a heard-of predicate, as each execution corresponds to the heard-of collection defined by the messages received before the end of the round. This gives rise to the heard-of predicate corresponding to the synchronous model. It always exists; what's left is to characterize it.

If we add crashes to the mix, there is the additional issue of defining the heard-of sets for crashed processes after they crash. We use the assumption, common in the literature on the Heard-Of model, that crashed processes still receive the messages from non-crashed processes. Obviously, if they're crashed, they cannot receive messages. But replacing them by silent processes that still wait for the upper bounds and receive all messages simplifies the predicates. It also doesn't change the part that the correct processes can observe – they never receive another message from a crashed process.[1] If we do that, then even with crashes, the way to implement rounds discussed above gives rise to a single heard-of predicate.

On the other hand, this neat story breaks down when switching to hard mode: asynchrony. In an asynchronous model, there is no upper bound on communication, by definition. This means that there is no clear-cut non trivial[2] rule for when to end rounds in general. Let's emphasize this point: no rule for deciding when to change round works for all asynchronous

[1]Another way to interpret this is that from the point of view of a correct process, there is no difference between a crashed process and a silent one. And thus we're free to choose the representation that is most helpful. Choosing the silent process allow us to define clean heard-of predicates, and thus is the choice taken.

[2]The rule that always allows the change of round will never block, but it also guarantees nothing in terms of heard-of predicate.

models. In this context, a rule that works is a rule that ensures progress of rounds – no process ever blocks forever at some round.

The absence of such a rule has dire consequences for the well-definiteness of the question: each rule that works for a given asynchronous model will implement its own heard-of predicate. Which one should be used to abstract the original model in the Heard-Of model? Is there always such a predicate? And how to compute it?

This chapter addresses these questions.

2.1.2 Overview

My approach to characterize asynchronous models in terms of heard-of predicates relies on the following intuition: what matters is the most constrainted predicate, among the set of predicates that can be implemented on top of the original model.

To express what constraint means in the context of the Heard-Of model, let's turn to a common mathematical duality – that of predicates and sets. Every predicate has a dual representation as a set: the set of all elements satisfying this predicate. So a heard-of predicate can be thought as the set of heard-of collections satisfying it; it constrains which heard-of collection is possible in an execution of the Heard-Of model. Then if there were less collections in it, it would be more constrained, because there would be even less choice of heard-of collection. Hence the most constrained heard-of predicate of a set of heard-of predicates is the one that is included in all the other predicates of the set – if it exists. Another perpective is that it's the minimal element by inclusion of the set of heard-of predicates that can be implemented on top of the original model.

In distributed computing, we usually say that a model is "strong" if it constrains heavily what can happen, and "weak" if it constrains lightly what can happen. Then the more constrained heard-of predicate of a set, if it exists, is the strongest in this sense.

I formalize this by introducing an intermediary step: delivered predicates, predicates on delivered collections. The latter capture the messages that are delivered from each round r, without considering the round of delivery. This is to contrast with heard-of collections, that only capture messages tagged by r if delivered at the receiver when the local round counter is $\leq r$. Because the round of delivery is not considered, computing the delivered predicate for a model does not require a rule for when to change round. Intuitively, the delivered predicate of a model is the heard-of predicate of the same model if it was synchronous (and thus every delivery happened on time). Going to delivered predicate before heard-of predicate allows the formal introduction of rounds without caring about asynchrony and the difficulties it entails.

Notice that a delivered predicate can be trivially implemented on top of the original model by maintaining a local round number, and then in repetition broadcasting a message tagged with it, and change round (with the receptions arriving at any point). So a delivered predicate is really the original model recast in terms of rounds, without guarantees about which message will be received on time and thus usable for each round. That is what going to heard-of predicates buys us.

Figure 2.1 shows this separation of the derivation in two steps. We start with the operational model, derive its delivered predicate, and then find the "strongest" heard-of predicate (the smallest when viewed as a set, the most constrained) that can be implemented by a rule (called a strategy) for this specific delivered predicate. Such a strongest predicate fully characterizes an asynchronous message-passing model in terms of the Heard-Of model.

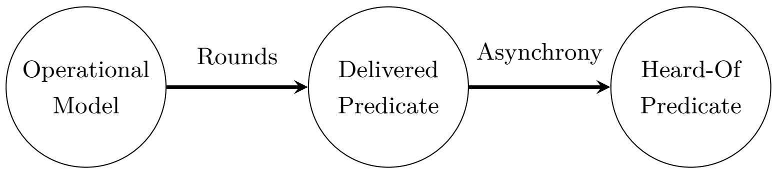


Figure 2.1: From classical model to heard-of predicate

That being said, a formalization is not enough by itself; there needs to be some way to compute such a characterizing predicate, and proving it is indeed the strongest. This problem becomes tractable by introducing two new ideas: families of strategies, and operations on predicates and strategies.

Recall that strategies are the rules that tell when processes can change rounds. Then families of strategies are sets of strategies that only depend on some limited part of the local state of a process. Because they are more limited, they're easier to study, and the heard-of predicates that they generate have specific forms.

For the second point, it's often easier to build complex delivered predicates and strategies by combining together simple ones. The operations defined in this chapter allow this, in a way that maintains important properties of the strategies. In some cases, the heard-of predicate characterizing the resulting delivered predicate can be found by combining the heard-of predicates characterizing the building blocks, using the same operations.

The results of this chapter are the following:

- The definition of delivered predicates and strategies, as well as operations on both, in Section 2.2.
- The formalization of the derivation of heard-of predicates from a delivered predicate and a strategy, in Section 2.3. This comes with a complete example: the asynchronous message-passing model with reliable communication and at most F permanent crashes.
- The study of oblivious strategies, the strategies only looking at messages for the current round, in Section 2.4. I provide a technique to extract a strategy dominating the oblivious strategies of the complex predicate from the strategies of its building blocks; exact computations of the generated heard-of predicates; and a sufficient condition on the building blocks for the result of operations to be dominated by an oblivious strategy.
- The study of conservative strategies, the strategies looking at all messages from previous and current round, as well as the round number[3], in Section 2.5. I provide a technique to extract a strategy dominating the conservative strategies of the complex predicate from the strategies of its building blocks; upper bounds on the generated heard-of predicates; and a sufficient condition on the building blocks for the result of operations to be dominated by a conservative strategy.
- A preliminary exploration of strategies using messages from "future" rounds, and an extended example where these strategies build stronger heard-of predicates that conservative and oblivious strategies. This is done is Section 2.6.

[3]They thus can't depend on messages from "future" rounds: messages tagged with a round greater than the round counter of the receiver

Finally, note that this chapter is a combination of two published papers coauthored with my advisors: "Characterizing Asynchronous Message-Passing Models Through Rounds" [54] published in OPODIS 2018, and "Derivation of Heard-Of Predicates From Elementary Behavioral Patterns" [55] published in FORTE 2020.

2.2 Delivered Predicates: Rounds Without Timing

In this chapter, the main information we're interested in is which messages are received, and at which round for the receiver. This means that the executions considered are not full executions of an algorithm where the content of messages matters – the important part are the emissions, deliveries, and changes of rounds.

2.2.1 Removing Timing Information

What makes the synchronous case easier boils down to the equivalence between the messages that are delivered at all, and those that are delivered on time. This cannot be replicated in the asynchronous case, as each asynchronous model requires a different rule for which messages to wait for before changing round. This also entails that it might be impossible to wait for all the messages that will be delivered and not block forever.

For example, in an asynchronous model with at most F crashes, it's safe to wait for $n - F$ messages before changing round, as at least $n - F$ processes will never crash. But in the asynchronous model with at most $F + 1$ crashes, doing so will get processes blocked in some cases.

On the other hand, even for such asynchronous models, we can study the predicate defined by the messages eventually delivered. Let's call this a **delivered predicate**, and it has the same formal definition as Definition 3.6 of heard-of collections and predicates – only the interpretation changes.

Definition 2.1 (Delivered Collections and Predicates)**.** Let Π a set of processes. Then an element c of $(\mathbb{N}^* \times \Pi) \mapsto \mathcal{P}(\Pi)$ is a **delivered collection**. The outputs of a delivered collection are the **delivered sets** of this collection.

An element of $\mathcal{P}((\mathbb{N}^* \times \Pi) \mapsto \mathcal{P}(\Pi))$ is a **delivered predicate** $\mathcal{DEL}$ for Π.

For examining the difference, recall that we're considering the Heard-Of model at a system level: we're implementing it. Then let's take an execution of some implementation (which needs to satisfy some constraints, defined later): a linear order of emissions, receptions and changes of rounds (a step where the local round counter is incremented) for each process. Then if each process changes round infinitely often, there's a delivered collection d and a heard-of collection h corresponding to this execution – just look at which messages sent to p tagged with round r where received at all by p (for d) and which were received when the round counter at p was $\leq r$ (for h). That is, for a round $r > 0$ and processes $p, q \in \Pi$, $q \in d(r, p)$ means that p received at some point the message of q annotated by r. On the other hand, $q \in h(r, p)$ means that p received (while it's round counter was $\leq r$) the message of q annotated by r. Hence the heard-of collection extracted from this execution captures which messages were waited for (and thus could be used at the algorithm level – but that's not treated here), whereas the delivered collection extracted from this execution captures which messages were received at all.

To find the delivered predicate corresponding to an asynchronous model, the intuition is to take its synchronous version, and then take the heard-of predicate that would be implemented by the rule for changing rounds in synchronous models. This is the delivered predicate for the model. This captures the strongest heard-of predicate that could be implemented on top of this asynchronous model, if processes could wait for all messages that will be delivered.

In general, they can't, since it requires knowing exactly what's happening over the whole distributed system. Nonetheless, the delivered predicate exists, and it plays the role of an ideal to strive for. The characterizing heard-of predicate of a model will be the closest over-approximation of the delivered predicate that can actually be implemented.

Now, let's look at some examples of delivered predicates. Starting with a classic, the asynchronous model with reliable communication, and at most F crash failures (where crashes can happen at any point).

Definition 2.2 (DEL_F^{crash})**.** The delivered predicate DEL_F^{crash} for the asynchronous model with reliable communication and at most F permanent crashes $\triangleq$

$$\left\{ c \textit{ a delivered collection} \;\middle|\; \forall r > 0, \forall p \in \Pi : \begin{array}{cc} & |c(r,p)| \geq n - F \\ \wedge & c(r+1,p) \subseteq K_c(r) \end{array} \right\}.$$

Why is this the delivered predicate for this model? Well, Charron-Bost and Schiper [37, Table 1] define it as the heard-of predicate of the synchronous version of this model. To prove it, we would need a formal definition of the delivered predicate for a given model. And this is hard because operational models are rarely formal to begin with, and when they are formal, it is often in incompatible ways.

Thus the best we can do right now is to give an informal argument for why, if you take the asynchronous model with reliable communication and at most F permanent crashes, and implement communication-closed rounds in any way that ensures an infinite number of rounds for every process, the messages received will form a delivered collection of DEL_F^{crash}. In the other direction, every collection of DEL_F^{crash} captures the message received in an execution of the implementation of rounds on top of the aforementioned asynchronous model.[4]

- Let t be an execution of an implementation of communication-closed rounds on top of the asynchronous model above, with the condition of ensuring an infinite number of rounds.

  Again, we consider that a crashed process will still receive all messages after it crashes. Since crashed processes never do anything else, it's not incoherent with what actually happens.

  This entails, in addition to reliable communication, that what determine if a message is received by a crashed process is the fact that it was sent by a non-crashed process. Since every process that has not crashed broadcasts, this entails that every process will eventually hear the message from every non-crashed process at this round. And since there's at most F crashes, that's at least $n - F$ messages per round. Hence $|c(r,j)| \geq n - F$.

  Also, if p hears from q at round $r+1$, then q sent the message before crashing. This means q did not crash at its own round r, and thus that the message it broadcasted at that round was sent, and will eventually be delivered. Hence $c(r+1,j) \subseteq K_c(r)$.

[4]This assumes that crashed processes are modelled as silent processes, as explained before in this chapter.

- Let c be a collection such that $\forall r > 0, \forall p \in \Pi : |c(r,p)| \geq n - F \wedge c(r+1,p) \subseteq K_c(r)$. This collection corresponds to the execution where the crashed process are the ones that stop broadcasting, because communication is reliable, so if $q \notin c(r,p)$, this means that q never sent its message to p tagged with r. From the model this means that it crashed during its broadcast at round r. Each crash thus happens at the first round where the crashed process is not heard by everyone, after sending the messages that are actually received at this round.

Later in this chapter, the heard-of predicate characterizing this delivered predicate (the most constrained one) is derived. But it's interesting to mention it here, as a comparison. This is $\mathcal{HO}_F$ (defined as $\{h \text{ a heard-of collection} \mid \forall r > 0, \forall p \in \Pi : |ho(r,p)| \geq n - F\}$ in Table 1.1). The difference lies in the kernel condition: DEL_F^{crash} ensures that if any message sent by p at round r is not eventually delivered, then no message will be delivered from p at rounds $> r$. Intuitively, p not broadcasting means that it crashed in the middle of sending messages at round r, and thus that it will never send messages for the next rounds. But this is not maintained by $\mathcal{HO}_F$. Why? Because the $n - F$ messages that are waited for are not necessarily the same at each process. So q might wait for a message from p at round r, but k might receive at least $n - F$ messages at round r without the one from p. And because of this, k cannot conclude that p crashed, because the message might just be late.

Here is another delivered predicate, this time for the asynchronous model without crashes, but with at most L messages lost in the whole execution (of the system level implementation of rounds).

Definition 2.3 (DEL_L^{loss})**.** The delivered predicate DEL_L^{loss} for the asynchronous model without crashes, and with at most L message losses $\triangleq$
$$\left\{ c \textit{ a delivered collection} \;\middle|\; \sum_{r>0, p\in\Pi} (n - |c(r,p)|) \leq L \right\}.$$

This one is not from Charron-Bost and Schiper [37], but we can apply the same reasoning than for the previous delivered predicate. Here the sum counts the number of messages that are never delivered. Since all the processes are correct, this corresponds to the number of lost messages.

For $L = 1$, the best known strategy (to our knowledge) implements $\mathcal{HO}_1^{countrounds}$ (See Table 1.1). What is lost in implementing a heard-of predicate on top of this delivered predicate is that instead of losing only one message over the whole execution, there might be one loss per round. This is explained in Section 2.6.

2.2.2 Building Delivered Predicates

Due to the difficulty of defining the delivered predicate for a given model, it might be hard to derive it. The more complex the model, the more complex the derivation of its delivered predicate. On the other hand, simple models are relatively easy to characterize by a delivered predicate.

This motivates the following proposal to solve the red part of Figure 2.2: building complex delivered predicates from simpler ones. That way, there will be no need to derive by hand the delivered predicates of complex models.

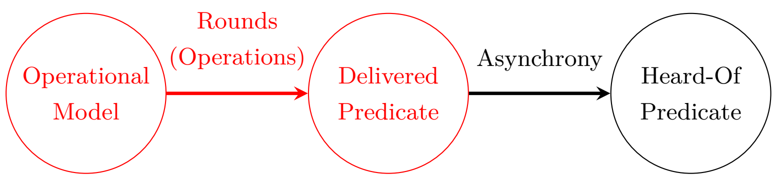


Figure 2.2: From classical models to delivered predicates

For example, consider a system where one process might crash and may or may not recover later on[5]. In some sense, this behavior is defined by having the delivered collections for one possible crash that never recover, and the delivered collections for one possible crash that must recover. This amounts to a union (or a disjunction); I write it $DEL_1^{canrecover} \triangleq DEL_1^{crash} \cup DEL_1^{recover}$.

The first predicate of this union is DEL_1^{crash}, the delivered predicate for at most one crash that never recovers. But what about the second one, $DEL_1^{recover}$? Intuitively, a process that can crash but must recover afterward is described by the behavior of DEL_1^{crash} which is shifted to the behavior of DEL^{total} (the predicate where all the messages are delivered) after some time. I call this the succession of these predicates, and write it $DEL_1^{recover} \triangleq DEL_1^{crash} \rightsquigarrow DEL^{total}$.

Finally, imagine adding another crash that cannot recover to the previous predicate. Thus a behavior where there might be one crashed process as constrained by DEL_1^{crash} and another crashed process as constrained by $DEL_1^{canrecover}$. I call it the combination (or conjunction) of these predicates, and write it $DEL_1^{crash} \bigotimes DEL_1^{canrecover}$.

The complete system is thus described by $DEL_1^{crash} \bigotimes ((DEL_1^{crash} \rightsquigarrow DEL^{total}) \cup DEL_1^{crash})$.

In the following, I also introduce an operator ω to express repetition. For example, a system where, repeatedly, a process can crash and recover is $(DEL_1^{crash} \rightsquigarrow DEL^{total})^{\omega}$.

Let's now define formally these operations.

Definition 2.4 (Operations on predicates)**.** Let P_1, P_2 be two delivered or heard-of predicates.

- The **union** of P_1 and P_2 is $P_1 \cup P_2$.
- The **combination** $P_1 \bigotimes P_2 \triangleq \{c_1 \bigotimes c_2 \mid c_1 \in P_1, c_2 \in P_2\}$, where for c_1 and c_2 two collections, $\forall r > 0, \forall p \in \Pi : (c_1 \bigotimes c_2)(r,p) = c_1(r,p) \cap c_2(r,p)$.
- The **succession** $P_1 \rightsquigarrow P_2 \triangleq \bigcup_{c_1 \in P_1, c_2 \in P_2} c_1 \rightsquigarrow c_2$,
  with $c_1 \rightsquigarrow c_2 \triangleq \{c \mid \exists r \geq 0 : c = c_1[1,r].c_2\}$ ($c_1[1,0]$ is the empty sequence).
- The **repetition** of P_1, $(P_1)^{\omega} \triangleq \{c \mid \exists (c_i)_{i \in \mathbb{N}^*}, \exists (r_i)_{i \in \mathbb{N}^*} : r_1 = 0 \wedge \forall i \in \mathbb{N}^* : (c_i \in P_1 \wedge r_i < r_{i+1} \wedge c[r_i + 1, r_{i+1}] = c_i[1, r_{i+1} - r_i])\}$.

The intuition behind these operations is the following:

- The union of two delivered predicates is equivalent to an OR on the two communication behaviors. For example, the union of the delivered predicate for one crash at round r and of the one for one crash at round $r+1$ gives a predicate where there is either a crash at round r or a crash at round $r+1$.

[5]If it does, we can assume that its memory is intact and no messages received in the meantime are lost, but that's not important for the system level implementation

- The combination of two behaviors takes every pair of collections, one from each predicate, and computes the intersection of the graphs at each round. Meaning, it adds the loss of messages from both, to get both behaviors at once. For example, combining DEL_1^{crash} with itself gives DEL_2^{crash}, the predicate with at most two crashes.
- For succession, the system starts with one behavior, then switch to another. The definition is such that if $r = 0$, then no prefix of c_1 is used (the first behavior never happens), but the second one must always happen.
- Repetition is the next logical step after succession: instead of following one behavior with another, the same behavior is repeated again and again. For example, taking the repetition of at most one crash results in a potential infinite number of crash-and-restart, with the constraint of having at most one crashed process at any time.

The usefulness of these observations comes from allowing the construction of interesting predicates from few basic ones. Let's take a simple family of basic blocks: $DEL_{1,r}^{crash}$, the delivered predicate of the model with at most one crash, at round r.

Definition 2.5 (At most 1 crash at round r)**.** $DEL_{1,r}^{crash} \triangleq$

$$\left\{ c \text{ a delivered collection} \;\middle|\; \exists \Sigma \subseteq \Pi : \begin{array}{c} |\Sigma| \geq n-1 \\ \wedge \quad \forall j \in \Pi \left(\begin{array}{lll} & \forall r' \in [1, r[: & c(r', j) = \Pi \\ \wedge & & c(r, j) \supseteq \Sigma \\ \wedge & \forall r' > r : & c(r', j) = \Sigma \end{array} \right) \end{array} \right\}.$$

In these predicates, before round r, every process receives every message. And at round r a crash might happen, which means that processes only receive messages from a subset Σ of Π of size $|\Pi| - 1$ from round $r + 1$ onwards. The subtlety at round r is that the crashed process (the only one in $\Pi \setminus \Sigma$) might crash while sending messages, and thus might send messages to some processes and not others.

Another fundamental predicate is the total one: the predicate containing a single collection c_{total}, the one where every process receives every message at every round.

Definition 2.6 (Total delivered predicate)**.** $DEL^{total} \triangleq \{c_{total}\}$, where c_{total} is the collection defined by $\forall r > 0, \forall p \in \Pi : c(r, p) = \Pi$.

Using these building blocks, many interesting and important delivered predicates can be built, as shown in Table 2.1. For example, let's take DEL_1^{crash}, the predicate with at most one crash. If a crash happens, it happens at one specific round r. DEL_1^{crash} is thus a disjunction for all values of r of the predicate with at most one crash at round r; that is, the union of $DEL_{1,r}^{crash}$ for all r.

2.3 Delivered In, Heard-Of Out

After defining delivered predicates and discussing how to find and/or build them, the next step is to study the heard-of predicates that can be implemented over a given delivered one. This is the red part of Figure 2.3, which works between two mathematical abstraction, and so is more formal.

Description	Expression
At most 1 crash	$DEL_1^{crash} = \bigcup\limits_{i=1}^{\infty} DEL_{1,i}^{crash}$
At most F crashes	$DEL_F^{crash} = \bigotimes\limits_{j=1}^{F} DEL_1^{crash}$
At most 1 crash, which will restart	$DEL_1^{recover} = DEL_1^{crash} \rightsquigarrow DEL^{total}$
At most F crashes, which will restart	$DEL_F^{recover} = \bigotimes\limits_{j=1}^{F} DEL_1^{recover}$
At most 1 crash, which can restart	$DEL_1^{canrecover} = DEL_1^{recover} \cup DEL_1^{crash}$
At most F crashes, which can restart	$DEL_F^{canrecover} = \bigotimes\limits_{j=1}^{F} DEL_1^{canrecover}$
No bound on crashes and restart, with only 1 crash at a time	$DEL_1^{recovery} = (DEL_1^{crash})^{\omega}$
No bound on crashes and restart, with max F crashes at a time	$DEL_F^{recovery} = \bigotimes\limits_{j=1}^{F} DEL_1^{recovery}$
At most 1 crash, after round r	$DEL_{1,\geq r}^{crash} = \bigcup\limits_{i=r}^{\infty} DEL_{1,i}^{crash}$
At most F crashes, after round r	$DEL_{F,\geq r}^{crash} = \bigcup\limits_{i=r}^{\infty} DEL_{F,i}^{crash}$
At most F crashes with no more than one per round	$DEL_F^{crash\neq} = \bigcup\limits_{i_1 \neq i_2 \neq \ldots \neq i_F} \bigotimes\limits_{j=1}^{F} DEL_{1,i_j}^{crash}$

Table 2.1: A list of delivered predicate built using our operations

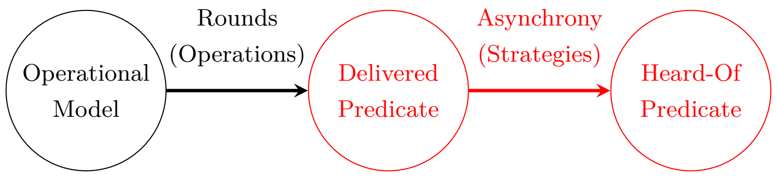


Figure 2.3: From delivered predicates to heard-of predicates

2.3.1 Executions

Executions of an algorithm are the bread-and-butter of the theory of distributed computing: they describe the behavior of systems formally enough to be analysed. Here, as we study the system-level implementation of the Heard-Of model, the executions we consider are not executions of an algorithm solving a distributed computing problem, but the executions of the implementation of a specific heard-of predicate. Hence these executions only track emissions, receptions and changes of rounds. Because the content of each message is not important, and we care about which messages will be received on time, the emissions are implicit: as long as a process changed round $r - 1$ times, it sent its messages for round r (which messages will depend on the delivered collection used, as explained in a few paragraphs). As for the local state of each process during this implementation, it contains a local round counter and the set of received messages.

The last thing that is missing here is the implementation algorithm: the rule that specifies when to change rounds. This is what I call a strategy, and will be defined shortly. For now, let's define executions as formal objects (sequences of events) that satisfy some basic constraints on the ordering of events. We then constrains them by requiring the delivery of exactly the messages from some delivered collection. The introduction of strategies constrains them some more, so that the executions allowed are the executions of an implementation of rounds using this strategy.

To summarize, executions are infinite sequences of events, either delivery of messages ($deliver(r, p, q)$, which represents the delivery at q of the message from p tagged with r), change to the next round for some process j ($next_j$), or a deadlock ($stop$). An execution must satisfy three rules: no message is delivered before it is sent, no message is delivered twice, and once there is a *stop*, the rest of the sequence can only be *stop*.

Definition 2.7 (Execution)**.** Let Π be a set of n processes. Let the set of transitions $T = \{next_j \mid j \in \Pi\} \cup \{deliver(r, k, j) \mid r \in \mathbb{N}^* \wedge k, j \in \Pi\} \cup \{stop\}$. Then, $t \in T^\omega$ is an **execution** $\triangleq$

- **(Delivery after sending)**
  $\forall i \in \mathbb{N} : t[i] = deliver(r, k, j) \implies \mathbf{card}(\{l \in [0, i[\mid t[l] = next_k\}) \geq r - 1$
- **(Unique delivery)**
  $\forall \langle r, k, j\rangle \in (\mathbb{N}^* \times \Pi \times \Pi) : \mathbf{card}(\{i \in \mathbb{N} \mid t[i] = deliver(r, k, j)\}) \leq 1$
- **(Once stopped, forever stopped)**
  $\forall i \in \mathbb{N} : t[i] = stop \implies \forall j \geq i : t[j] = stop$

Then we can start constraining executions by saying that they must be executions of a given delivered collection c. What does that mean? That if p changes round at least $r-1$ times in the execution, then it sends all messages tagged with r to processes q satisfying $p \in c(r, q)$, and these messages are delivered in the execution. Moreover, every delivery must be of such a message. For the executions of a delivered predicate, those are exactly the executions of the collections of the predicate.

Definition 2.8 (Execution of a delivered collection (and of a predicate))**.** Let c be a delivered collection. Then, $execs(c)$, the **executions of** $c \triangleq$

$$\left\{ t \text{ an execution} \;\middle|\; \begin{array}{l} \forall \langle r, k, j\rangle \in \mathbb{N}^* \times \Pi \times \Pi : \\ \quad (k \in c(r, j) \wedge \mathbf{card}(\{i \in \mathbb{N} \mid t[i] = next_k\}) \geq r - 1) \\ \quad \iff \\ \quad (\exists i \in \mathbb{N} : t[i] = deliver(r, k, j)) \end{array} \right\}$$

For DEL a delivered predicate, we write $execs(DEL) = \bigcup\limits_{c \in DEL} execs(c)$.

Definition 2.7 above casts behavior in term of changes to the system – the deliveries and changes of round. A dual perspective interprets behavior as the sequence of successive states of the system. In a distributed system, such states are the product of local states.

The local state of a process is the pair of its current round and all the received messages up to this point. Notably, such a local state doesn't contain the identity of the process. This is both because we never need this identity, and because not dealing with it allow an easier comparison of local states, since two distinct processes can have the same local state. A message is represented by a pair $\langle round, sender\rangle$ (instead of triplet like in deliver events, because the receiver is implicit – it's the process whose local state we're looking at). For a state q, $q(r)$ is the set of peers from which the process (with state q) has received a message tagged with round r.

Definition 2.9 (Local State)**.** Let $Q = \mathbb{N}^* \times \mathcal{P}(\mathbb{N}^* \times \Pi)$. Then $q \in Q$ is a **local state**.

For $q = \langle r, mes \rangle$, we write $q.round$ for r, $q.mes$ for mes and $\forall r' > 0 : q(r) \triangleq \{k \in \Pi \mid \langle r', k \rangle \in q.mes\}$.

Let t be an execution, $p \in \Pi$ and $i \in \mathbb{N}$. Then the local state of p in t after the prefix of t of size i is $q^t_p[i] \triangleq \langle |\{l < i \mid t[l] = next_p\}| + 1, \{\langle r, k \rangle \mid \exists l < i : t[l] = deliver(r, k, p)\}\rangle)$

Notice that such executions do not allow process to "jump" from say round 5 to round 9 without passing by the rounds in-between. Indeed, the Heard-Of model doesn't let processes decide when to change rounds: processes specify only which messages to send depending on the state, and what is the next state depending on the current state and the received messages. So it makes sense for a system-level implementation of heard-of predicates to do the same.

2.3.2 Strategies, and How to Build Them

Executions represent the link between delivered predicates and heard-of predicates: an execution of a delivered collection where all processes change round infinitely often defines a heard-of collection. This is done by looking, for each round r and process p, at the set of processes such that p received their message tagged by r when the round counter at p was $\leq r$.

But we cannot just take all executions of a delivered predicate $execs(DEL)$, and take the heard-of predicate defined by the heard-of collection for each execution. This is because not all of these executions ensure an infinite number of rounds for each process. As an example, for a delivered collection c, the execution where all messages from round 1 are delivered according to c (whatever the order) and then only *stop* transitions happen forever is an execution of c. Yet it blocks all processes at round 1 forever.

Strategies[6] solve the problem: they constrain executions, and for a strategy with the right property, the resulting executions always contain an infinite number of rounds for each process. A strategy is a set of states from which a process is allowed to change round. It can also be seen as a boolean function from the local states to $\{true, false\}$. It captures rules like "wait for at least F messages from the current round", or "wait for these specific messages".

Definition 2.10 (Strategy)**.** $f \subseteq Q$ is a **strategy**.

Strategies as defined above are predicates on states[7]. This makes them incredibly expressive; on the other hand, this expressivity creates difficulty in reasoning about them. To address this problem, we define families of strategies. Intuitively, strategies in a same family depend on a specific part of the state – for example the messages of the current round. Equality of these parts of the state defines an equivalence relation; the strategies of a family are strategies such that if a state q is in the strategy, then all states in the equivalence class of q are in the strategy.

Definition 2.11 (Families of strategies)**.** Let $\approx : Q \times Q \to bool$. The family of strategies defined by $\approx$, $family(\approx) \triangleq \{f \text{ a strategy} \mid \forall q_1, q_2 \in \Pi : q_1 \approx q_2 \implies (q_1 \in f \iff q_2 \in f)\}$

[6]The name comes from a previous iteration of this research, where the formalization was based on games. In the present context, a strategy is merely a rule to say when changing round is allowed, depending on the local state of the process.

[7]One limiting case is for the strategy to be empty – the predicate being just $False$. This strategy is clearly useless, and will be weeded out by the constraint of validity from Definition 2.13.

With strategies in the picture, we can constrain executions some more: the executions of a strategy on a delivered predicate are the executions of the implementation of rounds using the strategy, implementation on the operational model corresponding to the delivered predicate.

Thus let's next define the executions of a strategy. The intuition is simple: every change of rounds (an event $next_k$ for k a process) happens only if the local state of the process is in the strategy; and there is a fairness assumption that ensures that if the local state of some process k is continuously often in the strategy, then it will eventually change round (have a $next_k$ event) [8].

A subtlety hidden in this obvious intuition is that the "check" for changing round (whether the local state is in the strategy) doesn't necessarily happen at each reception; it can happen at any point. This captures an asynchronous assumption where processes do not decide when they are executed.

Definition 2.12 (Executions of a Strategy)**.** Let f be a strategy and t an execution. Then t is an **execution of** $f \triangleq t$ satisfies:

- **(All nexts allowed)** $\forall i \in \mathbb{N}, \forall p \in \Pi : (t[i] = next_p \implies q^t_p[i] \in f)$
- **(Fairness)** $\forall p \in \Pi : \mathbf{card}(\{i \in \mathbb{N} \mid t[i] = next_p\}) < \infty \implies \mathbf{card}(\{i \in \mathbb{N} \mid q^t_p[i] \notin f\}) = \infty$

For a delivered predicate DEL, we note $execs_f(DEL) \triangleq \{t \mid t \textit{ an execution of } f \wedge t \in execs(DEL)\}$.

The first property is obvious: processes only change round (the *next* transition) when their local state is in the strategy. Fairness requires more explanations: it ensures that the only way for a process p to be blocked at a round r is for p's local state to not be in f an infinite number of times. So if the local state of p is eventually always in f, the local state is outside of f only a finite number of times, and the execution must contain another $next_p$.

Going back to strategies, not all of them are equally valuable. In general, strategies with executions where processes are blocked forever at some round are less useful (to implement infinite sequences of rounds) than strategies without such executions. The validity of a strategy captures the absence of such an infinite wait.

Definition 2.13 (Validity)**.** An execution t is **valid** $\triangleq \forall p \in \Pi, \forall N > 0, \exists i \geq N : t[i] = next_p$.

Let DEL a delivered predicate and f a strategy. Then f is a **valid strategy** for DEL $\triangleq \forall t \in execs_f(DEL) : t$ is a valid execution.

Finally, analogously to how we can build complex predicates through operations, we can also build complex strategies through similar operations:

Definition 2.14 (Operations on strategies)**.** Let f_1, f_2 be two strategies.

- Their **union** $f_1 \cup f_2$.
- Their **combination** $f_1 \bigotimes f_2 \triangleq \{q_1 \bigotimes q_2 \mid q_1 \in f_1 \wedge q_2 \in f_2 \wedge q_1.round = q_2.round\}$, where for q_1 and q_2 at the same round r, $q_1 \bigotimes q_2 \triangleq \langle r, \{\langle r', k\rangle \mid r' > 0 \wedge k \in q_1(r') \cap q_2(r')\}\rangle$

[8]This is a weak fairness assumption, which requires a constant availability of the transition after some point to ensure it will happen. This is to be contrasted with a strong fairness assumption, which requires the availability infinitely often after some point to ensure that the transition will happen.

- Their **succession** $f_1 \rightsquigarrow f_2 \triangleq f_1 \cup f_2 \cup \{q_1 \rightsquigarrow q_2 \mid q_1 \in f_1 \land q_2 \in f_2\}$ where $q_1 \rightsquigarrow q_2 \triangleq \langle q_1.round + q_2.round, \{\langle r,k\rangle \mid r > 0 \land \begin{pmatrix} k \in q_1(r) & \text{if } r \leq q_1.round \\ k \in q_2(r - q_1.round) & \text{if } r > q_1.round \end{pmatrix}\}\rangle$
- The **repetition** of f_1, $f_1^\omega \triangleq \{q_1 \rightsquigarrow q_2 \rightsquigarrow ... \rightsquigarrow q_k \mid k \geq 1 \land q_1, q_2, ..., q_k \in f_1\}$.

The intuition behind these operations is analogous to the ones for predicates:

- The union of two strategies is equivalent to an OR of the two conditions. For example, the union of waiting for at least $n-F$ messages and waiting for all messages but the ones from p_1 gives a strategy that accepts change of round when more than $n-F$ messages are received or when all messages except the one from p_1 are received.
- The combination of two strategies takes all intersections of local states in the first strategy and local states in the second. For example, combining the strategy that waits at least $n-1$ messages for the current round with itself will wait for at least $n-2$ messages.
- For succession, the states accepted are those where messages up to some round correspond to an accepted state of the first strategy, and messages from this round up correspond to an accepted state of the second strategy.
- Repetition is the next logical step after succession: instead of following one strategy with another, the same strategy is repeated again and again.

2.3.3 Extracting Heard-Of Collections of Executions

If an execution is valid, then all processes go through an infinite number of rounds. That is, it captures the execution of a system-level implementation of rounds where no process blocks forever at some round. It thus makes sense to speak of the heard-of collection implemented by this execution: at the end of round r for process p, the messages from round r that were received by p define the heard-of set for r and p.

Definition 2.15 (Heard-Of Collections Generated by Executions and Heard-Of Predicate Generated by Strategies)**.** Let t be a valid execution. Then the **heard-of collection generated by** t, $h_t \triangleq$

$$\forall r \in \mathbb{N}^*, \forall p \in \Pi : h_t(r,p) = \left\{ k \in \Pi \;\middle|\; \exists i \in \mathbb{N} : \begin{pmatrix} & q_p^t[i].round = r \\ \land & t[i] = next_p \\ \land & \langle r,k\rangle \in q_p^t[i].mes \end{pmatrix} \right\}$$

Let DEL be a delivered predicate, and f be a valid strategy for DEL. Then the **heard-of predicate generated by** f **on** $DEL \triangleq \mathcal{HO}_f(DEL) \triangleq \{h_t \mid t \in execs_f(DEL)\}$.

Every valid strategy thus generates a heard-of predicate from the delivered predicate.

The way to go from a delivered predicate to a heard-of one is to design a valid strategy for the former that generates the latter. But that still does not answer the original question: among all the heard-of predicates one can generate from a given delivered predicate, which one should be considered as the characterization of the delivered predicate (and of the corresponding operational model)?

A heard-of predicate generated from a delivered one is an over-approximation of the latter. But to be able to solve as much problems as possible, as many messages as possible should be received on time. The characterizing heard-of predicate is thus the smallest such over-approximation of the delivered predicate, if it exists.

Let's formalize this intuition by defining a partial order on valid strategies for a delivered predicate, capturing the implication of the generated heard-of predicates (the inclusion when considered as sets). One strategy dominates another if the heard-of collections it generates are also generated by the other. Dominating strategies are then the greatest elements for this order. By definition of domination, all dominating strategies generate the same dominating heard-of predicate, which characterizes the delivered predicate.

Definition 2.16 (Domination Order, Dominating Strategy and Dominating Predicate)**.** Let *DEL* be a delivered predicate and let f and f' be two valid strategies for *DEL*. Then, f **dominates** f' for *DEL*, written $f' \prec_{DEL} f \triangleq HO_{f'}(DEL) \supseteq HO_f(DEL)$.

A greatest element for $\prec_{DEL}$, if it exists, is called a **dominating strategy** for *DEL*. Given such a strategy f, the **dominating predicate** for *DEL* is then $HO_f(DEL)$.

2.3.4 Two simple executions

In the following sections, we prove properties about dominating strategies, their invariance by the operations, and the heard-of predicate that they generate. These proofs rely on reasoning by contradiction: assume the theorem or lemma is false, and derive a contradiction. These contradictions take the form of proving that a valid strategy has an invalid execution; constructing specific executions is therefore the main technique in these proofs.

This section thus introduces two patterns for constructing executions: one from a delivered collection and a strategy, the other from a heard-of collection.

To do so, let's fix *ord* as some function taking a set and returning an ordered sequence of its elements – the specific ordering doesn't matter. This will be used to ensure the uniqueness of the executions, but the order has no impact on the results.

The standard execution extracts an execution from a delivered collection. It follows a loop around a simple pattern: deliver all the messages that were sent according to the delivered collection, then change round for all the processes which are allowed to do so by f.

About notation, $\prod_{i \in \mathbb{N}^*} x_i$ is the infinite concatenation $x_1.x_2.x_3....$

Definition 2.17 (Standard Execution of Strategy on Execution)**.** Let c be a delivered collection, and f be a strategy.

The **standard execution of** f **on** c is $st(f,c) \triangleq \prod_{r \in \mathbb{N}^*} dels_r.changes_r$, where

- $dels_1 \triangleq ord(\{deliver(1,p,q) \mid p \in c(1,q)\})$, the ordered sequence of deliveries for messages from round 1 that will be delivered eventually according to c.
- $changes_1 \triangleq ord(\{next_q \mid \langle 1, \{(1,p) \mid p \in c(1,q)\}\rangle \in f\})$, the ordered sequence of next transitions for processes for which the state resulting from the deliveries in $dels_1$ is in f.
- $\forall r > 1 : dels_r \triangleq ord(\{deliver(round^p_r, p, q) \mid next_p \in changes_{r-1} \land p \in c(round^p_r, q)\})$, with $round^p_r \triangleq 1 + \sum_{r' \in [1,r[} |\{next_p\} \cap changes_{r'}|$.

  This is the ordered sequence of deliveries of messages from processes that changed round during $changes_{r-1}$.

- $\forall r > 1 : changes_r \triangleq$
  $$\begin{cases} ord(\{next_q \mid \langle round^q_r, \{(r',p) \mid deliver(r',p,q) \in \bigcup\limits_{r'' \in [1,r]} dels_{r''}\}\rangle \in f\}) & \text{if it is not empty} \\ ord(\{stop\}) & \text{otherwise} \end{cases},$$
  with $round^p_r \triangleq 1 + \sum\limits_{r' \in [1,r[} |\{next_p\} \cap changes_{r'}|$.

  This is the ordered sequence of $next_p$ for processes such that their state after the delivered of $dels_r$ is in f.

The main property of a standard execution of f on c is that it is both an execution of f and an execution of c.

Lemma 2.18 (Correctness of Standard Execution)**.** *Let c be a delivered collection and f be a strategy. Then $st(f,c) \in execs_f(c)$.*

Proof. Let us first show that $st(f,c)$ is an execution by showing each point of Definition 2.7.

- **(Delivered after sending)** By Definition 2.17 of the standard execution there are $r-1$ transitions $next_p$ before the messages of p sent at round r are delivered.

- **(Delivered only once)** If a message sent at round r by p is delivered, we know from the previous point that p reaches round r before the delivery. Let r' such that $changes_{r'}$ contains the $r-1$ ith $next_p$ of $st(f,c)$.

  Then the message is delivered in $dels_{r'+1}$ by Definition 2.17 of the standard execution. And for all $r'' > r'+1$, if there are deliveries from p in $dels_{r''}$, this entails that $next_p \in changes_{r''-1}$, and thus that p is not anymore at round r. And by definition of $dels_{r''}$, it only delivers messages sent at the current rounds of processes.

  We conclude that there is only one delivery of the message.

- **(Once stopped, forever stopped)** By Definition 2.17 of the standard execution, if $changes_r$ contains only $stop$, this means that f does not allow any process to change rounds with the currently received messages. And by definition of del_r, it only delivers messages from processes that changed round in $changes_{r-1}$.

  Hence if there is some smallest r_{stop} such that $changes_{r_{stop}} = \{stop\}$, then $dels_{r_{stop}+1} = \emptyset$, which means the local states of processes do not change, and thus $changes_{r_{stop}+1} = \{stop\}$.

  Thus by induction, if there is some $stop$ in $st(f,c)$, the rest of the execution contains only $stop$ transitions.

Hence $st(f,c)$ is an execution.

Next, let's show that $st(f,c)$ is an execution of c. By Definition 2.8, this means the messages delivered are exactly those from c, for processes that reached the round where they send the message. By Definition 2.17 of the standard execution, all deliveries are from messages in c. And by Definition 2.17 of the standard execution, if p reaches round r, there is a smallest $r' > 0$ such that $round^p_{r'} = r$. This means that $dels_{r'}$ contains the deliveries of all the messages (r,p,q) such that $p \in c(r,q)$.

Hence $st(f,c)$ is an execution of c.

Finally, we show that $st(f,c)$ is an execution of f. We check the two conditions of Definition 2.12:

- **(All Nexts Allowed)** By Definition 2.17 of the standard executions, changes of round only occur when the local state is in f.
- **(Fairness)** If some process is blocked forever at some round, this means by Definition 2.17 of the standard execution that its local state was not in f for an infinite number of $changes_r$, and so for an infinite number of times.

We conclude that $st(f, c) \in execs_f(c)$. □

The other construction takes a heard-of collection, and gives an execution generating it on the total delivered collection. This canonical execution follows a simple pattern: at each round, deliver all messages from the heard-of sets of this round, and also all message undelivered from the previous round (the ones that were not in the heard-of sets of the previous round).

Definition 2.19 (Canonical Execution of a Heard-Of Collection)**.** Let *ho* be a heard-of collection.

The **canonical execution of** *ho* is $can(ho) \triangleq \prod_{r \in \mathbb{N}^*} dels_r.changes_r$, where

- $dels_1 \triangleq ord(\{deliver(1, p, q) \mid p \in ho(1, q)\})$, the ordered sequence of deliveries that happen at round 1 in h.
- $\forall r > 1 : dels_r \triangleq ord(\{deliver(r, p, q) \mid p \in ho(r, q)\} \cup \{deliver(r-1, p, q) \mid p \notin ho(r-1, q)\})$, the ordered sequence of deliverives that happen at round r in h.
- $\forall r > 0 : changes_r \triangleq ord(\{next_p \mid p \in \Pi\})$, the ordered sequence of next transitions, one for each process.

This canonical execution is an execution of any delivered predicate containing c_{total}, the collection where every message is delivered. Having this collection in a delivered predicate ensures that although faults might happen, they are not forced to do so.

Lemma 2.20 (Canonical Execution is an Execution of Total Collection)**.** *Let ho be a heard-of collection Then, the canonical execution $can(ho)$ of ho is an execution of c_{total}.*

Proof. First, $can(ho)$ is an execution by Definition 2.7 since it satisfies the three conditions:

- **(Delivered only once)** Every sent message is delivered either during the round it was sent or during the next one, and thus delivered only once.
- **(Delivered after sending)** Every message from round r is delivered after either $r-1$ or r $next_p$ transitions for the sender p, which means at round r or $r+1$. Hence the message is delivered after being sent.
- **(Once stopped, forever stopped)** No process stops, so the last condition for executions is trivially satisfied.

Furthermore, for each process p and round r, all the messages from p at round r are delivered in $can(ho)$, either at round r or at round $r+1$. Since the total collection is the collection where every message is delivered by Definition 2.6, this entails that $can(ho)$ is an execution of the total Delivered collection by Definition 2.8, and thus an execution of *DEL*. □

Lastly, the whole point of the canonical execution of *ho* is that it generates *ho*.

Lemma 2.21 (Canonical Execution Generates its Heard-Of Collection)**.** *Let ho be a heard-of collection. Then* $h_{can(ho)} = ho$.

Proof. By Definition 2.15,

$$\forall r > 0, \forall j \in \Pi : h_{can(ho)}(r,j) = \left\{ p \in \Pi \,\middle|\, \exists i \in \mathbb{N} : \begin{pmatrix} & q_j^{can(ho)}[i].round = r \\ \wedge & can(ho)[i] = next_j \\ \wedge & \langle r,p \rangle \in q_j^{can(ho)}[i].mes \end{pmatrix} \right\}.$$

By Definition 2.19 of a canonical execution, every $change_k$ of $can(ho)$ contains a $next_p$ for every $p \in \Pi$. So the r-th $next_p$ of $can(ho)$ happens at $change_r$ for every $p \in \Pi$. And also by Definition 2.19, the messages from round r delivered to j before $change_r$ are those in $dels_r$; that is, exactly the messages from processes in $ho(r,j)$. So for every process j, its r-th $next_j$ in $can(ho)$ happens at $change_r$, after it received only the messages from round r in $ho(r,j)$.

We conclude that $\forall r > 0, \forall j \in \Pi : h_{can(ho)}(r,j) = \left\{ p \in \Pi \,\middle|\, \exists i \in \mathbb{N} : \begin{pmatrix} & q_j^{can(ho)}[i].round = r \\ \wedge & can(ho)[i] = next_j \\ \wedge & \langle r,p \rangle \in q_j^{can(ho)}[i].mes \end{pmatrix} \right\} = ho(r,j)$.

Therefore $h_{can(ho)} = ho$. □

2.3.5 A Complete Example: At Most F Crashes

To provide a more concrete example, let's turn back to DEL_F^{crash} from Definition 2.2, the message-passing model with asynchronous and reliable communication, and at most F permanent crashes. We now give a dominating strategy for this predicate, as well as compute its heard-of predicate.

The folklore strategy for this model is to wait for at least $n - F$ messages before allowing the change of round.

Definition 2.22 (waiting for $n - F$ messages)**.** The strategy to wait for $n - F$ messages is $f_{n-F} \triangleq \{q \in Q \mid |\{k \in \Pi \mid \langle q.round, k \rangle \in q.mes\}| \geq n - F\}$

To see why this strategy is used in the literature, simply remark that at least $n-F$ messages must be delivered to each process at each round. Thus, waiting for that many messages ensures that no process is ever blocked. However, waiting for more will result in processes blocking forever if F crashes occur. Rephrased with the concepts introduced above, f_{n-F} is a valid strategy for DEL_F^{crash}.

Lemma 2.23 (Validity of f_{n-F})**.** f_{n-F} *is valid for* DEL_F^{crash}.

Proof. We proceed by contradiction: **Assume** f_{n-F} is invalid for DEL_F^{crash}. Thus by Definition 2.23, there exists an invalid $t \in execs_{f_{n-F}}(DEL_F^{crash})$. By Definition 2.23 of validity, $\exists p \in \Pi, \exists N, \forall i \geq N : t[i] \neq next_p$: there is a smallest round r such that some process j stays blocked at r forever in t. Because t is an execution of f, Definition 2.12 entails that infinitely many local states of j must be not in f_{n-F}; if it was not the case, the fairness condition would force the execution to contain another $next_j$.

Let also c_t be a delivered collection of DEL_F^{crash} such that $t \in execs(c)$.

We know by Definition 2.2 of DEL_F^{crash} that $|c_t(r,j)| \geq n - F$. The minimality of r and the fact that $t \in execs(c)$ then ensure by Definition 2.8 that all the messages in this delivered set are delivered at some point in t. By Definition 2.22 of f_{n-F}, the local state of j is then in f_{n-F} from this point on. By the fairness constraint of Definition 2.12, this **contradicts** the fact that there is never another $next_j$ in the suffix of t.

We conclude that f_{n-F} is valid for DEL_F^{crash}. □

The obvious next step is to prove that this strategy is dominating the predicate. But the proof given here depends on the heard-of predicate generated by f_{n-F}, which is thus computed first. This heard-of predicate was first given by Charron-Bost and Schiper [37] as a characterization of the asynchronous model with reliable communication and at most F crashes. The intuition behind it is that even in the absence of crashes, we can make all the processes change round by delivering any set of at least $n - F$ messages to them.

Theorem 2.24 (Heard-Of Characterization of f_{n-F})**.**
Let HO_F be the heard-of predicate defined in Table 1.1. Then $\mathcal{HO}_{f_{n-F}}(DEL_F^{crash}) = HO_F$.

Proof.
- ($\subseteq$). Let $ho \in \mathcal{HO}_{f_{n-F}}(DEL_F^{crash})$ and $t \in execs_{f_{n-F}}(DEL_F^{crash})$ an execution of f_{n-F} generating ho. By Definition 2.12 of the executions of f_{n-F}, processes change round (a $next_k$ event) only when their local state is in f_{n-F}. This means by Definition 2.22 that the local state q of processes satisfy $|\{k \in \Pi \mid \langle q.round, k\rangle \in q.mes\}| \geq n - F\}$: the process received at least $n - F$ messages tagged with the current value of its round counter. This in turns implies by Definition 2.15 of the heard-of collection of an execution that $\forall r \in \mathbb{N}^*, \forall j \in \Pi : |ho(r, j)| \geq n - F$.

- ($\supseteq$). Let ho a heard-of collection over Π such that $\forall r \in \mathbb{N}, \forall j \in \Pi : |ho(r, j)| \geq n-F$. Let t be the canonical execution of ho; since DEL_F^{crash} contains the total collection, t is an execution of DEL_F^{crash} by Lemma 2.20. To prove that t is also an execution of f_{n-F}, we proceed by contradiction: **assume** it is not an execution of f_{n-F}. By Definition 2.12, the problem stems either from breaking fairness or from some $next_p$ for some p at a point where the local state of p is not in f_{n-F}. Since by Definition 2.19 of a canonical execution, $\forall p \in \Pi : next_p$ appears an infinite number of times, the only possibility left is the second one: there is some $p \in \Pi$ and some $next_p$ transition in t that happens while the local state of p is not in f_{n-F}. Let r be the smallest round where this happens, and j the process to which it happens. By Definition 2.19 of a canonical execution, j received all the messages from $ho(r, j)$ in t before the problematic $next_j$. And $|ho(r, j)| \geq n - F$ by hypothesis. By Definition 2.22 of f_{n-F}, the local state of j is in f_{n-F} from this point on. By the fairness constraint of Definition 2.12, this **contradicts** the fact that j cannot change round at this point in t while t is an execution of f_{n-F}.

We conclude that $ho \in \mathcal{HO}_{f_{n-F}}(DEL_F^{crash})$. □

Finally, we want to vindicate the folklore intuition about this strategy: that it is optimal in some sense. Intuitively, waiting for more than $n - F$ messages per round means risking waiting forever, and waiting for less is wasteful. Our domination order captures this concept of optimality: we show that f_{n-F} is indeed a dominating strategy for DEL_F^{crash}. Therefore, $\mathcal{HO}_{f_{n-F}}(DEL_F^{crash})$ is the dominating predicate for DEL_F^{crash}.

Theorem 2.25 (f_{n-F} Dominates DEL_F^{crash})**.** *f_{n-F} dominates DEL_F^{crash}.*

Proof. Let f be a valid strategy for DEL_F^{crash}; the theorem follows by Definition 2.16 from proving that $f \prec_{DEL_F^{crash}} f_{n-F}$ – that is $\mathcal{HO}_{f_{n-F}}(DEL_F^{crash}) \subseteq \mathcal{HO}_f(DEL_F^{crash})$. We do that now.

Let $ho \in \mathcal{HO}_{f_{n-F}}(DEL_F^{crash})$, and let t be the canonical execution of ho. Since DEL_F^{crash} contains the total collection, t is an execution of DEL_F^{crash} by Lemma 2.20. We only need to prove that it is also an execution of f to conclude by Lemma 2.21 that f generates ho, and thus that the inequality above and the theorem hold.

We do so by contradiction. **Assume** t is not an execution of f. By Definition 2.12, it is either because the fairness condition is broken or because some $next_p$ for some process p happens when the local state of p is not in f. Since Definition 2.19 of canonical executions implies that t contains an infinite number of $next_p$ for every process $p \in \Pi$, the problem must come from some $next_j$ done by a process j when j's local state is not in f. Let r be the first round where this happens. At the point of the forbidden $next_j$, by Definition 2.19 of a canonical execution, j has received all the messages from previous rounds, and all the messages from $ho(r,j)$. Then $ho \in \mathcal{HO}_{f_{n-F}}(DEL_F^{crash})$ implies that $ho \in HO_F$ by Theorem 2.24. It then follows from the definition of HO_F in Table 1.1 that $ho(r,j)$ contains at least $n - F$ processes.

Let c_{block} be the delivered collection where all the processes from which j did not receive a message at the problematic *next* in t stop sending messages from round r onwards. So for rounds $> r$, the delivered set of any process k is always $ho(r,j)$. $\forall r' > 0, \forall k \in \Pi : c_{block}(r',k) = \begin{cases} \Pi & \textit{if } r' < r \\ ho(r,j) & \textit{otherwise} \end{cases}$

This is a delivered collection of DEL_F^{crash} by Definition 2.2: processes that stop sending messages never do again, and at most F processes do so because $ho(r,j)$ contains at least $n - F$ processes by the reasoning above.

Let $t' = st(f, c_{block})$ be the standard execution of f on c_{block}. Lemma 2.18 entails than t' is an execution of f on c_{block}. Since r is the smallest round in t with a wrong $next_j$, for all rounds $< r$ the local state of j is enough for f to allow the change of round. Thus by Definition 2.17 of standard executions, all $changes_k$ for $k < r$ contain a next transition for all processes in t'. By the same definition, all $dels_k$ for $k \leq r$ of t' contain the same deliveries for each process than the deliveres for j in the $dels_k$ of t. Hence in t', all the processes reach round r, all get the same state as j in t at round r, and thus they all block at this round, which means the suffix of t' is made of *stop* only. Hence t' is invalid, and so is f.

This **contradicts** the hypothesis that f is valid; we thus conclude that $ho \in \mathcal{HO}_f(DEL_F^{crash})$.

Therefore f_{n-F} dominates f by Definition 2.16, where f is any valid strategy for DEL_F^{crash}, which means that f_{n-F} dominates DEL_F^{crash} by Definition 2.16. $\square$

This means that when confronted with a model captured by DEL_F^{crash}, there is no point in remembering messages from past rounds – and messages from future rounds are simply buffered. Intuitively, messages from past rounds are of no use in detecting crashes in the current round. As for messages from future rounds, they actually serve to detect that a process has not crashed when sending its messages from the current round. For this to actually change the heard-of predicate, it would mean that some heard-of collection would be impossible to generate when using this future information. But this is not the case, as there is always an execution where no message from future rounds are delivered early (the canonical execution).

2.4 Carpe Diem: Oblivious Strategies Living in the Moment

Let's now turn to more general results about delivered predicates and strategies. Because of the generality of strategies, considering them all brings many issues in proving domination. Yet there exist interesting classes of strategies on which results can be derived.

Our first such class is the class of oblivious strategies: they depend only on the received messages from the current round. For example, f_{n-F} is an oblivious strategy, as it counts messages from the current round. Despite their apparent simplicity, some oblivious strategies dominate non-trivial delivered predicates, as in the case of f_{n-F} and $DEL_F^{crashes}$.

2.4.1 Definition and Expressiveness Results

Oblivious strategies are a family of strategies in the sense of Definition 2.11 – where the equivalence relation between the local states compare only the messages received for the current round.

Definition 2.26 (Oblivious Strategies and $Nexts_f$)**.** Let *obliv* be the function such that $\forall q \in Q : obliv(q) = \{k \in \Pi \mid \langle q.round, k\rangle \in q.mes\}$. Let $\approx_{obliv}$ the equivalence relation defined by $q_1 \approx_{obliv} q_2 \triangleq obliv(q_1) = obliv(q_2)$. The family of **oblivious strategies** $\triangleq family(\approx_{obliv})$.

For f an oblivious strategy, let $Nexts_f \triangleq \{obliv(q) \mid q \in f\}$. It uniquely defines f.

Thus an oblivious strategy reduces to a collection of sets, the sets of processes from which receiving a message in the current round is enough to change round. The strategy allows the change of round if, and only if, the processes heard at the current round form a set in this collection.

This provides a simple necessary condition on such a strategy f to be valid: its $Nexts_f$ set must contains all the delivered sets from the corresponding delivered predicate. If it does not, an execution would exists where the messages received at some round r by some process p are exactly this delivered set, which would block forever the process p and make the strategy invalid.

This simple necessary condition also proves sufficient.

Lemma 2.27 (Necessary and Sufficient Condition for Validity of an Oblivious Strategy)**.** *Let DEL be a delivered predicate and f be an oblivious strategy. Then f is valid for DEL $\iff$ $f \supseteq \{q \mid \exists c \in DEL, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r,p)\}$.*

Proof.

- ($\Rightarrow$) Let f be valid for *DEL*. We show by contradiction that f contains all local states q such that $\exists c \in DEL, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r,p)$. **Assume** there is some q_{block} for which it is not the case: then $\exists c \in DEL, r > 0$ and $j \in \Pi$ such that $obliv(q_{block}) = c(r,j)$ and $q \notin f$. By Definition 2.26, this means that for every q such that $obliv(q) = obliv(q_{block}) = c(r,j)$, we have $q \notin f$.

  Let $t = st(f,c)$ be the standard execution of f on c. This is a execution of f on c by Lemma 2.18. The sought contradiction is reached by proving that t is invalid. To do so, we split according to two cases: the first is the case where there is a blocking process before round r, and the other is the case where there is not blocking process before round r. This last case then uses the hypothesis on $c(r,j)$ to show that all processes block at r.

  - During one of the first $r - 1$ iterations of t, there is some process which cannot change round. Let r' be the smallest iteration where it happens, and k be a process unable to change round at the r'-ith iteration. By minimality of r', all the processes arrive at round r' in t; by Definition 2.17 of the standard execution, all messages for k from round r' are delivered before the $change_{r'}$ part of the iteration. Let q be the local state of k at the start of $change_{r'}$ in the r'-ith iteration, and let q' be any local state of k afterward. The above tells us that as long as $q'.round = q.round$, we have $obliv(q) = obliv(q')$ and thus $q' \notin f$. Therefore, k can never change round while at round r'.

    We conclude that t is invalid by Definition 2.23.
  - For the first $r - 1$ iterations, all the processes change round. Thus every one arrives at round r in the $r - 1$-ith iteration. By Definition 2.17 of the standard execution, all messages from the round are delivered before the $change_r$ part of the r-th iteration. Thus j is in a local state q at the $change_r$ part of the r-ith iteration such that $obliv(q) = c(r, j) = obliv(q_{block})$. By hypothesis, this means $q \notin f$ and thus that j cannot change round. Let q' be any local state of j afterward. The above tells us that as long as $q'.round = r$, we have $obliv(q) = obliv(q') = c(r, j)$ and thus $q' \notin f$. Therefore, j can never change round while at round r.

    Here too, t is invalid by Definition 2.23.

  Either way, we reach a **contradiction** with the validity of f. Therefore $f \supseteq \{q \mid \exists c \in DEL, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r, p)\}$.

- ($\Leftarrow$) Let DEL and f such that $f \supseteq \{q \mid \exists c \in DEL, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r, p)\}$. We show by contradiction that f is valid.

  Assume the contrary: there is some $t \in execs_f(DEL)$ which is invalid. Thus by Definition 2.23 of validity, there are some process blocked at a round forever in t. Let r be the smallest such round, and j be a process blocked at round r in t. By minimality of r, all the processes arrive at round r. By Definition 2.8 of an execution of DEL, there is a $c \in DEL$ such that t is an execution of c. Which means by Definition 2.8 of an execution of a collection that eventually all the messages from $c(r, j)$ are delivered.

  From this point on, every local state q of j satisfies $obliv(q) = c(r, j)$; thus we have $q \in f$ by hypothesis on f. Then the fairness condition of executions of f from Definition 2.12 imposes that j does change round at some point. We conclude that j is not blocked at round r in t, which **contradicts** the hypothesis that it is blocked forever at round r in t.

□

What happens when taking the oblivious strategy satisfying exactly this condition for validity? This results in a strategy dominating the other oblivious ones. It follows from the fact that this strategy waits for the minimum sets required to be valid; hence the name of minimal oblivious strategy.

Definition 2.28 (Minimal Oblivious Strategy)**.** Let DEL be a delivered predicate. The **minimal oblivious strategy** for DEL is $f_{min} \triangleq$
$\{q \mid \exists c \in DEL, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r, p)\}$.

Lemma 2.29 (Domination of Minimal Oblivious Strategy)**.** *Let DEL be a delivered predicate and f_{min} be its minimal oblivious strategy. Then f_{min} is a dominating oblivious strategy for DEL.*

Proof. First, f_{min} is valid for DEL by application of Lemma 2.27. Next, we take another oblivious strategy f, which is valid for DEL. Lemma 2.27 now gives us that $f_{min} \subseteq f$. Hence, when f_{min} allows a change of round, so does f. This entails by Definition 2.12 that all executions of f_{min} on DEL are also executions of f on DEL, and thus by Definition 2.15 that $HO_{f_{min}}(DEL) \subseteq HO_f(DEL)$.

We thus conclude from Definition 2.16 that f_{min} dominates any valid oblivious strategy for DEL. □

These strategies thus guarantee that every delivered predicate has a strategy dominating the oblivious ones, by giving a means to build it. Of course, a formal definition is not the same as a constructive definition, which motivates the study of minimal strategies through the operations, and their relations to the operations on the corresponding predicates.

2.4.2 Building oblivious strategies

One fundamental property of minimal oblivious strategies is their nice behaviour under the proposed operations (union, combination, succession and repetition). That is, they give minimal oblivious strategies of resulting delivered predicates. Although this holds for all operations, succession and repetition are not useful here, as the succession of two minimal oblivious strategies is equal to their union, and the repetition of a minimal oblivious strategy is equal to the strategy itself.

The first operation to study is therefore union. The minimal oblivious strategy of $DEL_1 \cup DEL_2$ and $DEL_1 \rightsquigarrow DEL_2$ is the same, as shown it the next theorem, and thus it's the union of the minimal oblivious strategies of DEL_1 and DEL_2.

Theorem 2.30 (Minimal Oblivious Strategy for Union and Succession)**.** *Let DEL_1, DEL_2 be two delivered predicates, f_1 and f_2 the minimal oblivious strategies for, respectively, DEL_1 and DEL_2. Then $f_1 \cup f_2$ is the minimal oblivious strategy for $DEL_1 \cup DEL_2$ and $DEL_1 \rightsquigarrow DEL_2$.*

Proof idea. Structurally, every proof in this subsection amounts to showing equality between the strategies resulting from the operations and the minimal oblivious strategy for the delivered predicate.

For a union, the messages that can be received at each round are the messages that can be received at each round in the first predicate or in the second. This is also true for succession. Given that f_1 and f_2 are the minimal oblivious strategies of DEL_1 and DEL_2, they accept exactly the states where the messages received from the current round are in a delivered set of DEL_1 or a delivered set of DEL_2. And thus $f_1 \cup f_2$ is the minimal oblivious strategy for $DEL_1 \cup DEL_2$ and $DEL_1 \rightsquigarrow DEL_2$. □

Proof. We first show that the minimal oblivious strategies of $DEL_1 \cup DEL_2$ and $DEL_1 \rightsquigarrow DEL_2$ are equal. By Definition 2.14, we thus need to prove that $\{q \mid \exists c \in DEL_1 \cup DEL_2, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r,p)\} = \{q \mid \exists c \in DEL_1 \rightsquigarrow DEL_2, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r,p)\}$.

- $(\subseteq)$ Let q such that $\exists c \in DEL_1 \cup DEL_2, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r,p)$.

  - If $c \in DEL_1$, then we take $c_2 \in DEL_2$, and take $c' = c[1, r].c_2$. Since by Definition 2.4 $c' \in c \rightsquigarrow c_2$, we have $c' \in DEL_1 \rightsquigarrow DEL_2$. And by definition of c', $c'(r, p) = c(r, p)$.

    We thus have c', p and r showing that $obliv(q) = c'(r, p)$, and thus q is in the set on the right.
  - If $c \in DEL_2$, then $c \in DEL_1 \rightsquigarrow DEL_2$ by Definition 2.4. We thus have c, p and r showing that $obliv(q) = c(r, p)$, and thus q is in the set on the right.

- ($\supseteq$) Let q such that $\exists c \in DEL_1 \rightsquigarrow DEL_2, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r, p)$.
  - If $c \in DEL_2$, then $c \in DEL_1 \cup DEL_2$ by Definition 2.4. We thus have c, p and r showing that $obliv(q) = c(r, p)$, and thus q is in the set on the left.
  - If $c \notin DEL_2$, there exist $c_1 \in DEL_1, c_2 \in DEL_2$ and $r' > 0$ such that $c = c_1[1, r'].c_2$ by Definition 2.4.
    * If $r \leq r'$, then by definition of c, we have $c(r, p) = c_1(r, p)$. We thus have c_1, p and r showing that $obliv(q) = c_1(r, p)$, and thus q is in the set on the left.
    * If $r > r'$, then $c(r, p) = c_2(r - r', p)$. We thus have c_2, p and $(r - r')$ showing that $obliv(q) = c_2(r - r', p)$, and thus q is in the set on the left.

We now show that $f_1 \cup f_2 = \{q \mid \exists c \in DEL_1 \cup DEL_2, \exists p \in \Pi, \exists r > 0 : obliv(q) = c(r, p)\}$, which allows us to conclude by Definition 2.28 that $f_1 \cup f_2$ is the minimal oblivious strategy for $DEL_1 \cup DEL_2$.

- Let $q \in f_1 \cup f_2$. We fix $q \in f_1$ (the case $q \in f_2$ is symmetric).

  Then because f_1 is the minimal oblivious strategy of DEL_1, by application of Lemma 2.27, $\exists c_1 \in DEL_1, \exists p \in \Pi, \exists r > 0$ such that $c_1(r, p) = obliv(q)$. Also $c_1 \in DEL_1 \subseteq DEL_1 \cup DEL_2$ by Definition 2.4. We thus have c_1, p and r showing that q is in the minimal oblivious strategy for $DEL_1 \cup DEL_2$.

- Let q such that $\exists c \in DEL_1 \cup DEL_2, \exists p \in \Pi, \exists r > 0 : c(r, p) = obliv(q)$. By Definition 2.4 of operations on strategies, and specifically union, c must be in DEL_1 or in $c \in DEL_2$; we fix $c \in DEL_1$ (the case DEL_2 is symmetric).

  Then Definition 2.28 gives us that q is in the minimal oblivious strategy of DEL_1, that is f_1. We conclude that $q \in f_1 \cup f_2$.

$\square$

For the same reason that succession is indistinguishable from union, repetition is indistinguishable from the original predicate: the delivered sets are the same, because every collection of the repetition is built from prefixes of collections of the original predicate. Thus the minimal oblivious strategy for a repetition is in fact the same strategy as the minimal oblivious strategy of the original predicate.

Theorem 2.31 (Minimal Oblivious Strategy for Repetition)**.** *Let DEL be a delivered predicate, and f be its minimal oblivious strategy. Then f is the minimal oblivious strategy for DEL^{ω}.*

Proof. We show that $f = \{q \mid \exists c \in DEL^{\omega}, \exists p \in \Pi, \exists r > 0 : \mathit{obliv}(q) = c(r,p)\}$, which allows us to conclude by Definition 2.28 that f is the minimal oblivious strategy for DEL^{ω}.

- ($\subseteq$) Let $q \in f$. By minimality of f for DEL, $\exists c \in DEL, \exists p \in \Pi, \exists r > 0 : \mathit{obliv}(q) = c(r,p)$.

  We take $c' \in DEL^{\omega}$ such that $c_1 = c$ and $r_2 = r$; the other c_i and r_i don't matter for the proof. By Definition 2.4 of operations on predicates, and specifically repetition, we get $c'(r,p) = c(r,p) = \mathit{obliv}(q)$. We have c', p and r showing that q is in the minimal oblivious strategy of DEL^{ω}.

- ($\supseteq$) Let q such that $\exists c \in DEL^{\omega}, \exists p \in \Pi, \exists r > 0 : \mathit{obliv}(q) = c(r,p)$. By Definition 2.4 of operations on predicates, and specifically repetition, there are $c_i \in DEL$ and $0 < r_i < r_{i+1}$ such that $r \in [r_i + 1, r_{i+1}]$ and $c(r,p) = c_i(r - r_i, p)$.

  We have found c_i, p and $(r - r_i)$ showing that q is in the minimal oblivious strategy for DEL. And since f is the minimal oblivious strategy for DEL, we get $q \in f$.

□

Combination is different from other operations, as combining collections is done round by round. Since oblivious strategies do not depend on the round, the combination of two oblivious strategies will accept the combination of any two states accepted, that is, it will accept any intersection of the delivered set of received messages from the current round in the first state and the delivered set of received messages from the current round in the second state. Yet when taking the combination of two predicates, maybe the collections are such that these two delivered sets used in the intersection above never happen at the same round, and thus never appear in the combination of collections.

To ensure that every intersection of pairs of delivered sets, one from a collection from each predicate, happens in the combination of predicates, we add an assumption: the symmetry of the predicate over processes and over rounds. This means that for any delivered set D of the predicate, for any round and any process, there is a collection of the predicate where D is the delivered set for some round and some process.

Definition 2.32 (Round Symmetric DEL)**.** Let DEL be a delivered predicate. DEL is **round symmetric** $\triangleq \forall c \in DEL, \forall r > 0, \forall p \in \Pi, \forall r' > 0, \forall q \in \Pi, \exists c' \in DEL : c(r,p) = c'(r',q)$

Theorem 2.33 (Minimal Oblivious Strategy for Combination)**.** *Let DEL_1, DEL_2 be two round symmetric delivered predicates, f_1 and f_2 the minimal oblivious strategies for, respectively, DEL_1 and DEL_2. Then $f_1 \bigotimes f_2$ is the minimal oblivious strategy for $DEL_1 \bigotimes DEL_2$.*

Proof idea. The oblivious states of $DEL_1 \bigotimes DEL_2$ are the combination of an oblivious state of DEL_1 and of one of DEL_2 at the same round, for the same process. Thanks to round symmetry, this translates into the combination of any oblivious state of DEL_1 with any oblivious state of DEL_2. Since f_1 and f_2 are the minimal oblivious strategy, they both accept exactly the oblivious states of DEL_1 and DEL_2 respectively. Thus, $f_1 \bigotimes f_2$ accept all the combinations of oblivious states of DEL_1 and DEL_2, and thus is the minimal oblivious strategy of $DEL_1 \bigotimes DEL_2$. □

Proof. We show that $f_1 \bigotimes f_2 = \{q \mid \exists c \in DEL_1 \bigotimes DEL_2, \exists p \in \Pi, \exists r > 0 : \mathit{obliv}(q) = c(r,p)\}$, which allows us to apply Lemma 2.28 to show that $f_1 \bigotimes f_2$ is the minimal oblivious strategy of $DEL_1 \bigotimes DEL_2$.

- Let $q \in f_1 \bigotimes f_1$. Then $\exists q_1 \in f_1, \exists q_2 \in f_2$ such that $q = q_1 \bigotimes q_2$. This also means that $q_1.round = q_2.round = q.round$.

  By minimality of f_1 and f_2, $\exists c_1 \in DEL_1, \exists p_1 \in \Pi, \exists r_1 > 0 : c_1(r_1, p_1) = obliv(q_1)$ and $\exists c_2 \in DEL_2, \exists p_2 \in \Pi, \exists r_2 > 0 : c_2(r_2, p_2) = obliv(q_2)$.

  Moreover, by Definition 2.32 of round symmetry, the hypothesis on DEL_2 ensures that $\exists c'_2 \in DEL_2 : c'_2(r_1, p_1) = c_2(r_2, p_2)$.

  We take $c = c_1 \bigotimes c'_2$. $obliv(q) = obliv(q_1) \cap obliv(q_2) = c_1(r_1, p_1) \cap c_2(r_2, p_2) = c_1(r_1, p_1) \cap c'_2(r_1, p_1) = c(r_1, p_1)$.

  We have c, p_1 and r_1 showing that q is in the minimal oblivious strategy for $DEL_1 \bigotimes DEL_2$.

- Let q such that $\exists c \in DEL_1 \bigotimes DEL_2, \exists p \in \Pi, \exists r > 0 : c(r, p) = obliv(q)$. By Definition 2.4 of operations on predicates, and specifically of combination, $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2 : c = c_1 \bigotimes c_2$.

  We take q_1 such that $q_1.round = r, obliv(q_1) = c_1(r, p)$ and $\forall r' \neq r : q_1(r') = q(r')$; we also take q_2 such that $q_2.round = r, obliv(q_2) = c_2(r, p)$ and $\forall r' \neq r : q_2(r') = q(r')$.

  Then $q = q_1 \bigotimes q_2$. And by Definition 2.28 of minimal oblivious strategies, f_1 and f_2 being respectively the minimal oblivious strategies of DEL_1 and DEL_2 imply that $q_1 \in f_1$ and $q_2 \in f_2$.

  We conclude that $q \in f_1 \bigotimes f_2$.

□

This subsection shows that as long as predicates are built from simple building blocks with known minimal oblivious strategy, the minimal oblivious strategy of the result can be explicitly constructed.

2.4.3 Computing Heard-Of Predicates

Once the minimal oblivious strategy computed, the next step is to extract the heard-of predicate for this strategy: the smallest predicate (all its collections are contained in the other predicates) generated by an oblivious strategy for this delivered predicate. This ends up being simple: it is the product of all delivered sets.

Definition 2.34 (Heard-Of Product)**.** Let $S \subseteq \mathcal{P}(\Pi)$. The **heard-of product generated by S**, $HOProd(S) \triangleq \{ha\ heard\text{-}of\ collection \mid \forall p \in \Pi, \forall r > 0 : h(r, p) \in S\ \}$.

Here is the intuition: defining a heard-of collection requires, for each round and each process, the corresponding heard-of set. A heard-of product is then the set of all collections that have heard-of sets from the set given as argument. So the totalsq heard-of predicate (containing only the total collection) is the heard-of product of the set Π. And $\mathcal{HO}_F$ is the heard-of product of all subsets of Π of size $\geq n - F$.

This segues into the following lemma, which links the $Nexts_f$ of some valid oblivious strategy and the heard-of predicate for this strategy: the predicate is the heard-of product of the $Nexts_f$.

Lemma 2.35 (Heard-Of Predicate of an Oblivious Strategy)**.** *Let DEL be a delivered predicate containing c_{total} and let f be a valid oblivious strategy for DEL. Then $\mathcal{HO}_f(DEL) = HOProd(Nexts_f)$.*

Proof.

- ($\subseteq$) To prove this first direction, we show that the heard-of sets of any collection in $\mathcal{HO}_f(DEL)$ are in $Nexts_f$. This then entails that $\mathcal{HO}_f(DEL) = HOProd(Nexts_f)$.

  By Definition 2.15 of the heard-of collection of an execution, every heard-of set contains the set of messages from the current round that was already received at the $next_p$ transition where the process p changed round. By Definition 2.12 of the executions of a strategy, such a $next_p$ transition can only happen if the local state of the process p is in f. And by Definition 2.26 of oblivious strategies, f contains exactly the states such that the messages received from the current round form a set in $Nexts_f$.

  Therefore the heard-of set of any collection generated by f on a collection of DEL are necessarily in $Nexts_f$.

- ($\supseteq$) Let ho be a heard-of collection such that $\forall r > 0, \forall j \in \Pi : ho(r, j) \in Nexts_f$. Let t be the canonical execution of ho. It is an execution by Lemma 2.20. By Definition 2.12 it is also an execution of f because at each round, processes receive a set of messages in $Nexts_f$. This entails that the local states are in f, by Definition 2.26 of oblivious strategies. Hence t is an execution of f on DEL. Since $t = can(ho)$, Lemma 2.21 implies that $h_t = ho$.

  We conclude that $ho \in \mathcal{HO}_f(DEL)$.

□

Thanks to this characterization, the heard-of predicate generated by the minimal strategies for the operations is computed in terms of the heard-of predicate generated by the original minimal strategies.

Theorem 2.36 (Heard-Of Predicate of Minimal Oblivious Strategies)**.** *Let DEL, DEL_1, DEL_2 be delivered predicates containing c_{total}. Let f, f_1, f_2 be their respective minimal oblivious strategies. Then:*

- $\mathcal{HO}_{f_1 \cup f_2}(DEL_1 \cup DEL_2) = \mathcal{HO}_{f_1 \cup f_2}(DEL_1 \rightsquigarrow DEL_2) = HOProd(Nexts_{f_1} \cup Nexts_{f_2})$
- *If DEL_1 or DEL_2 are round symmetric, then:* $\mathcal{HO}_{f_1 \bigotimes f_2}(DEL_1 \bigotimes DEL_2) = HOProd(\{n_1 \cap n_2 \mid n_1 \in Nexts_{f_1} \wedge n_2 \in Nexts_{f_2}\})$.
- $\mathcal{HO}_f(DEL^\omega) = \mathcal{HO}_f(DEL)$.

Proof. Obviously, we want to apply Lemma 2.35. Then we first need to show that the DELs contain c_{total}.

- By hypothesis, DEL_1 and DEL_2 contain c_{total}. Then $DEL_1 \cup DEL_2$ trivially contains it too by Definition 2.4 of operations on predicates.
- By hypothesis, DEL_1 and DEL_2 contain c_{total}. Then $DEL_1 \bigotimes DEL_2$ contains $c_{total} \bigotimes c_{total} = c_{total}$ by Definition 2.4 of operations on predicates.
- By hypothesis, DEL_1 and DEL_2 contain c_{total}. Then $DEL_1 \rightsquigarrow DEL_2 \supseteq DEL_2$ contains it too by Definition 2.4 of operations on predicates.

- By hypothesis, DEL contains c_{total}. We can recreate c_{total} by taking all $c_i = c_{total}$ and whichever r_i. Thus, DEL^{ω} contains c_{total} by Definition 2.4 of operations on predicates.

Next, the strategies $f_1 \cup f_2$, $f_1 \bigotimes f_2$ and f are the respective minimal oblivious strategies by Theorem 2.30, Theorem 2.33 and Theorem 2.31. They are also valid by Theorem 2.27.

Lastly, we need to show that the $Nexts_f$ for the strategies corresponds to the generating sets in the theorem.

- We show $Nexts_{f_1 \cup f_2} = Nexts_{f_1} \cup Nexts_{f_2}$, and thus that $HOProd(Nexts_{f_1 \cup f_2}) = HOProd(Nexts_{f_1} \cup Nexts_{f_2})$
  - ($\subseteq$) Let $n \in Nexts_{f_1 \cup f_2}$. Then $\exists q \in f_1 \cup f_2 : obliv(q) = n$. By Definition 2.4 of operations on predicates, and specifically union, $q \in f_1$ or $q \in f_2$. We fix $q \in f_1$ (the case $q \in f_2$ is symmetric). Then $n \in Nexts_{f_1}$ by Definition 2.26 of oblivious strategies.

    We conclude that $n \in Nexts_{f_1} \cup Nexts_{f_2}$.
  - ($\supseteq$) Let $n \in Nexts_{f_1} \cup Nexts_{f_2}$. We fix $n \in Nexts_{f_1}$ (as always, the other case is symmetric). Then $\exists q \in f_1 : obliv(q) = n$. As $q \in f_1$ implies $q \in f_1 \cup f_2$, we conclude that $n \in Nexts_{f_1 \cup f_2}$ by Definition 2.26 of oblivious strategies.
- We show $Nexts_{f_1 \bigotimes f_2} = \{n_1 \cap n_2 \mid n_1 \in Nexts_{f_1} \wedge n_2 \in Nexts_{f_2}\}$, and thus that $HOProd(Nexts_{f_1 \bigotimes f_2}) = HOProd(\{n_1 \cap n_2 \mid n_1 \in Nexts_{f_1} \wedge n_2 \in Nexts_{f_2}\})$.
  - ($\subseteq$) Let $n \in Nexts_{f_1 \bigotimes f_2}$. Then $\exists q \in f_1 \bigotimes f_2 : obliv(q) = n$. By Definition 2.4 of operations on predicates, and specifically of combination, $\exists q_1 \in f_1, \exists q_2 \in f_2 : q_1.round = q_2.round = q.round \wedge q = q_1 \bigotimes q_2$. This means $n = obliv(q) = obliv(q_1) \cap obliv(q_2)$.

    We conclude that $n \in \{n_1 \cap n_2 \mid n_1 \in Nexts_{f_1} \wedge n_2 \in Nexts_{f_2}\}$ by Definition 2.26 of oblivious strategies.
  - ($\supseteq$) Let $n \in \{n_1 \cap n_2 \mid n_1 \in Nexts_{f_1} \wedge n_2 \in Nexts_{f_2}\}$. Then $\exists n_1 \in Nexts_{f_1}, \exists n_2 \in Nexts_{f_2} : n = n_1 \cap n_2$. By Definition 2.26 of oblivious strategies, and because f_1 and f_2 are oblivious strategies, we can find $q_1 \in f_1$ such that $obliv(q_1) = n_1$, $q_2 \in f_2$ such that $obliv(q_2) = n_2$, and $q_1.round = q_2.round$.

    Then $q = q_1 \bigotimes q_2$ is a state of $f_1 \bigotimes f_2$. We have $obliv(q) = n_1 \cap n_2 = n$.

    We conclude that $n \in Nexts_{f_1 \bigotimes f_2}$ by Definition 2.26 of oblivious strategies.
- Trivially, $Nexts_f = Nexts_f$.

□

2.4.4 When Oblivious is Enough

Finally, the value of oblivious strategies depends on which delivered predicates have such a dominating strategy. DEL_F^{crash} does; let's now extend this result to delivered predicates satisfying the so-called common round property. This condition captures the fact that given any delivered set D, one can build, for any $r > 0$, a delivered collection where processes receive all the messages up to round r, and then they share D as their delivered set in round r. As a limit case, the predicate also contains the total collection.

Definition 2.37 (Common Round Property)**.** Let *DEL* be a Delivered Predicate. *DEL* has the **common round property** $\triangleq$

- **(Total collection)** *DEL* contains the total collection c_{total}.
- **(Common round)** $\forall c \in DEL, \forall r > 0, \forall j \in \Pi, \forall r' > 0, \exists c' \in DEL, \forall p \in \Pi : (\forall r'' < r' : c'(r'', p) = \Pi \wedge c'(r', p) = c(r, j))$

What the common round property captures is what makes DEL_F^{crash} be dominated by an oblivious strategy: if one process j might block at round r even after receiving all messages from round r in some collection c of DEL, and all messages from rounds $< r$, then there is a collection and an execution where all processes block in the same way. The collection ensures that the delivered collection gives each process the same delivered sets (Π) for rounds $< r$, and $c(r, j)$ at round r. The execution is the standard execution of this collection, that puts every process at round r in the same blocking state than j. Hence a deadlock. The conclusion is that any valid strategy should allow to change round when all messages from previous rounds are received, and the messages received for the current round form a delivered set from a collection of DEL.

Applying this reasoning to the canonical executions of heard-of collections from $HO_{f_{min}}(DEL)$ yields that the canonical executions are executions of any valid strategy for DEL (not only oblivious ones), and thus that $\forall f$ a valid strategy for DEL, $HO_{f_{min}}(DEL) \subseteq HO_f(DEL)$. That is to say, DEL is dominated by an oblivious strategy.

Theorem 2.38 (Sufficient Condition of Oblivious Domination)**.** *Let DEL be a delivered predicate satisfying the common round property. Then, there is an oblivious strategy which dominates DEL.*

Proof. Let f_{min} be the minimal oblivious strategy for *DEL* – it dominates the oblivious strategies for *DEL* by Lemma 2.29. We now prove that f_{min} dominates *DEL*. This amount to showing that for f' be a valid strategy for *DEL*, we have $f' \prec_{DEL} f_{min}$, that is $\mathcal{HO}_{f_{min}}(DEL) \subseteq \mathcal{HO}_{f'}(DEL)$. Let $ho \in \mathcal{HO}_{f_{min}}(DEL)$ and t be the canonical execution of *ho*. We show that t is an execution of f', which entails by Lemma 2.21 that $ho \in \mathcal{HO}_{f'}(DEL)$.

By Definition 2.37 of the common round property, *DEL* contains c_{total}. And by Lemma 2.20, t is an execution of c_{total}, and thus an execution of *DEL*. We now prove by contradiction it is also an execution of f' on *DEL*. **Assume** it is not. By Definition 2.12 of the executions of a strategy, the problem comes either from breaking fairness or from some $next_j$ for some process j at a point where the local state of j is not in f'. Since for every $j \in \Pi : next_j$ happens an infinite number of times in t by Definition 2.19 of a canonical execution, the only possibility left is the second one: some $next_j$ in t is done while the local state of j is not in f'. There thus exists a smallest r such that some proces j is not allowed by f' to change round when $next_j$ is played at round r in t.

Lemma 2.35 yields that $\mathcal{HO}_{f_{min}}(DEL) = HOProd(Nexts_{f_{min}})$. And by Definition 2.28 of minimal oblivious strategies, $Nexts_{f_{min}} = \{c(r', p) \mid c \in DEL \wedge r' > 0 \wedge p \in \Pi\}$. Thus $\exists c \in DEL, \exists r' > 0, \exists p \in \Pi : ho(r, j) = c(r', p)$. Next, by Definition 2.37 of the common round property, *DEL* satisfying this property allows us to build $c_{block} \in DEL$ such that $\forall r'' < r, \forall k \in \Pi : c_{block}(r'', k) = \Pi$ and $\forall k \in \Pi : c_{block}(r, k) = c(r', j) = ho(r, j)$.

Finally, we build $t_{block} = st(f', c_{block})$ the standard execution of f' on c_{block}. By Lemma 2.18, we know t_{block} is an execution of f on c_{block}. We then show that it is invalid by examining the two possibilities.

- During one of the first $r-1$ iterations of t_{block}, there is some process that cannot change round. Let r' be the smallest iteration where it happens, and k be a process unable to change round at the r'-ith iteration.

  By minimality of r', all the processes arrive at round r', and by definition of c_{block} they all receive the same messages as k before $changes_{r'}$. But that means every process has the same local state as k. Thus, all the processes are blocked at round r', there are no more next or deliveries, and t_{block} is therefore invalid by Definition 2.23 of validity.

- For the first $r-1$ iterations, every process changes round. Thus, everyone arrives at round r. By Definition 2.17 of the standard execution, all messages from round r are delivered before the $change_r$ section. The definition of c_{block} also ensures that every process received the same messages, that is all the messages from round $< r$ and all the messages from $ho(r, j)$. These are exactly the messages received by j in t at round r. But by hypothesis, j is blocked in this state in t. We thus deduce that all the processes are blocked at round r in t_{block}, and thus that it is an invalid execution by Definition 2.23 of validity.

Either way, we deduce that f' is invalid, which is a **contradiction**.

We conclude that t is an execution of f' on DEL. Lemma 2.21 therefore implies that $ho \in \mathcal{HO}_{f'}(DEL)$.

This entails that $\mathcal{HO}_{f_{min}}(DEL) \subseteq \mathcal{HO}_{f'}(DEL)$, and thus that $f' \prec_{DEL} f_{min}$. We conclude taht f_{min} dominates DEL by Definition 2.16. □

What's more, this condition is maintained by the operations. Hence any predicate built from ones satisfying this condition will still be dominated by an oblivious strategy.

Theorem 2.39 (Domination by Oblivious for Operations)**.** *Let DEL, DEL_1, DEL_2 be delivered predicates satisfying the common round property. Then $DEL_1 \cup DEL_2$, $DEL_1 \bigotimes DEL_2$, $DEL_1 \rightsquigarrow DEL_2$, DEL^ω also satisfy the common round property.*

Proof. Thanks to Theorem 2.38, we only have to show that the condition is maintained by the operations; the domination by an oblivious strategy follows directly from the Theorem 2.38.

The fact that c_{total} is still in the results of the operations was already shown in the proof of Theorem 2.36.

Then we show the invariance of the common round part.

- Let $c \in DEL_1 \cup DEL_2$. Thus $c \in DEL_1$ or $c \in DEL_2$. We fix $c \in DEL_1$ (the other case is symmetric). Then for $p \in \Pi, r > 0$ and $r' > 0$, we get a $c' \in DEL_1$ satisfying the condition of Definiton 2.37 by the hypothesis that DEL_1 satisfies the common round property. And since $DEL_1 \subseteq DEL_1 \cup DEL_2$, we get $c' \in DEL_1 \cup DEL_2$.

  We conclude that the condition still holds for $DEL_1 \cup DEL_2$.

- Let $c \in DEL_1 \bigotimes DEL_2$. Then $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2 : c = c_1 \bigotimes c_2$. For $p \in \Pi, r > 0$ and $r' > 0$, our hypothesis on DEL_1 and DEL_2 ensures that there are $c'_1 \in DEL_1$ satisfying the condition of Definition 2.37 for c_1 and $c'_2 \in DEL_2$ satisfying the condition of Definition 2.37 for c_2.

  We argue that $c' = c'_1 \bigotimes c'_2$ satisfies the condition of Definition 2.37 for c. Indeed, $\forall r'' < r', \forall q \in \Pi : c(r'', q) = c'_1(r'', q) \bigotimes c'_2(r'', q) = \Pi$ and $\forall q \in \Pi : c(r', q) = c'_1(r', q) \bigotimes c'_2(r', q) = c_1(r, p) \bigotimes c_2(r, p) = c(r, p)$.

We conclude that the condition of Definition 2.37 still holds for $DEL_1 \bigotimes DEL_2$.

- Let $c \in DEL_1 \rightsquigarrow DEL_2$. Since if $c \in DEL_2$ the condition of Definition 2.37 trivially holds by hypothesis, we study the case where succession actually happens. Hence, $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2, \exists r_{change} > 0 : c = c_1[1, r_{change}].c_2$. For $p \in \Pi, r > 0$ and $r' > 0$, we separate two cases.
  - if $r \leq r_{change}$, then our hypothesis on DEL_1 ensures that there is $c'_1 \in DEL_1$ satisfying the condition of Definition 2.37 for c_1. We argue that $c' = c'_1[1, r'].c_2 \in DEL_1 \rightsquigarrow DEL_2$ satisfies the condition of Definition 2.37 for c.

    Indeed, $\forall r'' < r', \forall q \in \Pi : c'(r'', q) = c'_1(r'', q) = \Pi$, and $\forall q \in \Pi : c'(r', q) = c_1(r, p) = c(r, p)$
  - if $r > r_{change}$, then our hypothesis on DEL_2 ensures that there is $c'_2 \in DEL_2$ satisfying the condition of Definition 2.37 for c_2 at p and $r - r_{change}$. That is, $c'_2[1, r' - 1] = c_{total}[1, r' - 1] \wedge \forall q \in \Pi : c'_2(r', q) = c_2(r - r_{change}, p)$ We argue that $c' = c'_2 \in DEL_1 \rightsquigarrow DEL_2$ satisfies the condition of Definition 2.37 for c.

    Indeed, $\forall r'' < r', \forall q \in \Pi : c'_2(r'', q) = \Pi$, and $\forall q \in \Pi : c'_2(r', q) = c_2(r - r_change, p) = c(r, p)$

  We conclude that the condition of Definition 2.37 still holds for $DEL_1 \rightsquigarrow DEL_2$.

- Let $c \in DEL^{\omega}$. Let (c_i) and (r_i) be the collections and indices defining c. We take $p \in \Pi, r > 0$ and $r' > 0$. Let $i > 0$ be the integer such that $r \in [r_i + 1, r_{i+1}]$. By hypothesis on DEL, There is $c'_i \in DEL$ satisfying the condition of Definition 2.37 for c_i at p and $r - r_i$. That is, $c'_i[1, r' - 1] = c_{total}[1, r' - 1] \wedge \forall q \in \Pi : c'_i(r', q) = c_i(r - r_i, p)$.

  We argue that $c'_i \in DEL$ satisfies the condition of Definition 2.37 for c. Indeed, $\forall r'' \leq r', \forall q \in \Pi$, we have: $c'_i(r'', q) = \Pi$ and $\forall q \in \ \Pi : c'_i(r', q) = c_i(r - r_i, p) = c(r, p)$.

  We conclude that the condition of Definition 2.37 still holds for DEL^{ω}.

□

Therefore, as long as the initial building blocks satisfying the common round property, so does the result of the operations – and thus the latter is dominated by its minimal oblivious strategy. A strategy that can be computed easily from the previous results in this section.

2.5 No Future: Conservative to the End

The class of considered strategies now broadens, by considering past rounds and the round number in addition to the present round. This is a generalization of oblivious strategies, that trades simplicity for expressivity, while retaining a nice structure.

2.5.1 Definition and Expressiveness Results

Definition 2.40 (Conservative Strategy)**.** Let *cons* be the function such that $\forall q \in Q$, $cons(q) \triangleq \langle q.round, \{\langle r, k\rangle \in q.mes \mid r \leq q.round\}\rangle$. Let $\approx_{cons}$ the equivalence relation defined by $q_1 \approx_{cons} q_2 \triangleq cons(q_1) = cons(q_2)$. The family of **conservative strategies** $\triangleq family(\approx_{cons})$.

We write $Nexts^C_f \triangleq \{cons(q) \mid q \in f\}$ for the set of conservative states in f. This uniquely defines f.

In analogy with the case of oblivious strategies, there is an intuitive necessary and sufficient condition for such a strategy to be valid for a given delivered predicate.

Lemma 2.41 (Necessary and Sufficient Condition for Validity of a Conservative Strategy)**.** *Let DEL be a delivered predicate and f be a conservative strategy. Then f is valid for DEL $\iff$ $f \supseteq \{q \in Q \mid \exists c \in DEL, \exists p \in \Pi, \forall r \leq q.round : q(r) = c(r,p)\}$.*

Proof.
- ($\Rightarrow$) Let f be valid for DEL. We show by contradiction that it satisfies the right-hand side of the above equivalence. **Assume** there is q_{block} a local state such that $\exists c \in DEL, \exists r > 0, \exists j \in \Pi : cons(q_{block}) = \langle r, \{\langle r', k\rangle \mid r' \leq r \wedge k \in c(r',j)\}\rangle$ and $q \notin f$. By Definition 2.40, this means that for every q such that $cons(q) = cons(q_{block} = \langle r, \{\langle r', k\rangle \mid r' \leq r \wedge k \in c(r',j)\}\rangle$, $q \notin f$.

  Let $t = st(f,c)$ be the standard execution of f on c. This is an execution of f on t by Lemma 2.18. The sought contradiction is reached by proving that t is invalid for DEL, and thus f is invalid for DEL too. To do so, we split according to two cases: the first is the case where there is a blocking process before round r, and the other uses the hypothesis on the prefix of c for j up to round r.

  – During one of the first $r-1$ iterations of t, there is some process which cannot change round. Let r' be the smallest iteration of the canonical execution where it happens, and k be a process unable to change round at the r'-ith iteration.

    By minimality of r', all the processes arrive at round r' in t; by Definition 2.17 of the standard execution, all messages for k from all rounds up to r' are delivered before the *change* part of the iteration. Let q the local state of k at the start of $change_{r'}$, and let q' be any local state of k afterward. The above tells us that as long as $q'.round = q.round$, we have $cons(q) = cons(q')$ and thus $q' \notin f$. Therefore, k can never change round while at round r'.

    We conclude that t is invalid for DEL by Definition 2.23.

  – For the first $r-1$ iterations, all the processes change round. Thus, every one arrives at round r in the $r-1$-th iteration. By Definition 2.17 of the standard execution, all messages from rounds up to r are delivered before the $change_r$ part of the r-th iteration. Thus j is in a local state q at the $change_r$ part of the r-ith iteration such that $cons(q) = \langle r, \{\langle r', k\rangle \mid r' \leq r \wedge k \in c(r',j)\}\rangle = cons(q_{block})$. By hypothesis, this means $q \notin f$ thus that j cannot change round. Let q' be any local state of j afterward. The above tells us that as long as $q'.round = q.round$, we have $cons(q) = cons(q')$ and thus $q' \notin f$. Therefore, j can never change round while at round r.

    Here too, t is invalid for DEL by Definition 2.23.

  Either way, we reach a **contradiction** with the validity of f for DEL.

- ($\Leftarrow$) Let DEL and f such that $\forall c \in DEL, \langle r, \{\langle r', k\rangle \mid r' \leq r \wedge k \in c(r',j)\}\rangle \in Nexts^C_f$. We show by contradiction that f is valid for DEL.

  Assume the contrary: there is some $t \in execs_f(DEL)$ that is invalid for DEL. Thus, there are some process blocked at a round forever in t. Let r be the smallest such round, and j be a process blocked at round r in t. By minimality of r, all the processes arrive at round r. By Definition 2.8 of an execution of *DEL*, there is a $c \in DEL$ such that t is an execution of c. Which means by Definition 2.8 of an execution of a collection that eventually all messages from all the delivered sets of j up to round r are delivered.

From this point on, every local state q of j satisfies $cons(q) = \langle r, \{\langle r', k\rangle \mid r' \leq r \wedge k \in c(r', j)\}\rangle$; thus we have $q \in f$ by hypothesis on f. Then the fairness condition of executions of f from Definition 2.12 imposes that j does change round at some point. We conclude that j is not blocked at round r in t, which **contradicts** the hypothesis that j is blocked forever at round r in t.

□

The strategy satisfying exactly this condition is the minimal conservative strategy of *DEL*, and it is a strategy dominating all the conservative strategies for this delivered predicate.

Definition 2.42 (Minimal Conservative Strategy)**.** Let *DEL* be a delivered predicate. Then the **minimal conservative strategy** for *DEL* is $f_{min} \triangleq$ the conservative strategy such that $f = \{q \in Q \mid \exists c \in DEL, \exists p \in \Pi, \forall r \leq q.round : q(r) = c(r, p)\}$.

Intuitively, when every message from a prefix is delivered, there is no message left from past and present; a valid conservative strategy thus has to accept the state, or it would be blocked forever.

Remark 2.43 (Prefix and conservative state of a prefix)**.** Intuitively, a prefix of a collection c for a process p at round r is the sequence of sets of messages received by p at rounds $\leq r$ in c. Then we can define a state corresponding to this prefix by fixing its round at r and adding to it the messages in the prefix. This is the conservative state of the prefix. The prefixes of a delivered predicate are then all the prefixes of all its collections.

Lemma 2.44 (Domination of Minimal Conservative Strategy)**.** *Let DEL be a delivered predicate and f_{min} be its minimal conservative strategy. Then f_{min} dominates the conservative strategies for DEL.*

Proof. First, f_{min} is valid for *DEL* by application of Lemma 2.41. Next, we take another conservative strategy f, valid for *DEL*. Lemma 2.41 gives us that $f_{min} \subseteq f$. Hence, when f_{min} allow a change of round, so does f. This entails by Definition 2.12 that all the executions of f_{min} for *DEL* are also executions of f for *DEL*, and thus by Definition 2.15 that the $\mathcal{HO}_{f_{min}}(DEL) \subseteq \mathcal{HO}_f(DEL)$.

We thus conclude from Definition 2.16 that f_{min} dominates any valid conservative strategy for DEL. □

2.5.2 Building conservative strategies

Like oblivious strategies, minimal conservative strategies give minimal conservative strategies of resulting delivered predicates.

Theorem 2.45 (Minimal Conservative Strategy for Union)**.** *Let DEL_1, DEL_2 be two delivered predicates, f_1 and f_2 the minimal conservative strategies for, respectively, DEL_1 and DEL_2. Then $f_1 \cup f_2$ is the minimal conservative strategy for $DEL_1 \cup DEL_2$.*

Proof. We only have to show that $f_1 \cup f_2$ is equal to Definition 2.42.

- ($\supseteq$) Let q be a state such that $\exists c \in DEL1 \cup DEL_2, \exists p \in \Pi$ such that $\forall r \leq q.round$: $q(r) = c(r,p)$. If $c \in DEL_1$, then $q \in f_1$, by Definition 2.42 of the minimal conservative strategy because f_1 is the minimal conservative strategy for DEL_1, and by application of Lemma 2.41. Thus, $q \in f_1 \cup f_2$. If $c \in DEL_2$, the same reasoning apply with f_2 in place of f_1. We conclude that $q \in f_1 \cup f_2$.

- ($\subseteq$) Let $q \in f_1 \cup f_2$. This means that $q \in f_1 \vee q \in f_2$. The case where it is in both can be reduced to any of the two. If $q \in f_1$, then by Definition 2.42 of the minimal conservative strategy and by minimality of f_1, $\exists c_1 \in DEL_1, \exists p_1 \in \Pi$ such that $\forall r \leq q.round : q(r) = c_1(r,p_1)$. $DEL_1 \subseteq DEL_1 \cup DEL_2$, thus $c_1 \in DEL_1 \cup DEL_2$. We found the c and p necessary to show q is in the minimal conservative strategy for $DEL_1 \cup DEL_2$. If $q \in f_2$, the reasoning is similar to the previous case, replacing f_1 by f_2 and DEL_1 by DEL_2.

□

For the other three operations, slightly more structure is needed on the predicates. More precisely, they have to be independent of the processes. Any prefix of a process p in a collection of the predicate is also the prefix of any other process q in a possibly different collection of the same *DEL*. Hence, the behaviors (fault, crashes, loss) are not targeting specific processes. This restriction fits the intuition behind many common fault models.

Definition 2.46 (Prefix Symmetric *DEL*)**.** Let *DEL* be a delivered predicate. *DEL* is **prefix symmetric** $\triangleq \forall c \in DEL, \forall p \in \Pi, \forall r > 0, \forall q \in \Pi, \exists c' \in DEL, \forall r' \leq r : c'(r',q) = c(r',p)$

This differs from the previous round symmetric *DEL*, in that here we focus on prefixes, while the other focused on rounds. Notice that none implies the other: round symmetry says nothing about the rest of the prefix, and prefix symmetry says nothing about the delivered sets when rounds are different.

Assuming prefix symmetry, the conservation of the minimal conservative strategy by combination, succession and repetition follows.

Theorem 2.47 (Minimal Conservative Strategy for Combination)**.** *Let DEL_1, DEL_2 be two prefix symmetric delivered predicates, f_1 and f_2 the minimal conservative strategies for, respectively, DEL_1 and DEL_2. Then $f_1 \bigotimes f_2$ is the minimal conservative strategy for $DEL_1 \bigotimes DEL_2$.*

Proof idea. Since f_1 and f_2 are the minimal conservative strategies of DEL_1 and DEL_2, $Nexts^C_{f_1}$ is the set of the conservative states of prefixes of DEL_1 and $Nexts^R_{f_2}$ is the set of the conservative states of prefixes of DEL_2. Also, the states accepted by $f_1 \bigotimes f_2$ are the combination of the states accepted by f_1 and the states accepted by f_2. And the prefixes of $DEL_1 \bigotimes DEL_2$ are the prefixes of DEL_1 combined with the prefixes of DEL_2 **for the same process**. Thanks to prefix symmetry, we can take a prefix of DEL_2 and any process, and find a collection such that the process has that prefix. Therefore the combined prefixes for the same process are the same as the combined prefixes of DEL_1 and DEL_2. Thus, $Nexts^C_{f_1 \bigotimes f_2}$ is the set of conservative states of prefixes of $DEL_1 \bigotimes DEL_2$, and $f_1 \bigotimes f_2$ is its minimal conservative strategy. □

Proof. We only need to show that $f_1 \bigotimes f_2$ is equal to Definition 2.42.

- ($\supseteq$) Let q be a state such that $\exists c \in DEL1 \bigotimes DEL_2, \exists p \in \Pi$ such that $\forall r \leq q.round$: $q(r) = c(r,p)$. By definition of c, $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2 : c_1 \bigotimes c_2 = c$. We take q_1 such that $q_1.round = q.round$ and $\forall r > 0$:
$\begin{pmatrix} q_1(r) = c_1(r,p) & \text{if } r \leq q.round \\ q_1(r) = q(r) & \text{otherwise} \end{pmatrix}$. We also take q_2 such that $q_2.round = q.round$ and $\forall r > 0 : \begin{pmatrix} q_2(r) = c_2(r,p) & \text{if } r \leq q.round \\ q_2(r) = q(r) & \text{otherwise} \end{pmatrix}$.

  First, f_1 and f_2 are valid for their respective predicate by Lemma 2.41 and Definition 2.42. Then by validity of f_1 and f_2 and by application of Lemma 2.41, we get $q_1 \in f_1$ and $q_2 \in f_2$. We also see that $q = q_1 \bigotimes q_2$. Indeed, for $r \leq q.round$, we have $q(r) = c(r,p) = c_1(r,p) \cap c_2(r,p) = q_1(r) \cap q_2(r)$; and for $r > q.round$, we have $q(r) = q(r) \cap q(r) = q_1(r) \cap q_2(r)$.

  Therefore $q \in DEL_1 \bigotimes DEL_2$.

- ($\subseteq$) Let $q \in f_1 \bigotimes f_2$. By Definition 2.14 of operations on strategies, and specifically combination, $\exists q_1 \in f_1, \exists q_2 \in f_2$ such that $q_1.round = q_2.round = q.round$ and $q = q_1 \bigotimes q_2$.

  Since f_1 and f_2 are minimal conservative strategies of their respective *DEL*s, by Definition 2.42 $\exists c_1 \in DEL_1, \exists p_1 \in \Pi$ such that $\forall r \leq q.round : q_1(r) = c_1(r,p_1)$; and $\exists c_2 \in DEL_2, \exists p_2 \in \Pi$ such that $\forall r \leq q.round : q_2(r) = c_2(r,p_2)$.

  By Definition 2.46 of prefix symmetry, the fact that DEL_2 is prefix symmetric implies that $\exists c_2' \in DEL_2$ such that $\forall r \leq q.round : c_2'(r,p_1) = c_2(r,p_2)$. Hence, $\forall r \leq q.round$: $q_2(r) = c_2'(r,p_1)$.

  By taking $c = c_1 \bigotimes c_2$, we get $\forall r \leq q.round : q(r) = q_1(r) \cap q_2(r) = c_1(r,p_1) \cap c_2(r,p_1) = c(r,p_1)$.

  We found c and p showing that q is in the minimal conservative strategy for $DEL_1 \bigotimes DEL_2$.

□

Theorem 2.48 (Minimal Conservative Strategy for Succession)**.** *Let DEL_1, DEL_2 be two prefix symmetric delivered predicates, f_1 and f_2 the minimal conservative strategies for, respectively, DEL_1 and DEL_2. Then $f_1 \rightsquigarrow f_2$ is the minimal conservative strategy for $DEL_1 \rightsquigarrow DEL_2$.*

Proof idea. Since f_1 and f_2 are the minimal conservative strategies of DEL_1 and DEL_2, $Nexts^C f_1$ is the set of the conservative states of prefixes of DEL_1 and $Nexts^C_{f_2}$ is the set of the conservative states of prefixes of DEL_2. Also, the states accepted by $f_1 \rightsquigarrow f_2$ are the succession of the states accepted by f_1 and the states accepted by f_2. And the prefixes of $DEL_1 \rightsquigarrow DEL_2$ are the successions of prefixes of DEL_1 and prefixes of DEL_2 **for the same process**. But thanks to prefix symmetry, we can take a prefix of DEL_2 and any process, and find a collection such that the process has that prefix.

Therefore the succession of prefixes for the same process are the same as the succession of prefixes of DEL_1 and DEL_2. Thus, $Nexts^C_{f_1 \rightsquigarrow f_2}$ is the set of conservative states of prefixes of $DEL_1 \rightsquigarrow DEL_2$, and is therefore its minimal conservative strategy. □

Proof. We only need to show that $f_1 \rightsquigarrow f_2$ is equal to Definition 2.42.

- ($\supseteq$) Let q be a state such that $\exists c \in DEL_1 \rightsquigarrow DEL_2, \exists p \in \Pi$ such that $\forall r' \leq q.round : q(r') = c(r', p)$. By Definition 2.4 of the operations on predicates, and specifically of succession, $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2, \exists r > 0 : c = c_1[1,r].c_2$.
  - If $r = 0$, then $c[1,r] = c_2[1,r]$, and thus $\forall r' \leq q.round : q(r') = c_2(r',p)$. First, f_2 is valid for DEL_2 by Lemma 2.41 and Definition 2.42. Then the validity of f_2 and Lemma 2.41 allow us to conclude that $q \in f_2$ and thus that $q \in f_1 \rightsquigarrow f_2$.
  - If $r > 0$, we have two cases to consider.
    - If $q.round \leq r$, then $\forall r' \leq q.round : q(r') = c_1(r',p)$ f_1 is also valid for DEL_1 by Lemma 2.41 and Definition 2.42. We conclude by validity of f_1 and application of Lemma 2.41 that $q \in f_1$ and thus that $q \in f_1 \rightsquigarrow f_2$.
    - If $q.round > r$, then $c[1, q.round] = c_1[1,r].c_2[1, q.round - r]$.
      We take q_1 such that $q_1.round = r$ and $\forall r' > 0$: $\begin{pmatrix} q_1(r') = c_1(r',p) & \text{if } r' \leq q_1.round \\ q_1(r') = q(r') & \text{otherwise} \end{pmatrix}$. We also take q_2 such that $q_2.round = q.round - r$ and $\forall r' > 0$: $\begin{pmatrix} q_2(r') = c_2(r',p) & \text{if } r' \leq q_2.round \\ q_2(r') = q(r' - q.round) & \text{otherwise} \end{pmatrix}$.
      Then by validity of f_1 and f_2 for their respective predicates, and by application of Lemma 2.41, we get $q_1 \in f_1$ and $q_2 \in f_2$. We also see that $q = q_1 \rightsquigarrow q_2$. Indeed, for $r' \leq q_1.round = r$, we have $q(r') = c(r',p) = c_1(r',p) = q_1(r')$; for $r' \in [q_1.round + 1, q.round]$, we have $q(r') = c(r',p) = c_2(r'-r,p) = q_2(r'-r)$ and for $r' > q.round$ we have $q(r') = q_2(r' - q.round)$.
      We conclude that $q \in f_1 \rightsquigarrow f_2$.

- ($\subseteq$) Let $q \in f_1 \rightsquigarrow f_2$. By Definition 2.14 of operations for strategies, specifically succession, there are three possibilities for q.
  - If $q \in f_1$, then by Definition 2.42 of the minimal conservative strategy and minimality of f_1 for DEL_1, we have $\exists c_1 \in DEL_1, \exists p_1 \in \Pi : \forall r \leq q.round : q(r) = c_1(r, p_1)$. Let $c_2 \in DEL_2$. We take $c = c_1[1, q.round].c_2$; we have $c \in c_1 \rightsquigarrow c_2$ by Definition 2.4 of operations for predicates.

    Then, $\forall r \leq q.round : q(r) = c_1(r,p_1) = c(r,p_1)$. We found c and p showing that q is in the minimal conservative strategy for $DEL_1 \rightsquigarrow DEL_2$ by Definition 2.42.
  - If $q \in f_2$, then by Definition 2.42 of the minimal conservative strategy and minimality of f_2 for DEL_2, we have $\exists c_2 \in DEL_2, \exists p_2 \in \Pi : \forall r \leq q.round : q(r) = c_2(r,p_2)$. As $DEL_2 \subseteq DEL_1 \rightsquigarrow DEL_2$ by Definition 2.4, thus $c_2 \in DEL_1 \rightsquigarrow DEL_2$.

    We found c and p showing that q is in the minimal conservative strategy for $DEL_1 \rightsquigarrow DEL_2$ by Definition 2.42.
  - There are $q_1 \in f_1$ and $q_2 \in f_2$ such that $q = q_1 \rightsquigarrow q_2$.

    Because f_1 and f_2 are the minimal conservative strategies of their respective DELs, then by Definition 2.42 $\exists c_1 \in DEL_1, \exists p_1 \in \Pi$ such that $\forall r \leq q.round : q_1(r) = c_1(r,p_1)$; and $\exists c_2 \in DEL_2, \exists p_2 \in \Pi$ such that $\forall r \leq q.round : q_2(r) = c_2(r,p_2)$.

    By Definition 2.46 of prefix symmetry, the fact that DEL_2 is prefix symmetric implies that $\exists c'_2 \in DEL_2 : \forall r \leq q.round : c'_2(r,p_1) = c_2(r,p_2)$. Hence, $\forall r \leq q.round : q_2(r) = c'_2(r,p_1)$.

By taking $c = c_1[1, q_1.round].c'_2$, we have $c \in c_1 \rightsquigarrow c'_2$. Then $\forall r \leq q.round = q_1.round + q_2.round$:

$$\left(\begin{array}{lll} q(r) & = q_1(r) & \\ & = c_1(r, p_1) & \text{if } r \leq q_1.round \\ & = c(r, p_1) & \\ q(r) & = q_2(r - q_1.round) & \\ & = c'_2(r - q_1.round, p_1) & \text{if } r \in [q_1.round + 1, q_1.round + q_2.round] \\ & = c(r, p_1) & \end{array}\right).$$

We found c and p showing that q is in the minimal conservative strategy for $DEL_1 \rightsquigarrow DEL_2$ by Definition 2.42.

□

Theorem 2.49 (Minimal Conservative Strategy for Repetition)**.** *Let DEL be a prefix symmetric delivered predicate, and f be its minimal conservative strategy. Then f^ω is the minimal conservative strategy for DEL^ω.*

Proof. We only have to show that f^ω is equal to Definition 2.42.

- ($\supseteq$) Let q be a state such that $\exists c \in DEL^\omega, \exists p \in \Pi$ such that $\forall r \leq q.round : q(r) = c(r, p)$. By Definition 2.4 of operations for predicates, and specifically of repetition, $\exists (c_i)_{i \in \mathbb{N}^*}, \exists (r_i)_{i \in \mathbb{N}^*}$ such that $r_1 = 0$ and $\forall i \in \mathbb{N}^* : (c_i \in DEL \wedge r_i < r_{i+1} \wedge c[r_i + 1, r_{i+1}] = c_i[1, r_{i+1} - r_i])$.

  Let k be the biggest integer such that $r_k \leq q.round$. We consider two cases.

  - If $r_k = q.round$, then $c[1, r] = c_1[1, r_2 - r_1].c_2[1, r_3 - r_2]...c_{k-1}[1, r_k - r_{k-1}]$. We take for $i \in [1, k-1]$: q_i the state such that $q_i.round = r_{i+1} - r_i$ and $\forall r > 0$:
  $$\left(\begin{array}{ll} q_i(r) = c_i(r, p) & \text{if } r \leq q_i.round \\ q_i(r) = q(r + \sum\limits_{j \in [1, i-1]} q_i.round) & \text{otherwise} \end{array}\right).$$
  First, f is valid for DEL by Lemma 2.41 and Definition 2.42. Then by validity of f and by application of Lemma 2.41, for $i \in [1, k-1]$ we have $q_i \in f$. We see that $\forall r > 0 : q(r) = (q_1 \rightsquigarrow ... \rightsquigarrow q_{k-1})(r)$. Indeed, $\forall r \in [r_i + 1, r_{i+1}] : q(r) = c(r, p) = c_i(r - r_i, p) = q_i(r - r_i)$; and for $r > q.round : q(r) = q_{k-1}(r - \sum\limits_{j \in [1, k-1]} q_i.round)$.

    We conclude that $q \in f^\omega$
  - If $q.round > r_k$, we can apply the same reasoning as in the previous case, the only difference being $c[1, r] = c_1[1, r_2 - r_1].c_2[1, r_3 - r_2]...c_{k-1}[1, r_k - r_{k-1}].c_k[1, r - r_k]$.

- ($\subseteq$) Let $q \in f^\omega$. By Definition 2.14 of operations for strategies, and specifically of repetition, $\exists q_1, q_2, ..., q_k \in f : q = q_1 \rightsquigarrow q_2 \rightsquigarrow ... \rightsquigarrow q_k$.

  By Definition 2.42 of the minimal conservative strategy and by minimality of f for DEL, $\exists c_1, c_2, ..., c_k \in DEL, \exists p_1, p_2, ..., p_k \in \Pi : \forall i \in [1, k] q_i = \langle q_i.round, \{\langle r, j \rangle \mid r \leq q_i.round \wedge j \in c_i(r, p_i)\}$.

  By Definition 2.46 of prefix symmetry, the fact that DEL is prefix symmetric implies that $\forall i \in [2, k], \exists c'_i \in DEL, \forall r \leq q_i.round : c'_i(r, p_1) = c_i(r, p_i)$.

We take $c = c_1[1, q_1.round].c'_2[1, q_2.round]...c'_{k-1}[1, q_{k-1}.round].c'_k$, thus $c \in c_1 \rightsquigarrow c'_2 \rightsquigarrow \dots \rightsquigarrow c'_k$. Then $\forall r \leq q.round = \sum\limits_{i\in[1,k]} q_i.round$, if $r \in [\sum\limits_{i\in[1,i-1]} q_i.round{+}1, \sum\limits_{i\in[1,i]} q_i.round]$, we have $\left(\begin{array}{ll} q(r) & = q_i(r - \sum\limits_{i\in[1,i-1]} q_i.round) \\ & = c_i(r - \sum\limits_{i\in[1,i-1]} q_i.round, p_1) \\ & = c(r, p_1) \end{array}\right)$.

We found c and p showing that q is in the minimal conservative strategy for DEL^{ω} by Definition 2.42.

□

2.5.3 Computing Heard-Of predicates of conservative strategies

The analogy with oblivious strategies breaks here: the heard-of predicate of conservative strategies is hard to compute, as it depends in intricate ways on the delivered predicate itself.

Yet it is still possible to compute interesting information on this HO: upper bounds. These are overapproximations of the actual HO, but they can serve for formal verification of LTL properties. Indeed, the executions of an algorithm for the actual HO are contained in the executions of the algorithm for any overapproximation of the HO, and LTL properties must be true for all executions of the algorithm. So proving the property on an overapproximation also proves it on the actual HO.

Theorem 2.50 (Upper Bounds on HO of Minimal Conservative Strategies)**.** *Let DEL, DEL_1, DEL_2 be delivered predicates containing c_{total}.*
Let $f^{cons}, f^{cons}_1, f^{cons}_2$ be their respective minimal conservative strategies, and $f^{obliv}, f^{obliv}_1, f^{obliv}_2$ be their respective minimal oblivious strategies. Then:

- $\mathcal{HO}_{f^{cons}_1 \cup f^{cons}_2}(DEL_1 \cup DEL_2) \subseteq HOProd(Nexts_{f^{obliv}_1} \cup Nexts_{f^{obliv}_2})$.
- $\mathcal{HO}_{f^{cons}_1 \rightsquigarrow f^{cons}_2}(DEL_1 \rightsquigarrow DEL_2) \subseteq HOProd(Nexts_{f^{obliv}_1} \cup Nexts_{f^{obliv}_2})$.
- $\mathcal{HO}_{f^{cons}_1 \bigotimes f^{cons}_2}(DEL_1 \bigotimes DEL_2) \subseteq HOProd(\{n_1 \cap n_2 \mid n_1 \in Nexts_{f^{obliv}_1} \wedge n_2 \in Nexts_{f^{obliv}_2}\})$.
- $\mathcal{HO}_{(f^{cons})^{\omega}}(DEL^{\omega}) \subseteq HOProd(Nexts_{f^{obliv}})$.

Proof. An oblivious strategy is a conservative strategy. Therefore, the minimal conservative strategy always dominates the minimal oblivious strategy. Hence, we get an upper bound on the heard-of predicate of the minimal conservative strategies by applying Theorem 2.36. □

2.5.4 When Conservative is Enough

Some examples above, like $DEL^{crash}_{1,\geq r}$ (see Table 2.1), are not dominated by oblivious strategies. But they are dominated by conservative strategies. This follows from a condition on delivered predicates and its invariances by the operations, similar to the case of oblivious strategies.

Let's start by showing that the condition implies the domination by a conservative strategy.

Definition 2.51 (Common Prefix Property)**.** Let DEL be a Delivered Predicate. DEL satisfies the **common prefix property** $\triangleq \forall c \in DEL, \forall p \in \Pi, \forall r > 0, \exists c' \in DEL, \forall q \in \Pi, \forall r' \leq r : c'(r', q) = c(r', p)$.

The common prefix property, as its name suggests, works just like the common round property but for a prefix. It ensures that for every prefix of a collection in the predicate, there exists a collection where every process shares this prefix.

Theorem 2.52 (Sufficient Condition of Conservative Domination)**.** *Let DEL be a delivered predicate satisfying the common prefix property Then, there is a conservative strategy which dominates DEL.*

Proof. The proof is exactly the same as for the oblivious case, except that the common prefix property gives us a delivered collection where every one has the same exact prefix as the blocked process in the canonical execution □

Theorem 2.53 (Domination by Conservative for Operations)**.** *Let DEL, DEL_1, DEL_2 be delivered predicates with the common prefix property. Then $DEL_1 \cup DEL_2$, $DEL_1 \bigotimes DEL_2$, $DEL_1 \rightsquigarrow DEL_2$, DEL^ω satisfy the common prefix property.*

Proof.

- Let $c \in DEL_1 \cup DEL_2$. Thus $c \in DEL_1$ or $c \in DEL_2$ by Definition 2.4. We fix $c \in DEL_1$ (the other case is symmetric). Then for $p \in \Pi$ and $r > 0$, we get a $c' \in DEL_1$. satisfying the condition of Definition 2.51. And since $DEL_1 \subseteq DEL_1 \cup DEL_2$, we get $c' \in DEL_1 \cup DEL_2$.

  We conclude that the common prefix property still holds for $DEL_1 \cup DEL_2$ by Definition 2.51.

- Let $c \in DEL_1 \bigotimes DEL_2$. Then $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2 : c = c_1 \bigotimes c_2$. For $p \in \Pi$ and $r > 0$, our hypothesis on DEL_1 and DEL_2 ensures that there are $c'_1 \in DEL_1$ satisfying the condition of Definition 2.51 for c_1 and $c'_2 \in DEL_2$ satisfying the condition of Definition 2.51 for c_2.

  We argue that $c' = c'_1 \bigotimes c'_2$ satisfies the condition of Definition 2.51 for c. Indeed, $\forall r' \leq r, \forall q \in \Pi : c(r', q) = c'_1(r', q) \bigotimes c'_2(r', q) = c_1(r', p) \bigotimes c_2(r', p) = c(r', p)$.

  We conclude that the condition of Definition 2.51 still holds for $DEL_1 \bigotimes DEL_2$.

- Let $c \in DEL_1 \rightsquigarrow DEL_2$. Since if $c \in DEL_2$, the condition trivially holds by hypothesis, we study the case where succession actually happens. Hence, $\exists c_1 \in DEL_1, \exists c_2 \in DEL_2, \exists r_{change} > 0 : c = c_1[1, r_{change}].c_2$. For $p \in \Pi$ and $r > 0$, our hypothesis on DEL_1 and DEL_2 ensures that there are $c'_1 \in DEL_1$ satisfying the condition for c_1 at r and $c'_2 \in DEL_2$ satisfying the condition for c_2 at $r - r_{change}$.

  We argue that $c' = c'_1[1, r_{change}].c'_2$ satisfies the condition for c. Indeed, $\forall r' \leq r, \forall q \in \Pi$, we have: if $r' \leq r_{change} : c'(r', q) = c'_1(r', q) = c_1(r', p) = c(r', p)$; and if $r' > r_{change}$: $c'(r', q) = c'_2(r' - r_{change}, q) = c_2(r' - r_{change}, p) = c(r', p)$.

  We conclude that the condition still holds for $DEL_1 \rightsquigarrow DEL_2$.

- Let $c \in DEL^\omega$. Let (c_i) and (r_i) be the collections and indices defining c. Let $p \in \Pi$ and $r > 0$. Then let i the integer such that $r \in [r_i + 1, r_{i+1}]$ By hypothesis on DEL, $\forall i' \leq i$ there are $c'_{i'} \in DEL$ satisfying the condition of Definition 2.51 for $c_{i'}$ and $r - r_{i'}$.

  We argue that $c' = \prod\limits_{i>0} c'_i[1, r_{i+1} - r_i]$ satisfies the condition of Definition 2.51 for c. Indeed, $\forall r' \leq r, \forall q \in \Pi$, we have: $\forall i' > 0 : r' \in [r_{i'} + 1, r_{i'+1}] \implies c'(r', q) = c'_{i'}(r' - r_{i'}, q) = c_{i'}(r' - r_{i'}, p) = c(r', p)$.

We conclude that condition of Definition 2.51 still holds for DEL^{ω}.

□

Therefore, as long as the initial building blocks satisfy the common prefix property, so does the result of the operations. Thus the latter is dominated by its minimal conservative strategy – a strategy that can be computed easily from the previous results in this section.

2.6 The future is now

In the above, the dominating strategy was at most conservative: only the past and present rounds were useful for generating heard-of collections. Nevertheless, messages from future rounds serve in some cases.

Let's go back to DEL_1^{loss}, the delivered predicate for at most one message loss presented in Section 2.2.1. The minimal oblivious strategy for this predicate is f_{n-1}. The minimal conservative strategy is a similar one, except that when it received a message from p at round r, it waits for all messages from p at previous rounds. But this does not change which messages the strategy waits for in the current round: $n-1$ messages, because one can always deliver all the messages from the past, and then the loss might be a message from the current round.

If on the other hand the strategy considers messages from the next round, it can ensure that at each round, at most one message among all processes is not delivered on time. The strategy presented here waits for either all messages from the current round, or for all but one messages from the current round and all but one message from the next round.

Definition 2.54 (Asymmetric Strategy)**.** Let $\mathit{after}: Q \mapsto \mathcal{P}(\Pi)$ such that $\forall q \in Q : \mathit{after}(q) = \{k \in \Pi \mid \langle q.round + 1, k\rangle \in q.mes\}$.
Then $f_{asym} \triangleq \left\{ q \in Q \;\middle|\; \begin{array}{l} \quad |\mathit{obliv}(q)| = |\Pi| \\ \vee \quad (|\mathit{after}(q)| = |\Pi| - 1 \wedge |\mathit{obliv}(q)| = |\Pi| - 1\} \end{array} \right\}$.

Intuitively, this strategy is valid for DEL_1^{loss} because at each round and for each process, only two cases exist:

- Either no message for this process at this round is lost, and it receives a message from every process;
- Or one message for this process is lost at this round, and it only receives $n-1$ messages. But all the other processes receive n messages (because none can be lost), thus change round and send their message from the next round. Since the one loss already happened, all these messages are delivered, and the original process eventually receives $n-1$ messages from the next round.

Lemma 2.55 (Validity of f_{asym})**.** *f_{asym} is valid for DEL_1^{loss}.*

Proof. We proceed by contradiction: **Assume** f_{asym} is invalid for DEL_1^{loss}. Thus, there exists $t \in execs_{f_{asym}}(DEL_1^{loss})$ invalid. Hence there is a smallest round r where some process j has infinitely often a state not in f_{asym}. Let also c be a delivered collection of DEL_1^{loss} such that $t \in execs(c)$.

Minimality of r entails that every process reaches round r and thus sends its messages to j, and $c \in DEL_1^{loss}$ entails that at most one of these messages can be lost. Thus j eventually receives $n-1$ messages from round r.

By hypothesis it doesn't receive all n messages, or it could change round. Thus j receives exactly $n-1$ messages from round r, which means that the only loss allowed by DEL_1^{loss} happens at round r.

But for j to block, it must never receives $n-1$ messages from round $r+1$. Yet the only loss is a message to j; thus every other process receives n messages at round r, changes round, and sends its message to j without loss. Hence j eventually receives $n-1$ messages from round $r+1$.

This **contradicts** the fact that j cannot change round at this point in t. □

This strategy also ensures that at most one process per round receives only $n-1$ messages on time – the others must receive all the messages. This vindicates the value of messages from future rounds for some delivered predicates, such as the ones with asymmetry in them.

Theorem 2.56 (Heard-Of Characterization of f_{asym})**.**
$\mathcal{HO}_{f_{asym}}(DEL_1^{loss}) = \mathcal{HO}_1^{countrounds}$.

Proof. First, we show $\subseteq$. Let $ho \in \mathcal{HO}_{f_{asym}}(DEL_1^{loss})$ and $t \in execs_{f_{asym}}(DEL_1^{loss})$ an execution of f_{asym} generating ho. By definition of the executions of f_{asym}, processes change round only when they received either n messages from the current round, or $n-1$ messages from the current round (and $n-1$ messages from the next one, but that is irrelevant to the heard-of predicate). Moreover, t is valid by definition, as it generates ho.

Let's now assume that at least one process j receives only $n-1$ messages on time for some round r in ho. By definition of f_{asym} and validity of t, we deduce that j also received $n-1$ messages from round $r+1$ while it was at round r. Hence every other process ended its own round r before j; the only possibility is that they received n messages from round r, because the alternatives require the reception of the message from j at round $r+1$

We conclude that for each round, at most one process receive only $n-1$ messages on time, which can be rewritten as $\forall r \in \mathbb{N}^* : \sum\limits_{j\in\Pi} |\Pi \setminus ho(r,j)| \leq 1$.

Then, we show $\supseteq$. Let ho a heard-of collection over Π such that $\forall r \in \mathbb{N}^* : \sum\limits_{j\in\Pi} |\Pi \setminus ho(r,j)| \geq 1$. The difficulty here is that the canonical execution of ho fails to be an execution of f_{asym}: when only $n-1$ messages from the current round are delivered to some process j, then the corresponding $next_j$ for this round will not be allowed by f_{asym}.

One way to deal with this issue is to start from the canonical execution of ho and move these incriminating $next_j$ after the deliveries of $n-1$ messages from the next round, and before the deliveries of messages from j in the next round.

In this way, every $next_j$ will happen when either n messages have been received from the current round, or $n-1$ messages from the current round and $n-1$ from the next one. We conclude that $ho \in \mathcal{HO}_{f_{asym}}(DEL_1^{loss})$. □

Does f_{asym} dominate DEL_1^{loss}? It would seem so, but proving it is harder than for f_{n-F} and DEL_F^{crash}. The reason is that the common round property of the latter allows the creation of deadlocks where every process is blocked in the same local state, which forces any valid strategy to accept this state. Whereas the whole reason the future serves in DEL_1^{loss} is because the latter doesn't have this property, and thus the local state of a process constrains the possible local states of other process.

That being said, the conjecture seems reasonable.

Conjecture 2.57 (Domination of f_{asym} on DEL_1^{loss}). *f_{asym} is a dominating strategy for DEL_1^{loss}.*

This example demonstrates two important points about strategies using the future: they are necessary for finding dominating strategies of some delivered predicates, and their study is not amenable to the same techniques as conservative and oblivious strategies.

2.7 Perspectives

2.7.1 Summary

In this chapter, I proposed a formalization of the heard-of predicate to use when studying a given operational model through the Heard-Of model. This formalization comes with the following methodology:

- Extract a delivered predicate from the informal model, either through direct analysis or by building the predicate from simpler building blocks through operations.
- Compute a dominating strategy for this delivered predicate, either by direct analysis or by combining dominating strategies for the simpler building blocks.
- Compute the heard-of predicate generated by this strategy on this delivered predicate.

This result captures the most constrained predicate which can be implemented on top of the initial model. Thus if a round-based algorithm is proved correct on this predicate, its correctness follows on the original model; and if no algorithm exists for a given problem on this predicate, this entails that no round-based algorithm solves the problem on the original model.

2.7.2 History of the research

The formalization in this chapter was first couched in terms of games. Processes aimed at never blocking forever, while the adversarial environment tried to ensure such a deadlock. Strategies, validity and domination were defined through games too, yet they worked in a very similar way to the current version.

The issue with this approach was its complexity, as shown by a cursory look at the first version of the paper [54] available in arXiv. Using games required an inordinate amount of administrative formal manipulations, for proving simple and intuitive results.

The idea of using delivered predicates instead followed from the negative but useful reviews given at DISC 2018, as well as from conversations with other researchers. This solution removed most of the complexity, while keeping intact the core of the intuition.

Except this false start, the other big difficulty in this research was the constraining of strategies and predicates to prove interesting results. At first we aimed at showing the existence of a dominating strategy among all strategies for all predicates, but we were brought back to reality by the issues of dealing with future messages. This resulted in constraints that focus on messages from past and present, and conditions on predicates which make these messages the only ones that matter.

2.7.3 Perspective

The obvious follow-up to this research is to tackle strategies which look into the future, and predicates like DEL_1^{loss} that are useful for such strategies. Doing so will require completely new proof techniques, as the one presented here implicitly rely on the ability to make only the messages in the past and present rounds matter.

A more straightforward extension of this work would be the study of more delivered predicates, both for having more building blocks to use with operations, and to derive more heard-of predicates.

Chapter 3

Clashing Heard-Of Predicates with Other Models

Sommaire

3.1 Introduction

3.1.1 Motivation

The most successful approach to abstract away distributed communication models is probably the introduction of failure detectors by Chandra and Toueg [16]. These are oracles giving information at each process on which other processes crashed. A notion of weakest failure detector for a problem naturally arises, one which captures the information necessary to solve a given problem in an asynchronous setting. Over the years, weakest failure detectors have been found for consensus [17], registers [56], NBAC [57], and others.

Despite all these achievements, failure detectors show several flaws as abstraction of uncertainty. Among them, the fact that the weakest failure detectors are highly dependent on the underlying model: whether there are crashes, how many of them, if processes are anonymous or not... Hence, we must study different failure detectors for different models, which brings us back to square one: the existence of too many models. Another issue with failure detectors stems from the formalism itself: depending on subtleties, there is either a weakest failure detector for each problem [58], or there isn't [59]. This comes from having a model with an asynchronous component (the actual algorithm) and a time-dependent component (the failure detector). As explored by Charron-Bost et al. [60], this sorely limits failure detectors. In essence, implementing an algorithm that uses some failure detector "loses" the time information. Hence the usual comparison relation between failure detector cannot have fundamental mathematical properties, such as reflexivity, and taking the easy way out through the reflexive closure still has problems. Namely, that the reflexive closure doesn't ensure that a failure detector is smaller by this order than another failure detector which allows strictly more histories for each failure pattern. Losing timing information also limits the weakest failure detector for reflexive and meaninful comparison relations to be time-insensitive (or stuttering).

All in all, these problems justify the search for another approach to distributed uncertainty. Obviously, we go with the one used in this thesis: the Heard-Of model.

Now, the natural question for any user of failure detectors is the following: is anything lost when going from failure detectors to the Heard-Of model? A lack of answer is partly to blame for the lack of acceptance of the latter. It would seem that the Heard-Of model requires more at first glance: all processes might decide (because there are no failures), rounds are communication-closed and thus late messages are never used. It is not ridiculous to assume that such constraints will reduce the ability to solve distributed problems.

This chapter delves into this question, and proves the equivalence of the Chandra-Toueg hierarchy of failure detectors with specific heard-of predicates. That is, I prove that any problem solvable in one is also solvable in the other. Along the way, I also explore various notions of solvability in the Heard-Of model, and their consequences for the equivalence.

3.1.2 Overview

The contributions are the following:

- The definition of heard-of predicates equivalent to Chandra-Toueg failure detectors. Among these, the predicate for the perfect failure detector was never used before.

- The proof of equivalence between these predicates and Chandra-Toueg failure detectors.

- The extension of equivalences above when all processes must decide in the Heard-Of model, with sufficient conditions.

3.1.3 Related Work

There is already a line of research studying the link between asynchronous models augmented by failure detectors, and round-based models with communication constraints. It started with Afek and Gafni [32], who related asynchronous shared-memory and synchronous message-passing with adversaries; and was followed by Raynal and Stainer [33], who studied simulations between asynchronous message-passing models with failure detectors and synchronous message-passing with adversaries.

However, the model studied here is not synchronous – it is the Heard-Of model, which only speaks about rounds, not how they are implemented. The thing is, fundamentally, neither of the papers above talks about synchrony: their round-based model is synchronous, but synchrony serves only for implementing rounds. Hence, it is not a necessity, just the natural way for distributed computing researchers to think about rounds. Therefore, the results of Afek and Gafni [32] and of Raynal and Stainer [33] both still work when replacing their synchronous model by the Heard-Of model, and conversely our results translate to their synchronous model too. Going with the Heard-Of model instead of the synchronous adversary one boils down to Occam's razor: it has less hypotheses for the same result.

Two other studies of the connection between rounds and failure detectors are the algorithm transformations of Biely et al. [61] and the comparison of the perfect failure detector with a synchronous model by Charron-Bost et al. [62].

Biely et al. [61] compare failure detectors with a partially synchronous model, but also a predecessor of the Heard-Of model. The results they derive are about the $\Diamond\mathcal{S}$ failure detector, and the predicate with an eventual source. Our derivation of the heard-of predicate for $\Diamond\mathcal{S}$ is similar; but the less trivial failure detectors to find equivalent predicates for are $\mathcal{P}$ and $\Diamond\mathcal{P}$, which this paper only treats through partial synchrony, not round-based properties.

On the other hand, Charron-Bost et al. [62] focus less on the power of rounds than on the power of synchrony. Their article indeed studies whether asynchrony augmented with a perfect failure detector is equivalent in terms of solvability to synchrony. They answer in the negative, by giving a problem which can only be solved if one process can know whether some message was sent or not. This is always possible with synchrony, thanks to the communication upper bound, but is sometimes impossible in asynchronous systems, even with the full power of $\mathcal{P}$.

Unexpectedly, this results applies to the Heard-Of model by putting it in the place of the asynchronous system, not the synchronous one. This is because the difference in power between synchronous and asynchronous models lies in being able to know if some message will ever be delivered. A synchronous model can, while an asynchronous model with failure detectors or the Heard-Of model can't. Hence a purely synchronous model is more powerful than the Heard-Of model.

That being said, the problem used in the paper is quite tailored to showing this separation, and has no known practical application. It's thus reasonable to conjecture that for all practical problems of distributed computation, synchrony is equivalent to the right properties on rounds.

3.2 Models and Previous Results

3.2.1 Models

This work investigates the links between two general models of distributed computing: the asynchronous model augmented with failure detectors, and the Heard-Of model. While the latter was already defined in Chapter 1, the former still needs to be formalized.

This starts with the definition of failures and failure detectors. These come straight from Chandra and Toueg [16], because their failure detectors are the ones studied here. For the rest of the chapter, the set Π of processes is fixed.

A failure detector is a collection of local modules, one at each node, that give a list of suspected process. The intuition is then that a failure pattern gives the set of crashed

Completeness \ Accuracy	Strong	Weak	Eventually Strong	Eventually Weak
Strong	Perfect $\mathcal{P}$	Strong $\mathcal{S}$	Eventually Perfect $\diamond\mathcal{P}$	Eventually Strong $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	Weak $\mathcal{W}$	$\diamond\mathcal{Q}$	Eventually Weak $\diamond\mathcal{W}$

Figure 3.1: Failure Detectors Defined in [16]

processes at time t (represented by a natural number) of the execution; and a failure detector takes a failure pattern and returns the set of history for the modules, that is of functions from a process p and a time t to the set of processes suspected by p at t.

Hence, a failure detector constrains the possible behaviors of its modules at each process depending on the failure pattern. Of course, the failure pattern is unknown to the algorithm; the only information about it is captured by the output of the failure detectors, that is by the histories.

Definition 3.1 (Failure pattern)**.** A **failure pattern** $F \triangleq$ a map $\mathbb{N} \to \mathcal{P}(\Pi)$. For F a failure pattern, $\mathit{crashed}(F) \triangleq \bigcup\limits_{t \in \mathbb{N}} F(t)$ and $\mathit{correct}(F) \triangleq \Pi \setminus \mathit{crashed}(F)$.

Definition 3.2 (Failure detector)**.** Let $FD : (\mathbb{N} \to \mathcal{P}(\Pi)) \to \mathcal{P}((\mathbb{N} \times \Pi) \to \mathcal{P}(\Pi))$. Then FD is a **failure detector**.

Table 3.1 provides a list of the failure detectors from Chandra and Toueg [16]; they will be defined in the chapter when needed.

Another fundamental property of distributed computing models is atomicity: which sequences of events are considered to happen in a single step. Atomicity plays a big role in limiting the uncertainty of distributed computations, by removing non-determinism and interleaving. Our two models differ in what atomicity they assume.

To define atomicity, we need to start with computations and executions. A computation is a partial order of events for each process, capturing the causal order between these events. An execution of a computation is a linearization of the execution, a total order that is coherent with the partial order. The following definition is standard, except for not putting receptions at a process p in the total order of events at p. The reason here is that in one of the models, we want to be able to reorder receptions, which is possible because we track when the receptions are used in the computation steps and thus in the following events.

Definition 3.3 (Distributed Computation and Execution)**.** Let *Events* be a set of events annotated by processes in Π. The possible events are:

- emissions of a single message
- broadcast, emissions of one message for each process.
- receptions of a single message
- receptions of multiple messages
- computation steps annotated by a set of messages.

Then a partial order $c = (E, \prec_c)$ on a subset E of *Events* is a **distributed computation** $\triangleq$ It satisfies the following constraints.

- For each process $p \in \Pi$, if m is a message and the computation step $comp_p$ has m in its annotation, then there's either a reception of m at p or a reception of multiple messages (including m) at p in c. If we call this reception event $receive_p(m)$, then $receive_p(m) \prec_c comp_p$.
- For each process $p \in \Pi$, the set of emissions (single emissions or broadcasts) and computation steps forms a chain (a totally ordered subset).
- For each message m, if the reception of m at some q (either single reception or multiple reception) is in c, then the emission of m at some p (either single emission or broadcast) is in c.
- For each message m, if $emission_p(m)$ is the emission event of m at the sender p and $receive_q(m)$ is the reception event of m at the receiver q, then $emission_p(m) \prec_c receive_q(m)$
- If there is a crash event for p, then this event is a maximal element for the set of events for p.

An **execution** t of $c \triangleq$ a linearization of c, where the time of each event is its position + 1 in t.

Definition 3.4 (Atomicity)**.** Let c be a distributed computation, and let A be a subset of events in c. Then c **is atomic** for $A \triangleq \exists t$ an execution of c such that all events in A happen one after the other, with no other event in between.

The asynchronous model studied is the wait-free model (asynchronous model with reliable communication, and at most $|\Pi| - 1$ crashes) augmented with a failure detector. The events on distributed computation for this model are reception, emission, and computation step; the latter are annotated with the messages used in it, to deal with atomicity in the simulations, and the output of the failure detector module. This model ensures atomicity at each process for every set of receptions, the next computation step and the next emissions of messages.

Definition 3.5 (Asynchronous Message-Passing model augmented with FD)**.** Let $\mathcal{FD}$ be a failure detector. $AMP[\mathcal{FD}]$ is the asynchronous model with reliable communication, at most $n - 1$ crashes, augmented with $\mathcal{FD}$.

A distributed computation in $AMP[\mathcal{FD}]$ is a distributed computation (Def 3.3) containing only single emissions, single receptions, crashes and computation steps annotated with a set of messages and failure detector output. It satisfies the additional conditions:

- At most $n - 1$ crashes per computation.
- Every message sent to a non-crashed process is eventually received in the computation.
- Every process without a crash event has an infinite number of computation steps.
- There is a linearization of the computation, such that the outputs of failure detectors are in a history of FD for the failure pattern defined by the crashes.

The executions of such a distributed computations are only those such that the outputs of failure detectors are in a history of $\mathcal{FD}$ for the failure pattern defined by the crashes.

The succession (at a single process) of (possibly no, or several) receptions used by the next computation step; the next computation step; the next (possibly no or several) emissions is atomic in $AMP[\mathcal{FD}]$.

(End of Definition)

To harmonize the notations, let's introduce an equivalent definition for the Heard-Of model. Here the events are broadcasts, receptions of multiple messages (every message of the current round according to the heard-of predicate), and computation steps for the change of rounds (and update of state). As for atomicity, it ensures that every succession at a process of a broadcast at round r, a reception at round r and a computation step at round r is atomic.

Definition 3.6 (Heard-Of model with predicate $\mathcal{HO}$)**.** Let $\mathcal{HO}$ be a heard-of predicate. $HO[\mathcal{HO}]$ is the Heard-Of model with communication predicate $\mathcal{HO}$.

A distributed computation in $HO[\mathcal{HO}]$ is a distributed computation (Def 3.3) containing only broadcasts tagged by the round, receptions of multiple messages tagged by the round, and computation steps tagged by the round and a set of messages. It satisfies the additional conditions:

- For each process, the set of local events forms an infinite chain (a totally ordered subset) built by a repetition of a broadcast, a reception and a communication step, forever.
- The sequence of all the receptions corresponds to a heard-of collection of $\mathcal{HO}$.

For every computation c of $HO[\mathcal{HO}]$, c is atomic for the succession (at a single process) of the broadcast for a round; the reception for a round; and the computation step for the round.

(End of Definition)

3.3 Simulations

The equivalence between the two models is proved using simulations from one to the other. A simulation from model M_1 to model M_2 takes an algorithm on M_1 and makes it run on model M_2, with an execution of the simulation on M_2 being coherent with an execution of the original algorithm on M_1. This implies that the fundamental properties of the model are maintained: atomicity and the failure detector/heard-of predicate properties.

The first step to defining simulations is through the notion of "solving a problem".

3.3.1 Specifications

Computability underlies the following equivalences. Thus the first step is to formalize the notion of a problem (called a specification), and what it means to solve such a problem.

Because $AMP[\mathcal{FD}]$ contains crashes, not every process has to decide in solving a specification. The processes that don't have to decide are the faulty processes, which correspond with the crashed processes in $AMP[\mathcal{FD}]$ (There are subtleties for $HO[\mathcal{HO}]$). The intuition about specifications is that they're defined for when no process crashes, and then the possible vector of outputs when crashes do happen just replace the output of crashed processes by $\perp$.

Taking a full output vector (or function from Π to the output values) and replacing some values by $\perp$ gives an output vector called a faulty extension of the initial output vector.

Definition 3.7 (Faulty extension)**.** Let $out : \Pi \rightarrow V_{out}$ be a function from processes to outputs, and let $F \subseteq \Pi$ be a set of faulty processes. The **faulty extension of** out **to** F is:

$$faulty_F(out) : \Pi \rightarrow (V_{out} \cup \{\bot\}) \triangleq \begin{cases} & \forall p \in \Pi \setminus F : faulty_F(out)(p) = out(p) \\ \wedge & \forall p \in F : faulty_F(out)(p) = \bot \end{cases}$$

Then a specification is defined by the possible complete output vectors for a given input vector, and extended such that if all processes in F are faulty, then the possible output vectors for a given input vector are all faulty extensions of possible complete output vectors for the subsets of F.

Definition 3.8 (Specification)**.** Let V_{in} and V_{out} be two sets of values.
$spec : ((\Pi \rightarrow V_{in}) \times \mathcal{P}(\Pi)) \rightarrow \mathcal{P}(\Pi \rightarrow (V_{out} \cup \{\bot\}))$ is a **specification** $\triangleq$

$$\forall in : \Pi \rightarrow V_{in} : \begin{cases} & spec(in, \emptyset) \subseteq \mathcal{P}(\Pi \rightarrow V_{out}) \\ \wedge & \forall F \subseteq \Pi, spec(in, F) = \{faulty_{F'}(f) \mid f \in spec(in, \emptyset) \wedge F' \subseteq F\} \end{cases}$$

Now, specifications are solved by algorithms. What does an algorithm on $AMP[\mathcal{FD}]$ looks like? The atomicity of the model guides us: each atomic step of such an algorithm includes receptions (possibly of no messages or of several messages); a computation step; then the emissions of multiple messages. Thus an algorithm on $AMP[\mathcal{FD}]$ gives the computation step after reception and the set of messages to send.

Definition 3.9 (Algorithm on $AMP[\mathcal{FD}]$)**.** Let $\mathcal{FD}$ be a failure detector, let Q be the set of local states with a write-once *decision* variable initalized at $\bot$, and let $M^{A_p}_{content}$ be the set of possible message contents. Then $A_p : \mathcal{P}(\Pi) \times \mathcal{P}(\Pi \times \Pi \times M^{A_p}_{content}) \times Q \rightarrow Q \times \mathcal{P}(\Pi \times M^{A_p}_{content})$ is a **local algorithm on** $AMP[\mathcal{FD}]$. A local algorithm A_p comes with an init function $init^{A_p}$ that takes an input value and gives the initial state of A_p for this input value. An **algorithm on** $AMP[\mathcal{FD}]$ is a family of local algorithms on $AMP[\mathcal{FD}]$ indexed by Π, with the same $M^{A_p}_{content}$ for each local algorithm.

The first parameter of A is the output of the local failure detector module; the second is the set of received messages from the start (where a message is a triplet (sender, receiver, content)); and the last is the current state. Then A returns the next state and the messages it sends next (where the sender is left implicit, since it's the process running the algorithm).

Note that we could define the second parameter as the set of received messages *since the last step*, but this information is never used in the actual algorithms. Using instead all the messages received from the start also simplifies the proofs.

Algorithm 1 show the pseudo-code corresponding to the execution of such an algorithm in $AMP[\mathcal{FD}]$. It explicitly uses the failure detector output despite no access to time, because the algorithm looks up the failure detector during the computation step, but is asynchronous and so has no way to track time accurately.

Definition 3.10 (Distributed Computation of Algorithm on $AMP[\mathcal{FD}]$)**.** Let $\mathcal{FD}$ be a failure detector and let A be an algorithm on $AMP[\mathcal{FD}]$ (with A_p the local algorithm at process p). Then a distributed computation of A on $AMP[\mathcal{FD}] \triangleq$ a distributed computation of $AMP[\mathcal{FD}]$ such that:

- Each computation step at p is annotated by a state: the state resulting from the application of A_p to the local state annotating the previous local computation step (or an intial state for some input value, if it's the first computation step of p), the set of received messages and the failure detector output annotating the computation step.

```
/* On reception of one or several messages */
Apply $A_p$ to $fd$ (the output of failure detector), the received messages and the
  current state to get the next state and messages to send;
Send messages from last application of $A_p$;
```

Algorithm 1: Pseudo-code for an execution of local algorithm A_p for process p on $AMP[\mathcal{FD}]$

- The messages annotating each computation step at p are the messages used in the computation given by applying A_p to: the local state, the set of received messages and the failure detector output.
- The emissions after a computation step at p are given by the application of A_p to the state of the process at the computation step, the received messages at this point and the failure detector output annotating the computation step.

(End of Definition)

Now, specifications constrain the output of such algorithms. Although the intuition of the output of an algorithm is simple, it becomes slightly more difficult for distributed systems – that is, when to stop the algorithm is not trivial. We sidestep this difficulty by defining the output of an algorithm as the vector of decision that never changes in the rest of the execution. Since *decision* is write-once and the number of processes is finite, such a vector always exists for an execution.

Definition 3.11 (Output of Algorithm on $AMP[\mathcal{FD}]$)**.** Let A be an algorithm on $AMP[\mathcal{FD}]$. Then the output *out* of a computation c of A on $AMP[\mathcal{FD}]$ with initial states q^p_{init} for process p $\triangleq$ the vector of *decision* variable indexed by Π such that for each $p \in \Pi$, there is a computation step after which $out[p] = decision_p$, and no following computation step at p ever changes this.

Definition 3.12 (Solving a specification on $AMP[\mathcal{FD}]$)**.** Let *spec* be a specification, and A be an algorithm on $AMP[\mathcal{FD}]$. A **solves** *spec* **on** $AMP[\mathcal{FD}] \triangleq \forall In : \Pi \to V_{in}, \forall c$ a computation of A on $AMP[\mathcal{FD}]$ where the initial state of each p is $init^{A_p}(In(p))$, if *out* is the output of c, then $out \in spec(\{(p, In(p)) \mid p \in \Pi\}, crashed(c))$, where $crashed(c)$ is the set of processes with a crash event in c.

Ordinarily in distributed computing, the only processes that don't need to decide are the crashed ones. Since there is no crash in the Heard-Of model, it seems natural to require that every process decides according to the specification. Yet this is not the definition of solving used by Raynal and Stainer [33]. Instead, they require the decision only for processes that form a set where all the members of the set hear each other infinitely often, all the other processes hear the processes of this set infinitely often, and no process outside of this set is heard infinitely often by a member of the set.

Intuitively, this set captures the processes that will be able to propagate their values, and to agree with each other. This is the definition of solving a specification that is used for the simulations in this section. It is justified, because weakly correct processes capture the processes that would have crashed in a classical asynchronous model. Then their decision is not really necessary, even in the Heard-Of model.

```
/* For each round $r \geq 1$                                          */
Send messages from last application of $A_p$ (or initial application for $r = 1$);
Receive messages from the Heard-Of oracle (from processes in the Heard-Of set);
Apply $A_p$ to $r$, the received messages and the current state to get the next state and
  messages to send;
```

Algorithm 2: Pseudo-code for an execution of local algorithm A_p for process p on $HO[\mathcal{HO}]$

Definition 3.13 (Eventual neighbors and strongly correct processes)**.** Let h be a heard-of collection, and let $i, j \in \Pi$. i is **eventual neighbor** of j, $i \overset{\infty}{\rightsquigarrow} j \triangleq \forall r > 0, \exists k \geq 0, \exists \lambda_0, \lambda_1, ..., \lambda_k \in \Pi : \lambda_0 = i \wedge \lambda_k = j \wedge \forall l \in [1, k] : \lambda_{l-1} \in h(r + l, \lambda_l) \vee \lambda_l = \lambda_{l-1}$.

i and j are **mutual eventual neighbors**, $i \leftrightsquigarrow j \triangleq i \overset{\infty}{\rightsquigarrow} j \wedge j \overset{\infty}{\rightsquigarrow} i$.

Finally, the **strongly correct processes** of h, $\mathcal{SC}(h) \triangleq$ the subset of Π such that

$$\begin{cases} \quad \forall i, j \in \mathcal{SC}(h) : i \leftrightsquigarrow j \\ \wedge \quad \forall k \notin \mathcal{SC}(h) : (\exists i \in \mathcal{SC}(h) : i \overset{\infty}{\rightsquigarrow} k \wedge \forall j \in \mathcal{SC}(h) : k \overset{\infty}{\not\rightsquigarrow} j) \end{cases}$$

Every process that is not strongly correct is **weakly correct**.

To give an intuition, consider a source: a process that is heard by everyone at each round. Then an eventual source is a process that becomes a source after a certain round. Such an eventual source is strongly correct. This proves important in the proofs, since all predicates in this chapter ensure an eventual source.

Lemma 3.14 (Eventual source is strongly correct)**.** *Let h be a heard-of collection, and $s \in \Pi$ be a process. Then $(\exists r > 0, \forall r' \geq r, \forall p \in \Pi : s \in h(r', p)) \implies s \in \mathcal{SC}(h)$.*

Proof. This is equivalent to saying that $\mathcal{SC}(h) = \{p \mid p \in \Pi \wedge p \leftrightsquigarrow s\}$. Let's take $k \in \Pi \setminus \{p \mid p \in \Pi \wedge p \leftrightsquigarrow s\}$. We want to prove that $s \overset{\infty}{\rightsquigarrow} k \wedge k \overset{\infty}{\not\rightsquigarrow} s$. The first follows by definition of s, and the second by the fact that $k \not\leftrightsquigarrow s$ – which would be the case if both $s \overset{\infty}{\rightsquigarrow} k$ and $k \overset{\infty}{\rightsquigarrow} s$.

We conclude that s is a strongly correct process of h. ☐

Just like the atomicity of $AMP[\mathcal{FD}]$ directed us towards the definition of an algorithm on this model, the atomicity of $HO[\mathcal{HO}]$ implies the form of its algorithms: a function that takes the current round number, the set of received messages (for the current round) and the local state, and which returns the next state and the set of messages it sends next.

Definition 3.15 (Algorithm on $HO[\mathcal{HO}]$)**.** Let $\mathcal{HO}$ be a failure detector, let Q be the set of local states with a write-once *decision* variable initalized at $\perp$, and let $M_{content}^{A_p}$ be the set of possible message contents used by A_p. Then $A_p : \mathbb{N} \times \mathcal{P}(\Pi \times \Pi \times M_{content}^{A_p}) \times Q \to (Q \times M_{content}^{A_p})$ is a **local algorithm on** $HO[\mathcal{HO}]$. A local algorithm A_p comes with an init function $init^{A_p}$ that takes an input value and gives the initial state of A_p for this input value. An **algorithm on** $HO[\mathcal{HO}]$ is a family of local algorithms on $HO[\mathcal{HO}]$ indexed by Π, with the same $M_{content}^{A_p}$ for each local algorithm.

Note that the round is in $\mathbb{N}$ instead of $\mathbb{N}^*$ to deal with the initial step of sending messages for round 1.

Algorithm 2 shows the pseudo-code corresponding to the execution of such an algorithm in the Heard-Of model.

Definition 3.16 (Distributed Computation of Algorithm on $HO[\mathcal{HO}]$)**.** Let $\mathcal{HO}$ be a failure detector and let A be an algorithm on $HO[\mathcal{HO}]$ (with A_p the local algorithm at process p). Then a distributed computation of A on $HO[\mathcal{HO}] \triangleq$ a distributed computation of $HO[\mathcal{HO}]$ such that:

- The messages annotating the computation step of p at round r are those received in the reception of p at round r.
- Each computation step at p is annotated by a state: the state resulting from the application of A_p to the local state annotating the previous local computation step (or some initial state for a choice of input if its the first), the set of received messages and the round number annotating the state.
- The broadcast after a computation step at p is given by the application of A_p to the state of the process at the computation step, the received messages at this point and the round annotating the computation step.

(End of Definition)

The output is defined analogously to the $AMD[\mathcal{FD}]$ case. And here too, A is an algorithm for a single process, but every process is assumed to share the same algorithm.

Definition 3.17 (Output of Computation of Algorithm on $HO[\mathcal{HO}]$)**.** Let A be an algorithm on $HO[\mathcal{HO}]$. Then the output *out* of a computation c of A on $HO[\mathcal{HO}]$ with initial states q^p_{init} for process $p \triangleq$ the vector of *decision* variable indexed by Π such that for each $p \in \Pi$, there is a computation step after which $out[p] = decision_p$, and no following computation step at p ever changes this.

Definition 3.18 (Solving a specification on $HO[\mathcal{HO}]$)**.** Let *spec* be a specification, and A be an algorithm on $HO[\mathcal{HO}]$. A **solves** *spec* **on** $HO[\mathcal{HO}] \triangleq \forall In : \Pi \rightarrow V_{in}, \forall c$ a computation of A on $HO[\mathcal{HO}]$ where the initial state of each p is $init^{A_p}(In(p))$, if *out* is the output of c, then $out \in spec(\{(p, In(p)) \mid p \in \Pi\}, \Pi \setminus \mathcal{SC}(ho))$,

A common criticism of the Heard-Of model lies in it forcing every process to decide, even if they would have crashed in the corresponding asynchronous model. Intuitively, this seems harder to do than only having non-crashed processes deciding. Limiting the decision to strongly correct processes can be criticized as sidestepping this fundamental issue. This is dealt with in Section 3.4, which shows that there are ways to ensure that every process decides.

For the rest of this section, solving a problem in the Heard-Of model is defined as above, forcing decision only for strongly correct processes.

3.3.2 Focus on Strong Completeness

In the rest of this chapter, all failure detectors considered satisfy strong completeness.

Definition 3.19 (Strong completeness (Chandra-Toueg))**.** Let FD be a failure detector. FD satisfies **strong completeness** $\triangleq \forall F$ a failure pattern, $\forall H \in FD(F), \exists t \in \mathbb{N}, \forall p \in crashed(F), \forall q \in correct(F), \forall t' \geq t : p \in H(q, t')$.

This property ensures that a process can wait for messages from unsuspected processes: either the message will arrive, or the process will end up suspected. The alternative is weak completeness, which only requires that for each crashed process, at least one correct process eventually detects the crash.

Definition 3.20 (Weak completeness (Chandra-Toueg))**.** Let FD be a failure detector. FD satisfies **weak completeness** $\triangleq \forall F$ a failure pattern, $\forall H \in FD(F), \exists t \in \mathbb{N}, \forall p \in crashed(F), \exists q \in correct(F), \forall t' \geq t : p \in H(q, t')$.

Focusing on strong completeness might seem very limiting. Yet this is hardly the first time such a focus was acted: Chandra and Toueg themselves [16] focus their paper on them. The reason being that weak completeness can be used to implement strong completeness: every process regularly sends its set of suspected processes to all others. And when a process receives such a set from say process p, the receiver process adds the set to its own set of suspected processes and removes p from it.

I don't include this implementation itself in the simulations because it doesn't work in the Heard-Of model: only trivial predicates (like waiting for at least one message) can be implemented in a communication-closed way on top of a weakly complete failure detector. This is because with weak completeness, the local detector offers no guarantee that a given correct process p will suspect any crashed process – only that some process will suspect each one. And because the Heard-Of model ensures communication-closedness, there is no way to keep exchanging the output of failure detecors until they get strongly complete, as is done in the usual simulation from weak completeness to strong completeness. Strong completeness thus needs to be implemented before simulating the Heard-Of model.

3.3.3 From $HO[\mathcal{HO}]$ to $AMP[\mathcal{FD}]$

The point of this section is to take an algorithm A on $HO[\mathcal{HO}]$ and build from it an algorithm B on $AMP[\mathcal{FD}]$. This is done through a simulation of $HO[\mathcal{HO}]$ running on $AMP[\mathcal{FD}]$. B is then the result of plugging A into this simulation.

This simulation is given in algorithm 3: processes simulate rounds by broadcasting a message, waiting for all the messages of this round from unsuspected processes, and using the call to A to compute the next state and messages to send. Applying this simulation to the failure detectors of interest gives rise to heard-of predicates.

Let's start with the Perfect Failure detector $\mathcal{P}$. In addition to strong completeness (satisfied by every failure detector in this section), $\mathcal{P}$ satisfies strong accuracy.

Definition 3.21 (Strong accuracy (Chandra-Toueg))**.** Let $\mathcal{FD}$ be a failure detector. $\mathcal{FD}$ satisfies **strong accuracy** $\triangleq \forall F$ a failure pattern, $\forall H \in \mathcal{FD}(F), \forall t \in \mathbb{N}, \forall p, q \in \Pi \setminus F(t) : p \notin H(q, t)$.

The heard-of predicate it simulates is defined as follows.

Definition 3.22 (Perfect Heard-Of Predicate)**.** $\mathcal{HO}^{perfect} \triangleq \{h$ a HO collection $\mid \forall r > 0, \forall p \in \Pi : (h(r, p) \neq \emptyset) \wedge (\forall r' \geq r + 2 : h(r', p) \subseteq \bigcap\limits_{q \in \Pi} h(r, q))\}$

This predicate says: if p is not heard by every other process at round r, then it will never be heard by anyone again from round $r + 2$ onwards. The $+2$ additive factor in the Perfect

```
/* Local state for process $i$                                              */
$\langle r, sim\_state\rangle$
/* Intial state of process $i$ in $B_i$ from initial state $q^i_{init}$ for $A_i$   */
$\langle 0, q^i_{init}\rangle$
/* Local algorithm $B_i$ on $AMP[\mathcal{FD}]$                                  */
$B_i(fd, received, \langle r, sim\_state\rangle) =$
    $\begin{cases} (\langle r+1, fst(simA)\rangle, \{(i,j,(r+1,m)) \mid (i,j,m) \in snd(simA)\}) & \text{if } cond\_fd \\ (\langle r, sim\_state\rangle, \emptyset) & \text{otherwise} \end{cases}$
where
    • $cond\_fd \triangleq \forall p \in \Pi \setminus fd, \exists m \in M^{A_i}_{content} : (p, i, (r, m)) \in received.$
    • $simA \triangleq A_i(r, \{(p, i, m) \mid (p, i, (r, m)) \in received\}, sim\_state).$
```

Algorithm 3: Algorithm B on $AMP[\mathcal{FD}]$ simulating an algorithm A on $HO[\mathcal{HO}]$

predicate might be confusing; why 2, and not 1 or 3? What matters here is that perfect failure detectors force correct processes to be at most one round apart, because they all wait for each other. Hence, a process p (at round r) detecting the crash of a process q means that q crashes when it was either at round $r-1$, r or $r+1$. The additive factor of 2 deals with the case where the process p detects at round r the crash of the process q itself at round $r+1$. Because q sent its messages for round r, and possibly also its messages for round $r+1$, there is uncertainty on which messages p will receive from q until round $r+2$.

We can now prove the simulation from $AMP[\mathcal{FD}]$ to $HO[\mathcal{HO}]$.

Theorem 3.23 (Simulation of $HO[\mathcal{HO}^{perfect}]$ on $AMP[\mathcal{P}]$)**.** *Let Spec be a specification. If there is an algorithm A solving Spec on $HO[\mathcal{HO}^{perfect}]$, then there is an algorithm B solving Spec on $AMP[\mathcal{P}]$.*

Proof. Let A be an algorithm solving *Spec* on $HO[\mathcal{HO}^{perfect}]$. The algorithm B is Algorithm 3, using A. We prove that B solves *Spec* on $AMP[\mathcal{P}]$.

Let c be a computation of B on $AMP[\mathcal{P}]$, with inputs In and faulty processes $crashed(c)$. The point is to build a computation c' of A on $HO[\mathcal{HO}^{perfect}]$ which respects the partial order of c. It must also be a computation of $HO[\mathcal{HO}^{perfect}]$, and thus must satisfy the conditions from Definition 3.6: being a distributed computation according to Definition 3.3, that for each process the set of events forms a infinite chain, that the messages provided to each simulated process at each round in c' form a collection of $\mathcal{HO}^{perfect}$, and the atomicity of $HO[\mathcal{HO}^{perfect}]$. Lastly, c' needs to be a computation of A on $HO[\mathcal{HO}^{perfect}]$, which means satisfying the conditions from Definition 3.16.

Then the output of c for B on $AMP[\mathcal{P}]$ will be the output of c' for A running on $HO[\mathcal{HO}^{perfect}]$, and thus that B solves *Spec* by definition of A.

Let's start by defining c'. First, let's define, for each event of c, to what it corresponds in c': no event, one event or multiple events.

- The receptions of messages used by a computation step at p are transformed into the reception event for the corresponding round at p, tagged with the round in the message.
- The receptions of messages never used in any computation step at p are not present in c'.

- The emissions of messages are transformed into the broadcast event for the corresponding round.
- The computation steps at p where no state change happens are removed.
- The computation steps at p where the state changes correspond to changes of rounds and thus are sent to the corresponding computation step at p in c'. They are tagged with the set of messages used in the call to A, and by the value of *sim_state* after the call to A.
- The crashes correspond to a set of broadcast, receptions and computation steps for all rounds greater than the one at which the process crashes. The receptions contain every message sent by non-crashed processes in c for round r.

  This amounts to replacing crashed process by processes that receive every message from non-crashed processes on time, but whose messages are never received on time (because the process is not in the heard-of sets after this round). This ensures the implementation of the right heard-of predicate, while having no influence on the algorithm because these silent processes never send another message that is received.

We only defined the set of events of c'; now let's define its partial order. The partial order of c' is defined by taking the reflexive closure of the order $\leq_{c'}$ defined by: $\forall e_1, e_2 \in c' : e_1 \leq_{c'} e_2$ if and only if:

- e_1 is $\mathit{broadcast}_p^r(m)$ and e_2 is $\mathit{receive}_q^r(M)$, where $m \in M$.
- e_1 is $\mathit{broadcast}_p^r(m)$ and e_2 is $\mathit{receive}_p^r(M)$.
- e_1 is $\mathit{receive}_p^r(M)$ and e_2 is comp_p^r.
- e_1 is comp_p^r and e_2 is $\mathit{broadcast}_p^{r+1}(m)$.

Now if two events e_1 and e_2 in c are such that $e_1 \prec_c e_2$ in c, and these events have the corresponding events e'_1 and e'_2 in c', then $e'_1 \prec_{c'} e'_2$. We can see this through a case-by-case analysis.

- if e_1 and e_2 happen at the same process p then there are multiple cases. We don't deal with the case where the events are a reception at p and an emission at p, as such events are either not ordered in c or follow from the transitivity. Same for the case where e_1 is a computation step at p and e_2 is a reception at p; or any case where e_1 is a crash, as this is impossible by definition of c.
  - if e_1 is the emission of a message at p and e_2 is any computation step at p greater than e_1, then e'_1 is a broadcast at a round smaller than the round of the computation step e'_2, and thus $e'_1 \prec_{c'} e'_2$.
  - if e_1 is a computation step at p and e_2 is any emission at p greater than e_1, then e'_1 is a computation step at a round smaller than the round of the broadcast step e'_2, and thus $e'_1 \prec_{c'} e'_2$.
  - if e_1 is the reception at p of a message used in a computation step and e_2 is this computation step at p, then e'_1 is the reception for the corresponding round and e'_2 is the computation step for this round. And by the definition of c' above, $e'_1 \prec_{c'} e'_2$.

  - if e_1 is any event at p and e_2 is the crash of process p, then for every event e'_2 in the set of events corresponding to the crash, it is either a broadcast, reception or computation step of a round greater than the round where e'_1 happens, and thus $e'_1 \prec_{c'} e'_2$ by definition of c'.

- if e_1 and e_2 happen at different processes, then there is a chain in c from e_1 to e_2. For the parts of the chain that are at some process, the discussion above shows that the order is maintained. The only case of events in c ordered at different process with nothing in between is the case where e_1 is an emission of a message used in some computation step and e_2 is the reception of this message. Then e'_1 is the broadcast where this message is sent and e'_2 the reception containing this message. And by the definition of c' above, $e'_1 \prec_{c'} e'_2$.

Then, one has to show that c' satisfies the properties of a computation of $HO[\mathcal{HO}^{perfect}]$ from Definition 3.6.

- **(Distributed Computation)** A distributed computation of $HO[\mathcal{HO}^{perfect}]$ is also a distributed computation according to Definition 3.3.

  - **(Reception of message used in computation event)** By definition of c', the message used in a computation event are exactly those used by the correponding computation event in c (or the messages from all non-crashed processes for crashed processes). Because c is a distributed computation, these messages were received in c before the computation event according to $\prec_c$. And by definition of c', the reception happens at the reception for the round of the computation event.
  - **(Local emissions and computation steps form a chain)** For each process $p \in \Pi$ and every round r, $\mathit{broadcast}_p^r \prec_{c'} \mathit{comp}_p^r$ by transitivity, and $\mathit{comp}_p^r \prec_{c'} \mathit{broadcast}_p^{r+1}$. Thus the broadcasts and computation steps at p do form a chain.
  - **(If reception, then emission)**. Every reception of c' corresponds to either a reception in c or a crashed event in c. In the first case, the fact that c is a distributed computation and the definition of c' ensures that the emission happens in c and also in c'. In the second case, by definition of the receptions corresponding to crashes, the messages received where sent by some non-crashed process, and thus the emission is in c'.
  - **(Emission smaller than reception)** This follows from the definition of $\prec_{c'}$.

- **(Local events form an infinite chain)** The question here is whether each process has an infinite number of computation steps, that is of changes of round; the specific form of the chain (broadcast, receive, computation step) follows from the definition of $\prec_{c'}$.

  For crashed processes, the crash is replaced by an infinite chain of events, which gives us the property. So we focus on correct processes.

  Let $p \in \Pi \setminus \mathit{crashed}(c)$ be a correct process of c. By definition, p never crashes, and by the strong completeness of $\mathcal{P}$, the failure detector at p eventually detects all crashes. Thus, for each round r and each process q, if p waits long enough, it will either receive the message of q for round r, or q will be suspected by the failure detector at p, because it crashed. Thus p always eventually hear from all non-suspected processes in Π. Then by Definition 3.5 of a computation of $AMP[\mathcal{P}]$, there will be a another computation step at p. At that point, *cond_fd* will be true, and thus calling B will increment the round number and and update the state according to A. Since this happen eventually for each round, p will have an infinite number of computation steps where it updates its state in c, and thus an infinite number of computation steps in c'.

- **(Messages heard at each round form a collection of $\mathcal{HO}^{perfect}$)** We extract a heard-of collection h from c', since every process has an infinite number of rounds. Based on how receptions were defined for crashed process, h is built with the actual received ones for non-crashed processes and the ones defined above for crashed processes. We now show that $h \in \mathcal{HO}^{perfect}$.

  First, because c is an execution of $AMP[\mathcal{P}]$, there are at most $n-1$ crashes, and thus at least one process is correct. It can never be suspected by $\mathcal{P}$ thanks to strong accuracy, and thus every non-crashed process waits for its message at each round. Since all the messages of the correct processes are sent, crashed processes also receive it on time (following the explanation above). So, $\forall r > 0, \forall p \in \Pi : h(r,p) \neq \emptyset$.

  Now we have to show that $\forall r > 0, \forall p \in \Pi : \forall r' \geq r+2 : h(r',p) \subseteq \bigcap\limits_{q \in \Pi} h(r,q)$.

  We do so by contradiction. **Assume** $\exists r > 0$ and $\exists r' \geq r+2$ two rounds, and $\exists p \in \Pi$ a process which has not crashed yet at round r' such that $h(r',p) \nsubseteq \bigcap\limits_{q \in \Pi} h(r,q)$. Thus, $\exists k \in \Pi$ such that $k \in h(r',p) \land k \notin \bigcap\limits_{q \in \Pi} h(r,q)$. This also means $\exists p_K \in \Pi : k \notin h(r,p_K)$.

  Here, p_K has not crashed yet at round r since $k \in h(r',p)$ implies that k has not crashed at round $r < r'$, and thus that it has sent its messages to every process. If p_K had crashed, then by the discussion above, it would have received the message from k before going to the next round. Since it is not the case, p_K has not crashed yet at round r.

  Then $k \notin h(r,p_K)$ means that p_K did not wait for the message from k at round r; on the other hand, p did receive the message from k at round $r' > r$. By Definition of B, this means that the failure detector of p_K suspected k at round r. Which, by strong accuracy of $\mathcal{P}$, means that k crashed when p_K was still at round r.

  Now, at which round was k, at most, when this crash happened? Recall that p_K was not crashed at round r, that is when k crashed. Which means that by strong accuracy, the failure detector at k never suspected p_K. Then k could not have gone beyond round $r+1 < r'$, because at round $r+1$, k had to wait the message of the non-crashed (and thus non-suspected) p_K, and p_K was still at round r, and thus had not sent yet its messages for round $r+1$.

  Then k never reached round r' before crashing, and thus never sent its message to p for this round. Which **contradicts** $k \in h(r',p)$, and thus the hypothesis.

  So, the messages received during the execution of B on $AMP[\mathcal{P}]$ form a collection of $\mathcal{HO}^{perfect}$.

- **(Atomicity)** By Definition 3.6, the atomicity of c' on $HO[\mathcal{HO}^{perfect}]$ depends on the broadcast $broadcast_p^r$, reception $receive_p^r$ and computation step $comp_p^r$ from the same round r. So we're looking for a linearization t' of c' such that for every process p and every round r, there are no other events in t' between $broadcast_p^r$ and $receive_p^r$, or between

$receive_p^r$ and $comp_p^r$. Showing that such a linearization exists is equivalent to showing that no other event e at p exists such that $broadcast_p^r \prec_{c'} e \prec_{c'} receive_p^r \vee receive_p^r \prec_{c'} e \prec_{c'} comp_p^r$.

We prove that no event satisfies the first inequality – the reasoning is analoguous for the second inequality.

Let e' be any event at p in c' distinct from $broadcast_p^r \prec_{c'}$ and $receive_p^r$. By definition of c', e' is tagged with a round number r'.

- If $r' < r$, then $e' \prec_{c'} broadcast_p^r$ by definition of c'.
- If $r' \geq r$, then $received_p^r \prec_{c'} e'$. by definition of c'.

Either way, e' is not between $broadcast_p^r \prec_{c'}$ and $receive_p^r$. Hence such event doesn't exist in c'.

Lastly, we have to show that c' is a computation of A on $HO[\mathcal{HO}^{perfect}]$.

- **(Annotation of computation steps by messages from round)** By definition of c', the messages annotating the computation step of p at round r are the messages received by p in its reception for round r.
- **(Annotation of computation step by state)** By definition of c', each computation step at p is annotated by the simulated state resulting from the application of A_p to the current round number, the set of received messages, and the previous simulated state.
- **(Message to send)** By definition of c', the broadcast of p at round r sends the message given by the application of A_p to $r-1$, the set of received messages at p from round $r-1$ and the simulated state of p before the computation step for round $r-1$ at p.

We conclude that B solves *Spec*. □

Now, there are weakest failure detectors than the perfect one. The classical way to weaken $\mathcal{P}$ is to weaken accuracy.

Definition 3.24 (Weak accuracy (Chandra-Toueg))**.** Let $\mathcal{FD}$ be a failure detector. $\mathcal{FD}$ satisfies **weak accuracy** $\triangleq \forall F$ a failure pattern, $\forall H \in \mathcal{FD}(F), \exists p \in correct(F), \forall t \in \mathbb{N}, \forall q \in \Pi \setminus F(t), p \notin H(q,t)$.

Here is the corresponding heard-of predicate.

Definition 3.25 (Strong Heard-Of Predicate)**.** $\mathcal{HO}^{strong} \triangleq \{h \text{ a HO collection} \mid (\bigcap_{r>0, p\in\Pi} h(r,p)) \neq \emptyset\}$

Theorem 3.26 (Simulation of $HO[\mathcal{HO}^{strong}]$ on $AMP[\mathcal{S}]$)**.** *Let Spec be a specification. If $\exists A$ an algorithm solving Spec on $HO[\mathcal{HO}^{strong}]$, then $\exists B$ an algorithm solving Spec on $AMP[\mathcal{S}]$.*

Proof. The mapping from a computation of B (the Algorithm 3 using A) on $AMP[\mathcal{S}]$ to a computation of A on $HO[\mathcal{HO}^{strong}]$ is defined in the same way as in the proof of Theorem 3.23.

The only parts of this proof that depend on the failure detector is the proof of progress for correct process (but it depends on strong completeness, which $\mathcal{S}$ also ensures), and the collection of received messages. Here, the reasoning is simpler: by weak accuracy, $\exists p \in \Pi$ such that p is a correct process that is never suspected. It always sends its messages at each round, and because it is never suspected, all the processes wait for it.

Hence, $p \in (\bigcap_{r>0, p\in\Pi} h(r,p))$, and thus the generated collection is in $\mathcal{HO}^{strong}$. $\square$

Finally, weaker failure detectors satisfy eventual variations of strong and weak accuracy.

Definition 3.27 (Eventually strong accuracy (Chandra-Toueg)). Let $\mathcal{FD}$ be a failure detector. $\mathcal{FD}$ satisfies **eventually strong accuracy** $\triangleq \forall F$ a failure pattern, $\forall H \in \mathcal{FD}(F), \exists t \in \mathbb{N}, \forall t' \geq t, \forall p, q \in \Pi \setminus F(t'), p \notin H(q,t')$.

Definition 3.28 (Eventually weak accuracy (Chandra-Toueg)). Let $\mathcal{FD}$ be a failure detector. $\mathcal{FD}$ satisfies **eventually weak accuracy** $\triangleq \forall F$ a failure pattern, $\forall H \in \mathcal{FD}(F), \exists p \in correct(F) \exists t \in \mathbb{N}, \forall t' \geq t, \forall q \in \Pi \setminus F(t'), p \notin H(q,t')$.

Definition 3.29 (Eventually Perfect Heard-Of Predicate).
$\mathcal{HO}^{\Diamond perfect} \triangleq \{h \text{ a HO collection} \mid \exists r_\Diamond, \forall r > r_\Diamond, \forall p \in \Pi : (h(r,p) \neq \emptyset) \wedge (\forall r' \geq r+2 : h(r',p) \subseteq \bigcap_{q\in\Pi} ho(r,q))\}$

Definition 3.30 (Eventually Strong Heard-Of Predicate).
$\mathcal{HO}^{\Diamond strong} \triangleq \{h \text{ a HO collection} \mid \exists r_\Diamond > 0 : (\bigcap_{r>r_\Diamond, p\in\Pi} h(r,p)) \neq \emptyset\}$

Theorem 3.31 (Simulation of $HO[\mathcal{HO}^{\Diamond perfect}]$ on $AMP[\Diamond\mathcal{P}]$). *Let Spec be a specification. If $\exists A$ an algorithm solving Spec on $HO[\mathcal{HO}^{\Diamond perfect}]$, then $\exists B$ an algorithm solving Spec on $AMP[\Diamond\mathcal{P}]$.*

Proof. This follows from Theorem 3.23, because $\mathcal{HO}^{\Diamond perfect}$ becomes $\mathcal{HO}^{perfect}$ after some point. Before, no guarantees are required; after, the same guarantees as for $\mathcal{HO}^{perfect}$ hold. $\square$

Theorem 3.32 (Simulation of $HO[\mathcal{HO}^{\Diamond strong}]$ on $AMP[\Diamond\mathcal{S}]$). *Let Spec be a specification. If $\exists A$ an algorithm solving Spec on $HO[\mathcal{HO}^{\Diamond strong}]$, then $\exists B$ an algorithm solving Spec on $AMP[\Diamond\mathcal{S}]$.*

Proof. This follows from Theorem 3.26, because $\mathcal{HO}^{\Diamond strong}$ becomes $\mathcal{HO}^{strong}$ after some point. Before, no guarantees are required; after, the same guarantees as for $\mathcal{HO}^{strong}$ hold. $\square$

3.3.4 From $AMP[\mathcal{FD}]$ to $HO[\mathcal{HO}]$

This simulation (Algorithm 4) is more involved: each process i sends control messages with the message its simulated process attempts to send. If all the processes that i hears from know of these messages, then i runs a simulation step.

Remark 3.33 (Explanation of Algorithm 4). The main difference between Algorithm 3 and Algorithm 4 lies in the nature of messages for the simulated processes: in Algorithm 3, every message given to the call to A is a message received, whereas in Algorithm 4 the messages given to the call to A were sent inside control messages.

So in Algorithm 4, messages sent and received by the algorithm are control messages that contains information about which message each simulated process wants to send. And once enough guarantees are satisfied, these implied messages are delivered to the simulated process.

```
/* Local state for process $i$                                                    */
$\langle sim\_state, fd, sim\_received, view, to\_send \rangle$
/* Intial state of process $i$ in $B_i$ from initial state $q^i_{init}$ for $A_i$   */
$\langle q^i_{init}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
/* Local algorithm $B_i$ on $HO[\mathcal{HO}]$                                     */
$B_i(r, received, \langle sim\_state, fd, sim\_received, view, to\_send \rangle) =$
```

- The new state *new_state* given by

  - $new\_state.sim\_state = \begin{cases} fst(sim\_A) & \text{if } cond\_fd \\ sim\_state & \text{otherwise} \end{cases}$
  - $new\_state.fd = new\_fd$
  - $new\_state.sim\_received =$
    $\begin{cases} sim\_received \\ \quad \cup \{(j,i,m) \mid (j,i,view_j) \in received \wedge (j,i,m) \in view_j\} & \text{if } cond\_fd \\ sim\_received & \text{otherwise} \end{cases}$
  - $new\_state.view = \begin{cases} view \cup (\bigcup_{(p,i,view_p) \in received} view_p) \\ \quad \cup snd(simA) \cup \{(i,i,\text{"}roundr\text{"})\} & \text{if } cond\_fd \\ view \cup (\bigcup_{(p,i,view_p) \in received} view_p) & \text{otherwise} \end{cases}$
  - $new\_state.to\_send = \begin{cases} snd(simA) \cup \{(i,i,\text{"}roundr\text{"})\} & \text{if } cond\_fd \\ to\_send & \text{otherwise} \end{cases}$

- The message to broadcast is *new_state.view*.

where

- $cond\_fd \triangleq \forall p \in (\Pi \setminus new\_fd) : to\_send \subseteq view_p$
- $new\_fd \triangleq \Pi \setminus (\{p \in \Pi \mid \exists (p,i,view_p) \in received\} \cup \{i\})$
- $new\_sim\_received = sim\_received \cup \{(j,i,m) \mid (j,i,view_j) \in received \wedge (j,i,m) \in view_j\}$
- $simA \triangleq A_i(new\_fd, new\_sim\_received, sim\_state)$

Algorithm 4: Algorithm B on $HO[\mathcal{HO}]$ simulating an algorithm A on $AMP[\mathcal{FD}]$

Now let's go over the meaning of the variables. fd is the simulated failure detector; *sim_state* is the state of the simulated process; *sim_received* is the set of messages received in simulation by the simulated process; *view* is the set of all the messages known by the process; *to_send* is the set of messages in the last computation step by the simulated process;

The algorithm initializes the variables in the obvious way, and then at each round, process i broadcasts its *view*. Messages are of the form $(sender, receiver, content)$. Once process i has received the messages for the current round (given by the HO oracle), it updates *view* by taking the union of received views (by construction, the content of a message from a process j is the view from j, $view_j$); it also updates fd to the set of process from whom a message was not received.

If all the heard processes have *to_send* in their *view*, process i runs a step of A. To do so, process i updates *sim_received* by adding all the messages to itself contained in the received views. The simulation step gives the next simulated state, and the new set of messages to send. Notice that process i always adds a new message from itself to itself to *to_send*; this ensures that a process that progresses was actually heard by the other processes, even in the case that its *to_send* was empty.

One additional subtlety here is that Algorithm 4 does not work for simulating $AMP[\mathcal{P}]$ from $HO[\mathcal{HO}^{perfect}]$. Indeed, it might generate executions where crashes are detected a bit in advance, and that invalidates strong accuracy.

Let's thus turn to the weaker failure detectors first. Then a variation of Algorithm 4 will take care of $\mathcal{P}$.

Theorem 3.34 (Simulation of $AMP[\mathcal{S}]$ on $HO[\mathcal{HO}^{strong}]$)**.** *Let Spec be a specification. If* $\exists A$ *an algorithm solving Spec on* $AMP[\mathcal{S}]$*, then* $\exists B$ *an algorithm solving Spec on* $HO[\mathcal{HO}^{strong}]$*.*

Proof. Let A be an algorithm solving *Spec* on $AMP[\mathcal{S}]$. The algorithm B is Algorithm 4 applied to A. We need to show that it indeed solves *Spec* on $HO[\mathcal{HO}^{strong}]$

Let c be a computation of B on $HO[\mathcal{HO}^{strong}]$, with inputs In and corresponding heard-of collection $ho \in \mathcal{HO}^{strong}$. The goal is to build a computation c' of A on $AMP[\mathcal{S}]$ which respects the partial order of c. It must also be a computation of $AMP[\mathcal{S}]$, and thus must satisfy the conditions from Definition 3.5: being a distributed computation according to Definition 3.3, at most $n-1$ crashes, every message sent to a non-crashed process is eventually received in the computation, every process without a crash event has an infinite number of steps, there is a linearization where the failure detector acts like $\mathcal{S}$, and the atomicity of $AMP[\mathcal{S}]$. Lastly, c' needs to be a computation of A on $AMP[\mathcal{S}]$, which means satisfying the conditions from Definition 3.10.

Then the output of c for B on $HO[\mathcal{HO}^{strong}]$ will be the output of c' for A running on $AMP[\mathcal{S}]$, and thus that B solves *Spec* by definition of A.

Let's start by defining c'.

- There is one computation steps in c' for each computation step in c where A is called, annotated with the failure detector output used in the call and the messages in *sim_received*, and the simulated state after the call to A.

- Crashes are for processes which don't have an infinite number of computation steps in c'. They happen just after the last computation step with a call to A (after the possible emissions from the next round)

- Receptions happen at the receiver end just before the first call to A with the corresponding message in *sim_received*. If there is no such step, the reception never happens.
- Emissions happen just after the computation step with a call to A which adds the message to *to_send*; these emissions happen before the possible crash. They also happen if and only if the corresponding receptions also happen in c'.

The partial order of c' is defined by the following rules:

- For every process, the set of local emissions and computation steps form the same chain than in c.
- Reception of a message m is smaller than the emission of m.
- A crash is a maximal element for the process. (this is for free, because no event in c' happens without a computation step in c where A is called, and the crash is bigger than all these events by definition and the first point above.)
- Reception of a message m at p is smaller than the first computation step where m is used.

This partial order is coherent with the partial order on c (except receptions with previous local events). Notice that every event in c' is placed according to a computation step calling A; this is because the messages used in the simulation are abstracted inside the view sent at each round. So the only constraint on the partial order of c' is whether the events in c' corresponding to two computation step cs_1 and cs_2 calling A in c are in the same order than cs_1 and cs_2.

Without loss of generality, let's assume that $cs_1 \prec_c cs_2$. Then we have the following case for e_1 and e_2 in c' corresponding respectively to cs_1 and cs_2.

- If cs_1 and cs_2 happen at the same process p, then $receives(cs_1) \prec_{c'} compStep(cs_1) \prec_{c'} emissions(cs_1) \prec_{c'} compStep(cs_2) \prec_{c'} emissions(cs_2)$ ($\prec_{c'} crash(cs_2)$ if there is a crash), because the set of local emissions and computations steps form the same chain in c' than the computation steps in c.
- If cs_1 and cs_2 happen at different processes p_1 and p_2, then there's a chain of messages from p_1 starting at or after the emission for cs_1 and arriving at p_2 at or after the receives for cs_2. So every event corresponding to cs_1 in c', with the exception of a possible crash, are necessarily smaller by $\prec_{c'}$ than any event corresponding to cs_2, with the exception of other receptions corresponding to cs_2.

Then, we have to show that c' satisfies the properties of a computation of $HO[\mathcal{HO}^{computation}]$ from Definition 3.6.

- **(Distributed Computation)** A distributed computation of $AMP[\mathcal{S}]$ is also a distributed computation. So it satisfies the conditions of Definition 3.3.
    - **(Reception of message used in computation event)** By definition of c', the message used in a computation event are exactly those used by the correponding computation event in c. Because c is a distributed computation, these messages were received in c before the computation event according to $\prec_c$. And by definition of c', the reception happens at some point before the computation event in c'.

- **(Local emissions and computation steps form a chain)** This follows from the fact that this holds in c because it is a distributed computation. Then the definition of c' says that the emissions and computations steps in c' form the same chains than in c.
- **(If reception, then emission)**. Every reception of c' corresponds to a reception in c. Then the fact that c is a distributed computation and the definition of c' ensures that the emission happens in c and also in c'.
- **(Emission smaller than reception)** This follows from the definition of $\prec_{c'}$.

- **(At most $n-1$ crashes)** A process has a crash in c' if it has only finitely many rounds in c where it calls A. And the condition to call A at a round is that every process heard at this round has the messages put in *to_send*.

  By Definition 3.25 of $\mathcal{HO}^{strong}$, there is at least one process s that is in every heard-of set of the collection ho for c. This entails that every one hears the message from this process at every round, and thus when s receive messages, the view of these processes must contain its own *to_send* eventually. Thus s always eventually calls A.

  Hence the processes simulated by s in c' has an infinite number of computation steps, and thus never crashes. We conclude that there are at most $n-1$ crashes in c'.

- **(Every message sent to a non-crashed process is eventually received)** By definition of c', if a message is sent, it is also received. So this is free.

- **(Correct processes have an infinite number of steps)** By definition of c', this means that every correct process of c' is simulated by a process in c that calls A infinitely many itme. And the condition for calling A is that every heard process has the local *to_send* in its view. This can be rephrased by saying that for every correct process p, if it always eventually hear from process q, then q must also always eventually hear from p. That is, p is strongly correct in c.

  We thus show that the strongly correct processes in t are correct processes in t'. Actually, we prove a stronger property that will serve for the failure detector part: $\forall p \in \Pi : p$ strongly correct in $t \iff p$ correct in t'.

  - ($\Rightarrow$) Let $p \in \Pi$ be strongly correct in c. We show that p is correct in c'.

    By definition of strongly correct process, $\forall q \in \Pi : p \overset{\infty}{\rightsquigarrow} q$. This entails that after each simulation step of p, the additional messages it puts in its view are eventually heard by every other process. Thus, every process that p hears from will eventually have heard of these messages, and p will change round.

    We conclude that p calls A an infinite number of time, and thus that p is correct in c'.
  - ($\Leftarrow$) Let $p \in \Pi$ be correct in c'. We show that it is strongly correct in c. We proceed by contradiction: **assume** p is weakly correct. By $\mathcal{HO}^{strong}$, we know that the source s is strongly correct. This entails that $p \overset{\infty}{\not\rightsquigarrow} s \land s \overset{\infty}{\rightsquigarrow} p$.

    By definition of s, p hears from s at each round; and by $p \overset{\infty}{\not\rightsquigarrow} s$, there is a round from which no message from p is ever heard by s. This entails that from this round, the condition for calling A never holds. Thus, p is faulty in c', which contradicts the hypothesis that p is correct in c'.

We conclude that p is strongly correct in c'.

- **(Linearization where Failure Detector in $\mathcal{S}$)** Finally, we need to show that there's a linearization of c' where the values of fd are in a history of S for the corresponding failure pattern. We take t' any linearization of c' satisfying the atomicity of $AMP[\mathcal{S}]$. Because t' only fixes the output of the failure detector at computation steps, the history of the failure detector is actually underspecified. We complete it by saying that no failure detector output changes between the changes visible in computation steps (we interpolate the values by keeping the output constant until the next value).

  - **(Strong completeness)** Let p be a correct process, q be a faulty process and s the source guaranteed by $\mathcal{HO}^{strong}$. Then $\exists r > 0$ such that starting on round r, q never calls A again. This means, among other things, that $q \overset{\infty}{\not\rightsquigarrow} s$. Because if $q \overset{\infty}{\rightsquigarrow} s$, messages from q would always eventually reach s, which would broadcast them in the next round. Then, q would call A.

    Moreover, for messages from q to never again reach s, they must never reach a process $k \in \Pi$ such that $k \overset{\infty}{\rightsquigarrow} s$.

    Now, p is correct. We showed above that this entails it is strongly correct in t. Since $s \overset{\infty}{\rightsquigarrow} p$ by definition of $\mathcal{HO}^{strong}$, we have $p \overset{\infty}{\rightsquigarrow} s$ by definition of strong correctness. Hence, by the reasoning above $q \overset{\infty}{\not\rightsquigarrow} p$. Therefore, p will eventually never hear from q again, thus suspecting it at every round. By our interpolation, the failure detector module at p suspects constantly q from this point on in t'.

    We conclude that fd satisfies strong completeness in t'

  - **(Weak accuracy)** For a process p to suspect a process q, p needs to have not received q's message at the round where A is called. But by definition of $\mathcal{HO}^{strong}$, $\exists s \in \Pi$ a source that is heard by all at each round. We conclude that s is never suspected by any process, and thus that fd satisfies weak accuracy in t'.

- **(Atomicity of $AMP[\mathcal{S}]$)** By Definition 3.5, the atomicity of $AMP[\mathcal{S}]$ depends on the receptions $receives_{cs}$, the next computation step cs and the following emissions $emissions_{cs}$ to be atomic. So we're looking for a linearization t' of c' such that for every process p and every round r, there are no other events in t' between the $receives_{cs}$ and cs, or between cs and $emissions_{cs}$. Showing that such a linearization exists is equivalent to showing that no other event e exists such that for $receives_{cs} \prec_{c'} e \prec_{c'} cs \vee cs \prec_{c'} e \prec_{c'} emissions_{cs}$.

  The first case is impossible because by definition of c', all events e' such that $receives_{cs} \prec_{c'} e'$ are either cs or bigger than cs by $\prec_{c'}$

  Then let's focus on the second one. Let e' be any event in c' distinct from cs and $emissions_{cs}$. Since both events happen at p, if some distinct events are causally between them, then one must happen at p too. So we can assume that e' happens at p, without loss of generality. Then e' cannot be a computation step or an emission because c' respects the chain of computation steps and sends in c. So it's a reception. But by definition of c', if some reception rec is $\prec_{c'}$ than some emission s, there's necessarily a computation step in between. It's not the case here, and thus e' doesn't exist.

Hence there is no such even e, and the linearity exists. We conclude that c' satisfies the atomicity of $AMP[\mathcal{S}]$.

Lastly, we have to show that c' is a computation of A on $AMP[\mathcal{S}]$.

- **(Annotation of computation steps by messages from round)** By definition of c', the messages annotating the computation step of p at round r are the messages used in the call to A_p, which are exactly those received by p before this computation step.
- **(Annotation of computation step by state)** By definition of c', each computation step at p is annotated by the simulated state resulting from the application of A_p to the new value of the failure detector new_fd, the set of received messages, and the previous simulated state.
- **(Message to send)** By definition of c', each emission at p sends a message given by the application of A_p to the new value of the failure detector new_fd, the set of received messages, and the previous simulated state.

There's one last detail one needs to address: the processes for which A must decide in c' are the same than the ones for which B must decide in c. We already proved it: the correct processes of c' are the strongly correct processes of c.

By everything above, since A will solve *Spec* for the correct processes of c', B will solve *Spec* for the strongly correct processes of c (and maybe some other processes as well, but this agree with the definition of faulty processes in a specification). □

Theorem 3.35 (Simulation of $AMP[\Diamond\mathcal{S}]$ on $HO[\mathcal{HO}^{\Diamond strong}]$)**.** *Let Spec be a specification. If $\exists A$ an algorithm solving Spec on $AMP[\Diamond\mathcal{S}]$, then $\exists B$ an algorithm solving Spec on $HO[\mathcal{HO}^{\Diamond strong}]$.*

Proof idea. Same proof as for the previous simulation, with an eventual source instead of just a source. Since all the properties of the source used in the previous proof depended on it being always eventually heard, they transfer to this predicate. □

Proof. The proof follows the same structure as the one for Theorem 3.34; the parts where the specific failure detector or predicate are used are in proving that strongly correct processes in t are correct in c', and in proving that fd satisfies the failure detector properties in c'.

The equivalence of strongly correct in c and correct in c' follows from the same reason as in the proof above, because there is an eventual source, and the proof relies on the eventual existence of such a source.

Finally, we need to show that there's a linearization of c' where the values of fd are in a history of $\Diamond S$ for the corresponding failure pattern. We take t' any linearization of c' satisfying the atomicity of $AMP[\Diamond\mathcal{S}]$. Because t' only fixes the output of the failure detector at computation steps, the history of the failure detector is actually underspecified. We complete it by saying that no failure detector output changes between the changes visible in computation steps (we interpolate the values by keeping the output constant until the next value).

- **(Strong completeness)** Let p be a correct process, q be a faulty process and s the eventual source guaranteed by $\mathcal{HO}^{\Diamond strong}$. Then $\exists r > 0$ such that starting on round r, q never calls A again. This means, among other things, that $q \overset{\infty}{\not\rightsquigarrow} s$. Because if $q \overset{\infty}{\rightsquigarrow} s$, messages from q would always eventually reach s, which eventually becomes a source, and thus would broadcast q's messages in the next round. Then, q would call A.

Moreover, for messages from q to never again reach s, they must never reach a process $k \in \Pi$ such that $k \overset{\infty}{\rightsquigarrow} s$.

Now, p is correct. We showed above that this entails it is strongly correct in c. Since $s \overset{\infty}{\rightsquigarrow} p$ by definition of $\mathcal{HO}^{\Diamond strong}$, we have $p \overset{\infty}{\rightsquigarrow} s$ by definition of strong correctness.

Hence, by the reasoning above $q \overset{\infty}{\not\rightsquigarrow} p$. Therefore, p will eventually never hear from q again, thus suspecting it at every round. Thus by our interpolation, the failure detector module at p suspects q constantly from this point on.

We conclude that fd satisfies strong completeness in t'

- **(Eventual weak accuracy)** For a process p to suspect a process q, p needs to have not receive q's message at the round where A is called. But by definition of $\mathcal{HO}^{\Diamond strong}$, $\exists r > 0, \exists s \in \Pi$ such that s is a source starting from round r. We conclude that s is eventually never suspected again by any process, and thus that fd satisfies eventual weak accuracy in t'.

□

Theorem 3.36 (Simulation of $AMP[\Diamond\mathcal{P}]$ on $HO[\mathcal{HO}^{\Diamond perfect}]$)**.** *Let Spec be a specification. If $\exists A$ an algorithm solving Spec on $AMP[\Diamond\mathcal{P}]$, then $\exists B$ an algorithm solving Spec on $HO[\mathcal{HO}^{\Diamond perfect}]$.*

Proof. The proof follows the same structure as the one for Theorem 3.34; the parts where the specific failure detector or predicate are used are in proving that strongly correct processes in c are correct in c', and in proving that fd satisfies the failure detector properties in some linearization of c'.

The equivalence of strongly correct in c and correct in c' follows from the same reason as in the proof above, because $\mathcal{HO}^{\Diamond perfect}$ entails the existence of an eventual source. Assume there is no eventual source. After the round r where the predicate starts constraining the collection, any process in Π is not heard by at least one process at some round $\geq r$. That entails that $\exists r' \geq r + 2$ such that $\forall r'' \geq r', \forall p \in \Pi : h(r'', p) = \emptyset$. This contradicts the fact that h is a collection of $\mathcal{HO}^{\Diamond perfect}$.

Finally, we need to show that there's a linearization of c' where the values of fd are in a history of $\Diamond P$ for the corresponding failure pattern. We take t' any linearization of c' satisfying the atomicity of $AMP[\Diamond\mathcal{P}]$. Because t' only fixes the output of the failure detector at computation steps, the history of the failure detector is actually underspecified. We complete it by saying that no failure detector output changes between the changes visible in computation steps (we interpolate the values by keeping the output constant until the next value).

- **(Strong completeness)** This follows from the proof of strong completeness for Theorem 3.35, because $\mathcal{HO}^{\Diamond perfect} \implies \mathcal{HO}^{\Diamond strong}$.

  Indeed, if no process is eventually a source, this means that after the round where the predicate starts constraining heard-of sets, all processes will eventually not be heard by everyone in a round. By definition of $\mathcal{HO}^{\Diamond perfect}$, this means that eventually no process will be in any heard-of sets, contradicting the fact that no heard-of set is ever empty.

- **(Eventual strong accuracy)** Let r be the round from which the predicate constrains heard-of sets in c. Then $\forall p \in \Pi$, if p is strongly correct, it is a source starting from r on.

  Assume p is not a source, then $\exists r' \geq r \wedge q \in \Pi : p \notin h(r', q)$. By $\mathcal{HO}^{\Diamond perfect}$, this entails that $\forall r'' > r' + 1, \forall k \in \Pi : p \notin h(r', p)$. Thus, $\forall p \overset{\infty}{\not\rightsquigarrow} k$. By the reasoning above, there is an eventual source $s \in \Pi$. Thus, $s \overset{\infty}{\rightsquigarrow} p \wedge p \overset{\infty}{\not\rightsquigarrow} s$. And by Lemma 3.14, s is strongly correct. This contradicts the strong correctness of p.

  Since all strongly correct processes are sources starting from round r, they are never suspected after round r. Thus, no correct process in t' is ever suspected after round r. We conclude that fd satisfies eventual strong accuracy in t'.

□

For the failure detector $\mathcal{P}$, the simulation is slightly different. The issues stem from strong accuracy: the first simulation (Algorithm 4) might detect crashes a few rounds in advance. This is because q might stop receiving messages from p before the last round of p, and in that case q is not constrained anymore by p and might progress and call A (suspecting p). Although this is not detecting crashes of correct processes, it still invalidates strong accuracy. The trick to solving this problem uses one specificity of $\mathcal{HO}^{perfect}$: weakly correct processes can eventually detect their weak correctness.

Remark 3.37 (Explanation of Algorithm 5)**.** The difference with Algorithm 4 is that processes track the set of processes suspected by everyone they ever heard of, and add it to their own messages. This allows us to deal with the issue of the previous simulation regarding the failure detector $\mathcal{P}$: that processes crashes can be detected slightly in advance.

As long as the processes whose crash has been detected in advance do not hear from the process suspecting them, the execution is indistinguishable from one where the processes crashed before being suspected. Issues arise when the process hears from the one suspecting it, and then sends some messages – the causality here forbids reordering the messages from the soon-to-crash process as happening before the suspicion. The addition to this simulation ensures that if it is the case, the soon-to-crash process will stop being simulated, and thus cannot make causality an issue.

Theorem 3.38 (Simulation of $AMP[\mathcal{P}]$ on $HO[\mathcal{HO}^{perfect}]$)**.** *Let Spec be a specification. If* $\exists A$ *an algorithm solving Spec on* $AMP[\mathcal{P}]$*, then* $\exists B$ *an algorithm solving Spec on* $HO[\mathcal{HO}^{perfect}]$*.*

Proof. The proof follows the same structure as the one for Theorem 3.34; most notably, the computing of c' from c is done the same way. We only change the part using the properties of failure detectors. One difference with the proof of Theorem 3.34 is that we don't show that strongly correct processes in c are exactly the correct processes in c'. Instead we show that strongly correct processes in c are correct in c', and that will suffice to derive that B solves the specification from A's definition.

So we only need to show that strongly correct processes in c are correct in c', and that fd satisfies the failure detector properties in c'.

We first show that strongly correct processes in c are correct in c'. Notice that any process in t that does not always broadcast eventually stops being heard, by definition of $\mathcal{HO}^{perfect}$. Since no heard-of set is empty by definition of $\mathcal{HO}^{perfect}$, it means at least one process always broadcasts. And a source is strongly correct, by Lemma 3.14. Strongly correct processes in

```
/* Local state for process $i$                                                  */
$\langle sim\_state, fd, all\_fd, sim\_received, view, to\_send\rangle$
/* Intial state of process $i$ in $B_i$ from initial state $q^i_{init}$ for $A_i$     */
$\langle q^i_{init}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle$
/* Local algorithm $B$ on $HO[\mathcal{HO}^{perfect}]$                              */
$B_i(r, received\langle sim\_state, fd, all\_fd, sim\_received, view, to\_send\rangle) =$
```

- The new state new_state given by
  - $new\_state.sim\_state = \begin{cases} fst(sim\_A) & \text{if } cond\_fd \\ sim\_state & \text{otherwise} \end{cases}$
  - $new\_state.fd = new\_fd$
  - $new\_state.all\_fd = new\_all\_fd$
  - $new\_state.sim\_received =$ $\begin{cases} sim\_received \cup \{(j,i,m) \mid \\ (j,i,(all\_fd_j, view_j)) \in received \wedge (j,i,m) \in view_j\} & \text{if } cond\_fd \\ sim\_received & \text{otherwise} \end{cases}$
  - $new\_state.view =$ $\begin{cases} view \cup (\bigcup_{(p,i,(all\_fd_p, view_p)) \in received} view_p) \\ \cup snd(simA) \cup \{(i,i,(new\_all\_fd, "roundr"))\} & \text{if } cond\_fd \\ view \cup (\bigcup_{(p,i,(all\_fd, view_p)) \in received} view_p) & \text{otherwise} \end{cases}$
  - $new\_state.to\_send =$ $\begin{cases} snd(simA) \cup \{(i,i,(new\_all\_fd, "roundr"))\} & \text{if } cond\_fd \\ to\_send & \text{otherwise} \end{cases}$
- The message content to broadcast is $(all\_fd, view)$.

where

- $cond\_fd \triangleq i \notin new\_all\_fd \wedge \forall p \in (\Pi \setminus new\_fd) : to\_send \subseteq view_p$
- $new\_fd \triangleq \Pi \setminus (\{p \in \Pi \mid \exists (p,i,(all\_fd_p, view_p)) \in received\} \cup \{i\})$
- $new\_all\_fd \triangleq all\_fd \cup new\_fd \cup (\bigcup_{(p,i,(all\_fd_p, view_p)) \in received} all\_fd_p)$;
- $new\_sim\_received = sim\_received \cup \{(j,i,m) \mid (j,i,(all\_fd_j, view_j)) \in received \wedge (j,i,m) \in view_j\}$
- $simA \triangleq A_i(new\_fd, new\_sim\_received, sim\_state)$

Algorithm 5: Algorithm B on $HO[\mathcal{HO}^{perfect}]$ simulating an algorithm A on $AMP[\mathcal{P}]$

t are those that always broadcast: if they did not, they would eventually never be heard by anyone, including the always broadcasting process mentioned above. That would contradicts their strong correctness.

Therefore, strongly correct processes in t are always heard, and thus the messages they receive always contain their own messages from the previous round. Thus they satisfy the condition $cond\_fd$ an infinite number of times, and they call A an infinite number of times too.

We conclude that strongly correct processes in t are correct in t'.

Finally, we need to show that there's a linearization of c' where the values of fd are in a history of P for the corresponding failure pattern. We take t' any linearization of c' satisfying the atomicity of $AMP[\mathcal{P}]$. Because t' only fixes the output of the failure detector at computation steps, the history of the failure detector is actually underspecified. We complete it by saying that no failure detector output changes between the changes visible in computation steps (we interpolate the values by keeping the output constant until the next value).

- **(Strong completeness)** Let p be a correct process in t' and q be a faulty process in t'. Because q is faulty in t' and thus in c', it is not strongly correct in c by the above. This implies that it eventually stops broadcasting, and by definition of $\mathcal{HO}^{perfect}$, it is eventually never heard again by other processes.

  Thus, $\exists r > 0$ such that starting at round r, p does not hear q. Hence, q is always added to fd in computation steps from this round on, and it is always suspected by p for all the following calls to A.

  We conclude that fd satisfies strong completeness in t'.

- **(Strong accuracy)** Strong accuracy requires that every suspected process has already crashed. Proving that a linearization satisfying this property exists means proving that there's no causal chain from the event where q suspects p to p and back to q again. Or said differently, no computation step or emission at p are greater by $\prec_{c'}$ than the computation step where q suspects p. Then for some choice of t', fd satisfies strong accuracy in t'.

  We proceed by contradiction: **assume** that some event at p, either a computation step or an emission, happens by $\prec_{c'}$ in between p being suspected by q and p's crash.

  By definition of causality in asynchronous distributed systems, this means that there is a chain of messages from q to p, starting after the suspicion at q and reaching p before the assumed event at p. By definition of Algorithm 5, each message in this chain will transport the all_fd of the sender, and thus the final message reaching p will contain p in its all_fd, because p is in all_fd_q. Then the condition $cond\_fd$ for calling A will always be false for p, and thus it will never make another call to A.

  This **contradicts** the hypothesis that p makes another computation step or emission.

  We thus conclude that for some choice of t' (it doesn't matter which one for strong completeness), fd satisfies strong accuracy in t'.

$\square$

3.4 Extending to the All-Deciding Case

As mentioned above, one of the specificities of the simulations from the previous section is that not every process needs to decide in the definition of solvability for the Heard-Of model. It is only required that the strongly correct processes (the ones which can communicate which each other infinitely often) decide. However, strictly speaking, there is no crash in the Heard-Of model, only forever silent or unheard processes. Thus, it seems natural to ask that all processes decide. In this section, I explore two ways of converting the partial solutions given by the simulations of the previous section into full solutions.

Definition 3.39 (Strongly Solving a Specification on $HO[\mathcal{HO}]$)**.** Let *spec* be a specification, and A be an algorithm for $HO[\mathcal{HO}]$. A **strongly solves** *spec* **on** $HO[\mathcal{HO}] \triangleq \forall in : \Pi \to V_{in}, \forall out$ the output of A: $out \in spec(In, \emptyset)$.

3.4.1 Full Monotony: Extending the Final Decisions

For some specifications, given a set of values decided by the strongly correct processes, each weakly correct process can decide a value coherent with theses decisions, and with all the concurrent decisions. I call this full monotony, and it underlies the first approach for strongly solving specifications.

Definition 3.40 (Full Monotony)**.** A specification (*spec*) on Π is **fully-monotone** $\triangleq \exists dec : (\Pi \times \mathcal{P}(\Pi \times V_{out})) \to V_{out}$:
$\forall in : \Pi \to V_{in}, \forall F \subsetneq \Pi, \forall out \in spec(in, F), \forall out' \in spec(in, \emptyset)$ such that $\forall p \in \Pi \backslash F : out'(p) = out(p), \forall p \in \{q \mid out(q) = \bot\}, \exists out_{full} \in spec(in, \emptyset)$:

$$\left(\begin{array}{ll} & out_{full}(p) = dec(p, \{(q, v) \mid v \in V_{out} \wedge out(q) = v\}) \\ \wedge & \forall q \neq p : out_{full}(q) = out'(q) \end{array} \right)$$

Theorem 3.41 (Simulation of $AMP[\mathcal{FD}]$ on $HO[\mathcal{HO}]$ for Fully-monotone Specifications)**.** *Let spec be a fully-monotone specification. Let* $(\mathcal{FD}, \mathcal{HO}) \in \{(\mathcal{P}, \mathcal{HO}^{perfect}), (\mathcal{S}, \mathcal{HO}^{strong}), (\Diamond\mathcal{P}, \mathcal{HO}^{\Diamond perfect}), (\Diamond\mathcal{S}, \mathcal{HO}^{\Diamond strong})\}$. *If* $\exists A$ *an algorithm solving spec on* $AMP[\mathcal{FD}]$*, then* $\exists B$ *an algorithm strongly solving spec on* $HO[\mathcal{HO}]$.

Proof idea. Being full-monotone ensures that weakly correct processes can decide coherently when given the decision of some strongly correct processes. In the considered predicates, there is an eventual source, which is strongly correct by Lemma 3.14. Hence after the decision of some strongly correct process, this decision value is propagated to everyone to decide accordingly. □

Proof. Thanks to the simulations in the previous section, for such an algorithm A, we have B' solving *spec* on $HO[\mathcal{HO}]$. The way to go from B' to B is for every process that decides in a call to A to stop simulating, and to broadcast its decision at each round (in a special form, so that the message can be distinguished from a normal view). Every process that receives a decision decides for itself and propagates this decision.

By full-monotony, there is a function *dec* that, given a decision for a subset of processes, returns a decision value for the other processes coherent with what has been decided. Thus, the only issue is to know whether every process will receive a decision value.

This follows from the fact that all our predicates have an eventual source, and such an eventual source is strongly correct by Lemma 3.14. Since the simulation from A to B' ensures the decision of strongly correct processes, the source will finish and propagate its decision value.

We conclude that B strongly solves *spec* on $HO[\mathcal{HO}]$. □

Corollary 3.42 (Simulation of Consensus, Set-agreement, Leader Election and Splitter)**.** *Let* $(\mathcal{FD}, \mathcal{HO}) \in \{(\mathcal{P}, \mathcal{HO}^{perfect}), (\mathcal{S}, \mathcal{HO}^{strong}), (\Diamond\mathcal{P}, \mathcal{HO}^{\Diamond perfect}), (\Diamond\mathcal{S}, \mathcal{HO}^{\Diamond strong})\}$*. For spec the specification of consensus, set-agreement, leader election or splitter, if* $\exists A$ *an algorithm solving spec on* $AMP[\mathcal{FD}]$*, then* $\exists B$ *an algorithm strongly solving spec on* $HO[\mathcal{HO}]$*.*

Proof. For each problem, we show the full monotony by giving an explicit *dec* function.

- **Consensus**: The *dec* function takes the set of decisions, use the sole decided value (since the decisions solve consensus) as the decision for the current process. This maintains correctness, since every process still decides the same value.
- **Set-agreement**: For each set of decided value and each process, *dec* returns one of the decided value. This maintains correctness, since the number of decided values stays the same.
- **Leader Election**: The output is the leader name, *dec* is the same as for consensus.
- **Splitter** [63]: If only *right* (respectively *down*) was decided, *dec* returns *down* (respectively *right*); otherwise (both *right* and *down* values, and/or presence of a *stop*), *dec* returns *right* or *down* arbitrarily.

□

However, not all specifications are fully-monotone, for instance renaming [64], where processes must all choose a distinct name from a namespace, isn't. Weakly correct processes cannot decide from the decision values of the strongly correct processes alone, because the decision values also need to be distinct. So two weakly correct processes risk deciding the same value.

This motivates another approach for strongly solving specifications.

3.4.2 Local Specifications: Agreeing on a Full Decision

Instead of extending a partial decision, processes could agree on a complete decision from the start. That is, if consensus is solvable, and if each process by itself can produce a possible complete decision coherent with the initial values (without knowing them, except its own), then the result above for strongly solving consensus would transfer to other specifications. A local specification is such a specification, where from a single input value, there is at least one allowed output vector coherent with any input with this single value.

Definition 3.43 (Local Specification)**.** Let *spec* be a specification on Π. *spec* is a **local specification** $\triangleq$ there is a function $local\_dec : (\Pi \times V_{in}) \to (\Pi \to V_{out})$ such that $\forall p \in \Pi, \forall v_{in} \in V_{in}, \forall in : \Pi \to V_{in} : in(p) = v_{in} \implies local\_dec(p, v_{in}) \in spec(in, \emptyset)$.

Theorem 3.44 (Simulation of $AMP[FD]$ on $HO[\mathcal{HO}]$ for Local Specifications)**.** *Let spec be a local specification. Let* $(\mathcal{FD}, \mathcal{HO}) \in \{(\mathcal{P}, \mathcal{HO}^{perfect}), (\mathcal{S}, \mathcal{HO}^{strong}), (\Diamond\mathcal{P}, \mathcal{HO}^{\Diamond perfect}), (\Diamond\mathcal{S}, \mathcal{HO}^{\Diamond strong})\}$*. If* $\exists A$ *an algorithm solving consensus on* $AMP[\mathcal{FD}]$*, then* $\exists B$ *an algorithm strongly solving spec on* $HO[\mathcal{HO}]$*.*

Proof. It follows from Corollary 3.42: if there is an algorithm solving consensus on $AMP[\mathcal{FD}]$, it can be strongly solved on $HO[\mathcal{HO}]$. For a local specification, each process can compute an allowed output vector from its input value. Then a consensus is solved on these proposed output vectors, and every process decides the value corresponding to itself in the chosen vector. □

Renaming, although not fully-monotone, is a local specification: any naming scheme in the namespace is a possible complete decision. Hence, renaming is also solvable with the predicates corresponding to the failure detectors of the Chandra-Toueg hierarchy.

Corollary 3.45 (Simulation of Renaming). *Let* $(\mathcal{FD}, \mathcal{HO}) \in \{(\mathcal{P}, \mathcal{HO}^{perfect}), (\mathcal{S}, \mathcal{HO}^{strong}), (\Diamond\mathcal{P}, \mathcal{HO}^{\Diamond perfect}), (\Diamond\mathcal{S}, \mathcal{HO}^{\Diamond strong})\}$. *If* $\exists A$ *an algorithm solving consensus on* $AMP[\mathcal{FD}]$*, then* $\exists B$ *an algorithm strongly solving renaming on* $HO[\mathcal{HO}]$.

Proof. Consensus is a fully-monotone specification. Hence by Corollary 3.42, there is an algorithm B_{cons} strongly solving consensus for each of the corresponding heard-of predicates. Then B is simply the algorithm of running B_{cons} where each process proposes an allowed output vector to renaming, and decides its name in the chosen vector at the end. This works because renaming is a local specification: any allocation of distinct names to processes is an allowed output vector. □

Every example given of fully-monotone specification is also a local specification. So why not limit ourselves to the latter? First, the value of a local specification depends on strongly solving consensus, which is proven using the full-monotony of consensus. Secondly, this dependence on consensus means that local specification probably generalizes less to other predicates than full-monotony: solving consensus is a really strong requirement.

3.5 Conclusion

3.5.1 Summary

The equivalences proven above are evidence than for many important problems, the Heard-Of model is at least as powerful as asynchronous models with failure detectors. If the decision is limited to strongly correct processes, then the equivalence is a straightforward equivalence between the predicates given here and the failure detectors from the Chandra-Toueg hierarchy. When forcing everyone to decide, many interesting problems like consensus and renaming stay equivalent in terms of computability between the two models. Therefore, for these important cases, forcing communication-closed rounds doesn't weaken the power of the model.

3.5.2 History of the research

Every aspect of this research struggled because of the Perfect failure detector. I wanted to find corresponding predicates for the Chandra-Toueg hierarchy from the start of my thesis, but the one for the Perfect failure detector kept eluding me. I probably discovered the final version a couple of times, but failed to convince myself that it was the right one.

Similarly, this failure detector forced me to lower my ambitions about these results. Instead of proving a sweeping equivalence with all failure detectors using the same simulation, I had to deal with a specific simulation for this one alone, and thus to limit the previous equivalence to concrete failure detectors, since it didn't generalize completely.

3.5.3 Perspectives

Among remaining issues are the generalization of the equivalence to other failure detectors, and the lookout for problems that are neither local nor fully-monotone.

Chapter 4

Leveraging Heard-Of Predicates

Sommaire

4.1 Introduction

4.1.1 Motivation

As seen in the previous chapters, the Heard-Of model uses the concept of round for formalizing many different models within a common framework. What's left to do is to prove results on these formalized models. Researchers provided, among other things, new algorithms for consensus in the original paper by Charron-Bost and Schiper [37]; characterizations for consensus solvability by Coulouma et al. [41] and Nowak et al. [42]; a characterization for approximate consensus solvability by Charron-Bost et al. [39]; a study of k-set agreement by Biely et al. [44]; and more.

But the techniques used for proving these results tend to be ad-hoc – very specific to some model or setting. Hence every new paper carry the burden of finding new techniques. What would help here is a general approach to proving impossibility results and bounds on round-based models.

Actually, there is at least one example of a general mathematical technique used in this context: the characterization of consensus solvability through point-set topology by Nowak et al. [42].

I propose what might be seen as an extension to higher dimension of this intuition, by applying combinatorial topology (instead of point-set topology) to bear on k-set agreement (instead of just consensus).

Combinatorial topology abstracts the reasoning around knowledge and indistinguishability behind many impossibility results in distributed computing. It thus provide generic mathematical tools and methods for deriving such results [19]. Moreover, this approach is the only one that managed to prove impossibility results and characterization of solvability of the k-set agreement [65], our focus problem.

4.1.2 Overview

Concretely, this chapter focus on closed-above heard-of predicates, that is predicates where constraints happen round-per-round, and the set of communication graphs allowed is the closure-above of a set of graphs. These predicates capture many safety properties: the ones requiring some underlying structure in communication, like having an underlying star, ring or tree. This is a strict generalization of the predicates with a fixed communication graph considered by Castañeda et al. [66].

For this class of predicates, I derive upper bound and lower bounds on the k for which k-set agreement is solvable. And while the proofs of the bounds use combinatorial topology, they are stated in terms of variants of the domination number, a well-known and used combinatorial number on graphs.

- First closed-above predicates are defined in Section 4.2.
- Then various upper bounds on k for k-set agreement in one round are proved in Section 4.3. These bounds correspond to algorithms solving the k-set agreement in one round. I present them first because they do not rely on combinatorial topology, and they serve to introduce combinatorial numbers of use later.
- Next, Section 4.4 introduces the combinatorial topology necessary for our lower bounds, both the basic definitions and the main technical lemma.
- Then its lower bounds on k which are proved, still for one round, in Section 4.5. These bounds leverage combinatorial topology, but are state in terms of graph properties only. these bounds use combinatorial topology, but are stated in terms of graph properties.

- Finally, Section 4.6 generalizes both upper and lower bounds to the case of multiple rounds.

The content of this chapter comes from the paper "K-set agreement bounds in round-based models through combinatorial topology" published at PODC 2020, and written with Armando Castañeda.

4.1.3 Related Works

Combinatorial Topology Combinatorial Topology was first applied to the problem of k-set agreement in wait-free shared memory by Herlihy and Shavit [19], Saks and Zaharoglou [67] and Borowsky and Gafni [68].

Beyond these first forays, many other results got proved through combinatorial topology. Among others, we can cite the lower bounds for renaming by Castañeda and Rajsbaum [69] and the derivation of lower-bounds for message-passing by Herlihy and Rasjbaum [70]. There is even a result by Alistarh et al. [71] showing that traditional proof techniques (dubed extension-based proofs) cannot prove the impossibility of k-set agreement in specific shared-memory models, whereas techniques from combinatorial topology can.

For a full treatment of combinatorial topology applied to distributed computing, see Herlihy et al. [19].

Combination of Topology and Round-based models Two papers at least applied topology (combinatorial or not) to general round-based models in order to study agreement problems: Godard and Perdereau [72] and Nowak et al. [42].

Godard and Perdereau [72] used similar tools of combinatorial topology to study k-set agreement in message adversary models; the difference lies in the constraints they consider on communication. Whereas we focus on closed-above predicates, they limit themselves to models with at most f omission failures in a fixed arbitrary communication graph. The two are actually incomparable: when the fixed graph is not the full graph, their model is not closed-above; and closed-above models might have additional constraints than a number of possible message loss.

Nowak et al. [42] proposed a characterization of consensus for general round-based models (not necessarily oblivious) using point-set topology. Their results are both stronger (because it is a complete characterization for general predicates) and weaker (because it only treats consensus) than ours. The point-set topology approach seems limited to 1-set agreement (consensus), and going to k-set agreement might require combinatorial topology.

4.2 Definitions

Every result in this chapter assumes a model where processes know all identities of other processes, and notably know from who they received each message.

4.2.1 Closed-above predicates

The predicates studied here are of a restricted form, where the graph for each round is decided independently of the other rounds. The predicate is thus entirely characterized by the set of allowed graphs. I call these predicates *oblivious*, following Coulouma et al [41].

Definition 4.1 (Oblivious predicates)**.** Let $\mathcal{HO}$ be a communication model. Then $\mathcal{HO}$ is **oblivious** $\triangleq \exists S \subseteq Graphs_\Pi : \mathcal{HO} = S^\omega$.

Intuitively, oblivious predicates capture safety properties: bad things that must not happen. Or equivalently, good things that must happen at every round. Usually, these good properties are related to connectivity, like containing a cycle or a spanning tree. Since such a property tends to be invariant when more messages are sent, a natural constraint on oblivious predicates is to be defined by a set of subgraphs.

Definition 4.2 (Closed-above predicates)**.** Let $\mathcal{HO}$ be an oblivious predicate. Then $\mathcal{HO}$ is **closed-above** $\triangleq \exists S \subseteq Graphs_\Pi : \mathcal{HO} = (\bigcup\limits_{G \in S} \uparrow G)^\omega$, where $\uparrow G \triangleq \{H \mid V(H) = V(G) \wedge E(H) \supseteq E(G)\}$.

We call the graphs in S the **generators** of $\mathcal{HO}$.

If S is a singleton, then $\mathcal{HO}$ is **simple closed-above**.

Classical examples of closed-above predicates are the non-empty kernel predicate $\mathcal{HO}^{nek}$ (see Table 1.1) and the non-split predicate $\mathcal{HO}^{nosplit}$ (see Table 1.1), used notably by Charron-Bost et al. [39] for characterizing the solvability of *approximate consensus* – the variant of consensus where the decided values should be less than ε apart, where $\varepsilon > 0$ is fixed beforehand. Another closed-above predicate is the one satisfying the tournament property of Afek and Gafni [32], which they show is equivalent to wait-free read-write shared memory. In general, oblivious predicates where the constraint on rounds is about how much message can be lost/not on-time are closed-above.

Which segues into an example of an oblivious predicate which is **not** closed-above: the one generated by all graphs containing a cycle, except the clique. Here the constrains not only limits which messages might be lost/on-time, but also forces at least one such loss.

Closed-above predicates also have a good trade-off between expressivity and simplicity, since the "combinatorial data" used to build them is contained in a small number of graphs.

Finally, the patterns expected by safety properties tend to be independent of which processes play which roles – what matters is the existence of a ring or spanning tree, not who is where on it. I call these closed-above predicates *symmetric* predicates: their set of possible graphs possible for each round is closed under permutation.

Definition 4.3 (Symmetric predicates)**.** Let $\mathcal{HO}$ be a closed-above predicate, and S be the set of graphs generating it. Then $\mathcal{HO}$ is **symmetric** $\triangleq S = Sym(S)$, where $Sym(S) = \{\pi(G) \mid G \in S \wedge \pi : Graphs_\Pi \rightarrow Graphs_\Pi$ a permutation on graphs permuting the processes$\}$.

In the rest of the paper, I only consider closed-above predicates, both symmetric and not.

4.2.2 Oblivious algorithms

Because most applications of combinatorial topology to distributed computing aim towards impossibility results, the traditional algorithms considered err on the side of power: full information protocols, which exchange at each round the view of everything ever heard by the process. For example, after a couple of rounds, views will contain nested sets of views, themselves containing views, recursively until the initial values.

In contrast, I focus on oblivious algorithms. That is, each process only remembers the initial values and their corresponding processes, not who sent them or when. This information amounts to a function from Π to the set of initial values (with a $\perp$ when the value is not known). In turn, these algorithms lose the ability to trace the path of the value.

Oblivious algorithms can be viewed as full-information protocols whose decision map (the function from final view to decision value) depends only on the set of known pairs (process,initial value). The full-information protocol might still be used for deciding when to apply the decision map, but this map loses everything except the known pairs (process,initial value). That is, the decision map is constrained to decide similarly in situations where it received the same information about the initial configuration, whatever the history of messages.

Definition 4.4 (Oblivious algorithm)**.** Let $\mathcal{A}$ be a full-information protocol, with decision map δ. Then $\mathcal{A}$ is an **oblivious algorithm** $\triangleq \forall v$ a view: $\delta(v) = \delta(flat(v))$,
where $flat(v) = \bigcup\limits_{(p,v_p)\in v} flat((p, v_p))$
and $flat((p, v_p)) = \begin{cases} \{(p, v_p)\} & \text{if } v_p \text{ is a singleton from } V_{in} \\ flat(v_p) & \text{otherwise} \end{cases}$

The corresponding definition of solvability is that there is an algorithm, a number of rounds r and a decision map such that after running the algorithm for r rounds, applying the decision map gives decision that form an accepted output.

4.2.3 K-set agreement

The focus problem of this chapter is k-set agreement – the weakening of consensus where at most k different values can be decided.

Definition 4.5 (K-set agreement)**.** An algorithm A solves the k-set agreement for $k > 0 \triangleq$ the decided values satisfy the following two properties:

- **Validity**. Every decided value is an initial value.
- **Agreement**. The set of decided values is of size $\leq k$.
- **Termination**. Every process decides eventually some value

This choice of problem follows from two considerations. First, Nowak et al. [42] already completely solved the consensus for heard-of predicates (message adversaries in their terminology). The problem thus cannot be consensus, and k-set agreement is the usual choice to go beyond consensus. Second, k-set agreement is the most important problem for which combinatorial topology techniques seems necessary. Neither the failure detector approach nor the knowledge-based approach managed to extract as much results as combinatorial topology for this problem. Hence it feels like both an interesting and promising problem to study here.

4.3 One round upper bounds: a start without topology

Although lower bounds on k are the target, they require upper bounds to gauge their strength. Let's thus start with upper bounds on k-set agreement for closed-above predicates. Another advantage of starting with these upper bounds is that they rely on concrete algorithms, and also use generalizations of the classical domination number that will be used for our lower bounds. Lastly, the bounds here in this section and the next one only work on the one round case. Bounds for multiple rounds depend on these one round bounds.

These bounds follow from a very simple algorithm for solving k-set agreement.

Definition 4.6 (One round k-set agreement algorithm)**.** Assume that the set of initial values to k-set agreement is totally ordered. Then the **one round k-set agreement algorithm** $1 - round \triangleq$ the algorithm where each process

- broadcasts its initial value;
- and decides the minimum value it received.

4.3.1 Simple closed-above predicates: almost too easy

Recall that the domination number of a graph is the size of its smallest dominating set, that is the minimum size of a set of nodes whose set of outgoing neighbors is Π. Note that graphs here have self-loops – that is, the outgoing neighbors of a set $S \subseteq \Pi$ contains S. For ease of representation, these loops are not drawn in the figures.

Definition 4.7 (Domination number)**.** Let G be a graph. Then its **domination number** $\gamma(G) \triangleq min\{i \in [1, n] \mid \exists P \subseteq \Pi : |P| = i \wedge \bigcup\limits_{p \in P} Out_G(p) = \Pi\}$.

Because the simple closed-above predicate generated by G only allows graphs containing G, these graphs' domination number is at most $\gamma(G)$. This entails a very simple upper bound on k-set agreement. The algorithm depends explicitly on G: it is parameterized with a minimum dominating set of G, which serve to synchronize the decided values among processes.

Definition 4.8 (One round k-set agreement algorithm)**.** Assume that the set of initial values to k-set agreement is totally ordered. Then the **one round k-set agreement algorithm parametrized by a set of process**, $1 - roundParam \triangleq \forall S \subseteq \Pi$, $1 - roundParam(S)$ is the algorithm where each process

- broadcasts its initial value;
- and decides the minimum value it received from a process of S. If no such value were received, it decides the minimal value it received from any process.

Theorem 4.9 (Upper bound on k-set agreement by $\gamma(G)$)**.** *Let G be a graph, and let D be a minimum dominating set of G. Then Algorithm $1 - roundParam(D)$ solves $\gamma(G)$-set agreement in one round on the simple closed-above predicate generated by G.*

Proof. Because D is a dominating set, every process receives at least one value from it, so every process decides a value from a process of D. Finally, since the minimum dominating set has at most $\gamma(G)$ distinct values, at most $\gamma(G)$ values are decided, and thus $1 - roundParam(D)$ solves $\gamma(G)$-set agreement. □

The tightness of this bound follows from Castañeda et al. [66, Thm 5.1]: the oblivious predicate with a single graph G cannot solve k-set agreement in one round for $k < \gamma(G)$. Hence the weaker simple closed-above predicate generated by G cannot solve it either.

Still, simple closed-above predicates are somewhat artificial, as can be seen in the proof: the algorithm is parametrized with a precomputed minimum dominating set of the known subgraph contained in the actual communication graph. A more realistic take would add uncertainty to which graph is the underlying subgraph; the next step is thus general closed-above predicates.

4.3.2 General closed-above predicates: tweaking of upper bounds

For general closed-above predicates, there is not a single subgraph but a set of possible underlying subgraphs. This makes the previous approach inapplicable: the algorithm cannot hardcode a dominating set because the underlying subgraph is not known.

This new issue motivates the definition of a weakening of the domination number: the equal-domination number of a set of graphs. Intuitively, any set of that many processes forms a dominating set in all the graphs considered.

Definition 4.10 (Equal-Domination number of a set of graphs)**.** Let S be a set of graphs. Then its **equal-domination number** $\gamma^{eq}(S) \triangleq \max\limits_{G \in S} \gamma^{eq}(G)$, where $\gamma^{eq}(G) = min\{i \in [1,n] \mid \forall P \subseteq \Pi : |P| = i \implies \bigcup\limits_{p \in P} Out_G(p) = \Pi\}$.

Then the algorithm from Definition 4.6 solves $\gamma^{eq}(S)$-set agreement in one round for the closed-above predicate generated by S – and it doesn't need a minimum dominating set as a parameter.

Theorem 4.11 (Upper bound on k-set agreement by $\gamma^{eq}(S)$ for general closed-above predicates)**.** *Let S be a set of graphs. Then Algorithm $1-round$ solves $\gamma^{eq}(S)$-set agreement in one round on the closed-above predicate generated by S.*

Proof. Let P be a set of $\gamma^{eq}(S)$ processes with the smallest initial values. They have thus at most $\gamma^{eq}(S)$ distinct initial values. By Definition 4.10 of $\gamma^{eq}(S)$, P dominates every graph in S, and thus every graph in the closed-above predicate generated by S.

Thus taking the minimum after one round will result in deciding one of those initial values, and thus one of at most $\gamma^{eq}(S)$ values. We conclude that the algorithm from solves $\gamma^{eq}(S)$-set agreement after one round on the closed-above predicate generated by S. □

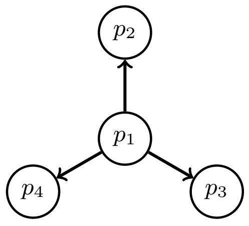
Since the equal-domination number is independent of which process does what, it is the same for any permutation of the graph. This entails an upper bound on symmetric predicates as a corollary.

Corollary 4.12. *Let S be a set of graphs. Then Algorithm $1-round$ solves $\gamma^{eq}(S)$-set agreement in one round on the closed-above predicate generated by $Sym(S)$.*

Now, the natural question to ask is the tightness of this bound.

The answer depends on the graphs. To see it, let's look at another combinatorial number: covering numbers. Given fewer processes than the equal-domination number of the graph, they do not always form a dominating set. Nonetheless, they might still get heard by some minimum number of processes. We call such minimums the covering numbers of the graph: the i-th covering number of G is, given any set of i processes, the minimum number of processes hearing this set in G.

Definition 4.13 (Covering numbers of a set of graphs)**.** Let S be a set of graphs. Then $\forall i < \gamma^{eq}(S)$, its i-th covering number $cov_i(S) \triangleq \min\limits_{G \in S} cov_i(G)$, where $cov_i(G) \triangleq \min\limits_{\substack{P \subseteq \Pi \\ |P|=i}} |(\bigcup\limits_{p \in P} Out_G(p))|$.



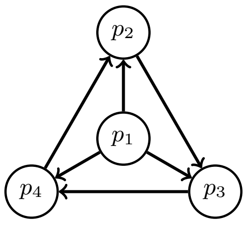


Figure 4.1: Two examples of communication graphs

These numbers capture the ability of a set of processes to disseminate their values in the graph. Taking the i processes with the smallest initial values, at least $cov_i(S)$ processes will hear them, and thus choose one of the i smallest initial values. This then gives a solution to $(i + (n - cov_i(S)))$-set agreement in one round.

Theorem 4.14 (Upper bounds on k-set agreement by covering numbers for general closed-above predicates)**.** *Let S be a set of graphs. Then $\forall i \in [1, \gamma^{eq}(S)[$: Algorithm $1-round$ solves $(i + (n - cov_i(S)))$-set agreement in one round on the closed-above predicate generated by S.*

Proof. For a set of i processes with the i smallest initial values, they will reach at least $cov_i(S)$ processes after the first round. Thus these processes will decide one of the i values when taking the smallest value they received.

As for the rest of the processes, we can't say anything about what they will receive, and thus we consider the worst case, where they all decide differently, and not one of the i smallest values. Then the number of decided values is at most $i + (n - cov_i(S))$, and the theorem follows. □

The covering numbers are also independent of processes names; this entails a similar upper bound on symmetric predicates as a corollary.

Corollary 4.15. *Let S be a set of graphs. Then $\forall i \in [1, \gamma^{eq}(S)[$: Algorithm $1-round$ solves $(i + (n - cov_i(S)))$-set agreement in one round on the closed-above predicate generated by $Sym(S)$.*

When is this new bound better than the one using the equal-domination number? When there is some i such that $n - cov_i(S) < \gamma^{eq}(S) - i$. Let's take the symmetric predicates generated by the two graphs in Figure 4.1.

In the first predicate, $n - cov_i(S) < \gamma^{eq}(S) - i$ never happens, because every covering number of a star equals 1 (the biggest set of outgoing neighbors different from Π contains only one process), and its equal-domination number equals n (because when taking only $n-1$ processes, the center of the star might not be in there). Thus $n - cov_i(S) = n - 1 \geq \gamma^{eq}(S) - i = n - i$.

On the other hand, this is the case in the second predicate, because $cov_2(S) = 3$ and $\gamma^{eq}(S) = 4$. Thus $n - cov_2(S) = 4 - 3 = 1 < \gamma^{eq}(S) - i = 4 - 2 = 2$. Hence the upper bound with covering numbers ensure 3-set agreement solvability while the upper bound with the equal-domination number only ensures 4-set agreement solvability.

4.3.3 Intuitions on upper and lower bounds

Why do these upper bounds hold? Because the underlying graphs betray some minimal connectivity of sets of processes, in the form of combinatorial number. From these, it follows how far the minimal values will spread in the worst case, and thus the bound on the maximum number of values decided.

On the other hand, the lower bounds will follow from studying how much values can spread in the best case. Why? Because the more values can spread, the more processes can distinguish between initial configurations, and the more they have a chance to decide correctly. Ensuring enough indistinguishability thus entails an impossibility at solving k-set agreement.

This indistinguishability is linked to higher-dimension connectivity in combinatorial topology [19, Thm. 10.3.1]; Let's thus turn to the topological approach to distributed computing for the lower bounds.

4.4 Elements of combinatorial topology

4.4.1 Preliminary definitions

First, we need to introduce the mathematical objects that this approach uses. These are simplexes and complexes. A simplex is simply a set of values, and can be represented as a generalization of a triangle in higher dimensions. Simplexes capture configurations in general, be them initial configurations, intermediate configurations, or decision configurations.

Definition 4.16 (Simplex)**.** Let $Cols$ and $Views$ be sets. Then $\sigma \subseteq Cols \times Views$ is a **simplex** on $Cols$ and $Views$ (or colored simplex) $\triangleq \forall p \in Cols : |\{v \in Views | (p,v) \in \sigma\}| \leq 1$.

For projections, $col(\sigma)$ or $names(\sigma) \triangleq \{p \in Cols \mid \exists v \in Views : (p,v) \in \sigma\}$. And $views(\sigma) = \{v \in Views \mid \exists p \in Cols : (p,v) \in \sigma\}$. We also write $view_\sigma(p)$ for the $v \in Views$ such that $(p,v) \in \sigma$: $view_\sigma(p) \triangleq \{v \in Views | (p,v) \in \sigma\}$

The **dimension** of σ is $|\sigma| - 1$.

A simplex τ is a **proper face** of $\sigma \triangleq \tau \subsetneq \sigma$.

Although Views could be any set for readability, the traditional view is a set of pairs, the first element being a process name, and the second being either another view or an initial value. For more details, refer to [19].

Then a complex is a set of simplexes that is closed under inclusion. It captures all considered configurations.

Definition 4.17 (Complex)**.** Let $Cols$ and $Views$ be sets. Then $C \subseteq \mathcal{P}(Cols \times Views)$ is a **simplicial complex** on $Cols$ and $Views$ (or colored simplicial complex) $\triangleq$

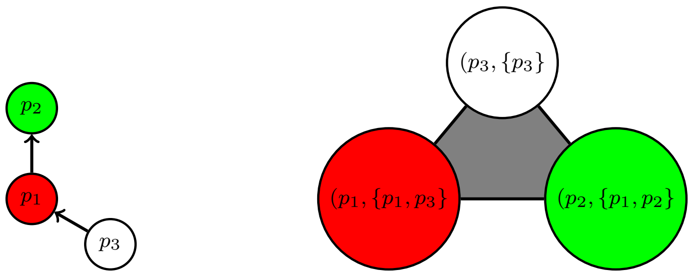
- $\forall (p,v) \in Cols \times Views : \{(p,v)\} \in C$.
- $\forall \sigma, \tau$ simplexes on $Cols \times Views$: $\sigma \in C \land \tau \subseteq \sigma \implies \tau \in C$.

The **facets** of $C \triangleq \{\sigma \in C \mid \forall \tau \in C : \sigma \subseteq \tau \implies \tau = \sigma\}$.

The **dimension** of C is the maximum dimension of its facets. C is called **pure** if all its facets have the same dimension.

How to go from heard-of predicates, which are generated by graphs, to simplexes and complexes?

Starting with a single graph G, the uninterpreted simplex of this graph is the simplex capturing the configuration after a round using G in terms of who hears from whom. It disregards input values, which makes it uninterpreted. Figure 4.2 shows a graph and its corresponding uninterpreted simplex.



(a) Graph (b) Uninterpreted simplex

Figure 4.2: A graph and its uninterpreted simplex

Definition 4.18 (Uninterpreted simplex of a graph)**.** Let G be a graph. Then the **uninterpreted simplex of** G is $\sigma_G \triangleq$ the colored simplex $\{(p, In_G(p) \mid p \in \Pi\ \}$.

Given a set of graphs S representing the possible graphs, the previous definition generalizes to the uninterpreted complex of the oblivious predicate $\mathcal{HO}$ generated by S.

Definition 4.19 (Uninterpreted complex of an oblivious predicate)**.** Let $\mathcal{HO}$ be an oblivious predicate generated by a set of graphs S. Then the **uninterpreted complex of** A is $C_{\mathcal{HO}} \triangleq$ the complex whose facets are exactly the $\{\sigma_G \mid G \in S\}$.

4.4.2 Uninterpreted complexes of closed-above predicates

It so happens that closed-above predicates give rise to uninterpreted complex that are easy to define and study. Indeed, they are unions of pseudospheres, where pseudospheres are colored complexes topologically equivalent to n-spheres. These pseudospheres have already been used in the literature to study multiple predicates of computation [19, Chap. 13].

Definition 4.20 (Pseudospheres [19, Def 13.3.1])**.** Let $V_1, V_2, ..., V_n$ be sets. Then the **pseudosphere complex** $\varphi(\Pi; V_i \mid i\ \in [1, n]) \triangleq$

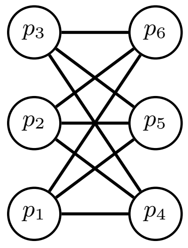
- $\forall i, \forall v \in V_i : (P_i, v)$ is a vertex of C.
- $\forall J \subseteq [1, n] : \{(P_j, v_j) \mid j \in J, v_j \in V_j\}$ is a simplex of C iff all P_j are distinct.

These complexes work like a generalization of complete bipartite graphs in n dimensions. Recall that a complete bipartite graph is a graph that can be split into two sets of nodes, the nodes of each set not linked between them and each node of one set linked to all nodes of the other set. For example, Figure 4.3a is a bipartite graph.

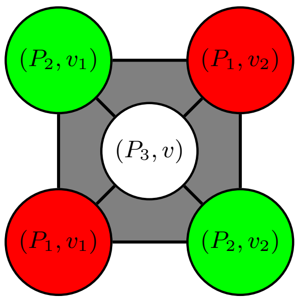
Now a pseudosphere is the same, except that nodes can be partitioned into n sets, no simplex contains more than one element of each set as a vertex, and all the simplexes built from one element of each set are in the complex. Figure 4.3b is an example of a pseudosphere built from processes P_1, P_2, P_3, and the three sets $V_1 = \{v_1, v_2\}$, $V_2 = \{v_1, v_2\}$ and $V_3 = \{v\}$.

Among other things, pseudospheres are closed under intersection, and are $(n-2)$ connected.

Lemma 4.21 (Intersection of pseudospheres [19, Fact 13.3.4])**.** $\varphi(\Pi; U_i \mid i \in [1, n]) \cap \varphi(\Pi; V_i \mid i \in [1, n]) = \varphi(\Pi; U_i \cap V_i \mid i \in [1, n])$.



(a) Bipartite graph



(b) Pseudosphere

Figure 4.3: A bipartite graph and a pseudosphere

One advantage of pseudosphere is that they have high connectivity [19, Def. 3.5.6]. Intuitively, connectivity concerns the (non-)existence of high-dimensional generalization of holes in the complexes. Since pseudospheres are topologically equivalent to spheres [19, Sect. 13.3], they only have these holes in their highest dimension.

Lemma 4.22 (Connectivity of pseudospheres [19, Cor. 13.3.7])**.** $\varphi(\Pi; V_i \mid i \in [1, n])$ *is* $(n-2)$*-connected, where* $n \triangleq |\{i \in [1, n] \mid V_i \neq \emptyset\}|$*.*

The connectivity of the uninterpreted complex for a simple closed-above predicate follows, because such a complex is a pseudosphere. Intuitively, for any process p, its possible views are exactly the upward closure of its view in the defining graph G. Then the n-simplexes of the uninterpreted complex are exactly the simplex one can build with one such view for each process.

Lemma 4.23 (Uninterpreted complex of a simple closed-above predicate is a pseudosphere)**.** *Let* $\mathcal{HO}$ *be a simple closed-above predicate generated by* G*. Then* $C_{\mathcal{HO}} = \varphi(\Pi; \{S \mid In_G(P_i) \subseteq S \subseteq \Pi\} \mid i \in [1, n])$*.*

Proof.

- ($\subseteq$). Let σ be a n-simplex of $C_{\mathcal{HO}}$. By definition of $C_{\mathcal{HO}}$, it is the uninterpreted simplex of a graph $H \in\uparrow G$. This in turn means that $\forall p \in \Pi : view_\sigma(p) = In_H(p) \supseteq In_G(p)$.

  Thus $\sigma = \{(P_i, In_H(P_i)) \mid i \in [1, n]\} \subseteq \varphi(\Pi; \{S \mid In_G(P_i) \subseteq S \subseteq \Pi\} \mid i \in [1, n])$.

- ($\supseteq$). Let σ be a n-simplex of $\varphi(\Pi; \{S \mid In_G(P_i) \subseteq S \subseteq \Pi\} \mid i \in [1, n])$. Then $\forall p \in \Pi : view_\sigma(p) \supseteq In_G(p)$. Thus σ is the uninterpreted simplex of a graph H such that $\forall p \in \Pi : In_H(p) \supseteq In_G(p)$.

  We conclude that $H \in\uparrow G$ and thus that $\sigma \in C_{\mathcal{HO}}$

□

It follows instantly that the uninterpreted complexes of simple closed-above predicates are $(|\Pi| - 2)$-connected.

Corollary 4.24 (Connectivity of the uninterpreted complex of a simple closed-above predicate)**.** *Let* $\mathcal{HO}$ *be a simple closed-above predicate generated by* G*. Then* $C_{\mathcal{HO}}$ *is* $(|\Pi| - 2)$*-connected.*

From this corollary and the closure of pseudospheres by intersection, let's now deduce a similar characterization of the connectivity for general closed-above predicates.

But to do so the main tool for studying connectivity of simplicial complexes needs to be introduced: the nerve lemma. This result uses a cover of a complex: a set of subcomplexes such that their union gives the initial complex.

Intuitively, the nerve lemma says that if you provide a cover of a complex that is "nice enough", then the connectivity of the initial complex can be deduced from the connectivity of the nerve complex, which has a vertex for each element of the cover and simplexes for sets of vertexes with non empty intersections.

Definition 4.25 (Nerve complex)**.** Let C be a simplicial complex, $(C_i)_{i \in I}$ a cover of C. Then the **nerve complex** of this cover, $\mathcal{N}(C_i \mid I) \triangleq$ the complex generated by

- the vertices are the C_i;
- and the simplexes are the sets $\{C_i \mid i \in J\}$ for $J \subseteq I$ such that $\bigcap\limits_{i \in J} C_i \neq \emptyset$

The nerve complex captures how the elements of the cover are glued together. What is needed for the nerve lemma is that the cover doesn't hide any hole; if it did, then the correspondence between the connectivity would be lost.

Lemma 4.26 (Nerve lemma [73, Thm 15.24])**.** *Let C be a simplicial complex, $(C_i)_{i \in I}$ a cover of C and $k \geq 0$. Then*

$$\left(\forall J \subseteq I : \left(\begin{array}{rl} & dim(\bigcap\limits_{i \in J} C_i) \geq (k - |J| + 1) \\ \wedge & \bigcap\limits_{i \in J} C_i = \emptyset \end{array} \right) \right)$$

$\implies$ *(C is k-connected $\iff$ $\mathcal{N}(C_i \mid I)$ is k-connected).*

Now the connectivity of uninterpreted complexes for general closed-above predicates follows.

Theorem 4.27 (Connectivity of the uninterpreted complex of a closed-above predicate)**.** *Let $\mathcal{HO}$ be a closed-above predicate generated by S. Then $C_{\mathcal{HO}}$ is $(|\Pi| - 2)$-connected.*

Proof. From the proof of Theorem 4.23, we know that $C_{\mathcal{HO}}$ is a union of pseudospheres: $C_{\mathcal{HO}} = \bigcup\limits_{G \in S} C_G$. We want to apply the nerve lemma to this cover. First, by Theorem 4.24, C_G is $(n-2)$-connected.

As for the intersection of any set I of C_G, we have two properties. First, it cannot be empty, since all C_G must contain the uninterpreted simplex of the complete graph on Π, by definition of $\uparrow G$. This gives us that the nerve complex is a simplex, and thus ∞-connected.

And second, the intersection is also a pseudosphere, by application of Lemma 4.21. Indeed, these are intersections of pseudospheres with the same processes which have an non-empty intersection for each color: the view of this process in the complete graph.

We can thus conclude by application of the nerve lemma and Theorem 4.24. □

4.4.3 Interpretation of uninterpreted complexes

Uninterpreted complexes only go so far; at some point, we need to consider initial values.

Definition 4.28 (Interpretation of uninterpreted simplex)**.** Let σ be an uninterpreted simplex on Π and τ be a $(n-1)$-simplex colored by Π. Then the **interpretation of σ on** τ, $\sigma(\tau) \triangleq \{(p, V) \mid p \in \Pi \wedge (v \in V \implies (\exists q \in \mathit{view}_\sigma(p) : v = \mathit{view}_\tau(q)))\}$

Then the same intuition can be applied to a full uninterpreted complex.

Definition 4.29 (Interpretation of uninterpreted complex)**.** Let $\mathcal{A}$ be an uninterpreted complex on Π and $\mathcal{I}$ be a pure $(n-1)$ complex colored by Π. Then the **interpretation of $\mathcal{A}$ on** $\mathcal{I}$, $\mathcal{A}(\mathcal{I}) \triangleq \bigcup_{\substack{\tau \text{ a facet of } \mathcal{I} \\ \sigma \text{ a facet of } \mathcal{A}}} \sigma(\tau)$

These interpretations give protocol complexes, on which known result on computability are applicable.

4.4.4 A Powerful Tool

On the combinatorial topology front, our results leverage two main tools: the impossibility result on k-set agreement based on connectivity [19, Thm. 10.3.1], and a way to compute the connectivity of a complex from the way it is built. This section develops the second idea.

First, let's define the concept of shellability. This builds on the boundary complex of a simplex, which is simply the complex formed by the faces of the simplex.

Definition 4.30 (Boundary complex)**.** Let σ be a simplex. Then the boundary complex of σ $\triangleq$ the complex formed by all proper faces of σ.

Definition 4.31 (Shellability)**.** Let $\mathcal{A}$ be a pure complex of dimension d. Then $\mathcal{A}$ is **shellable** $\triangleq$ there is an ordering $\varphi_1, \dots, \varphi_r$ of its facets such that for every $1 \leq t \leq r-1$,

$$\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi_{t+1}$$

is a pure subcomplex of dimension $d-1$ of the boundary complex of φ_{t+1}, i.e., of $\mathrm{skel}^{d-1}\, \varphi_{t+1}$.

The ordering is a **shelling order**.

The intuition here is that the complex is the union of simplexes of dimension d, and there is an order in which to add simplexes, so that the new simplex is connected to the rest by $d-1$ simplexes, some of its own facets. In the concrete case of 2-simplexes (triangles) for example, they must be connected to the rest by 1-simplexes (edges).

Here, unions and intersections apply to the complexes induced by the facet and all its faces.

For example, the complex in Figure 4.4a is shellable, but the one in Figure 4.4b is not.

Given a shelling order $\varphi_1, \dots, \varphi_r$ of a complex $\mathcal{A}$, $(\bigcup_{i=1}^{t} \varphi_i) \cap \varphi_{t+1}$ is the union of the complexes induced by some $(d-1)$-faces $\tau_1, \dots, \tau_s$ of φ_{t+1}, by definition of shellability. Each τ_j is a face of a facet σ_j of $\cup_{i=1}^{t} \varphi_i$, hence φ_{t+1} and σ_j share a $(d-1)$-face. Then,

$$\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi_{t+1} = \bigcup_{j=1}^{s} (\varphi_{t+1} \cap \sigma_j).$$

The following technical result also serves in the following proofs.

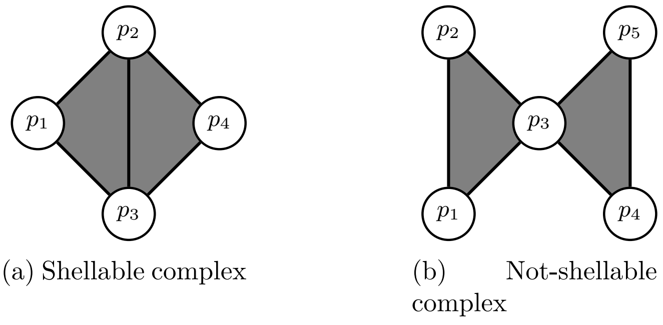


Figure 4.4: Examples of a complex that is shellable and one that is not

Lemma 4.32 (Shellability of simplex boundary [19, Thm 13.2.2])**.** *Let $\mathcal{A}$ be a pure $(d-1)$-dimensional sub-complex of the boundary complex of a simplex of dimension d. Then $\mathcal{A}$ is shellable, and any sequence of its facets is a shelling order for $\mathcal{A}$.*

Finally, the last piece of the puzzle is this straightforward corollary of the nerve lemma for a cover with two elements.

Corollary 4.33 (Two elements nerve lemma)**.** *Let C and K be k-connected complexes. If $C \cap K$ is $(k-1)$-connected, then $C \cup K$ is k-connected.*

Now I can state the main technical result of the section. It extends the result from Castañeda et al. [66] and adapts it to the interpretation of complexes needed here. While Castañeda et al. studied the complex given by the interpretation of a single graph (to capture predicates like $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ [25]), what matters care about the complex resulting of the interpretation of a set of graphs.

If each input simplex gets sent into a complex, and both the output complexes and the mapping are "nice", then the interpreted complex is highly connected.

Lemma 4.34. *Let $\mathcal{A}$ be a pure shellable complex of dimension d, $\mathcal{B}$ a complex, $(\mathcal{B}_i)_{i\in I}$ a cover of $\mathcal{B}$, and $\ell \geq 0$ an integer. Suppose that there is a bijection α between the facets of $\mathcal{A}$ and the elements of $(\mathcal{B}_i)_{i\in I}$ such that:*

1. *For every facet φ' of $\mathcal{A}$ and every pure d-subcomplex $\bigcup_{i=1}^{t} \varphi_i \subseteq \mathcal{A}$ satisfying that $\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi' = \bigcup_{i=1}^{s} (\varphi' \cap \sigma_i)$ for some of $\mathcal{A}$'s facets $\sigma_1, \ldots, \sigma_s$, with each σ_i and φ' sharing a $(d-1)$-face, it holds that $\left(\bigcup_{i=1}^{t} \alpha(\varphi_i)\right) \cap \alpha(\varphi') = \bigcup_{i=1}^{s} (\alpha(\varphi') \cap \alpha(\sigma_i))$.*
2. *For every $t \geq 0$ and every collection $\varphi_0, \varphi_1, \ldots, \varphi_t$ of $t+1$ facets of $\mathcal{A}$ with each φ_i and φ_0 sharing a $(d-1)$-face, it holds that $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is least $(\ell - t)$-connected.*

Then, $\mathcal{B}$ is ℓ-connected.

Proof. We prove the claim by induction on ℓ.

- **(Base case)** $\ell = 0$. We need to prove that $\mathcal{B}$ is 0-connected, by induction on the length of a shelling order of $\mathcal{A}$. Fix a shelling order $\varphi_1, \ldots, \varphi_m$ of $\mathcal{A}$, so $\mathcal{B} = \bigcup_{i=1}^{m} \alpha(\varphi_i)$.
  - **(Base case)** $\mathcal{B} = \alpha(\varphi_1)$. By hypothesis **(2)**, for the case $t = 0$, $\mathcal{B} = \alpha(\varphi_1)$ is at least $l - t = 0$-connected.

  - **(Induction step)** Suppose that $\bigcup_{i=1}^{r-1} \alpha(\varphi_i)$ is 0-connected, for some $2 \leq r < m$. We have that $\alpha(\varphi_r)$ is 0-connected by hypothesis **(2)**, as above. We show that $\left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cap \alpha(\varphi_r)$ is (-1)-connected, namely, non-empty, and then Corollary 4.33 imply that $\mathcal{B} = \left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cup \alpha(\varphi_r)$ is 0-connected. By definition of shellability,

$$\left(\bigcup_{i=1}^{r-1} \varphi_i\right) \cap \varphi_r = \tau_1 \cup \ldots \cup \tau_s,$$

    where each τ_j is a face of dimension $(d-1)$ of φ_r. For each τ_j there is a facet σ_j of $\bigcup_{i=1}^{r-1} \varphi_i$ such that $\tau_j \subset \sigma_j$. Thus, φ_r and σ_j share a $(d-1)$-face and

$$\left(\bigcup_{i=1}^{r-1} \varphi_i\right) \cap \varphi_r = \bigcup_{j=1}^{s} \left(\varphi_r \cap \sigma_j\right).$$

    By hypothesis (1) we have that

$$\left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cap \alpha(\varphi_r) = \bigcup_{j=1}^{s} \left(\alpha(\varphi_r) \cap \alpha(\sigma_j)\right).$$

    Each σ_j shares a $(d-1)$-face with φ_r, so hypothesis (2), with $t = 1$, implies that $\alpha(\varphi_r) \cap \alpha(\sigma_j)$ is at least (-1)-connected, which implies that $\left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cap \alpha(\varphi_r)$ is non-empty.

- **(Induction step)** Suppose that the statement of the theorem holds for $\ell - 1$, and consider a shelling order $\varphi_1, \ldots, \varphi_m$ of $\mathcal{A}$. Our aim is to show that $\mathcal{B} = \bigcup_{i=1}^{m} \alpha(\varphi_i)$ is ℓ-connected.

  As in the base case, we proceed by induction on the length of the shelling order.

  - **(Base case)** $\mathcal{B} = \alpha(\varphi_1)$. By hypothesis **(2)**, for the case $t = 0$, $\mathcal{B} = \alpha(\varphi_1)$ is at least $\ell - 0 = \ell$-connected.
  - **(Induction step)** Suppose that $\bigcup_{i=1}^{r-1} \alpha(\varphi_i)$ is ℓ-connected, for some $2 \leq r < m$. We have that $\alpha(\varphi_r)$ is ℓ-connected by hypothesis **(2)** as above. If we show that $\left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cap \alpha(\varphi_r)$ is $(\ell-1)$-connected, Corollary 4.33 implies that $\bigcup_{i=1}^{r} \alpha(\varphi_i)$ is ℓ-connected.

    To do so, we use the theorem for $\ell - 1$. As seen before, there are facets $\sigma_1, \ldots, \sigma_s$ of $\bigcup_{i=1}^{r-1} \varphi_i$ such that each σ_j and φ_r share a $(d-1)$-face,

$$\left(\bigcup_{i=1}^{r-1} \varphi_i\right) \cap \varphi_r = \bigcup_{j=1}^{s} \left(\varphi_r \cap \sigma_j\right) \text{ and } \left(\bigcup_{i=1}^{r-1} \alpha(\varphi_i)\right) \cap \alpha(\varphi_r) = \bigcup_{j=1}^{s} \left(\alpha(\varphi_r) \cap \alpha(\sigma_j)\right).$$

    Let $\mathcal{B}' = \bigcup_{j=1}^{s} \left(\alpha(\varphi_r) \cap \alpha(\sigma_j)\right)$. Let $\lambda_1, \ldots, \lambda_{s'}$ be simplexes among the σ_j's such that $\alpha(\varphi_r) \cap \alpha(\lambda_i) \neq \alpha(\varphi_r) \cap \alpha(\lambda_{i'})$ for $i \neq i'$, and the $\alpha(\varphi_r) \cap \alpha(\lambda_i)$ still form a cover of $\mathcal{B}'$: $\mathcal{B}' = \bigcup_{i=1}^{s'} \left(\alpha(\varphi_r) \cap \alpha(\lambda_i)\right)$. Let $\mathcal{A}' = \bigcup_{i=1}^{s'} \left(\varphi_r \cap \lambda_i\right)$. Note that $\mathcal{A}'$ is pure of dimension $d - 1$ and is a subcomplex of the boundary complex of φ_r.

    By Lemma 4.32, $\mathcal{A}'$ is shellable. The facets of $\mathcal{A}'$ are the intersections $\varphi_r \cap \lambda_i$. Consider the bijection $\beta(\varphi_r \cap \lambda_i) = \alpha(\varphi_r) \cap \alpha(\lambda_i)$ between the facets of $\mathcal{A}'$ and our cover of $\mathcal{B}'$. Now, let $\varphi_r \cap \lambda$ be any facet of $\mathcal{A}'$ and $\bigcup_{i=1}^{m'} (\varphi_r \cap \lambda'_i)$ be any pure $(d-1)$-subcomplex of $\mathcal{A}'$. Note that every pair of facets of $\mathcal{A}'$ share a $(d-2)$-face as both are $(d-1)$-faces of φ_r. Then, $\varphi_r \cap \lambda$ and each $\varphi_r \cap \lambda'_i$ share a face of dimension $d-2$, and thus we can write

$$\left(\bigcup_{i=1}^{m'} (\varphi_r \cap \lambda'_i)\right) \cap (\varphi_r \cap \lambda) = \bigcup_{i=1}^{m'} ((\varphi_r \cap \lambda) \cap (\varphi_r \cap \lambda'_i))$$

and

$$\left(\bigcup_{i=1}^{m'} \beta(\varphi_r \cap \lambda'_i)\right) \cap \beta(\varphi_r \cap \lambda) = \bigcup_{i=1}^{m'} (\beta(\varphi_r \cap \lambda) \cap \beta(\varphi_r \cap \lambda'_i)).$$

We conclude that hypothesis (1) of the theorem holds for $\mathcal{A}'$, $\mathcal{B}'$ and β.

Finally, consider any collection $\varphi_r \cap \lambda'_0, \dots, \varphi_r \cap \lambda'_{t'}$ of $t'+1$ facets of $\mathcal{A}'$. As already noted, each of them and the first one share a $(d-2)$-face. We have that

$$\tau = \bigcap_{i=0}^{t'} \beta(\varphi_r \cap \lambda'_i) = \bigcap_{i=0}^{t'} (\alpha(\varphi_r) \cap \alpha(\lambda'_i)) = \alpha(\varphi_r) \cap \bigcap_{i=0}^{t'} \alpha(\lambda'_i).$$

As said above, the λ'_i's are facets of $\mathcal{A}$ and each of them and φ_r share a $(d-1)$-face. By hypothesis (2) with $t = t'+1$, τ is of at least $(\ell - t) = (\ell - (t'+1)) = ((\ell-1)-t')$-connected. Then, hypothesis (2) of the theorem holds for $\mathcal{A}'$, $\mathcal{B}'$, β and $\ell - 1$.

We have all hypothesis to use the theorem with $\mathcal{A}'$ and $\mathcal{B}'$ and $\ell - 1$. Therefore, $\mathcal{B}'$ is $(\ell-1)$-connected, and then $\cup_{i=1}^{r} \alpha(\varphi_i)$ is ℓ-connected.

□

4.5 One round lower bounds: a touch of topology

As before, we start with the simple closed-above case, where the predicate is the closure of a single graph. In this case the tight lower bound follows from Castañeda et al. [66, Thm 5.1], as mentioned above.

Theorem 4.35 (Lower bound on k-set agreement for simple closed-above predicates)**.** *Let $\mathcal{HO}$ a simple closed-above predicate generated by the graph G. Let $k \le \gamma(G)$. Then k-set agreement is not solvable on $\mathcal{HO}$ in a single round.*

Let's thus focus on general closed-above predicates. Here the underlying structure of the protocol complex is leveraged through two tools: the main lemma from Section 4.4.4, as well as two graph parameters: the distributed domination number over a set of graphs, and the max-covering numbers of a set of graphs.

Definition 4.36 (Distributed domination number of a set of graphs)**.** Let S be a set of graphs. Then the **distributed domination number** S,

$$\gamma^{dist}(S) \triangleq min \left\{ i > 0 \;\middle|\; \begin{array}{l} \forall P \subseteq \Pi, \forall S_i \subseteq S : \\ (|P| = i \wedge |S_i| = \min(i, |S|)) \\ \implies \bigcup\limits_{G \in S_i} Out_G(P) = \Pi \end{array} \right\}.$$

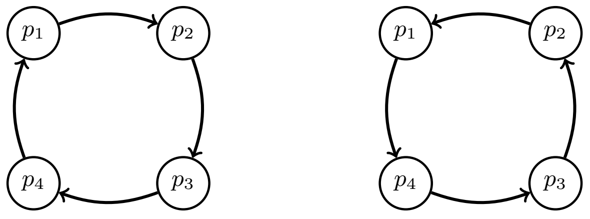


Figure 4.5: The two graphs generating a predicate with different domination numbers

Figure 4.5 gives a set S of graphs for which $\gamma^{eq}(S)$ and $\gamma^{dist}(S)$ differ. Indeed, $\gamma^{eq}(S)$ equals 3 in this context, because some sets of two processes (for example $\{p_1, p_2\}$ is not a dominating set in both graphs). On the other hand, $\gamma^{dist}(S)$ equals 2. That's because any set of two processes reach all processes when considering the neighbors in both graphs.

So a set of $\gamma^{eq}(S)$ processes dominates each graph of S separately, whereas a set of $\gamma^{dist}(S)$ processes might not dominate any graph of S, but it dominates every subset of $\gamma^{dist}(S)$ graphs of S when all the neighbors in all the graphs are considered together. Thus $\gamma^{dist}(S) \leq \gamma^{eq}(S)$. Fitting, considering the former is used in lower bounds and the latter in upper bounds.

Next, the max-covering numbers are quite subtle. For $i < \gamma^{dist}(S)$, the i-th max-covering number of S is the maximum number of processes hearing a set of i processes, summed over i graphs in S. That is, the max-covering numbers capture how much values can be disseminated in the best case. They serve in lower bounds by giving a best case scenario to focus on for proving impossibility.

Definition 4.37 (Max covering numbers of a set of graphs)**.** Let S be a set of graphs and $i < \gamma^{dist}(S)$. Then the i-th max-covering number of S,

$$\textit{max-cov}_i(S) \triangleq \max_{\substack{P\subseteq\Pi, |P|=i \\ S_i\subseteq S, |S_i|=\min(i,|S|) \\ \bigcup\limits_{G\in S_i} Out_G(P)\neq\Pi}} |(\bigcup_{G\in S_i} Out_G(P))|.$$

We also define the i-th max-covering coefficients on S,

$$M_i(S) \triangleq \begin{cases} \left\lfloor \frac{n-i-1}{\textit{max-cov}_i(S)-i} \right\rfloor & \text{if } \textit{max-cov}_i(S) > i \\ n-i & \text{if } \textit{max-cov}_i(S) = i \end{cases}$$

Let's now state the lower bound for general closed-above predicate. Notice that there is an additional constraint: the closed-above predicate is not generated only by the clique. If it was, then the l of the lower bound could be equal to -1 (because the $\gamma^{dist}(S)$ of a clique is 1) and thus the bound would talk about 0-set agreement, which doesn't exists.

Since every k-set agreement is trivially solvable on the clique, this constraint does not reduce the application of the bound.

Theorem 4.38 (Lower bound on k-set agreement for general closed-above predicates)**.** *Let $\mathcal{HO}$ be a closed-above predicate generated by the set of graphs S, such that S is not the singleton set of the clique..*
Let $l = \min(\gamma^{dist}(S), \min\limits_{t\in[1,\gamma^{dist}(S)[} t + M_t(S)) - 2$. Then $(l+1)$-set agreement is not solvable on $\mathcal{HO}$ in a single round.

Proof. It is known that when the protocol complex is k-connected, non trivial $k+1$-set agreement is impossible. Herlihy et al. [19] gives an example derivation for colorless protocols, and

Castañeda et al. [66] give one for colored protocols. We thus prove that the protocol complex generated by $\mathcal{HO}$ after one round is l-connected.

As we said, we want to apply Lemma 4.34. Our $\mathcal{A}$ is the pseudosphere $\Psi(\Pi, [0,k])$, our $\mathcal{B}$ is $C_{\mathcal{HO}}(\mathcal{A})$ and our mapping α sends a facet σ of $\mathcal{A}$ on $C_{\mathcal{HO}}(\sigma) = \bigcup\limits_{G\in S} C_G(\sigma)$.

- Let φ' be a facet of $\mathcal{A}$ and take a pure d-subcomplex $\bigcup_{i=1}^{t} \varphi_i \subseteq \mathcal{A}$ satisfying that $\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi' = \bigcup_{i=1}^{s} (\sigma_i \cap \varphi')$ for some of $\mathcal{A}$'s facets $\sigma_1, \ldots, \sigma_s$, with each σ_i and φ' sharing a $(d-1)$-face.

  We want to show that $\left(\bigcup_{i=1}^{t} \alpha(\varphi_i)\right) \cap \alpha(\varphi') = \bigcup_{i=1}^{s} (\alpha(\sigma_i) \cap \alpha(\varphi'))$.

  - Let τ be a simplex of $\left(\bigcup_{i=1}^{t} \alpha(\varphi_i)\right) \cap \alpha(\varphi')$. Since $\left(\bigcup_{i=1}^{t} \alpha(\varphi_i)\right) \cap \alpha(\varphi') = \bigcup_{i=1}^{t} (\alpha(\varphi_i) \cap \alpha(\varphi'))$ by distributivity of intersection on union, we have some $i \in [1,t]$ such that τ is a simplex of $\alpha(\varphi_i) \cap \alpha(\varphi')$.

    Now, α sends a simplex σ to $C_{\mathcal{HO}}(\sigma)$; thus $\alpha(\varphi') \cap \alpha(\varphi_i) = C_{\mathcal{HO}}(\varphi') \cap C_{\mathcal{HO}}(\varphi_i) = (\bigcup\limits_{G\in S} C_G(\varphi')) \cap (\bigcup\limits_{G\in S} C_G(\varphi_i)) = \bigcup\limits_{G,H\in S} C_G(\varphi') \cap C_H(\sigma_i)$. Hence τ being a simplex of $\alpha(\varphi_i) \cap \alpha(\varphi')$ means that $\exists G, H \in S$ for which every process of τ has its view in both $C_G(\varphi_i)$ and $C_H(\varphi')$. And a view is completely defined by the value received from other processes. That is, $\forall (p,v) \in \tau, \forall (q, v_q) \in v : (q, v_q) \in \varphi_i \cap \varphi'$.

    By our equation $\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi' = \bigcup_{i=1}^{s} (\sigma_i \cap \varphi')$ there is $l \in [1,s]$ such that $\varphi_i \cap \varphi' \subseteq \sigma_l \cap \varphi'$. Then all $(q, v_q) \in v$ are also in $\sigma_l \cap \varphi'$. We conclude that all $(q, v_q) \in v$ are in σ_l and in φ', and thus τ is a simplex of $\alpha(\sigma_l) \cap \alpha(\varphi')$.
  - Let τ be a simplex of $\bigcup_{i=1}^{s} (\alpha(\sigma_i) \cap \alpha(\varphi'))$. Thus there is some $i \in [1,s]$ such that τ is a simplex of $\alpha(\sigma_i) \cap \alpha(\varphi')$. That is, $\forall (p,v) \in \tau, \forall (q, v_q) \in v : (q, v_q) \in \sigma_i \cap \varphi'$.

    Then, by our equation $\left(\bigcup_{i=1}^{t} \varphi_i\right) \cap \varphi' = \bigcup_{i=1}^{s} (\sigma_i \cap \varphi')$, there is $l \in [1,t]$ such that $\forall (p,v) \in \tau, \forall (q, v_q) \in v : (q, v_q) \in \varphi_l$. We conclude that all $(q, v_q) \in v$ are in φ_l and in φ', and thus τ is a simplex of $\left(\bigcup_{i=1}^{t} \alpha(\varphi_i)\right) \cap \alpha(\varphi')$.

- Now we want to study the connectivity of the intersection of well-chosen facets. Let $t \geq 0$ and $\varphi_0, \varphi_1, \ldots, \varphi_t$ be $t+1$ facets of $\mathcal{A}$ with each φ_i and φ_0 sharing a $(d-1)$-face. We want to prove that $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is $l-t$ connected.

  Because $l \leq \gamma^{dist}(S) - 2$, we only have to consider $t < \gamma^{dist}(S)$, because $l - \gamma^{dist}(S) \leq -2$, and thus in the case $t \geq \gamma^{dist}(S)$, there is no constraint to satisfy on the connectivity of the intersection.

  Now, each $\alpha(\varphi_i)$ is in fact $C_{\mathcal{HO}}(\varphi_i) = \bigcup\limits_{G\in S} C_G(\varphi_i)$. We start by developing the $C_{\mathcal{HO}}(\varphi_i)$ into the union of the $C_G(\varphi_i)$ and applying the distributivity of intersection over union on this big intersection:

$$\begin{aligned} \bigcap_{i\in[0,t]} C_{\mathcal{HO}}(\varphi_i) &= \bigcap_{i\in[0,t]} \bigcup_{G\in S} C_G(\varphi_i) \\ &= \bigcup_{G_0,G_1,\ldots,G_t\in S} \bigcap_{i\in[0,t]} C_{G_i}(\varphi_i) \end{aligned}$$

  As always, we naturally get a cover of our space. We thus use the Nerve Lemma.

This requires first a computation of connectivity for the $\bigcap_{i=0}^{t} C_{G_i}(\varphi_i)$. We are taking the intersection of pseudospheres, which gives a new pseudosphere by Lemma 4.21. To compute its connectivity, we need to now how many processes end up with a non-empty set of view, by Lemma 4.22.

Let us assume that the φ_i are all distinct; if not we can remove the duplicate and start with a lower t. Then, because they all intersect with φ_0 on a $(d-1)$ face, we have $\bigcap_{i=0}^{t} \varphi_i$ of dim $(d-t)$. That is, in these input simplexes, there are $(d-t)$ processes with the same input value across all φ_i. Or equivalently, there are t processes with different values for some φ_i.

Let P be the set of t processes with sometimes different initial values across the φ_i. Then the processes disappearing from $\bigcap_{i=0}^{t} C_{G_i}(\varphi_i)$ are the ones receiving the values from P.

But for $t < \gamma^{dist}(S)$, we know either all processes receive the values from P, or at most $\mathit{max\text{-}cov}_t(S)$ do. Thus $\bigcap_{i=0}^{t} C_{G_i}(\varphi_i)$ is either empty or a pseudosphere with $(n - \mathit{max\text{-}cov}_t(S))$ processes with a non empty set of views. It is therefore empty or $(n - \mathit{max\text{-}cov}_t(S) - 2)$-connected by Lemma 4.22.

Let us index the subsets of S of size t whose intersection is non-empty. Then let J be a set of index of size $\leq M_t(S)$. We now show that $\bigcap\limits_{j \in J} \bigcap\limits_{\substack{i \in [0,t] \\ G_i \in S_j}} C_{G_i}(\varphi_i)$ is either empty or $(M_t(S) - 2 - |J| + 1) = (M_t(S) - |J| - 1)$-connected.

If P dominates any set S_j for $j \in J$, then the intersection for this set is empty, and thus the intersection of the intersections of J is also empty. Let us thus assume from this point that P does not dominate any set S_j with $j \in J$.

Then in each $G \in S_j$, P talks to at most $\mathit{max\text{-}cov}_t(S)$ processes. Of these, there are $\mathit{max\text{-}cov}_t(S) - t$ who are not in P. This means that in the worst case, P talks to $|J|(\mathit{max\text{-}cov}_t(S) - t)$ processes not in P. Thus in this worst case, the number of processes hearing the views of P is $t + |J|(\mathit{max\text{-}cov}_t(S) - t)$.

Therefore, the intersection is $(n - t - |J|(\mathit{max\text{-}cov}_t(S) - t) - 2)$-connected.

First we treat the case where $\mathit{max\text{-}cov}_t(S) = t$, and thus where $M_t(S) = n - t$. We have an intersection that is $(n - t - 2)$-connected, and $n - t - 2 \geq n - t - |J| - 1$, since $|J| \geq 1$. This actually holds for any J, even when $|J| > M_t(S)$. Hence the nerve complex of our cover in this case is a simplex, and thus ∞-connected. We conclude by the nerve lemma 4.26 that $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is $(M_t(S) - 2)$-connected.

Now let us turn to the case where $\mathit{max\text{-}cov}_t(S) > t$. Notice that $n - t - |J|(\mathit{max\text{-}cov}_t(S) - t) - 2 = (n - t - 1) - |J|(\mathit{max\text{-}cov}_t(S) - t) - 1 \geq (\mathit{max\text{-}cov}_t(S) - t)(M_t(S) - |J|) - 1$. And since $M_t(S) \geq |J|$ and $\mathit{max\text{-}cov}_t(S) > t$, we have $n - t - |J|(\mathit{max\text{-}cov}_t(S) - t) - 2 \geq M_t(S) - |J| - 1$. Among other things, this means that $\forall J$ such that $|J| = M_t(S)$, $\bigcap\limits_{j \in J} \bigcap\limits_{\substack{i \in [0,t] \\ G_i \in S_j}} C_{G_i}(\varphi_i)$ is (-1)-connected and thus not empty.

This implies that the nerve complex contains the $M_t(S)$ skeleton of a higher dimensional simplex. And such a skeleton is at least $(M_t(S) - 1)$-connected. We conclude by the nerve lemma 4.26 that $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is $(M_t(S) - 2)$-connected.

For $t < \gamma^{dist}(S)$, we thus have that $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is $(M_t(S) - 2)$-connected. And since $(t + M_t(S) - 2) \geq l$, we have $M_t(S) - 2 \geq l - t$, and thus $\bigcap_{i=0}^{t} \alpha(\varphi_i)$ is $(l - t)$-connected.

The two conditions of Lemma 4.34 are satisfied; we conclude that $C_{\mathcal{HO}}(\mathcal{A})$ is l-connected, and thus k-set agreement is unsolvable in one round on predicate $\mathcal{HO}$. □

The term depending on $\gamma^{dist}(S)$ in the lower bound serves when the max-covering numbers are not sufficient to distinguish adversaries with different properties. Consider for example the symmetric predicates generated by all unions of s stars, with $s \leq n$. Then for those graphs, for $t < \gamma^{dist}(S)$, we have $\textit{max-cov}_t(S) =$ t, and thus $M_t(S) = n - t$. Hence the minimum over the $t + M_t(S) - 2$ is $n - 2$.

But this would mean that $(n-1)$-set agreement is impossible for $s < n$, whereas we can clearly solve 2-set agreement for $s = n - 1$, for example. What depends on s is $\gamma^{dist}(S)$ itself. More precisely, $\gamma^{dist}(S) = n - s + 1$, because given P, we can consider only the graph where the s centers of stars are in $\Pi \setminus P$, up until the point where $|P| > n - s$.

Hence our lower bound shows that for the symmetric union of s stars, $(n-s)$-set agreement is impossible in one round. Given that our upper bounds above tell that $(n - s + 1)$-set agreement is possible in one round for this predicate, the bound is tight.

Finally, the bound can be specialized for symmetric predicates.

Corollary 4.39 (Lower bounds for symmetric closed-above predicate)**.** *Let G be a graph different than the clique. Let* $l = \min\left(\gamma^{dist}(Sym(G)), \min\limits_{t\in[1,\gamma^{dist}(Sym(G))[} \begin{pmatrix} t + \left\lfloor \frac{n-t-1}{t(\textit{max-cov}_t(\{G\})-t)} \right\rfloor & \textit{if } \textit{max-cov}_t(\{G\}) > t \\ & \textit{if } \textit{max-cov}_t(\{G\}) = t \end{pmatrix}\right) - 2$
Then $(l + 1)$-set agreement is not solvable on the predicate generated by $Sym(\uparrow G)$ in a single round.

Proof. We simply compute $M^t(Sym(G))$ from $M_t(\{G\})$. First, notice that if $\textit{max-cov}_t(\{G\}) = t$, then no other process hears any set of t processes. This is invariant by permutation, and thus $\textit{max-cov}_t(S) = t$, and $M_t(Sym(G)) = n - t$.

We now turn to the case $\textit{max-cov}_t(\{G\}) = t$. In the worst case, we have a $P \subseteq \Pi$ of size t that hits $\textit{max-cov}_t(\{G\})$ processes in G. Of these, $\textit{max-cov}_t(\{G\}) - t$ processes are not in P. By permutation, we can take, in the worst case, $t - 1$ other graphs where these $\textit{max-cov}_t(\{G\}) - t$ are completely different. That is, in the worst case, P touches $\textit{max-cov}_t(Sym(G)) = t + t(\textit{max-cov}_t(\{G\}) - t)$ processes. And thus $M_t(Sym(G) = \left\lfloor \frac{n-t-1}{\textit{max-cov}_t(Sym(G))-t} \right\rfloor = \left\lfloor \frac{n-t-1}{t(\textit{max-cov}_t(G)-t)} \right\rfloor$. □

Notice that all these lower bounds are valid for general algorithms, not only oblivious ones. The reason is that a one round full information protocol is an oblivious algorithm.

4.6 Multiple rounds

Given that the focus on oblivious algorithms, a natural approach to extending the lower bounds to the multiple rounds case is to look at the product of graphs. By product, I mean the graph of the paths with one edge per graph. Thus the products of r graphs capture who will hear who after r corresponding communication rounds.

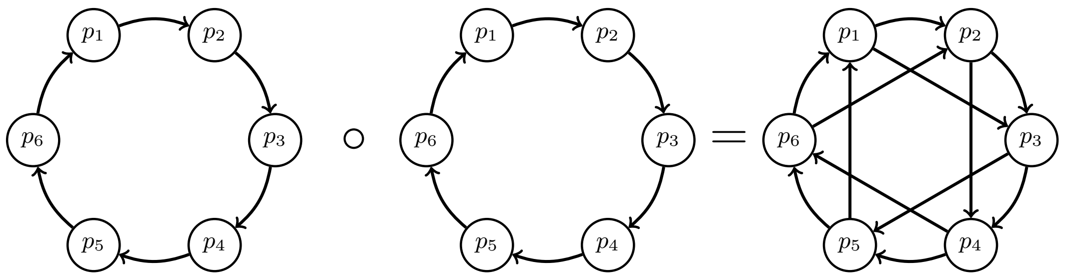
Definition 4.40 (Graph path product)**.** Let G and H be graphs with auto-loops ($\forall v \in \Pi : (v,v) \in E(G) \wedge (v,v) \in E(H)$). Then their **graph path product** $G \circ H \triangleq$ the graph (Π, E) such that $\forall u,v \in \Pi : (u,v) \in \Pi \implies \exists w \in \Pi : (u,w) \in E(G) \wedge (w,v) \in E(H)$.

Since the result is still a graph, the lower bound for one round still applies. At least, if the resulting graph still satisfy the hypotheses of the lower bounds. It does, although product doesn't maintain closure-above. This subtlety is explained in the next subsection.

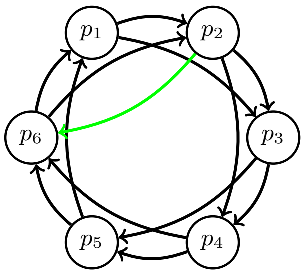
4.6.1 Closure-above is not invariant by product, but its still works

What is the pitfall mentioned above? Quite simply, that the product of two closed-above predicates does not necessarily gives a closed-above predicate. This follows from the fact that the closure-above of a product of graphs doesn't always equal the product of the closure-above of the graphs.

Let's take an example: the product of a cycle with itself.



Then the following graph cannot be built by extending the cycles and taking the product:



Why? Simply put, adding the new edge to either of the two cycles necessarily creates other edges in the product. Adding an edge from p_2 to any other node than p_3 or p_4 also creates new edges; so does adding an edge to p_4 and then an edge from p_4 to p_6, or an edge from p_3 to p_6 in the second graph.

Hence the product of the closure above of this cycle with itself is not the closure-above of the squared cycle. To put it differently, closure-above is not invariant by the product operation.

Nonetheless, the bell does not toll for our hopes of extending the bounds. What is used in the lower bound proofs above is not closure-above itself, but its consequences: being a union of pseudospheres containing the full simplex, such that for each pseudosphere, all graphs contain the smallest graph.

All three properties are present in a specific subset of the product of two simple closed-above predicates: all products where edges might be added to the last graph in the product but not to the other. Each added edge only alters the view of its destination, since it is in the second graph, and multiple added edges don't interfere because they are all added to the same graph. Hence the views of processes can change one at a time, and thus give a pseudosphere. Since adding no edge gives the original product and adding all missing edges gives the clique, the other two properties also hold Then taking this subset of the product of two general closed-above predicates result in a union of pseudosphres, one for each product of the underlying graphs.

Therefore relevant subcomplexes exists within the product of closed-above predicates, and then the lower bounds only depend on the properties of the underlying product of graphs.

4.6.2 Upper bounds for multiple rounds

Even though the section before just explained how to deal with lower bounds for multiple rounds, let's still start by giving upper bounds for multiple rounds. This is for the same reason as in the one round case: the upper-bounds require no combinatorial topology, and they allow us to introduce concepts needed for the lower bounds.

The algorithm is the same than for a single round, except that the exchange of values runs for r rounds. Note that this algorithm is parameterized by the number of rounds it uses.

Definition 4.41 (Multiple rounds k-set agreement algorithm)**.** Let $r > 0$. Assume that the set of initial values to k-set agreement is totally ordered. Then the r **round k-set agreement algorithm** $r - rounds \triangleq$ the algorithm where each process

- broadcasts the set of pairs (process,initial value) that it knows for r rounds;
- and then decides the minimum value it received.

First, a little preliminary result is required for upper bounds: that the product of closed-above predicates is included in the closure-above of the product.

Lemma 4.42 (Product and inclusion for closed-above)**.** *Let G and H be two graphs. Then $\uparrow G \circ \uparrow H \subseteq \uparrow (G \circ H)$.*

Proof. Let $K \in \uparrow G \circ \uparrow H$. Thus $\exists G' \in \uparrow G, \exists H' \in \uparrow H : K = G' \circ H'$. Let $u, v \in \Pi$ such that $(u, v) \in G \circ H$. We show that $(u, v) \in K$; this will entail that $K \in \uparrow (G \circ H)$.

Because $(u, v) \in G \circ H$, $\exists w \in \Pi : (u, w) \in E(G) \wedge (w, v) \in E(H)$. But $G' \in \uparrow G$ and $H' \in \uparrow H$, therefore $(u, w) \in E(G') \wedge (w, v) \in E(H')$. We conclude that $(u, v) \in G' \circ H' = K$. □

Hence taking the closure-above of the products of our graphs over-approximate the actual predicate after r rounds. And thus algorithms working on these approximations work on the actual predicate.

Now, let's start with simple closed-above predicates. Just like for the one round case, they are completely characterized by the domination number of their underlying graph.

Definition 4.43 (Multiple rounds k-set agreement algorithm)**.** Let $r > 0$. Assume that the set of initial values to k-set agreement is totally ordered. Then the r **round k-set agreement algorithm parametrized by a set of process**, $r - roundsParam \triangleq \forall S \subseteq \Pi$, $r - roundsParam(S)$ is the algorithm where each process

- broadcasts the set of pairs (process,initial value) that it knows for r rounds;

- and decides the minimum value it received from a process of S. If no such value were received, it decides the minimal value it received from any process.

Theorem 4.44 (Upper bound (multiple rounds) for simple closed-above predicates)**.** *Let $\mathcal{HO}$ be a simple closed-above predicate generated by the graph G. Let $r > 0$ and D be a minimum dominating set of G^r. Then Algorithm $r - roundsParam(D)$ solves $\gamma(G^r)$-set agreement in r rounds on $\mathcal{HO}$.*

Proof. Because $r - roundsParam(D)$ is an oblivious algorithm, at round r every process p takes its decision from only the pairs of (process, initial values) it has received in the r rounds. This is equivalent to saying that p received all the initial values of its incoming neighbors in the product of the r communication graphs.

By Lemma 4.42, this product of graph is included in $\uparrow G^r$. Thus every processe receives at least the initial value from its incoming neighbors in G^r. Since D is a dominating set of G^r, this means every process decides an initial value from a process of D. And since $|D| = \gamma(()G^r)$, at most $\gamma(()G^r)$ values are decided.

The algorithm thus solves $\gamma(()G^r)$-set agreement on $\mathcal{HO}$. □

But for general closed-above predicates, one cannot use the domination number itself, because one cannot know which of the underlying graphs will be there. As in the one round case, these bounds rely on the equal-domination number and covering numbers.

Theorem 4.45 (Upper bound (multiple rounds) on k-set agreement by $\gamma^{eq}(S)$ for general closed-above models)**.** *Let $\mathcal{HO}$ be a general closed-above predicate generated by the set of graphs S. Let $r > 0$. Then Algorithm $r - rounds$ solves $\gamma^{eq}(S^r)$-set agreement in r rounds on $\mathcal{HO}$.*

Proof. Because $r - rounds$ is an oblivious algorithm, at round r every process p takes its decision from only the pairs of (process, initial values) it has received in the r rounds. This is equivalent to saying that p received all the initial values of its incoming neighbors in the product of the r communication graphs.

By Lemma 4.42, this product of graph is included in $\bigcup_{G_1,\ldots,G_r \in S} \uparrow \bigcirc_{i=1}^{r} G_i = \uparrow S^r$. Thus there is a graph G in S^r such that every process receives at least the initial value from its incoming neighbors in G.

Let P be a set of $\gamma^{eq}(S^r)$ processes with the smallest initial values. They have thus at most $\gamma^{eq}(S^r)$ distinct initial values. By Definition 4.10 of $\gamma^{eq}(S^r)$, P dominates every graph in S^r, including G. Thus taking the minimum value received will result in deciding one of those initial values, and thus one of at most $\gamma^{eq}(S^r)$ values.

We conclude that the algorithm solves $\gamma^{eq}(S^r)$-set agreement in r round on $\mathcal{HO}$. □

Theorem 4.46 (Upper bounds (multiple rounds) on k-set agreement by covering numbers for general closed-above predicates)**.** *Let $\mathcal{HO}$ be a general closed-above predicate generated by the set of graphs S. Let $r > 0$. Then $\forall i \in [1, \gamma^{eq}(S^r)[$: Algorithm $r - rounds$ solves $(i + (n - cov_i(S^r)))$-set agreement in r rounds on $\mathcal{HO}$.*

Proof. Because $r - rounds$ is an oblivious algorithm, at round r every process p takes its decision from only the pairs of (process, initial values) it has received in the r rounds. This is equivalent to saying that p received all the initial values of its incoming neighbors in the product of the r communication graphs.

By Lemma 4.42, this product of graph is included in $\bigcup_{G_1,\dots,G_r\in S} \uparrow \bigodot_{i=1}^{r} G_i = \uparrow S^r$. Thus there is a graph G in S^r such that every process receives at least the initial value from its incoming neighbors in G.

For a set of i processes with the i smallest initial values, they will reach at least $cov_i(S^r)$ processes in G. Thus these processes will decide one of the i values when taking the smallest value they received.

As for the rest of the processes, we can't say anything about what they will receive, and thus we consider the worst case, where they all decide differently, and not one of the i smallest values. Then the number of decided values is at most $i + (n - cov_i(S^r))$, and the theorem follows. $\square$

One issue with these bounds is that they require the computation of possibly many products, as well as the computation of the combinatorial numbers for a lot of graphs. One alternative is to forsake the best bound possible for one that can be computed using only the numbers for the initial graphs.

This hinges on covering number sequences. Recall that the i-th covering number of a graph is the minimum number of processes hearing a set of i processes that do not broadcast. In a sense, it gives the guaranty of propagation of information by a set of i processes.

That's the whole story for one round. But what happens when you do multiple rounds? Then, if the i-th covering number of the graph is greater than i, this means that in the next rounds, the minimum number of people who will hear the value of the i initial processes is the cov_i-th covering number. And if this number is greater than cov_i, this repeats.

Covering number sequences capture this process. One can also see them as the sequences of covering numbers for powers of the graph.

Definition 4.47 (Covering number sequences)**.** Let G be a graph. Then the **i-th covering numbers sequence** of $G \triangleq (s_j^i)_{j\ \in\mathbb{N}^*}$ such that $s_1^i = cov_i(G)$ and $\forall k \geq 1 : s_{k+1}^i = \begin{pmatrix} |\Pi| & \textit{if } s_k^i \geq \gamma^{eq}(G) \\ cov_{s_k^i}(G) & \textit{if } s_k^i < \gamma^{eq}(G) \end{pmatrix}$

Armed with these sequences, an upper bound follows from G directly.

Theorem 4.48 (Upper bounds on k-set agreement by covering numbers sequences)**.** *Let $\mathcal{HO}$ be a simple closed-above predicate defined by the graph G on Π. Then if the i-th covering sequence of G reaches $|\Pi|$ for the first time at round t, $\forall r \geq t$: Algorithm $r - rounds$ solves i-set agreement on the predicate $\mathcal{HO}$.*

Proof. Let S be the set of the i smallest initial values. There are at least k processes with one of these values. Now the i-th covering sequence of G gives us a lower bound on the number of processes who know these values for each round. We show this by induction on the index j of the sequence elements.

- **(Base case)** $j = 1$. Then after one round, all processes who heard from our i initial processes know their initial values; this number is lower bounded by $cov_i(G)$.
- **(Induction step)** $j = t + 1$ and the result holds $\forall j \leq t$. Notably, s_t^i lower bounds the number of processes knowing one of the i initial values after t rounds.

– if $s_t \geq \gamma^{eq}(G)$ then whatever the set of processes knowing one of the i values after round t, they form a dominating set. And thus every process will know at least on of these values after round $t+1$, corresponding to $s^i_{t+1} = n$.
– If $s^i_t < \gamma^{eq}(G)$, then not all sets of size s^i_t are dominating sets. But they all reach at least $cov_{s^i_t}(G)$ processes. Thus after round $t+1$, at least that many processes know at least one of the i initial values.

Hence, if the i-th covering sequence reaches n after say round t, all processes know at least one initial value from every set of i processes. Notably, every process knows one of the i smallest initial values, and thus choosing the smallest value ensures that i-set agreement is solved.

Thus $\forall r \geq t$: Algorithm $r - rounds$ solves i-set agreement on $\mathcal{HO}$. □

This bound generalizes to general closed-above predicates by generalizing the covering numbers sequences to a set of graphs.

Definition 4.49 (Covering numbers sequences for sets of graphs)**.** Let s be set of graphs. Then the ***i*-th covering numbers sequence** of $S \triangleq (s_j)_{j \in \mathbb{N}^*}$ such that $s_1 = \min\limits_{G \in S} cov_i(G)$ and

$$\forall k \geq 1 : s_{k+1} = \begin{pmatrix} n & \textit{if } s_k \geq \max\limits_{G \in S} k_{eq-dom}(G) \\ \min\limits_{G \in S} cov_{s_k}(G) & \textit{if } s_k < \max\limits_{G \in S} k_{eq-dom}(G) \end{pmatrix}$$

Theorem 4.50 (Upper bounds on k-set agreement by covering numbers sequences for general closed-above predicates)**.** *Let S be a set of graphs on Π. Then if the i-th covering sequence of S reaches $|\Pi|$ for the first time at round t, $\forall r \geq t$: Algorithm $r-rounds$ solves i-set agreement on the closed-above predicate generated by S.*

Proof. If the i-th covering number sequence of S reaches n after step t, this means that every set of i processes is heard by everyone after t rounds. In particular, the i processes with the smallest initial values will be heard by everyone.

Hence $\forall r \geq t$: Algorithm $r - rounds$ solves i-set agreement. □

4.6.3 Lower bounds for multiple rounds

In the same way as upper bounds, lower bounds generalize cleanly to multiple rounds.

Theorem 4.51 (Lower bound (multiple rounds) on k-set agreement for simple closed-above predicates)**.** *Let $r > 0$ and let $\mathcal{HO}$ a simple closed-above predicate generated by the graph G different from the clique.*
Then $(\gamma(G) - 1)$-set agreement is not solvable on $\mathcal{HO}$ in r rounds by an oblivious algorithm.

Proof. Because we only consider oblivious algorithms, we can consider the graphs given by the products of r graphs of $\mathcal{HO}$ as generating a predicate, and apply our bound for one round. The trick is to know if $(\uparrow G)^r$ is itself a simple closed-above predicate. One would think that probably $(\uparrow G)^r = \uparrow G^r$, but this is not true in general, as shown in the examples at the beginning of the section.

On the other hand, $(\uparrow G)^r$ contains a subcomplex which has all the properties that we use in the proof of our lower bound: $G^{r-1}. \uparrow G$.

- It is a pseudosphere. Indeed, such a complex contains the uninterpreted complex of $G^{r-1}.G = G^r$; and each edge added to G only changes the view of one process in the product, the one receiving the new message.

  Hence we can change the view of each process independently of the others (because we only add messages to the last graph, and thus no two such messages can interfere with each other to create a new path).

- It contains the full simplex. This follows from the fact that $\uparrow G$ contains the clique, and our product operation maintain this graph.

- It is included in the complex $\uparrow G^r$. This follows from the fact that adding messages to G can only add messages to the product, and thus all considered graphs contain G^r and thus are in $\uparrow G^r$.

This means that this subcomplex can be treated just like the complex of $\uparrow G^r$ in our lower bound for one round. The theorem follows. $\square$

Theorem 4.52 (Lower bound (multiple rounds) on k-set agreement for general closed-above predicates)**.** *Let $r > 0$ and let $\mathcal{HO}$ be a closed-above predicate generated by the set of graphs S, such that S is not the singleton set of the clique..*
Let $l = \min(\gamma^{dist}(S^r), \min_{t\in[1,\gamma^{dist}(S^r)[} t + M_t(S^r)) - 2$. Then $(l+1)$-set agreement is not solvable on $\mathcal{HO}$ in r rounds by an oblivious algorithm.

Proof. The idea is the same that for the proof above, except that for each product of r graphs $G_1.G_2.\dots.G_r$, we consider the complex of the graphs in $G_1.G_2.\dots.G_{r-1}.\uparrow G_r$.

This satisfies the same three properties used in our lower bound proof than $\uparrow (G_1.G_2.\dots.G_r)$, and thus our lower bound gives the same result.

This means we can consider the union of our complexes like the the union of the complexes for $\uparrow (G_1.G_2.\dots.G_r)$ for each product of r graphs of S, that is like the complex of the general closed-above predicate generated by S^n. $\square$

As a concrete applications of these bounds, let's consider a classical family of subgraphs: stars.

Definition 4.53 (Star graph)**.** Let G be a graph. Then G is a **star graph** $\triangleq \exists c \subseteq \Pi : G = (V, \{c\} \times \Pi)$.

Theorem 4.54 (Lower bound for stars)**.** *Let S be the set of graphs which are unions of s star graphs with different centers. Then $(n-s)$-set agreement is not solvable in the closed-above predicate generated by S.*

Proof. First, we have $\forall r > 0 : \gamma^{dist}(S^r) = \gamma^{dist}(S) = n - s + 1$.

The first equality follows from the fact that the product of any star graph with itself is the same initial star graph; hence $S \subseteq S^r$, and a set of $n-s$ processes in the $n-s$ graphs with stars at the other s processes does not dominate these graphs. This actually gives $\gamma^{dist}(S^r) \geq \gamma^{dist}(S)$; the other direction follows form the fact that $\forall G \in S^r, \exists H \in S : G \in\uparrow H$. Hence every set of i graphs in S^r will have more edges than some set of i graphs in S, and thus will be easier to dominate.

	One round	Multiple (r) rounds
Upper Bound by $\gamma^{eq}(S)$	$\gamma^{eq}(S)$	$\gamma^{eq}(S^r)$
Upper Bound by $cov_i(S)$	$i + (n - cov_i(S))$	$i + (n - cov_i(S^r))$
Lower Bound	$\min\begin{pmatrix}\gamma^{dist}(S), \\ \min\limits_{t\in[1,\gamma^{dist}(S)[} t + M_t(S)\end{pmatrix} - 1$	$\min\begin{pmatrix}\gamma^{dist}(S^r), \\ \min\limits_{t\in[1,\gamma^{dist}(S^r)[} t + M_t(S^r)\end{pmatrix} - 1$

Table 4.1: Overview of the main results of the chapter.

As for the second equality, it follows from the fact that from a set of $n - s + 1$ processes, and among $n - s + 1$ distinct graphs, at least one of the process is the center of a star in one of the graph, and thus the processes dominates the graphs.

On the other hand, $\forall t \in [1, \gamma^{dist}(S) - 1] : t + M^t(S^r) - 2 = n - 2$. This equality comes from the fact that any set of t processes that does not dominate a given set of t graphs of S^r is distinct from the stars of these t graphs. Thus they are silent, and then $max\text{-}cov_t(S^r) = t$ and $M^t(S^r) = n - t$. We thus have $t + M^t(S^r) - 2 = t + n - t - 2 = n - 2$.

Therefore $\min(\gamma^{dist}(S^r), \min\limits_{t\in[1,\gamma^{dist}(S^r)[} t + M_t(S^r)) - 2 = \min(n - s + 1, n) - 2 = n - s - 1$. We conclude by Theorem 4.52. □

4.7 Conclusion

4.7.1 Summary

In this chapter, I proved upper and lower bounds on k-set agreement for closed-above predicates, the class of heard-of predicates defined by subgraphs that must be present in the communication graph at each round. These predicates encompass many message-passing models of distributed computing focused on safety properties. Table 4.1 summarizes these bounds.

Regarding the bounds themselves, although their proofs leverage combinatorial topology, all the bounds here are expressed in terms of combinatorial numbers of the graphs. That is, these bounds can be used without any knowledge of combinatorial topology. Yet combinatorial topology was instrumental in showing such sweeping results.

4.7.2 History of the research

This research comes from a collaboration with Armando Casteñada. Armando came to Toulouse to work with my advisors in the summer of 2018, when I proposed a collaboration to him on applying combinatorial topology to the Heard-Of model.

This spawned a back and forth, which resulted in a proof of the idempotence of many protocol complexes linked with the Heard-Of model, and subsequent lower bounds for the k-set agreement. These results were written up for PODC 2019. Yet a week before the submission, we realized that I made a mistake which greatly limited the breadth of the result.

The paper was thus scrapped, and the research started anew. During my two-month research stay in Mexico in May/June 2019, Armando proposed what would become the closed-above predicates, and the results followed from here.

4.7.3 Perspectives

Two direct extensions of this research would be: bounds on predicates that are not closed-above, maybe not even oblivious; and an analysis of the tightness of the lower bounds.

Although closed-above predicates are a natural subsets, they still limit greatly the models that can be studied. But predicates not closed-above, or not even oblivious, would not reduce to the one round case, and thus would require new ideas.

For the tightness, it's hard to know whether there's a big gap between the upper and lower bounds proven in this chapter. My intuition would say that they are close, but further analysis is required to prove or disprove this conjecture.

Chapter 5

Conclusion

5.1 Summary

The Heard-Of model aims to solve a fundamental problem of distributed computing: the overabundance of incomparable models. It abstracts away many different parameters into a formal property about infinite sequences of graphs. This elegant formalism then makes it possible to prove sweeping results linking models, and to go from one to another in as systematic way.

Yet it is not used much by researchers. This thesis addresses three angles on this problem:

- **How to find, in a principled way, the right heard-of predicate to study a given model**. For someone to use the Heard-Of model instead of another one, the latter must be expressed in the former. I proposed to do this by adding an intermediary step between the operational models and the heard-of predicates: the delivered predicates, which capture only the addition of rounds to the model, not any subtleties of asynchrony. I then showed how to derive delivered predicates from operational models – through combination of operations –, and how to derive heard-of predicates from delivered predicates – through strategies.

- **The existence of predicates equivalent in terms of solvability with specific models**. Once a model is expressed through a heard-of predicate, a natural question to ask is whether something was lost in this translation. I gave an element of answer by finding heard-of predicates equivalent to the Chandra-Toueg hierarchy of failure detectors. Because this equivalence is couched in terms of solvability, I also explored the different notions of solvability in the Heard-Of model, for which such equivalences can be proven, and how to move between them.

- **How to prove results on heard-of predicates using general powerful techniques.** Finally, a heard-of predicate serves to prove results. Here the elegant formalization makes things more difficult, as heard-of collection tend to be mathematically more complex to handle than classical models of distributed computation. A need thus exists for a systematic approach to derive computability and complexity results. I started one such approach by applying combinatorial topology tools to the solvability of k-set agreement problem. When limiting ourselves to specific classes of heard-of predicates, these tools generate valuable lower and upper bounds on the k for which k-set agreement is solvable.

Although the problems with the Heard-Of model are far from completely solved, the research from this thesis cements the relevance of this model for studying distributed computing. It entails that common models can be abstracted by heard-of predicates, without losing too much power, and that results can be proved on these models with general techniques.

5.2 Perspectives

When thinking about what still needs to be done, I see two parallel threads: studying further the three aspects of the Heard-Of model investigated in this thesis; and exploring other aspects of the Heard-Of model.

5.2.1 Further perspective on my work

Finding the right heard-of predicate In Chapter 2, the perspectives focused on the next steps in the study of delivered predicates and strategies. Zooming out, one question becomes obvious: is there always a characterizing heard-of predicate for a given delivered predicate?

My intuition say that there is. Yet this is not trivial to prove, because there might be an infinite sequence of improving strategies without a concrete limit. The vocabulary smells of point-set topology, and that's where my intuition tells me that the proof of this statement – or its refutation – lies.

Even if delivered predicates without a characterizing heard-of predicate exist, they might be so pathological as to not matter in practice. Which means that the next best thing to proving the existence of a characterizing heard-of predicate is to delineate the set of delivered predicates without one.

Comparing operational models and heard-of predicate Here too, the perspectives of the corresponding chapter dealt with the concrete next step. This leave us free to ask the underlying question: is there an operational model without an equivalence heard-of predicate?

I believe the answer is yes, yet I failed to find any such example. Maybe its because every example is pathological, or maybe the models I explored are too well-behaved for that. Either way, finding such a model would be a very important result, even more so if it reveals what is lost in going to rounds.

If all models have an equivalent heard-of predicate, what to do becomes slightly more tricky. The lack of formalization of operation models – that's what the Heard-Of model is here for, after all – means that no sweeping result can be proved on "every operational model". The best one might hope for is a result on a large class of operation models which allows formalization.

Proving results on heard-of predicates The research program for this last direction points more towards mathematical understanding than a deep concept: the study of the protocol complexes of heard-of predicates. My intuition screams that this is the right approach to analyzing distributed problems in the Heard-Of model, and that the only obstacle in the way of this approach is our inability to prove topological property about the relevant complexes.

What is left to discover is whether the right mathematical tool already exists in the combinatorial topology literature, or if distributed computing researchers will need to forge their own.

5.2.2 Other questions

I already proposed many other open problems in my SIGACT News Distributed Computing Column [3]; this subsection presents some of them in more detail.

Translations between heard-of predicates From the start, the Heard-Of model comes with a notion of translation between predicates. Such a translation from predicate HO_1 to predicate HO_2 captures the possibility of using multiple rounds of HO_1 to implement one round of HO_2. Intuitively, the same messages are sent for multiple rounds, and after that many rounds, all that was received satisfy the property defining HO_2.

Studying the closure of predicates through such translations would provide theoretical and practical advances: the former thanks to the definition of equivalence classes of predicates, and the latter by allowing one to check if an algorithm exists for a predicate by checking if an algorithm exists for any predicate in its closure. As of now, the only in-depth examination of these translations lies in Schmid et al. [40].

Probabilistic Heard-Of model The Heard-Of model is deterministic. Given an algorithm and a collection, the behavior of the system is completely determined. Yet probabilistic models and reasoning prove an important part of distributed computing. It's thus only natural to ask how one might add probability in this model.

I see two distinct approach. The first is to allow probabilistic algorithms. This has the benefits of changing nothing about how predicates are defined; it only makes it harder to study the behavior of the system. The other, more original take, is to make the predicate probabilistic. Instead of saying that an algorithm should be correct for any collection in the predicate, having a measure of probability on the set defined by the predicate would allow a probabilistic definition of correctness: the algorithms must finish with probability 1, for example. If previous work is an indication – like Ben-Or's randomized consensus algorithm [74] –, then this relaxation will yield fascinating new algorithms.

Repeated tasks The original paper by Charron-Bost and Schiper [37] focuses on consensus. One limitation of consensus comes from it being a one-shot task: you solve it once, and then you're done. Yet many problems in distributed computing fall on the repeated task side: one must solve something over and over again. A perfect example is the implementation of distributed objects.

This question intersect with the detection of termination, the composition of algorithms in the Heard-Of model, and whereas the deterministic behavior given the input and the collection must be maintained at all cost. Work in this direction is lacking, although some preliminary research by Andrei et al. [48] attempts to tackle it.

Bibliography

[1] K. R. Popper, *The Logic of Scientific Discovery.* London: Hutchinson, 1934. (Cited in page 2.)

[2] R. Hoffmann, "What might philosophy of science look like if chemists built it?," *Synthese*, vol. 155, no. 3, pp. 321–336, 2007. (Cited in page 2.)

[3] A. Shimi, "The splendors and miseries of rounds," *SIGACT News*, vol. 50, pp. 35–50, Sept. 2019. (Cited in pages 3, 7, and 126.)

[4] T. Elrad and N. Francez, "Decomposition of distributed programs into communication-closed layers," *Science of Computer Programming*, vol. 2, no. 3, pp. 155–173, 1982. (Cited in page 4.)

[5] M. Chaouch-Saad, B. Charron-Bost, and S. Merz, "A reduction theorem for the verification of round-based distributed algorithms," in *Proceedings of the 3rd International Workshop on Reachability Problems*, RP '09, pp. 93–106, Springer-Verlag, 2009. (Cited in page 4.)

[6] A. Damien, C. Dragoi, A. Militaru, and J. Widder, "Reducing asynchrony to synchronized rounds," *CoRR*, vol. abs/1804.07078, 2018. (Cited in page 4.)

[7] K. V. Gleissenthall, R. G. Kici, A. Bakst, D. Stefan, and R. Jhala, "Pretend synchrony: Synchronous verification of asynchronous distributed programs," *Proc. ACM Program. Lang.*, vol. 3, pp. 59:1–59:30, Jan. 2019. (Cited in page 4.)

[8] A. Cornejo and F. Kuhn, "Deploying wireless networks with beeps," in *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pp. 148–162, Springer-Verlag, 2010. (Cited in pages 5 and 17.)

[9] A. Cornejo, A. Dornhaus, N. Lynch, and R. Nagpal, "Task allocation in ant colonies," in *Proceedings of the 28th International Conference on Distributed Computing*, DISC'14, pp. 46–60, Springer-Verlag, 2014. (Cited in page 5.)

[10] S. Gilbert, J. Maguire, and C. Newport, "On bioelectric algorithms: A novel application of theoretical computer science to core problems in developmental biology," *CoRR*, vol. abs/1809.10046, 2018. (Cited in page 5.)

[11] I. Chlamtac and S. Kutten, "On broadcasting in radio networks - problem analysis and protocol design," *IEEE Transactions on Communications*, vol. 33, pp. 1240–1246, Dec. 1985. (Cited in pages 5 and 16.)

[12] F. Kuhn, N. Lynch, and R. Oshman, "Distributed computation in dynamic networks," in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pp. 513–522, ACM, 2010. (Cited in page 5.)

[13] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, pp. 133–169, May 1998. (Cited in pages 5 and 19.)

[14] F. P. Junqueira, B. C. Reed, and M. Serafini, “Zab: High-performance broadcast for primary-backup systems,” in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN ’11, pp. 245–256, IEEE Computer Society, 2011. (Cited in page 5.)

[15] B. M. Oki and B. H. Liskov, “Viewstamped replication: A new primary copy method to support highly-available distributed systems,” in *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC ’88, pp. 8–17, ACM, 1988. (Cited in pages 5 and 19.)

[16] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” *J. ACM*, vol. 43, pp. 225–267, Mar. 1996. (Cited in pages 5, 18, 65, 67, 68, and 75.)

[17] T. D. Chandra, V. Hadzilacos, and S. Toueg, “The weakest failure detector for solving consensus,” *J. ACM*, vol. 43, pp. 685–722, July 1996. (Cited in pages 5 and 65.)

[18] S. Rajsbaum and M. Raynal, “Mastering concurrent computing through sequential thinking,” *Commun. ACM*, vol. 63, pp. 78–87, Dec. 2019. (Cited in page 5.)

[19] M. Herlihy, D. Kozlov, and S. Rajsbaum, *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann Publishers Inc., 1st ed., 2013. (Cited in pages 5, 98, 99, 105, 106, 107, 109, 110, and 113.)

[20] M. Herlihy and N. Shavit, “The topological structure of asynchronous computability,” *J. ACM*, vol. 46, pp. 858–923, Nov. 1999. (Cited in pages 5 and 6.)

[21] E. Arjomandi, M. J. Fischer, and N. A. Lynch, “A difference in efficiency between synchronous and asynchronous systems,” in *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC ’81, pp. 128–132, ACM, 1981. (Cited in page 6.)

[22] P. Fraigniaud, A. Korman, and D. Peleg, “Towards a complexity theory for local distributed computing,” *J. ACM*, vol. 60, pp. 35:1–35:26, Oct. 2013. (Cited in page 6.)

[23] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 1st ed., 2009. (Cited in page 6.)

[24] O. Goldreich, *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1st ed., 2008. (Cited in page 6.)

[25] D. Peleg, *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. (Cited in pages 6, 15, 16, and 110.)

[26] A. Korman and S. Kutten, “On distributed verification,” in *Proceedings of the 8th International Conference on Distributed Computing and Networking*, ICDCN’06, pp. 100–114, Springer-Verlag, 2006. (Cited in page 6.)

[27] M. Göös and J. Suomela, “Locally checkable proofs,” in *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC ’11, pp. 159–168, ACM, 2011. (Cited in page 6.)

[28] M. Ghaffari, D. G. Harris, and F. Kuhn, “On derandomizing local distributed algorithms,” in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 662–673, Oct. 2018. (Cited in page 6.)

[29] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer, “Distributed verification and hardness of distributed approximation,” in *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC ’11, pp. 363–372, ACM, 2011. (Cited in page 6.)

[30] N. Santoro and P. Widmayer, “Time is not a healer,” in *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89*, pp. 304–313, Springer-Verlag New York, Inc., 1989. (Cited in page 7.)

[31] M. Raynal and M. Roy, “A note on a simple equivalence between round-based synchronous and asynchronous models,” in *Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing*, PRDC ’05, pp. 387–392, IEEE Computer Society, 2005. (Cited in pages 7 and 18.)

[32] Y. Afek and E. Gafni, “A simple characterization of asynchronous computations,” *Theor. Comput. Sci.*, vol. 561, pp. 88–95, Jan. 2015. (Cited in pages 7, 18, 66, 67, and 100.)

[33] M. Raynal and J. Stainer, “Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors,” in *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC ’13, pp. 166–175, ACM, 2013. (Cited in pages 7, 18, 66, 67, and 72.)

[34] E. Gafni, “Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony,” in *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’98, (New York, NY, USA), pp. 143–152, ACM, 1998. (Cited in pages 7 and 18.)

[35] I. Keidar and A. Shraer, “Timeliness, failure-detectors, and consensus performance,” in *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’06, pp. 169–178, ACM, 2006. (Cited in page 8.)

[36] R. Guerraoui, “Indulgent algorithms (preliminary version),” in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’00, pp. 289–297, ACM, 2000. (Cited in page 8.)

[37] B. Charron-Bost and A. Schiper, “The heard-of model: computing in distributed systems with benign faults,” *Distributed Computing*, vol. 22, pp. 49–71, Apr. 2009. (Cited in pages 8, 9, 10, 13, 18, 26, 27, 39, 98, and 127.)

[38] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *J. ACM*, vol. 32, pp. 374–382, Apr. 1985. (Cited in pages 9 and 18.)

[39] B. Charron-Bost, M. Függer, and T. Nowak, “Approximate consensus in highly dynamic networks: The role of averaging algorithms,” in *Automata, Languages, and Programming*, pp. 528–539, 2015. (Cited in pages 9, 98, and 100.)

[40] U. Schmid, M. Schwarz, and K. Winkler, "On the strongest message adversary for consensus in directed dynamic networks," in *Structural Information and Communication Complexity*, pp. 102–120, Springer International Publishing, 2018. (Cited in pages 9 and 127.)

[41] É. Coulouma, E. Godard, and J. Peters, "A characterization of oblivious message adversaries for which consensus is solvable," *Theoretical Computer Science*, vol. 584, pp. 80–90, 2015. (Cited in pages 9, 98, and 99.)

[42] T. Nowak, U. Schmid, and K. Winkler, "Topological characterization of consensus under general message adversaries," in *2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, 2019. (Cited in pages 9, 98, 99, and 101.)

[43] A. R. Balasubramanian and I. Walukiewicz, "Characterizing consensus in the heard-of model," in *31st International Conference on Concurrency Theory (CONCUR 2020)* (I. Konnov and L. Kovács, eds.), vol. 171, (Dagstuhl, Germany), pp. 9:1–9:18, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. (Cited in page 9.)

[44] M. Biely, P. Robinson, M. Schmid, Ulrich Schwarz, and K. Winkler, "Gracefully degrading consensus and k-set agreement in directed dynamic networks," *Theoretical Computer Science*, vol. 726, pp. 41–77, 2018. (Cited in pages 9 and 98.)

[45] B. Charron-Bost, H. Debrat, and S. Merz, "Formal verification of consensus algorithms tolerating malicious faults," in *Stabilization, Safety, and Security of Distributed Systems*, pp. 120–134, Springer Berlin Heidelberg, 2011. (Cited in page 9.)

[46] O. Marić, C. Sprenger, and D. Basin, "Cutoff bounds for consensus algorithms," in *Computer Aided Verification*, pp. 217–237, Springer International Publishing, 2017. (Cited in page 9.)

[47] C. Drăgoi, T. A. Henzinger, and D. Zufferey, "Psync: A partially synchronous language for fault-tolerant distributed algorithms," *SIGPLAN Not.*, vol. 51, pp. 400–415, Jan. 2016. (Cited in page 9.)

[48] A. Damian, C. Drăgoi, A. Militaru, and J. Widder, "Communication-closed asynchronous protocols," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), pp. 344–363, 2019. (Cited in pages 15 and 127.)

[49] M. Haundefinedćkowiak, M. Karoundefinedski, and A. Panconesi, "On the distributed complexity of computing maximal matchings," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pp. 219–225, 1998. (Cited in page 16.)

[50] F. Kuhn, N. Lynch, and R. Oshman, "Distributed computation in dynamic networks," in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, (New York, NY, USA), pp. 513–522, ACM, 2010. (Cited in page 17.)

[51] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari, "Design patterns in beeping algorithms: Examples, emulation, and analysis," *Information and Computation*, vol. 264, pp. 32–51, 2019. (Cited in page 17.)

[52] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988. (Cited in page 19.)

[53] M. Hutle and A. Schiper, "Communication predicates: A high-level abstraction for coping with transient and dynamic faults," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 92–101, June 2007. (Cited in page 19.)

[54] A. Shimi, A. Hurault, and P. Quéinnec, "Characterizing asynchronous message-passing models through rounds," in *22nd Int'l Conf. on Principles of Distributed Systems (OPODIS 2018)*, pp. 18:1–18:17, 2018. (Cited in pages 25 and 62.)

[55] A. Shimi, A. Hurault, and P. Queinnec in *Formal Techniques for Distributed Objects, Components, and Systems*, (Cham), pp. 133–149, Springer International Publishing, 2020. (Cited in page 25.)

[56] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, "Shared memory vs message passing," *Tech. Rep. IC/2003/77, EPFL.*, 2003. (Cited in page 65.)

[57] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg, "The weakest failure detectors to solve certain fundamental problems in distributed computing," PODC '04, pp. 338–346, 2004. (Cited in page 65.)

[58] P. Jayanti and S. Toueg, "Every problem has a weakest failure detector," PODC '08, pp. 75–84, 2008. (Cited in page 66.)

[59] V. Bhatt and P. Jayanti, "On the existence of weakest failure detectors for mutual exclusion and k-exclusion," DISC'09, pp. 311–325, 2009. (Cited in page 66.)

[60] B. Charron-Bost, M. Hutle, and J. Widder, "In search of lost time," *Inf. Process. Lett.*, vol. 110, pp. 928–933, Oct. 2010. (Cited in page 66.)

[61] M. Biely, M. Hutle, L. D. Penso, and J. Widder, "Relating stabilizing timing assumptions to stabilizing failure detectors regarding solvability and efficiency," SSS'07, pp. 4–20, Springer-Verlag, 2007. (Cited in page 67.)

[62] B. Charron-Bost, R. Guerraoui, and A. Schiper, "Synchronous system and perfect failure detector: Solvability and efficiency issue," in *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, DSN '00, (Washington, DC, USA), pp. 523–532, IEEE Computer Society, 2000. (Cited in page 67.)

[63] M. Moir and J. H. Anderson, "Wait-free algorithms for fast, long-lived renaming," *Science of Computer Programming*, vol. 25, no. 1, pp. 1–39, 1995. (Cited in page 93.)

[64] A. Castañeda, S. Rajsbaum, and M. Raynal, "The renaming problem in shared memory systems: An introduction," *Computer Science Review*, vol. 5, no. 3, pp. 229–251, 2011. (Cited in page 93.)

[65] S. Chaudhuri, "More choices allow more faults: Set consensus problems in totally asynchronous systems," *Information and Computation*, vol. 105, no. 1, pp. 132–158, 1993. (Cited in page 98.)

[66] A. Castañeda, P. Fraigniaud, A. Paz, S. Rajsbaum, M. Roy, and C. Travers, “A topological perspective on distributed network algorithms,” in *Structural Information and Communication Complexity*, pp. 3–18, 2019. (Cited in pages 98, 102, 110, 112, and 114.)

[67] M. Saks and F. Zaharoglou, “Wait-free k-set agreement is impossible: The topology of public knowledge,” *SIAM J. Comput.*, vol. 29, pp. 1449–1483, Mar. 2000. (Cited in page 99.)

[68] E. Borowsky and E. Gafni, “Generalized FLP impossibility result for T-resilient asynchronous computations,” in *Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC ’93, pp. 91–100, ACM, 1993. (Cited in page 99.)

[69] A. Castañeda and S. Rajsbaum, “New combinatorial topology bounds for renaming: the lower bound,” *Distributed Computing*, vol. 22, pp. 287–301, Aug 2010. (Cited in page 99.)

[70] M. Herlihy and S. Rajsbaum, “The topology of distributed adversaries,” *Distributed Computing*, vol. 26, pp. 173–192, Jun 2013. (Cited in page 99.)

[71] D. Alistarh, J. Aspnes, F. Ellen, R. Gelashvili, and L. Zhu, “Why extension-based proofs fail,” *CoRR*, vol. abs/1811.01421, 2018. (Cited in page 99.)

[72] E. Godard and E. Perdereau, “k-set agreement in communication networks with omission faults,” in *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, vol. 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 8:1–8:17, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. (Cited in page 99.)

[73] D. N. Kozlov, *Combinatorial Algebraic Topology*, vol. 21 of *Algorithms and computation in mathematics*. Springer, 2008. (Cited in page 108.)

[74] M. Ben-Or, “Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols,” PODC ’83, pp. 27–30, Association for Computing Machinery, 1983. (Cited in page 127.)

Résumé

La théorie des systèmes distribués, à l'inverse de l'informatique théorique séquentielle, étudie énormément de modèles différents et incomparables. En effet, elle se concentre sur l'incertitude inhérente de la communication et de la synchronisation, qui peut se modéliser de milles façons différentes. Cela entraine une explosition du nombre de modèles à considérer, et avec une difficulté à tous les garder en tête, les comprendre et les comparer.

Une solution récente à ce problème utilise le concept de rounds – ou tours : une structure où chaque processus envoie à tous un message annoté avec son numéro de round courant, attend un certain nombre de messages annotés avec ce même numéro, et puis utilise ses messages pour ses calculs locaux, avant de passer au round suivant en incrémentant son numéro. Aucune proposition n'utilise cette idée aussi bien, à mon avis, que le modèle Heard-Of de Charron-Bost et Schiper. Ce dernier offre comme avantage significatif une absence – ou au moins une minimisation – d'hypothèses opérationelles : pas de synchronie, pas d'adversaires, pas même de fautes. Tout revient aux messages qui sont reçus par les processus à temps – c'est-à-dire avant la fin du round correspondant chez le recepteur. Ces contraintes sur les messages forment des collections heard-of, qui elles même forment des prédicats heard-of, l'équivalent des modèles classiques.

Malgré les promesses du modèle Heard-Of, il ne jouit pas de la popularité qu'il mérite au sein de la communauté des systèmes distribués. Je conjecture que cela vient de trois problèmes non encore résolus, qui limitent son utilité pour les chercheurs. Ces trois problèmes sont : comment trouver le prédicat heard-of qui correspond à un modèle classique donné ; quelles pertes surviennent lors de cette conversion ; et comment prouver des résultats généraux sur des prédicates heard-of.

Ma thèse explore ces trois questions, et fournit des premiers éléments de réponse : une formalisation de la dérivation d'un prédicat heard-of correspondant à un certain modèle, avec en supplément une méthodologie pour simplifier cette dérivation en décomposant le modèle de départ en une combinaison de modèles plus simples ; une équivalence entre de nouveaux prédicat heard-of et les modèles à messages asynchrones augmentés de détecteurs de fautes, ainsi qu'une exploration des hypothèses derrière cette équivalence et ses prédécesseurs dans la littérature ; et des résultats d'impossibilité pour le k-set agreement à travers la topologie combinatoire, se concentrant sur une classe de prédicats heard-of qui capture de nombreuses propriétés de sûreté.

Mots clés :

Abstract

Distributed computing differs from sequential computing mainly through its abundance of incomparable models. Whereas everything goes back to Turing machines in sequential computing, distributed computing models the inherent uncertainties of communication and synchronization, in many equally meaningful ways that don't fall under the umbrella of one true model. The need to study them all then leads to the complex landscape of distributed computing models.

A recent approach for dealing with this difficulty proposes to unify models through communication-closed rounds: sequences of steps where everyone sends a message tagged with the current round number, waits for messages with this same round number, and then uses them to compute its next state and change round. The most promising take on this approach, in my opinion, is the Heard-Of model of Charron-Bost and Schiper. One significant advantage of this model over alternatives lies in its lack of operational assumptions: no synchrony, no adversary, not even failures. Everything follows from which message is received on time – before the end of the corresponding round at the receiver. Collections capture these possible patterns of messages received on time, and predicates over these collections capture models of communication.

Yet this model lacks the attention that it deserves from the research community. I believe the reason lies on the following three unsolved problems: how to find the heard-of predicate corresponding to a given model; is anything lost in this translation; and how to prove general results on heard-of predicates.

This thesis addresses all three, and provides elements of answers: a formalization of how to derive the most meaningful heard-of predicate for a given model, along with a methodology for simplifying this derivation by decomposing the original model into a combination of simpler ones; an equivalence between new heard-of predicates and asynchronous message-passing with failure detectors, along with an analysis of the underlying assumptions of such and previous equivalences; and general impossibility results for k-set agreement using combinatorial topology, and focusing on a class of heard-of predicates capturing many safety properties.

Keywords: Distributed Systems, Rounds, Heard-Of Model