Maintaining Simultaneously Consistent Views of a Distributed System using Common Knowledge

Tal Mizrahi

Maintaining Simultaneously Consistent Views of a Distributed System using Common Knowledge

Research Thesis

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in Electrical Engineering

Tal Mizrahi

Submitted to the Senate of
the Technion - Israel Institute of Technology

Elul, 5766 Haifa September, 2006

This Research Thesis Was Done Under The Supervision of
A/Prof. Yoram Moses in the Department of Electrical Engineering

Acknowledgement

I would like to express my uttermost gratitude to my advisor, A/Prof. Yoram Moses, for his devoted guidance throughout this research. His vision and extraordinary intuition have inspired me in my work.

My warm thanks to my parents, Tsipi and Joe, for their encouragement and support, to my brother, Amit, for all the LaTeX tips, and of course to my lovely wife, Hagit, for putting up with me during these long years in the Technion.

The Generous Financial Assistance of the Technion is Gratefully Acknowledged.

Contents

List of Figures

Abstract

In a distributed system it is often necessary for all processes to maintain a consistent view of the world. However, the existence of communication failures in a system may prevent processes from obtaining a consistent view. *Continuous consensus* is the problem of having each process i maintain at each time k an up-to-date core $M_i[k]$ of information about the past, so that the cores at all processes are guaranteed to be identical. Our analysis assumes an unreliable synchronous system, with an upper bound of t failures. The notion of continuous consensus enables a new perspective on classical problems such as the consensus or the Simultaneous Byzantine Agreement (SBA) problems, and allows for simpler analysis. A simple algorithm for continuous consensus in fault-prone systems with crash and sending omission failures called CONCON is presented, based on a knowledge-based analysis. Continuous consensus is shown to be closely related to common knowledge. Via this connection, the characterization of common knowledge by Moses and Tuttle is used to prove that CONCON is optimal—it produces the largest possible core at any given time. Finally, a second algorithm is presented that provides an optimum *uniform* solution to continuous consensus, in which all processes (faulty and nonfaulty) maintain the same core information at any given time.

List of Symbols and Abbreviations

$A(r,m)$	The set of active processes at (r,m).
$B_i(k)$	Set of bad processes computed by i at time $k+1$.
$b_i(k)$	Number of bad processes computed by i at time $k+1$.
β	Labelling function - determines which messages are successful.
C	Common knowledge among all processes.
C_N	Common knowledge among the nonfaulty process.
c	Short for $crit_i(k)$.
$D_S\varphi$	Distributed knowledge of φ among the processes in S.
$d(r,k)$	The difference between $\mathcal{N}(r,k)$ and k (used in the computation of waste).
$d_i^{(m)}(k)$	The difference between $\mathcal{N}_i^{(m)}(k)$ and $k-m$.
$dest_i(m)$	The destination of i at time m.
E	Edges of the communication graph.
$E_S\varphi$	Everyone in the set S knows φ.
$\mathcal{E}$	Set of monitored events.
$\mathcal{E}(\mathsf{V})$	The set of monitored external inputs determined by the view V.
e	An event or an external input.
F_ℓ	Set of potentially faulty processes in ℓ^{th} iteration of the fixed-point construction.
$\hat{F}$	The fixed-point of F_ℓ.
FIP	Full Information Protocol.

fm	Failure model.
$G_i(k)$	Set of good processes computed by i at time $k+1$.
g	A good process.
h	Index of the last iteration in the fixed-point construction.
$\mathsf{horizon}_i(k)$	The horizon of i at time k.
I	Set of possible input assignments in the system.
i	Process name.
j	Process name.
$K_i\varphi$	Process i knows φ.
k	Time / round counter.
$\hat{k}$	The final time in the fixed-point construction.
κ	Time / round counter.
$\mathcal{L}$	Logical language.
$Latest_i[k]$	Process i's estimation of the critical time for k in CONCON.
$Latest^{\mathsf{u}}_x[k]$	Process x's estimation of the critical time for k in UNICONCON.
ℓ	Time / round counter.
ℓ^{w}	The round in which the waste is reached.
ℓ^{lw}	The round in which the local waste is reached.
λ	The empty view.
$M_i[k]$	The core of i at time k.
$M^{\mathsf{C}}_i[m]$	Process i's core according to CONCON.
$M^{\mathsf{u}}_x[k]$	Process x's core according to UNICONCON.
m	Time / round counter.
N	The set of nonfaulty processes.

$\mathcal{N}(r,k)$	The number of failures discovered up to round k.
$\mathcal{N}_i^{(m)}(k)$	Number of failures discovered by i between rounds m and k.
n	Number of processes in the system.
$\mathbb{P}$	Set of processes in the system.
P	Protocol.
Φ	The set of primitive propositions in the logical language.
$\Phi_{\mathcal{E}}$	The set of primitive propositions confined to facts about external inputs.
φ	Fact or proposition.
R	System.
r	Run.
S_ℓ	Set of potentially nonfaulty processes in ℓ^{th} iteration of the fixed-point construction.
$\hat{S}$	The fixed-point of S_ℓ.
Σ_i	Set of initial local states of i.
t	Upper bound on the number of failures in the system.
V	Set of nodes in the communication graph.
$\mathsf{V}_i(m)$	View of process i at time m (subgraph of V).
$W(r)$	The waste or r.
$W_i^{(m)}(k)$	The local waste at time k w.r.t time m.
X	Contents of the core.
x	An arbitrary (possibly faulty) process.
z	An arbitrary (possibly faulty) process.
ζ	Input assignment function - assigns an input from $\mathcal{E}$ to every point.

Chapter 1

Introduction

1.1 Motivation

Maintaining consistency of the information held by different nodes in an unreliable distributed system is a challenging problem. Whether it is a system whose nodes share resources, or a system in which critical decisions are made according to the most recent data at hand, it is highly important to maintain consistency of the nodes' information.

A core of identical information maintained at different sites allows decisions performed in a distributed manner to be compatible with each other. Moreover, with the rapid growth of the internet over the last two decades, distributed systems operations are no longer restricted to being internal. In many cases, external elements interact concurrently with different nodes of the system. This places a stronger emphasis on the need to present a consistent view to the world at different nodes at a given time. A core of information that is guaranteed to be identical at all sites at any given time, and contains as much information as possible, is a very desirable tool in implementing such a consistent view. This thesis deals with the design of efficient protocols for maintaining such a core.

The challenge, however, is that different nodes in a distributed system typically have asymmetric information. Part of this information—the facts that are common knowledge—is identical for all agents and, moreover, can in principle be identified by each agent. By acting on information that is common knowledge, agents are guaranteed to be acting on consistent information available to all agents.

1.2 Previous Work

The role of common knowledge for consistent simultaneous actions has been firmly established in the literature [2, 3, 4, 5]. Dwork and Moses [3] presented an optimal[1] solution to simultaneous Byzantine agreement in the presence of crash failures, by using the notions of *clean rounds* and *waste*. They proved that simultaneous agreement can be reached exactly when the value of at least one agent's initial vote becomes common knowledge. Moses and Tuttle [4] extended this work to the more complex (sending) omission failure model, and presented optimal solutions for a broader class of simultaneous choice problems. Implicit in the latter work is the computation of a core of information that characterizes the common knowledge at any given point in time. This computation is based on a subtle fixed-point construction.

Providing an up-to-date consistent picture of the system at different sites can sometimes alleviate the need to explicitly activate voting or agreement protocols to handle individual transactions (see, for example, [6]). Weaker guarantees than simultaneous consistency are popular, where consistency is guaranteed over time: If one process can determine that an event has occurred, the others will eventually know this as well [7]. These weaker consistency conditions are essential in some systems since simultaneous coordination requires nontrivial common knowledge, and this is not attainable in truly asynchronous systems [2].

1.3 Research Goals

The current work introduces the *continuous consensus* problem, in which a core $M_i[k]$ of information is continuously maintained at every correct process i in the system. All

[1]Throughout this work we use the term *optimal* referring to the time it takes to reach agreement or consensus, rather than the computation complexity.

local copies of the core must be identical at all times k, and every interesting event from a set of possible events, $\mathcal{E}$, should eventually enter the core. The continuous consensus problem is studied in synchronous systems with crash and omission failures. We assume an upper bound of t failures in every run of our system. We shall show that the analysis of continuous consensus (CC for short) shown in this work allows for a simple and elegant solution to the problem of maintaining simultaneously consistent views in the system.

The continuous consensus problem generalizes many problems having to do with simultaneous coordination. For example, in the distributed *firing squad* problem [8], the system may receive an alarm message from the outside world. If a correct process receives such a message, then it is required that at some later point all correct processes "fire" simultaneously. In addition, "firing" is not allowed to occur in different rounds by different processes (hence in a non-simultaneous fashion), nor is it allowed to take place in a run before an alarm message has been received in the system. Clearly, if the arrival of an alarm message is a monitored event in a continuous consensus protocol, then the presence of an alarm in the shared core can be used as a necessary and sufficient condition for firing. Continuous consensus can also be used as a generalization of simultaneous versions of Byzantine agreement and the consensus problem (cf. [3]), as well as for the class of simultaneous choice problems of [4].

In this work we present an algorithm called ConCon that solves the CC problem. ConCon is **optimal** in providing at any given time the largest and most informative core possible. The algorithm will provide the most up-to-date consistent picture of the system, without the need to explicitly activate voting or agreement protocols to handle individual transactions.

A variant of the CC problem, which we call *uniform continuous consensus* (*UCC*), requires that the core be consistent among *all* processes in the systems, rather than just among the correct ones. We present UniConCon, which is a variant of ConCon that solves the UCC problem.

Our solutions to CC and UCC rely on a knowledge-based analysis. A close connection is shown between continuous consensus and common knowledge: it is shown that the core of shared information, $M_i[k]$ is common knowledge among the correct processes at time k. Moreover, the optimality of ConCon is proven using the characterization of common knowledge in the crash and omission failure models given in [4]. Our analysis also aims at extending Dwork and Moses' analysis of clean rounds and waste [3] to the more complicated omission model.

1.4 Outline

This work is organized as follows. Chapter 2 provides a formal definition of our system, and some technical background to the notions described in later chapters. Chapter 3, which is the heart of this work, presents the continuous consensus problem and its solution, ConCon, as well as proving its correctness and optimality. Chapter 4 describes the uniform continuous consensus problem, and introduces UniConCon. Finally, a few concluding remarks are presented in Chapter 5. Some of the detailed proofs are given in Appendix A.

Chapter 2

Preliminaries

Our treatment of the continuous consensus problem will be driven by a knowledge-based analysis. A general approach to modelling knowledge in distributed systems was initiated in [2] and given a detailed foundation in [5] (most relevant to the current work are Chapters 4 and 6). The lion's share of technical analysis in this thesis will be performed with respect to a single protocol, which gives rise to a specific class of systems. For ease of exposition, our definitions will be tailored to this particular setting.

2.1 The Communication Network

We consider a synchronous network with $n \geq 2$ possibly unreliable processes, denoted by $\mathbb{P} = \{1, 2, \ldots, n\}$. Each pair of processes is connected by a two-way communication link. Processes correctly identify the sender of every message they receive. They share a discrete global clock that starts out at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round $k+1$ taking place between time k and time $k+1$. Each process starts in some *initial state* at time 0. Then, in every following round, the process first sends a set of messages to other processes, and then receives messages sent to it by other processes during the same round. In addition, a process may also receive requests for service from clients external to the system (think, for example, of deposits and withdrawals at branches of a bank), or input from sensors with information about the world outside of the system (e.g., smoke detectors). Finally, the process may perform local computations based on the messages it has received. The history of an infinite execution of such a network will be called a *run*.

2.2 Nature's Role: Inputs and Failures

We think of a solution to the continuous consensus problem as a protocol operating (or playing) against an adversary called *nature*. Nature determines two central aspects of any given run: inputs and failures.

Inputs. We consider a setting in which every process starts out in an initial local state from some set Σ_i, and can receive an external input in any given round k (this input is considered as *arriving* at time k). The initial local state of each process can be thought of as its external input at time 0. We represent the external inputs in an infinite execution as follows. Define a set $V = \mathbb{P} \times \mathbb{N}$ of *process-time nodes* (or *nodes*, for short). An *(external) input assignment* is a function ζ associating with every (initial) node $\langle i, 0\rangle$ at time 0 an initial state from Σ_i and with each node $\langle i, k\rangle$ an input from a set of possible inputs, $\mathcal{I}$. Our analysis is independent of the type or structures of the elements of $\mathcal{I}$.

Failures. The second aspect of a run that is determined by nature is the identity of the faulty processes, and the details of their faulty behavior. These depend on the particular failure model being assumed. In this thesis we consider two closely-related failure models, called the *crash* model and the *sending omission* model, which is a generalization of the crash model. For simplicity, a process will be considered faulty in a run if it displays faulty behavior at any point during the run. In the crash failure model, a faulty process *crashes* in some round $k \geq 1$. In this case, it behaves correctly in the first $k-1$ rounds and sends no messages from round $k+1$ on. During its crashing round k, the process may succeed in sending messages on an arbitrary subset of its channels. In the sending omission model (or the omission model for short), a faulty process may omit to send messages in any given round. It sends messages only according to its protocol (it cannot misrepresent or lie), and nature determines for every round what subset of its messages will successfully be delivered. We remark that even faulty processes receive all messages sent to them over non-blocked channels. If a message is not delivered, its sender is necessarily faulty. We formally represent the *failure pattern* in a given run via an edge-labelled graph (V, E, β), where V is the set of process-time nodes defined above, and $E = \{(\langle i,k\rangle, \langle j,k+1\rangle) : i \neq j, k \geq 0\}$. An edge $e = (\langle i,k\rangle, \langle j,k+1\rangle) \in E$ stands for the round $k+1$ communication in the channel from i to j. The labelling function $\beta : E \to \{\mathtt{Y}, \mathtt{N}\}$ captures when such channels are blocked and when they operate correctly. Intuitively, $\beta(e) = \mathtt{N}$ means that e is blocked for communication, while $\beta(e) = \mathtt{Y}$ means that it is not blocked. In the latter case, a message on e, if sent, will be delivered.

Nature's combined contribution to a run r is captured by an *execution graph*. This is a labelled graph $\mathsf{G}^r = (V, E, \zeta, \beta)$ with labels ζ on the vertices giving the input assignment and labels β on the edges defining the failure pattern.[1] Notice that all execution graphs over n processes have the same edge and vertex sets (V, E)— a complete grid of $n \times \mathbb{N}$ nodes, with edges from each node $u \in V$ at level k to all nodes of level $k+1$ with a process name different from u's. Different execution graphs G differ only in the labelling functions ζ and β. Figure 2.1 contains an illustration of the nodes of an execution graph, with some of the edges describing round $k+1$. Observe that all edges from one time point to the next are in the graph—some are crossed, depicting their being blocked by β, while the others are available for communication.

We now consider particular subgraphs of $\mathsf{G} = (V, E, \zeta, \beta)$ that will be useful later on. Given a node $u = \langle i,m\rangle \in V$, we denote by V_u the set of nodes containing the nodes $\langle i,\ell\rangle \in V$ for all $\ell \leq m$ as well as all nodes $u' = \langle j,k\rangle \in V$ such that there is a directed path of edges of E from u' to u , in which all edges are labelled 'Y' . Intuitively, V_u contains all nodes about which u has potentially received information either directly or via a sequence of messages. We define the *maximal potential view* (or *view* for short) at node $u = \langle i,m\rangle$ in G, denoted by $\mathsf{V}_i(m)$, to be the subgraph of G generated by V_u , i.e.,

[1]The run r appears in the superscript of G^r. Throughout the thesis, we omit explicit reference to the run whenever it is clear from context.

$\mathsf{V}_i(m) = (V', E', \zeta', \beta') = (V_u, E \restriction V_u, \zeta \restriction V_u, \beta \restriction E')$. See Figure 2.1 for an illustration of a view $\mathsf{V}_i(m)$. The view $\mathsf{V}_S(k)$ of a set $S \subseteq \mathbb{P}$ of processes at a time k is defined to be the union of the graphs $\mathsf{V}_j(k)$, over all $j \in S$. We say that a view V is *contained* in a view V' if the nodes of V are a subset of those in V', and V is the subgraph of V' generated by these nodes. Thus, every node in V has the same external input as in V', and for every pair of nodes in V, they are connected by an edge in V exactly if they are connected by one in V'. In some cases, it will be convenient to talk about views $\mathsf{V}_S(k)$ at a time $k < 0$. The view in this case is denoted by λ. It is called an *empty* view, and is contained in every possible view $\mathsf{V}_{S'}(k')$.

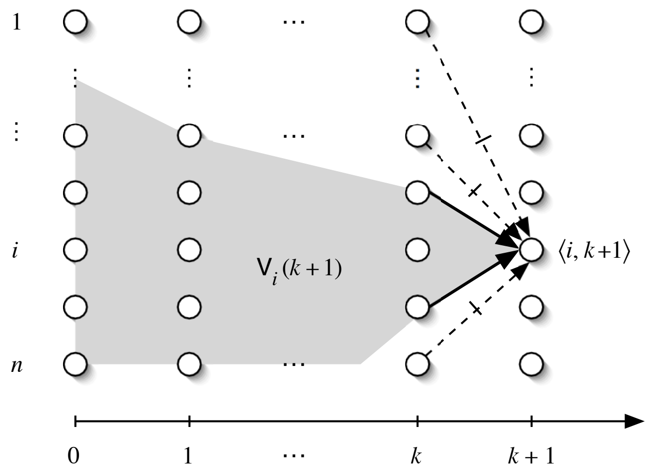


Figure 2.1: An execution graph and i's view $\mathsf{V}_i(k+1)$.

2.3 Full-Information Protocols

A *full-information protocol* (FIP) is one in which processes have *perfect recall* and observe all incoming messages and external inputs that they receive. Moreover, in every round, every process sends a message encoding all of its information to all other processes. It is not hard to show that in any such protocol a process is able to reconstruct $\mathsf{V}_i(k)$ from its information at time k. Without loss of generality, we will assume for the sake of concreteness that the local state of a process is maintained in the form of a view $\mathsf{V}_i(k)$, and the message sent by i in round $k+1$ is $\mathsf{V}_i(k)$.

Since in a FIP a message is sent on every channel in every round, the execution graph describes all aspects of a run: i.e., what inputs are received by the processes, which processes are faulty, and, for every message, whether or not it is delivered. Moreover, the contents of delivered messages can also be derived from the graph G. From now on we shall identify a *run r* of a FIP with its execution graph G^r. Since a run of FIP is determined by the inputs and failures, we sometimes denote such a run by $r = \text{FIP}(\zeta, \beta)$. It is a folk theorem, perhaps first proven formally in a fault-prone setting by Coan [9], that any deterministic protocol can be simulated by a FIP.

Using a FIP as we defined it above may be quite inefficient in terms of communication complexity. However, in practice a FIP may be implemented quite efficiently (see, for example, [5]). Further discussion of the communication complexity of the FIP in our context appears in Section 3.1.

2.4 Definition of the Continuous Consensus Problem

We now specify the continuous consensus problem formally. With respect to a set $\mathcal{E}$ of *monitored events*, we would like each process i to hold a copy of a shared list of events of $\mathcal{E}$. An event is defined based on the messages delivered in the run, and on initial states and external inputs that processes have received (and the times at which they were received). The precise definition of $\mathcal{E}$ will depend on the application. We define a *continuous consensus* (*CC*) service to be a distributed protocol that at all times $k \geq 0$ provides each process i with a *core* $M_i[k]$ of events of $\mathcal{E}$. In every run of this protocol the following properties are required to hold, for all nonfaulty processes i and j.

Accuracy: All events in $M_i[k]$ occurred in the run.

Consistency: $M_i[k] = M_j[k]$ at all times k.

Completeness: If an event $e \in \mathcal{E}$ is known to process j at any point, then $e \in M_i[k]$ must hold at some time k.

The consistency property guarantees that the information in the local lists is in fact shared among the nonfaulty processes at any given time. Since an event is in $M_i[k]$ for some nonfaulty process i only if it is also in $M_j[k]$ for all other nonfaulty processes j, it follows that a process i may know of the occurrence of a monitored event $e \in \mathcal{E}$ long before e is in $M_i[k]$. In many cases it is, of course, desirable to have the shared list in a continuous consensus application be as up-to-date as possible. A variant of this problem, which we call *uniform continuous consensus* (*UCC*), is defined similarly, but Accuracy and Consistency should hold for arbitrary processes and not just for nonfaulty ones. Completeness, however, is still restricted to events that are known to nonfaulty processes: If an event $e \in \mathcal{E}$ is known to a *nonfaulty* process j at any point, then $e \in M_i[k]$ must hold (for *all* processes i, of course) at some time k.

2.5 Systems and Knowledge

Generally speaking, we identify a *system* with a set R of runs. For a general protocol, a run r is an infinite sequence of states, and there is a well defined *local state* $r_i(m)$ for every process i and time m. For the FIP we identify runs with execution graphs, while in general every execution graph will determine a run of a protocol P (cf. [4, 5]). The systems that we study in this thesis are thus parameterized by a tuple (n,t,fm,I), where $n \geq 2$ is the number of processes, t is a bound on the number of faulty processes in a run (where $t \leq n-2$), $fm \in \{\text{crash},\text{omission}\}$ is a failure model, and I is a nonempty set of (external) input assignments. The exact identity and internal structure of I are application-dependent. A FIP system $R = R(n,t,fm,I)$ is defined to be the set of all runs of the FIP with n processes, at most t of which fail according to the failure model fm, and where initial states and external inputs conform to one of the input assignments in I. Our definitions imply that, in a precise sense, the inputs in a FIP system R are independent from the failures that occur (and hence carry no information about them): If there are runs $r, r' \in R$ where $r = \text{FIP}(\zeta, \beta)$ and $r' = \text{FIP}(\zeta', \beta')$, then R will also contain the run $r'' = (\zeta', \beta)$.[2]

Our analysis makes use of the knowledge that processes achieve at different times in various runs. As is standard in the literature, formulas will be considered true or false at a *point*, which is a pair (r, m) consisting of a run $r \in R$ and a time $m \in \mathbb{N}$. Moreover, since what is known at a point (r, m) may depend on what is true at other points, we define truth with respect to a system R. Let $\Phi = \{p, q, p', \ldots\}$ be a set of propositions. Intuitively, a proposition is a basic primitive fact. An example of a relevant proposition in the context of continuous consensus is "$\zeta(i, k) = e$", stating the arrival of an external input $e \in I$ at a given process i at time k. Given a system R, each proposition $p \in \Phi$ is identified with a set $[\![p]\!]$ of points of R. A proposition $p \in \Phi$ holds at (r, m), which we denote by $(R, r, m) \models p$, if $(r, m) \in [\![p]\!]$. For simplicity, we identify the set of monitored events $\mathcal{E}$ with a subset $\Phi_{\mathcal{E}} \subseteq \Phi$, and restrict monitored events to depend only on the external inputs in the current run. Thus, if $q \in \Phi_{\mathcal{E}}$ then, for all input assignments ζ and runs $r = \text{FIP}(\zeta, \beta)$ and $r' = \text{FIP}(\zeta, \beta')$, and times m, we will have that $(R, r, m) \models q$ iff $(R, r', m) \models q$. The core maintained by a continuous consensus algorithm consists of a set $X \subseteq \Phi_{\mathcal{E}}$ at any given time. We note that in our context the truth value of every $q \in \Phi_{\mathcal{E}}$ is independent of the time, m. Thus, in a given system, R, the truth value of q depends only on the run, r. We say that an event q has *occurred* in a run r if $(R, r, m) \models q$ for all $m \geq 0$. The core constructed by our proposed protocols will consist of the monitored events that are determined by a particular view computed at any given point. In order to describe such cores more formally, we denote

$$\mathcal{E}(\mathsf{V}) \triangleq \{q \in \Phi_{\mathcal{E}} : (R, r, m) \models q \text{ for all points } (r, m) \text{ whose execution graph contains } \mathsf{V}\}.$$

[2]Recall that the input assignments in I establishes the initial states of processes as well as the external inputs they receive. Often, initial states may be independent of external inputs, and the inputs at one process may be independent of those at another. But our definitions do not require such independence. There could be strong correlation among inputs in I. Our definitions only imply that inputs carry no information about failures and vice-versa.

Notice that $\mathcal{E}$ is monotone: If V is contained in V', then $\mathcal{E}(\mathsf{V}) \subseteq \mathcal{E}(\mathsf{V}')$.

We construct a logical language $\mathcal{L}$ by closing Φ under Boolean connectives $\wedge$ and $\neg$, and under modal knowledge operators K_i, D_S, E_S, C and C_N where $i \in \mathbb{P}$ and $S \subseteq \mathbb{P}$ is a set of processes. Here K_i stands for process i's knowledge, D_S corresponds to the *distributed knowledge* that is implicit in the set of processes S, E_S refers to facts known to *every* process in S, C stands for common knowledge, and C_N stands for common knowledge among the nonfaulty processes. The semantics of the Boolean operators is standard; we now review the definitions for K_i and D_S. Common knowledge and "everyone knows" are defined in the next subsection. The formal definitions (cf. [5]) of satisfaction for knowledge and distributed knowledge formulas are briefly stated as follows:

$$(R,r,m) \models K_i\varphi \text{ iff } (R,r',m') \models \varphi \text{ for all} (r',m') \text{ such that } r' \in R \text{ and } r_i(m) = r'_i(m'). \tag{2.1}$$

$$(R,r,m) \models D_S\varphi \text{ iff } (R,r',m') \models \varphi \text{ for all } (r',m') \text{ such that } r' \in R \text{ and } r_j(m) = r'_j(m') \text{ for all } j \in S. \tag{2.2}$$

A process knows φ by this definition if its local state (which captures the information it has access to) implies that φ holds. Distributed knowledge is defined similarly, but is based on the combined information available to the members of a set S of processes. In a full-information protocol, the distributed knowledge of S is equivalent to the knowledge of a process whose local state at a point (r,m) of R is the view $\mathsf{V}^r_S(m)$.

Knowledge in the FIP has a number of useful properties. For example, suppose that process i receives messages in round $k+1$ from the processes in the set S. Then, by construction, $\mathsf{V}_S(k)$ is contained in i's view $\mathsf{V}_i(k+1)$ at the end of the round. As a result, all facts about the past that are distributed knowledge of S at time k are known by i at time $k+1$. This observation plays a role in the solution to the continuous consensus problem described in the next section.

2.6 Common Knowledge

We define $E_S\varphi$ or "*everyone* in S *knows* φ" as follows:

$$E_S\varphi \triangleq \bigwedge_{i\in S} K_i(\varphi) \tag{2.3}$$

The formula $C_S\varphi$ is true if everyone in S knows φ, everyone in S knows that everyone in S knows φ, etc. Define $E_S^1\varphi \triangleq E_S\varphi$, and $E_S^m\varphi \triangleq E_S E_S^{m-1}\varphi$. Thus *common knowledge of* φ *among the processes in S*, denoted $C_S\varphi$, is defined as the infinite conjunction of E_S^m:

$$C_S\varphi \triangleq \varphi \wedge E_S\varphi \wedge E_S E_S\varphi \wedge \cdots \wedge E_S^m\varphi \wedge \cdots \tag{2.4}$$

Specifically, we have an interest in two particular cases: $S = N$, or $S = \mathbb{P}$. We denote common knowledge among the nonfaulty processes by C_N. $C_{\mathbb{P}}$ stands for common knowledge among all the processes in our system, and is denoted C for short.

We present an equivalent semantic definition of satisfaction for C_N, which will be more useful in the context of our analysis. Rather than defining common knowledge as an infinite conjunction of "everyone knows", we define it in term of N-reachability.[3] We say that two points (r,m) and (r',m) are *N-neighbors*, and write $(r',m) \sim_N (r,m)$, if there is some process j that is nonfaulty in both r and r' for which $r'_j(m) = r_j(m)$. In this case we say that the points (r,m) and (r',m) are *indistinguishable* by j. The "$\sim_N$" relation is also called the *similarity* relation. The point (r',m) is *N-reachable from* (r,m) in R, if there is a finite sequence of points $(r,m) = (r^0,m),(r^1,m),\ldots,(r^k,m) = (r',m)$ such that $(r^\ell,m) \sim_N (r^{\ell+1},m)$ holds for every $0 \le \ell < k$. Thus, N-reachability is the transitive closure of the $\sim_N$ relation. Moreover, it is an equivalence relation that defines a partition over the points of a system R. Common knowledge among the nonfaulty processes is then formalized by:

[3]The fact that in our systems a process can always distinguish between points (r,m) and (r',m') with $m \neq m'$ simplifies the definitions here slightly.

$$(R,r,m) \models C_N\varphi \quad \text{iff} \quad (R,r',m) \quad \models \varphi \text{ for all points } (r',m) \text{ that are } N\text{-reachable from } (r,m) \text{ in } R \tag{2.5}$$

A formal proof for the equivalence of the two definitions of common knowledge, in Equations 2.4 and 2.5 may be found in [5].

In other words, a fact φ is common knowledge among the processes in N at a point (r,m) if φ is valid in all the points that are N-reachable from (r,m). Very similarly, we can define common knowledge among *all* processes, C, by replacing the N in the definition of C_N by $\mathbb{P}$, the set of all processes in the system. More precisely, we say that the point (r',m) is *reachable* from (r,m) in R if there is a finite sequence of points of R $(r,m) = (r^0,m),(r^1,m),\ldots,(r^k,m) = (r',m)$ such that for every $0 \le \ell < k$ there is some $j = j_\ell$ for which $r^\ell_j(m) = r^{\ell+1}_j(m)$. Then

$$(R,r,m) \models C\varphi \quad \text{iff} \quad (R,r',m) \quad \models \varphi \text{ for all points } (r',m) \text{ that are reachable from } (r,m) \text{ in } R \tag{2.6}$$

Chapter 3

Solving Continuous Consensus

3.1 The CC Problem

The continuous consensus problem was formally defined in Section 2.4. In this section we present our solution to this problem, which is the CONCON algorithm. We start by presenting a simple protocol which solves the CC problem, after which we present CONCON and prove its correctness. In the following subsection we provide some background which enables us to present the SIMPLE algorithm immediately afterwards.

Clean Rounds

As mentioned in the introduction, the concept of *clean rounds* played a critical role in the analysis of Simultaneous Byzantine Agreement (SBA) by Dwork and Moses [3] in the crash model. In this model, a round of communication is clean if no new failure is discovered in the round. Following a clean round, all processes can have the same information about the past. Once it is common knowledge that a round was clean, the information available to nonfaulty processes before this round becomes common knowledge. We shall now present a formal definition of clean rounds.

We define the set of *Active* processes in (r,k), denoted $A(r,k)$, as the set of processes that did not fail in the first k rounds of r.[1] We say that the failure of process p is *discovered* in round k of r if k is the first time at which p's faulty behavior is distributed knowledge, i.e., k is the first time at which $(r,k) \models D(\text{“}p\ is\ faulty\text{”})$ holds. In this context $D(\cdot)$ refers to distributed knowledge among the active processes, i.e., $D_{A(r,k)}(\cdot)$.

Formally, a *clean round* is a round in which no process failure is discovered by the active processes. A round which is not *clean* is referred to as *dirty*. Notice that it is possible that the failure of a process p will be discovered in round k (which is thus a dirty round), whereas some of the processes may learn of p's failure in round $k+1$. Even so, if no other failures are discovered in round $k+1$, it is a clean round, since p's failure was discovered in round k.

An important property of a clean round, presented in [3], is the following: If round k of r is clean, then every fact of which there is distributed knowledge in $k-1$, becomes known to everyone at the end of round k. It is an inherent property of the crash failure model, that in a round in which process p fails, it may successfully send information to any subset of the other processes. As a result, information sent by p in round k may be known to some of the processes in k, and not known to others. On the other hand, the importance of clean rounds lies in the fact that all active processes successfully send their messages, and thus at the end of this round they have consistent views of the system. This property is phrased in the following theorem.

Theorem 3.1 (Dwork and Moses) *Let $(r,k-1) \models D\varphi$. If round k of r is clean, then $(r,k) \models E\,\varphi$.*

Once again, by $E\,\varphi$ we mean everyone among the active processes knows φ.

We say that a fact φ is a fact *about the initial configuration* of the run r if its truth value is uniquely determined by the set of initial states Σ in r. Similarly, we say that a fact φ is about the first m rounds of r if its truth depends on the first m rounds of r. An

[1]The set of nonfaulty processes in the run is a subset of the active processes at time k. While the term *nonfaulty* refers to the entire run, the term *active* refers to a specific time, and thus a process j who is active at time k may crash at some later time, and is thus considered faulty in the run.

important connection between clean rounds and common knowledge is presented in a theorem which was shown in [3] and is quoted below:

Theorem 3.2 (Dwork and Moses) *Let* clean *be the fact "a clean round has occurred", and let* φ *be a fact about the initial configuration. If* $(r,k) \models C\,(\text{clean})$ *then*

$$(r,k) \models D\varphi \quad iff \quad (r,k) \models C\varphi$$

The theorem implies that once it becomes common knowledge that a clean round has occurred between times 0 and k, all information about the initial configuration becomes common knowledge.

A direct corollary of Theorem 3.2 generalizes the result, and shows that if the appearance of a clean round in the time interval $[m,k]$ is common knowledge at time k, then all information about the first m rounds becomes common knowledge.

Corollary 3.3 Let φ be a fact about the first m rounds. Let $clean(m,k)$ be the fact "a clean round occurred between times m and k". If $(r,k) \models C\,(clean(m,k))$ then

$$(r,k) \models D\varphi \quad iff \quad (r,k) \models C\varphi$$

Theorem 3.2 is a key tool in the characterization of common knowledge in [3]. However, while the definition of clean rounds, as well as the theorem, are valid in the crash model, the (sending) omission failure model is significantly more complex, since information about failures can be kept by faulty processes for a long while without reaching nonfaulty processes. Such information can then be delivered to nonfaulty processes only much later. Thus, in the omission model the information about failures evolves in a much more erratic fashion. Hence in the omission model clean rounds no longer play the same role as they do in the crash failure model. Previous to this work, no direct analog to the notion of a clean round was found in the omissions model. The notion of clean rounds in the omission model will be discussed in later sections.

A Simple Protocol for Continuous Consensus

Before proceeding to present the CONCON protocol, we shall first present a very simple protocol for Continuous Consensus. In the scope of this subsection, we assume the crash failure model.

```
SIMPLE(i)

    for every round $k \geq 0$   do
1       send local state and receive messages according to FIP
2       $m \leftarrow k - (t+2)$
3       $M_i[k] \leftarrow \begin{cases} \mathcal{E}(\lambda) & \text{if } k < t+2 \\ \mathcal{E}(\mathsf{V}_{A_i(r,m+1)}(m)) & \text{otherwise.} \end{cases}$
```

Figure 3.1: The SIMPLE CC protocol for process *i*.

In the protocol, every process *i* computes the core in time *k* as the view of the *active* processes $t+2$ rounds before time *k*. On line 1 every process sends and receives messages according to a full information protocol. On line 3, *i* computes the core by assigning the view of the active processes $t+2$ rounds beforehand. We denote by $A_i(r,m+1)$ the set of *active* processes at time $m+1$ according to *i*'s view **at time** $k = m+t+2$, i.e., the processes not known to have dropped any messages up to round $m+1$:

$$A_i(r,m+1) \triangleq \{\ j\ :(r,m+t+2) \models \neg K_i(j \text{ crashed up to round } \ m+1)\} \qquad (3.1)$$

Note that there are $t+1$ rounds between $m+1$ and *k*, during which there must occur a clean round. Intuitively, by Theorem 3.1 we have that all facts that are distributed knowledge to the nonfaulty processes at time $m+1$ become known to all processes by time *k*, and in particular every nonfaulty *i* can compute $A_i(r,m+1)$ and $\mathsf{V}_{A_i(r,m+1)}(m)$ on line 3.

Lemma 3.4 The SIMPLE protocol solves the Continuous Consensus problem.

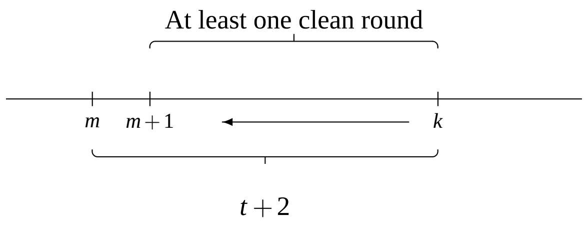


Figure 3.2: Illustration of the SIMPLE CC Protocol.

Proof: In order to prove that SIMPLE is a CC protocol, we have to prove that the three properties of CC hold.

Since $M_i[k]$ is defined as $\mathcal{E}(\mathsf{V})$ for some view V in the run, all events in $M_i[k]$ have occurred, and thus the Accuracy property holds. For completeness, assume that an event $e \in \mathcal{E}$ is known to a nonfaulty process j at time m, and is thus included in j's view at time m. Since j is nonfaulty, in particular it is active at time $m+1$, and thus from line 3 in *Simple*, the event e will appear in the core of every nonfaulty process i no later than time $m+t+2$; completeness follows. Finally, for consistency, we have to prove that if i and j are nonfaulty, then for all k we have $M_i[k] = M_j[k]$. Since at time k, both i and j produce the same m in line 2, all we have to prove is that $A_i(r,m+1) = A_j(r,m+1)$, and that $\mathsf{V}_{A(r,m+1)}(m)$ is available to i and j at time k. Since there is a bound of t failures in the system, during a time interval of $t+1$ rounds, there *must* be a clean round. From Theorem 3.1, we deduce that any fact which is distributed knowledge among the active processes at time $m+1$, is known to every correct process at time k. In particular, there is distributed knowledge at $m+1$ about the identity of the processes in $A_i(r,m+1)$, and thus at time k every correct process will have a copy of $A_i(r,m+1)$, and specifically $A_i(r,m+1) = A_j(r,m+1) = A(r,m+1)$. Moreover, since there is distributed knowledge about $\mathsf{V}_{A(r,m+1)}(m)$ at time $m+1$, it is necessarily available to every correct process by k, and in particular to i and j. It follows that $\mathsf{V}_{A_i(r,m+1)}(m) = \mathsf{V}_{A_j(r,m+1)}(m)$, and thus

$M_i[k] = M_j[k]$. The consistency property follows, and we are done.

■

Notice that process i's computation of $\mathcal{E}(\mathsf{V}_{A_i(r,m+1)}(m))$ on line 3 of the protocol uses the set of active processes, $A_i(r,m+1)$, at $m+1 = k-(t+1)$, while the joint view $\mathsf{V}_{A_i(r,m+1)}(m)$ of this set refers to round $m = k-(t+2)$. This follows from the fact that if an event e occurs at a process z at time m, in order for e to appear in the core, the protocol requires that z is active for at least one round after m, so that z will be able to notify the correct processes of e's occurrence. More formally, the contents of $\mathsf{V}_{A(r,m+1)}(m)$ is distributed knowledge among the correct processes at time $m+1$, since all processes in $A(r,m+1)$ are guaranteed to have successfully sent their messages to the correct processes in round $m+1$. On the other hand, $\mathsf{V}_{A(r,m+1)}(m+1)$ is not necessarily distributed knowledge among the correct processes at $m+1$, since if a faulty process $z \in A(r,m+1)$ fails to send all of its messages in round $m+2$, then any event e that occurs at z at time $m+1$ (and thus $e \in \mathsf{V}_{A(r,m+1)}(m+1)$) is not distributed knowledge among the correct processes at $m+1$, and in fact, since we are dealing exclusively with crash failures in this subsection, no correct process will ever learn of e.

Good and Bad Processes

As mentioned in the previous subsection, the analysis in [3] used a set of *active* processes, whose view of the system defined which rounds were potentially clean. The definition of active processes is useful in the crash model. However, when considering the omission model, both faulty and nonfaulty processes can actively send messages, and an appropriate analogue to the set of active processes is more difficult to define. While in the crash model the failure of a process becomes known to all the other processes at most one round after its occurrence, in the omission model a process i may omit a message to a faulty process j without any nonfaulty process ever noticing i's faulty behavior. In this example, one might wonder whether i should be considered *active* or not. Moreover, in the omission model, defining the set of processes *known* to have failed may need a bit of fine tuning as well, compared to our characterization of this set in the crash model.

The crux of our protocol, ConCon, which we present for the omission model, depends on finding the appropriate replacement for the role played in the crash model for the number of failures known to i. This is suitably generalized by the following two definitions of sets of *good* and *bad* processes for i with respect to time k. Process i determines the identity of these sets one round later, at time $k+1$. The first set, which we denote $G_i(k)$, consists of the processes that appear to i to have been nonfaulty at time k (and so, in particular, behaved correctly in round $k+1$):

$$G_i^r(k) \quad \triangleq \quad \{\ j\ :(R,r,k+1) \models \neg K_i(j \text{ is faulty})\ \} \tag{3.2}$$

As usual, we drop the superscript r from terms when it is clear from context. Intuitively, $G_i(k)$ is the set of processes who have not presented a faulty behavior up to round k, and furthermore, have managed to pass their messages in round $k+1$, allowing them to share the information they know at time k with the other processes. Notice that $G_i(k)$ is defined in terms of i's knowledge at the end of the following round $k+1$. In the crash failure model, $G_i(k)$ is the set of processes that i receives messages from in round $k+1$, while in the omissions model it is a possibly strict subset of these processes, since i can exclude j from $G_i(k)$ based on a report that j failed to send a message to a different process i'. The G stands for *good*. We associate $G_i(k)$ with time k rather than $k+1$ because the view of the members of $G_i(k)$ *at time* k turns out to be especially important. It serves a central role in determining the contents of the shared core. In addition, this view facilitates the computation of the core, by allowing the definition a set of *bad* processes associated with time k, which we denote by $B_i(k)$. This set consists of the processes that are distributedly known at time k to the members of $G_i(k)$ to be faulty:

$$B_i^r(k) \quad \triangleq \quad \{\ j\ :\ (R,r,k) \models D_{G_i(k)}(j \text{ is faulty})\ \} \tag{3.3}$$

Intuitively, B is the set of processes which are known by processes in G to have failed.

Thus, every process in B is necessarily faulty (while processes in G are not necessarily nonfaulty). Recall that i receives messages from all members of $G_i(k)$ in round $k+1$. Hence, every member of $B_i(k)$ is known by i at time $k+1$ to be faulty. It follows that $B_i(k) \cap G_i(k) = \emptyset$ for all $i \in \mathbb{P}$. Notice, however, that while $B_i(k)$ and $G_i(k)$ are disjoint, they are not necessarily complements. There are a number of different scenarios that may cause these sets not to be complements. For example, consider a process j that behaves correctly for the first k rounds and fails to send a message to i in round $k+1$. Process j is then excluded from $G_i(k)$, since i knows that j is faulty. Moreover, $j \notin B_i(k)$ because no process could have observed faulty behavior of j in the first k rounds; hence $j \notin G_i(k) \cup B_i(k)$ and the sets are not complements.

There is a close connection between the identity of $B_i(k)$ and the sets $G_j(k-1)$. Observe that, by definition of G_j, a process j knows, at time m that another process j' is faulty exactly if $j' \notin G_j(m-1)$. In the crash and omission models, the set of processes distributedly known to be faulty by a set S of processes is simply the union of those known to be faulty by the members of S. It thus follows that

$$B_i(k) = \bigcup_{j \in G_i(k)} (\mathbb{P} \setminus G_j(k-1)). \tag{3.4}$$

We denote by $b_i(k)$ the cardinality of $B_i(k)$. Since $B_i(k)$ consists of faulty processes, necessarily $b_i(k) \le t$. Figure 3.3 illustrates the sets $B_i(k)$ and $G_i(k)$ in the FIP given a particular execution graph.

The Core

By definition, a Continuous Consensus task maintains at any given time a *core* of shared information. As defined in Chapter 2, the core $M_i[k]$ is a subset of $\mathcal{E}(\mathsf{V})$ w.r.t. a view V. In our solution to the CC problem, which is described in the following sections, the core is represented by a time $c < k$, and a set F, such that $M_i[k]$ is uniquely determined by the joint view of the processes in F at time c, i.e., $M_i[k] = \mathcal{E}(\mathsf{V}_F(c))$. We call c

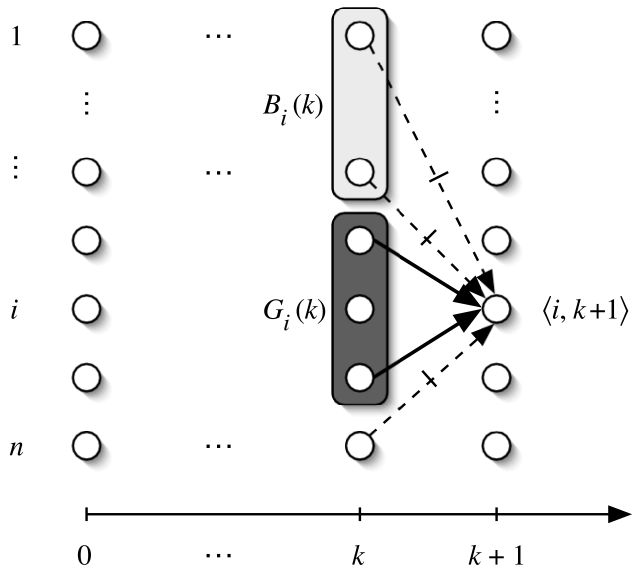


Figure 3.3: Sets $G_i(k)$ & $B_i(k)$ are based on $\mathsf{V}_i(k+1)$.

the *critical time* of k, denoted $c = crit_i(k)$, and F the *critical set* of k. In the context of ConCon, we define the critical set, F, as the set of good processes at time c, i.e. $F = G_i(c)$. The challenge in ConCon is to compute at each time, k, the appropriate critical time, $crit_i(k)$. In the following subsections we shall prove that by choosing the core as $M_i[k] = \mathcal{E}(\mathsf{V}_{G_i(c)}(c))$, we obtain an optimum solution for CC.

We define i's *destination* at time c, denoted $dest_i(c)$, as the first time, k, at which $\mathcal{E}(\mathsf{V}_{G_i(c)}(c)) \subseteq M_i[k]$ holds. Intuitively, $dest_i(c)$ is the first time at which facts which are distributed knowledge at time c join the core. In particular, if $c = crit_i(k)$, then $dest_i(c) = crit_i^{-1}(c) = k$, however, it is also possible that for some $c' < crit_i(k)$ we will also have $dest_i(c') = k$. Formally, we say that $dest_i(m) = k$ exactly if $crit_i(k) \geq m$, and $crit_i(k-1) < m$.

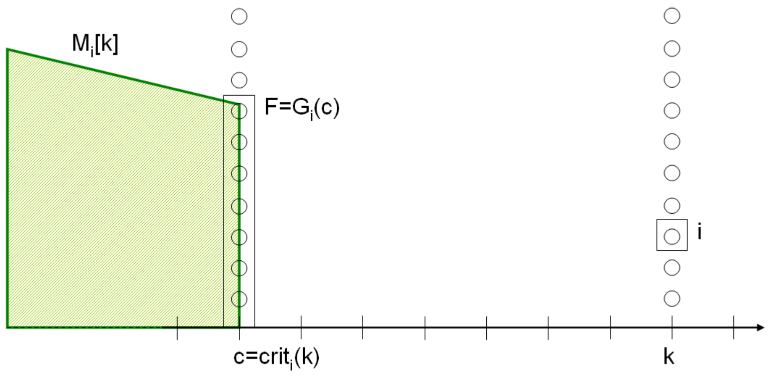


Figure 3.4: The core computed at time k.

The Horizon

The analysis of simultaneous agreement in the crash failure model in [3] showed that knowing that a clean round occurred within a certain time interval enables processes to obtain a consistent view of information about earlier times. Thus a key goal in CONCON will be to compute, in every round, a margin of time in which a clean round is guaranteed to have occurred.[2]

The analysis in the following subsections will show that if at time $k+1$, process i knows that at least f processes failed before time k, then it knows that at least one clean round must happen between time k and time $h_i(k) = k+t+1-f$. We think of $h_i(k)$ as i's *horizon* for time k, denoted $\mathsf{horizon}_i(k)$. It turns out that if i is a good process, then $\mathsf{V}_i(k)$ will be common knowledge by time $\mathsf{horizon}_i(k)$. Intuitively, the properties of the crash-failure model show that there will be a latest (*critical*) time c with $k \leq c < \mathsf{horizon}_i(k)$ such that round $c+1$ is clean, and after which all nonfaulty processes can predict the same horizon (i.e., $\mathsf{horizon}_i(c) = \mathsf{horizon}_j(c)$ for all processes i and j). We shall show that the same holds in the omission model as well. Moreover, we shall show that at time $\ell = \mathsf{horizon}_i(c)$, every nonfaulty process knows that c is the critical time (for ℓ).

[2]At this point this is an intuitive description of our motivation in CONCON. A formal definition of clean rounds in the omission model will be presented in Section 3.4.

This makes it common knowledge that round $c+1$ was clean, allowing for an efficient solution to simultaneous agreement. (Continuous consensus is a strict generalization of simultaneous agreement.) Finally, it will then follow that the core information at time ℓ is the view of the good processes in the critical time, c.

Formally, we define i's *horizon* for time m as:

$$\mathsf{horizon}_i(m) = m + t + 1 - b_i(m) \tag{3.5}$$

The value of $\mathsf{horizon}_i(m)$ is defined for all $m \geq 0$. Recall that $b_i(m)$ is i's estimation for the number of "bad" processes, whose failure has been discovered up to round m.

Since no failures are known initially, we have that $b_i(0) = 0$. Thus, by definition of $\mathsf{horizon}_i(m)$, we have that $\mathsf{horizon}_i(0) = t + 1$. The facts that the processes communicate according to the full-information protocol, and that the number of failures is bounded from above by t, imply that $b_i(m) \leq b_i(m+1) \leq t$ for all m. The following observation immediately follows.

Observation 3.5 For all $m \geq 0$:

(i) $\mathsf{horizon}_i(m) \geq m + 1$, and

(ii) $\mathsf{horizon}_i(m+1) \leq \mathsf{horizon}_i(m) + 1$.

Notice from (ii) that the horizon cannot move forward by more than one at every round. It can move backwards more rapidly, however. If $b_i(m+1) = b_i(m) + d + 1$ then $\mathsf{horizon}_i(m+1) = \mathsf{horizon}_i(m) - d$. This happens when $d+1$ new failures are discovered in round m by processes that are still trusted by i at time $m+1$.

CONCON

The CONCON protocol, shown in Figure 3.5, is run by each process i individually. CONCON is used in both the crash and the omission failure model. Observe, however, that the value of $\mathsf{horizon}_i(k-1)$ on line 2 of CONCON is a function of $b_i(k-1)$ which, in turn, is based on the knowledge of process i as well as on that of other processes. This knowledge is evaluated with respect to different systems R for each of the failure models.

The analysis presented in [4] is, roughly speaking, "backward looking": at any point in the execution, a process computes what it knows to be common knowledge. On the other hand, the analysis in [3] takes a "forward looking" approach: every process computes at what time facts about the initial configuration of the run become common knowledge. Our approach in ConCon is hybrid — both forward and backward looking. At the end of every round k of the protocol, each process i performs two tasks: One is to update a current estimate (upper bound) for when events in the view $\mathsf{V}_{G_i(k-1)}(k-1)$ of $G_i(k-1)$ will be part of the shared core view (*forward* looking). The other is to determine the shared core at time k, at the end of the current round. For this purpose, the value in $crit_i(k) \triangleq Latest_i[k]$ is considered the *critical time* for the core at time k (*backward* looking). In the protocol text, we use the term $\mathsf{horizon}_i$, which is defined in Eq.3.5. Notice that $\mathsf{horizon}_i(m)$ is easily computable based on $\mathsf{V}_i(m+1)$.

```
ConCon(i)

0  $Latest_i[\ell] \leftarrow -1$ for all $\ell \geq 1$
   for every round $k \geq 1$   do
1      send local state and receive messages according to FIP
2      compute $G_i(k-1)$, $B_i(k-1)$ and $\mathsf{horizon}_i(k-1)$
3      $Latest_i[\mathsf{horizon}_i(k-1)] \leftarrow k-1$
4      $c \leftarrow Latest_i[k]$     ▷ $c$ is the critical time for $k$, denoted $crit_i(k)$
5      $M_i[k] \leftarrow \begin{cases} \mathcal{E}(\lambda) & \text{if } c = -1 \\ \mathcal{E}(\mathsf{V}_{G_i(c)}(c)) & \text{otherwise.} \end{cases}$
   endfor
```

Figure 3.5: The ConCon protocol for process i.

In the protocol, each process performs the same set of actions in every round. Round $k \geq 1$ starts at time $k-1$ and ends at time k. The first part of each round's computation consists of communicating according to the full-information protocol on line 1. In the next part, on line 2, process i computes an upper bound on the time at which a view of time k will become included in the core. Finally, on lines 4 and 5 it records in $M_i[k]$ the view which is the core information at the current time k. We denote by $crit_i(k)$ the value of c that i sets in round k on line 4.

To compute the value of the index $\mathsf{horizon}_i(k-1)$ used on line 2 of ConCon, process i needs to know the value of $b_i(k-1)$ at the end of round k. Since a process j is in $G_i(k-1)$ if i does not know that j is faulty at time k, it follows that i receives round k messages from all members in $G_i(k-1)$. Thus, i has a copy of $\mathsf{V}_{G_i(k-1)}$ at time k. In particular, i can compute $B_i(k-1)$ and $b_i(k-1)$, as desired. Observe that steps 2 and 3 of the ConCon protocol depend only on the failures that occur in the run. As a result, the nodes in $\mathsf{V}_{G_i(c)}(c)$ are independent of the (external) input assignment of the run. Finally, observe that, for all $m < k$, process i has a copy of $\mathsf{V}_{G_i(m)}(m)$. Thus, process i is able to compute $\mathcal{E}(\mathsf{V}_{G_i(c)}(c), k)$ on line 5 of ConCon.

We now state two useful properties of ConCon. The first says that the horizon is an upper bound on the time at which current round information is contained in the shared core. Indeed, given Observation 3.5(i) above this will imply that every round's information will become common knowledge within a fixed bound of roughly $t-f$ rounds, where f is the number of failures discovered. The second says that once the core is not empty, the critical time increases by at least one in every time step. Moreover, every round is assigned a critical time.

Proposition 3.6 For all nonfaulty processes i and times m and ℓ:

(a) if $\mathsf{horizon}_i(m) \leq \ell$ then $crit_i(\ell) \geq m$, and

(b) if $crit_i(\ell) \neq -1$ then $crit_i(\ell) < crit_i(\ell+1)$.

We now present the following technical lemma that is based on Observation 3.5, and will assist us in the proof of Proposition 3.6.

Lemma 3.7 If $k < k'$ and $\mathsf{horizon}_i(k) \leq \ell < \mathsf{horizon}_i(k')$, then there exists a time $\hat{k}$ with $k \leq \hat{k} < k'$ such that (a) $\mathsf{horizon}_i(\hat{k}) = \ell$ and (b) $\mathsf{horizon}_i(\hat{k}+1) = \ell + 1$.

Proof: Assume that $k < k'$ and $\mathsf{horizon}_i(k) \leq \ell < \mathsf{horizon}_i(k')$. By Observation 3.5(ii), the function $\mathsf{horizon}_i$ can advance only in steps of one. Hence, $\mathsf{horizon}_i(k'') = \ell$ for some intermediate time $k \leq k'' < k'$, establishing part (a) of the claim. Let $\hat{k} = \max\{k'' : k'' < k' \text{ and } \mathsf{horizon}_i(k'') = \ell\}$. We claim that $\mathsf{horizon}_i(\hat{k}+1) > \mathsf{horizon}_i(\hat{k})$. By definition of $\hat{k}$, we have that $\mathsf{horizon}_i(\hat{k}+1) \neq \ell$. Recall that $\mathsf{horizon}_i(k') > \ell$ and $\hat{k} < k'$. Hence, if $\mathsf{horizon}_i(\hat{k}+1) < \ell$ then $\hat{k}+1 < k'$. We can now apply part (a) to $\hat{k}+1 < k'$ to obtain that $\mathsf{horizon}_i(h) = \ell$ for some h such that $\hat{k}+1 \leq h < \ell$. This contradicts the maximality of $\hat{k}$. It follows that $\mathsf{horizon}_i(\hat{k}+1) > \ell$. Finally, since $\mathsf{horizon}_i(\hat{k}+1) > \mathsf{horizon}_i(\hat{k})$, we have by Observation 3.5(ii) that $\mathsf{horizon}_i(\hat{k}+1) = \mathsf{horizon}_i(\hat{k}) + 1$, and we are done.

■

Proof of Proposition 3.6: For part (a), assume that $\mathsf{horizon}_i(m) \leq \ell$. Since $\mathsf{horizon}_i(\ell) > \ell$ we have by Lemma 3.7(a) that $\mathsf{horizon}_i(\hat{k}) = \ell$ for some $\hat{k}$ such that $m \leq \hat{k} < \ell$. In particular, $Latest_i[\ell] = \hat{k} \geq m$ after line 3 is executed in round $\hat{k}+1$. Since $\hat{k} < \ell$, we have that $\hat{k}+1 \leq \ell$. Moreover, since the value of $Latest_i[\ell]$ is nondecreasing in time, it follows that $Latest_i[\ell] \geq \hat{k}$ when line 4 is reached in round ℓ. It now follows by line 4 that $crit_i(\ell) \geq \hat{k} \geq m$, proving part (a). For part (b), assume that $crit_i(\ell) = m \neq -1$. Then, from the definition of $\mathsf{horizon}_i(m)$, we have that $\mathsf{horizon}_i(m) = \ell$. Since $\mathsf{horizon}_i(\ell) > \ell$, we have by Lemma 3.7(b) that there exists $\hat{k}$ such that $m \leq \hat{k} < \ell$ and $\mathsf{horizon}_i(\hat{k}+1) = \ell + 1$. Applying part (a) we obtain that $crit_i(\ell+1) \geq \hat{k}+1$. Since $m \leq \hat{k}$, we have that $\hat{k}+1 > m$, and thus $crit_i(\ell+1) > m = crit_i(\ell)$, which completes the proof.

■

So far we have looked at the properties of the protocol as executed by a single non-faulty process in isolation. The correctness of the algorithm depends on the relationship between executions of different processes in the same run. The following theorem shows the main correctness claim for ConCon, namely that all cores agree at all times.

Theorem 3.8 *The* ConCon *protocol solves the continuous consensus problem.*

Proof: Since $M_i[\ell]$ is a view of the run, all events in $M_i[\ell]$ have occurred, and thus the Accuracy property holds. Completeness requires every $q \in \Phi_{\mathcal{E}}$ that is known to a nonfaulty process j will eventually appear in $M_i[\ell]$. Suppose that $K_j q$ holds no later than time k. Notice that $j \in G_i(k)$ for every nonfaulty process i, since j is nonfaulty. By definition, $\mathsf{horizon}_i(k) \leq k+t+1$, and Proposition 3.6(a) implies that $crit_i(k+t+1) \geq k$. It follows that $\mathcal{E}(\mathsf{V}_j(k))$ will be contained in $M_i[k+t+1]$. In particular, we have that $q \in M_i[k+t+1]$, and we have Completeness.

Finally, for Consistency, we need to show that $M_i[\ell] = M_j[\ell]$ for all times $\ell \geq 0$ and nonfaulty processes i and j. The variable $crit_i(\ell)$ is assigned the value of $Latest_i[\ell]$ in round ℓ by line 4. Lines 0 and 3 guarantee that $Latest_i[m] \geq -1$ and $Latest_j[m] \geq -1$ holds for all indices m at all times. It follows that $crit_i(\ell) \geq -1$. We distinguish two cases. First suppose that $crit_i(\ell) = crit_j(\ell) = -1$. In this case we have by line 5 and the fact that $crit_i(\ell) = crit_j(\ell) = -1$ that $M_i[\ell] = M_j[\ell] = \lambda$ as desired. Second, suppose without loss of generality that $m = crit_i(\ell) \neq -1$. We claim that $G_j(m) \supseteq G_i(m)$ and $crit_j(\ell) \geq crit_i(\ell)$. If $G_j(m) \not\supseteq G_i(m)$, then j knows at time $m+1$ of some $z \in G_i$ that is faulty. Since j is nonfaulty, $j \in G_i(m+1)$, and hence $z \in B_i(m+1)$ so that $b_i(m+1) > b_i(m)$ and $\mathsf{horizon}_i(m+1) \leq \mathsf{horizon}_i(m) = \ell$. It follows from Proposition 3.6(a) that $crit_i(\ell) \geq m+1$, contradicting the assumption that $crit_i(\ell) = m$. Since $G_j(m) \supseteq G_i(m)$ it follows that $\mathsf{V}_{G_i(m)}(m)$ is contained in $\mathsf{V}_{G_j(m)}(m)$ and hence that $B_j(m) \supseteq B_i(m)$. This implies that $b_j(m) \geq b_i(m)$, and thus $\mathsf{horizon}_j(m) \leq \mathsf{horizon}_i(m)$. Again by Proposition 3.6(a) we can conclude that $crit_j(\ell) \geq m = crit_i(\ell)$, and the claim is established. Moreover, since $crit_i(\ell) > -1$, it follows that $crit_j(\ell) \neq -1$. Applying this argument to j instead of i, we obtain also that $G_j(m) \subseteq G_i(m)$ and $crit_j(\ell) \leq crit_i(\ell)$. We thus have that $crit_i(\ell) = crit_j(\ell)$ and that $G_i(m) = G_j(m)$. Finally, since $G_i(m) = G_j(m)$, we have by line 5 that $M_i[\ell] = \mathsf{V}_{G_i(m)}(m) = \mathsf{V}_{G_j(m)}(m) = M_j[\ell]$ and we are done.

■

More efficient implementations. The CONCON protocol sends messages according to FIP, so that messages get larger over time. In many cases it is possible to derive a more space- and communication-efficient implementation of CONCON. In order to simulate the CONCON protocol a process i must in particular have enough information about (knowledge regarding) failures to enable the computation of the $\mathsf{horizon}_i$ at all points. The value of $\mathsf{horizon}_i(k)$ depends on $b_i(k)$ which, in turn, can be computed once i knows the sets $G_j(k-1)$ for all $j \in G_i(k)$ (see Eq.3.4). This can be achieved if, in every round $k+1$ each process i sends $G_i(k-1)$ to all other processes. The set $G_i(k-1)$ consists of the processes that i does not know to be faulty at time k, which is when the round $k+1$ message is prepared. This set can be encoded as a string of n bits. Initially, process i knows of no failures. At every time $k > 0$, it can compute $G_i(k-1)$ by detecting a process as faulty exactly if it is either reported as faulty in one of the $G_j(k-2)$ it has received in round k, or if the process in question has failed to deliver a message to i. It can be checked that the sets $G_i(k-1)$ computed under this scheme are the same as they are using FIP. In order to carry out step 4 of CONCON, we must also guarantee that all information about monitored events from $\mathcal{E}$ be passed to everyone. In applications in which there are only a few interesting events (e.g., fire alarms) then representing the relevant data regarding them can be done succinctly. It follows that there is a protocol for continuous consensus that is equivalent to CONCON, but sends short messages and uses little space beyond that needed for the shared core $M_i[k]$ and the aspects of the view needed to determine the events of $\Phi_{\mathcal{E}}$.

3.2 Continuous Consensus and Common Knowledge

We have developed the CONCON protocol using intuitions obtained from the analysis of common knowledge in fault-prone systems. In fact, continuous consensus is closely related to the problem of computing common knowledge. We now formalize this connection, and use it in order to prove the optimality of ConCon.

The continuous consensus problem is specified in terms of the behavior of the nonfaulty processes, and does not require correct action from faulty ones. It was shown in [4] (see also [10, 5]) that the appropriate variant of common knowledge corresponding to such a situation is common knowledge among the *nonfaulty* processes, for which our language has the operator C_N.

We say that a formula φ is *valid in R*, and write $R \models \varphi$, if $(R,r,m) \models \varphi$ for all points (r,m) with $r \in R$. Recall that, by Theorem 3.8, $M_i[k] = M_j[k]$ holds for every pair of nonfaulty processes i and j. For every possible state $X \subseteq \mathcal{E}$ of the core, we define a proposition $\text{Core} = X$ that is true at a point (r,m) exactly if $M_i^r[m] = X$ for all nonfaulty processes i in r. We can now show a strong connection between common knowledge and continuous consensus:

Proposition 3.9 Let P be a protocol for continuous consensus and let R_{P} be the set of all runs of P with execution graphs in $R = R(n,t,fm,I)$. Then for all X we have

$$R_{\mathsf{P}} \models ((\text{Core} = X) \equiv C_N(\text{Core} = X)) \ .$$

Proof: We shall prove that $\text{Core} = X$ holds iff $C_N(\text{Core} = X))$. The 'if' direction is trivial, since every fact that is common knowledge is necessarily true. For the 'only if' direction assume that $M_i^r[m] = X$ for all nonfaulty processes i in r. Define the proposition p as "$\text{Core} = X$". It suffices to show that, for all $r,r' \in R$, if $(R,r,m) \models p$ and $(r,m) \sim_N (r',m)$ then $(R,r',m) \models p$, and the claim will follow by induction. Assume that $(R,r,m) \models p$ and $(r,m) \sim_N (r',m)$. Thus, $r_j(m) = r'_j(m)$ for some process j that is nonfaulty in both runs. Since $(R,r,m) \models p$ we have that $\text{Core} = X$ holds at (r,m); since j is nonfaulty it follows that, in particular, $M_j^r[m] = X$. Since $r_j(m) = r'_j(m)$, we have that $M_j^{r'}[m] = X$ as well. Finally, since P solves continuous consensus, we have by Consistency that $M_i^{r'}[m] = X$ for all nonfaulty i in r', and hence $(R,r',m) \models p$.

■

Proposition 3.9 implies that the contents of the core in any protocol P for continuous consensus (e.g., CONCON) are common knowledge among the nonfaulty processes at every point. Hence, an event can be entered into the local copies of the core only once its occurrence has become common knowledge. We shall now argue that the CONCON protocol places events in the cores $M_i[m]$ as early as possible. This will establish that CONCON is an optimum protocol for continuous consensus. More formally, we prove the following

Theorem 3.10 *If R is a* FIP *system, $r \in R$, i is nonfaulty in r, P is a correct protocol for continuous consensus, i's core at (r,m) under* CONCON *is $M_i^{\mathsf{C}}[m]$, and i's core under P is $M_i^{\mathsf{P}}[m]$, then $M_i^{\mathsf{P}}[m] \subseteq M_i^{\mathsf{C}}[m]$.*

In Theorem 3.10 we assume, without loss of generality, that P is a FIP. As Lemma A.2 in the appendix shows, the same argument holds for the general case, when P is an arbitrary protocol. Theorem 3.10 shows that CONCON is optimal in terms of recording events in the core as early as possible. We prove the theorem by showing that the core $M_i^{\mathsf{C}}[m]$ produced by CONCON is precisely the view of the run that is common knowledge at (r,m). Moses and Tuttle [4] completely characterized the connected components of the N-reachability relation systems for FIP in crash and omission models, thereby characterizing common knowledge as well. To set up the necessary background for the proof, we now briefly review their fixed-point construction and related characterization of common knowledge. The construction is performed individually by every process i based on its view $\mathsf{V}_i^r(m)$ at a given point (r,m). It defines a sequence of pairs (k_ℓ, S_ℓ) consisting of a time and set of processes, for $\ell \geq 0$. In the construction, F_ℓ denotes the set $\{j : (R,r,k_\ell) \models D_{S_\ell}(j \text{ is faulty})\}$ of processes known at time k_ℓ to be faulty by processes in S_ℓ. The sets S_ℓ and F_ℓ are analogous to, but in general distinct from, to the sets $G_i(k)$ and $B_i(k)$ in CONCON. The construction proceeds inductively as follows.

Base: Set $k_0 = m$ and $S_0 = \{i\}$.

Step: Set $k_{\ell+1} = m - (t+1-|F_\ell|)$ and $S_{\ell+1} = \mathbb{P} \setminus F_\ell$.

As Moses and Tuttle show, the F_ℓ's form a nonincreasing sequence of sets of processes. As a result, the S_ℓ's form a nondecreasing sequence of sets of processes, and the k_ℓ's form a descending sequence of indices. Since $|F_\ell| \leq t$, for some index h we must have that $F_h = F_{h-1}$. When this happens for the first time, the construction reaches a fixed-point because $S_{h+1} = S_h$ and $k_{h+1} = k_h$. We use $\hat{k} = \hat{k}(r,m)$ and $\hat{S} = \hat{S}(r,m)$ to denote the first values k_h and S_h at which a fixed-point is reached. The construction can reach two types of fixed-points. One in which $\hat{k} < 0$ (and $\hat{S} = \mathbb{P}$), and the other in which $\hat{k} \geq 0$. To accommodate the former case, we define $\mathsf{V}_{\mathbb{P}}(m') = \lambda$ for all $m' < 0$. (Recall that λ is used to denote the empty view.) We remark that at the fixed point, F_h is the complement of S_h. Since S_0 is a singleton and all members of every F_ℓ are faulty, the assumption that $t \leq n-2$ guarantees that $h \geq 1$ at the fixed point.

The final step of the construction is:

Output: The view $\mathsf{V}_{\hat{S}}(\hat{k})$. (We denote this view by $\hat{\mathsf{V}}_i[r,m]$.)

As shown in [4], process i's local state $\mathsf{V}_i(k_0)$ at time k_0 contains $\mathsf{V}_{S_\ell}(k_\ell)$ for all $k_\ell \geq 0$. As a result, process i can compute all of the stages of the construction at time $m = k_0$ based on its local state there. Let $r = \textsc{fip}(\zeta,\beta)$. When the construction is performed at (r,m), the sets S_ℓ and F_ℓ depend only on the β component (failures) in r. It follows that the final output $\hat{\mathsf{V}}_i[r,m]$ of the construction depends only on β and on the restriction of ζ (the inputs) to the nodes of $\hat{\mathsf{V}}[r,m]$.[3] Figure 3.6 illustrates an example computation of the fixed-point construction.

The fixed-point construction is shown to characterize N-reachability relation (and hence also the common knowledge) in the crash and omission models:

Proposition 3.11 (Moses and Tuttle) Let r and r' be runs of a FIP system R, and assume

[3]We shall denote by $\hat{\mathsf{V}}[r,m]$ the view $\hat{\mathsf{V}}_j[r,m]$ obtained by the nonfaulty processes j in r, since $\hat{\mathsf{V}}_j[r,m]$ is the same for all nonfaulty processes j, by Proposition 3.11.

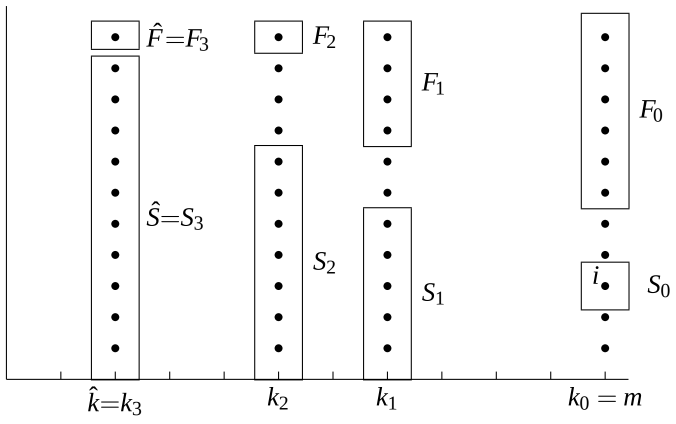


Figure 3.6: An instance of Moses and Tuttle's fixed-point construction by i at time m.

that i is a nonfaulty process in r and j a nonfaulty process in r'. Then (r',m) is N-reachable from (r,m) iff $\hat{\mathsf{V}}_j[r',m] = \hat{\mathsf{V}}_i[r,m]$.

In other words, Proposition 3.11 states that the fixed-point construction performed by a nonfaulty process i at (r,m) outputs the same view as the one it outputs for any nonfaulty process at any point in the N-connected component of (r,m). Proposition 3.11 thus implies that, in a precise sense, $\hat{\mathsf{V}}[r,m]$ summarizes and uniquely determines the set of facts that are common knowledge at any given point (r,m). As a result, we can show that only input events of $\mathcal{E}$ that appear in $\hat{\mathsf{V}}$ are common knowledge among the nonfaulty processes:

Corollary 3.12 Let $q \in \Phi_{\mathcal{E}}$, let P be a FIP, and assume that i is nonfaulty in r. Then $(R_{\mathsf{P}},r,m) \models C_N q$ *iff* $q \in \mathcal{E}(\hat{\mathsf{V}}_i[r,m])$.

Proof: Fix $r = \text{FIP}(\zeta,\beta) \in R$ and $m \geq 0$, let i be nonfaulty in r, let $q \in \Phi_{\mathcal{E}}$ and let $\mathsf{V} = \hat{\mathsf{V}}_i[r,m]$. To prove the 'if' direction, assume that $q \in \mathcal{E}(\mathsf{V})$. By Proposition 3.11, $\mathsf{V} = \hat{\mathsf{V}}_j(r',m)$ for every nonfaulty processor j at a point (r',m) that is N-reachable from (r,m). It follows that V is contained in $\mathsf{G}^{r'}$, and $q \in \mathcal{E}(\mathsf{V})$ implies that $(R,r',m) \models q$. Since this holds for all points N-reachable from (r,m), we have that $(R,r,m) \models C_N q$.

For the ‘only if’ direction, assume that $q \notin \mathcal{E}(\mathsf{V})$. By definition of $\mathcal{E}(\mathsf{V})$, it follows that $(R,r',m) \not\models q$ for some run $r' = \textsc{fip}(\zeta',\beta')$ whose execution graph contains V. In particular, ζ and ζ' agree on the inputs at the nodes of V. Consider the run $r'' = \textsc{fip}(\zeta',\beta)$ with the same failure pattern (β) as in r, and the same external inputs (ζ') as in r'. The assumption that inputs in $R = R(n,t,fm,I)$ are independent of failures and of each other ensures that $r'' \in R$. Since $q \in \Phi_{\mathcal{E}}$ it depends only on the external inputs. Since $(R,r',m) \not\models q$, and the fact that r'' shares ζ' with r' implies that $(R,r'',m) \not\models q$. Recall, that the nodes and edges of the execution graph in $\mathsf{V} = \hat{\mathsf{V}}_i[r,m]$ depend only on the failure pattern β. Moreover, since r and r'' share the same failure pattern β, process i is nonfaulty in r'' as it is in r. Since, in addition, ζ' agrees with ζ on the inputs assigned to nodes of V, it follows that $\hat{\mathsf{V}}[r'',m] = \mathsf{V} = \hat{\mathsf{V}}[r,m]$. Proposition 3.11 implies that (r'',m) is N-reachable from (r,m). Hence, from $(R,r'',m) \not\models q$ we obtain that $(R,r,m) \not\models C_N q$ and we are done. ■

Based on this characterization, we can prove our claim that ConCon is optimal:

Proof of Theorem 3.10: Fix a run r of $\textsc{fip}$. We will abuse notation slightly and denote the runs of both P and ConCon with execution graph G^r by the same name r, and $\hat{\mathsf{V}}[r,m]$ by $\hat{\mathsf{V}}$. By Proposition 3.9, the events in $M_i^{\mathsf{P}}[m]$ are common knowledge. By definition of the core, $M_i^{\mathsf{P}}[m] \subseteq \Phi_{\mathcal{E}}$. Hence, Corollary 3.12 implies that $M_i^{\mathsf{P}}[m] \subseteq \mathcal{E}(\hat{\mathsf{V}})$. We will show that $\hat{\mathsf{V}}$ is contained in $\mathsf{V}_{G(c)}(c)$ for $c = crit_i^r(m)$. Since $M_i^{\mathsf{C}}[m] = \mathcal{E}(\mathsf{V}_{G(c)}(c))$, this will imply that $M_i^{\mathsf{C}}[m] \supseteq \mathcal{E}(\hat{\mathsf{V}}) \supseteq M_i^{\mathsf{P}}[m]$, from which the claim follows. The case in which $\hat{\mathsf{V}} = \lambda$ is immediate, since λ is by definition contained in all views, including $\mathsf{V}_{G(c)}(c)$. It remains to consider the case in which $\hat{\mathsf{V}} \neq \lambda$. In this case, let $\hat{k} = k_h$ and $\hat{S} = S_h$ be the fixed-point values in the Moses and Tuttle construction performed in the run r. Since $\hat{\mathsf{V}} \neq \lambda$ we have that $\hat{k} \geq 0$. Moreover, recall that the construction ends with $h \geq 1$ since $t \leq n-2$. Since k_h is the first place at which a fixed-point is obtained, we obtain that $k_h < k_{h-1} \leq k_0$. Observe that the sets F_ℓ in the fixed-point construction contain only faulty processes. Since $i \in S_0$ and i is nonfaulty, it follows that $i \in S_\ell$

for every $\ell \leq h$. In particular, $i \in S_{h-1}$. By definition of S_h and F_{h-1}, we have that $S_h = \{j : (R,r,k_{h-1}) \models \neg D_{S_{h-1}}(j \text{ is faulty})\}$. Since $i \in S_{h-1}$, we have in particular that $(R,r,k_{h-1}) \models \neg K_i(j \text{ is faulty})$, for all $j \in S_h$. It follows that $S_h \subseteq G_i(k_{h-1}-1)$, where G_i is the set of *good* processes according to i in CONCON. From $k_h < k_{h-1}$ we have that $k_h \leq k_{h-1} - 1$. The perfect recall property of the FIP implies that the sets $G_i(k)$ are monotonically nonincreasing. Thus, $S_h \subseteq G_i(k_{h-1}-1) \subseteq G_i(k_h)$. It follows that $B_i(k_h) \supseteq F_h$, and hence also that $b_i(k_h) \geq |F_h|$. Since $k_h = \hat{k}$ is the fixed point, we have that $k_h = m - (t+1-|F_h|)$ and hence $m = k_h + t + 1 - |F_h|$. By definition of CONCON, $\mathsf{horizon}_i(k_h) = k_h + t + 1 - b_i(k_h)$. Since $b_i(k_h) \geq |F_h|$, we obtain that $\mathsf{horizon}_i(k_h) \leq m$. By Proposition 3.6(a) we have that $crit_i^r(m) \geq \hat{k}$. It follows that $\mathsf{V}_{G(c)}(c) \supseteq \hat{\mathsf{V}}$ and we are done.

■

Extending the core. So far, the monitored events that we allowed (which we identified with the propositions in $\Phi_{\mathcal{E}}$) have been facts about the external inputs to the system. This is reasonable because such facts are independent of the protocol used to implement continuous consensus, and such events allow us to monitor information that is relevant in a broad range of applications. It is possible, however, to extend the core and allow it to monitor events that are concerned with the failure pattern as well as the external inputs. In the context of the FIP this could actually determine all communications and all message contents. We say that a proposition $p \in \Phi$ is *objective* if its truth depends on the failure pattern and input assignment of the run. Namely, if there is a set T_p of pairs (ζ, β) such that, for every run $r' = \text{FIP}(\zeta', \beta')$ and time m', we have that $(R, r', m') \models p$ exactly if $(\zeta', \beta') \in T_p$. In particular, an objective proposition is independent of time. Notice that the propositions in $\Phi_{\mathcal{E}}$, which depend on the external input component, ζ, of the run, are by definition objective propositions. Another feature of the new definition is that an objective proposition has the same truth value in runs of different protocols, because T_p is independent of the protocol being followed.

We can now define an *extended* continuous consensus problem, whose specification differs from the original problem only in that the core consists of a set $\Phi'_{\mathcal{E}} \subseteq \Phi$ of objective facts. The proof of Theorem 3.8 shows that the view $\mathsf{V}_{G_i(c)}(c)$ computed by a nonfaulty process i in the ConCon protocol at a point (r,k) is shared among the nonfaulty processes, and hence contained in $\hat{\mathsf{V}}_i[r,k]$. The proof of Theorem 3.10 shows that $\mathsf{V}_{G_i(c)}(c)$ contains $\hat{\mathsf{V}}_i[r,k]$. Thus, the ConCon protocol executed at nonfaulty processes in fact computes $\hat{\mathsf{V}}[r,k]$ at each point (r,k). The information in $\hat{\mathsf{V}}[r,k]$ can imply that certain processes are faulty, and that particular messages were sent successfully in the run while others were not. By Proposition 3.9 all of this information is common knowledge at (r,k). But there may be (objective) facts about the failure pattern that are common knowledge at (r,k) and do not appear explicitly in $\mathsf{V}[r,k]$. For example, there are no explicit failures and/or successful message deliveries in the empty view λ. Nevertheless, if, for example, $\hat{\mathsf{V}}_i[r,3] = \lambda$ then among other things the nonfaulty processes did not discover t failures in the first round. This translates into an objective fact q about the run. And since, as shown in Proposition 3.9, the identity of $\hat{\mathsf{V}}[r,k]$ for the nonfaulty processes is common knowledge, $C_N q$ would hold at $(r,3)$ in this case. In order to obtain an optimum solution for the extended continuous consensus problem, we need to add this type of fact to the core. We do this by replacing the definition of $\mathcal{E}(\mathsf{V})$ used in ConCon by

$$\mathcal{E}'(\mathsf{V},k) \triangleq \{q \in \Phi'_{\mathcal{E}} : (R,r,k) \models q \text{ for all points } (r,k) \text{ such that } \hat{\mathsf{V}}[r,k] = \mathsf{V}\}.$$

We denote by ConCon′ the protocol obtained from ConCon by replacing $\mathcal{E}(\lambda)$ by $\mathcal{E}'(\lambda,k)$ and replacing $\mathcal{E}(\mathsf{V}_{G_i(c)}(c))$ by $\mathcal{E}'(\mathsf{V}_{G_i(c)}(c),k)$ on line 5. Since ConCon′ computes the view $\hat{\mathsf{V}}[r,m]$ at every point (r,m), we obtain an analogous result to Corollary 3.12 for arbitrary objective propositions in $\Phi'_{\mathcal{E}}$. An analogous proof to that of Theorem 3.10 can now be shown to yield:

Corollary 3.13 ConCon′ is an optimum protocol for the extended continuous consensus problem.

3.3 Waste and ConCon

Waste

The concept of *waste* was defined in [3]. In this section we shall review the definition of *waste* and extend the term further to define *local waste*. The task discussed in [3] was obtaining agreement in a crash failure model about the initial configuration of the system. *Waste* was defined in the context of this analysis. *Local waste* is a generalization of *waste*, which helps us analyze when agreement can be reached about any fact in the system (rather than just about time $k = 0$). Moreover, our definition of local waste will apply to both the crash and the omission models.

Recall that according to Theorem 3.2, once it becomes common knowledge that a clean round has occurred, all the facts about the initial configuration become common knowledge. It is easy to show that at time $t+1$ it is common knowledge that a clean round has occurred during the first $t+1$ rounds, and thus all the facts about the initial configuration are common knowledge. However, if $k+j$ failures are discovered by the end of round k, Dwork and Moses' analysis in [3] shows that by the end of round $t+1-j$ there must be a clean round. Moreover, it can be shown that by the end of round $t+1-j$ it is common knowledge that a clean round has occurred. From the point of view of an adversary trying to delay the agreement as much as possible, this may be considered as a "waste" of j failures.

We now formally define the waste of a run, $W(r)$. First, define $\mathcal{N}(r,k)$ as the number of faulty processes discovered by time k in r. Next, we define $d(r,k)$ as the difference between $\mathcal{N}(r,k)$ and k, and finally we can define the waste, $W(r)$, as the maximal value of $d(r,k)$:

$$\mathcal{N}(r,k) \triangleq max\{j : (r,k) \models D(\text{``}j\ processes\ have\ failed\text{''})\} \tag{3.6}$$

$$d(r,k) \triangleq \mathcal{N}(r,k)\ -\ k \tag{3.7}$$

$$W(r) \triangleq \max_{k \geq 0} d(r,k) \tag{3.8}$$

Note that in this subsection the operators D and C refer to distributed knowledge and common knowledge w.r.t. the set of active processes. The analysis in [3] shows that at the end of round $t+1-W(r)$ it is common knowledge that a clean round has occurred in the run. Thus at $t+1-W(r)$ all facts about the initial configuration of the nonfaulty processes become common knowledge. The concept is summarized in the following lemma from [3], which is quoted here without proof:

Lemma 3.14 Let φ be a fact about the initial configuration. Then:

$$(r,t+1-W(r)) \models D\varphi \ \textit{iff} \ (r,t+1-W(r)) \models C\varphi.$$

Lemma 3.14 implies that at the end of round $t+1-W(r)$, any fact known to an active process [4] at time $k=0$ becomes common knowledge. An immediate corollary of Lemma 3.14 is the following:

Corollary 3.15 Let φ be a fact about the initial configuration. If $(r,0) \models D_N\varphi$ then $(r,t+1-W(r)) \models C_N\varphi$.

Proof: Assume $(r,0) \models D_N\varphi$. It is easy to see that also $(r,t+1-W(r)) \models D_N\varphi$, and thus $(r,t+1-W(r)) \models D\varphi$. By Lemma 3.14 we have that $(r,t+1-W(r)) \models C\varphi$, i.e., that φ is common knowledge among the active processes, $A(r,t+1-W(r))$. It is straightforward from the definition of common knowledge that any fact which is common knowledge among the active processes, is also common knowledge among any subset of this set. Since $N \subseteq A(r,t+1-W(r))$, we obtain $(r,t+1-W(r)) \models C_N\varphi$, and we are done.

■

[4] An active process w.r.t. time $t+1-W(r)$

We refer to the last round, ℓ^{w} in which $d(r,k)$ reaches its maximal value as the round in which *the waste was reached*, i.e., $\ell^{\mathrm{w}} = max\{\arg\max_{k\geq 0} d(r,k)\}$. An interesting property, which is straightforward from the definition of waste, the round following the last time the waste is reached — round $\ell^{\mathrm{w}}+1$ — is a clean round.

A Generalized Definition of Waste

The concept of *waste* was defined in the context of the crash model, and was aimed at analyzing when facts about time $k=0$ can be agreed upon. *Local waste* is an extension to the definition of waste, which relates to any time $k \geq 0$. In addition, our analysis in this subsection will be relevant to both the crash and the omission models. A key difference between waste and local waste is that waste is a property of the run, $W(r)$, whereas the local waste, $W_i^{(m)}$, is a value which is computed locally with respect to a specific time m, and a process i. Moreover, two different nonfaulty processes, i and j, may compute different local wastes w.r.t. a time m, i.e., $W_i^{(m)} \neq W_j^{(m)}$. We omit the r from our definitions, as it is clear from context. We now present three definitions that lead to the notion of local waste:

$$\mathcal{N}_i^{(m)}(k) \triangleq b_i(k) - b_i(m) \tag{3.9}$$

$$d_i^{(m)}(k) \triangleq \mathcal{N}_i^{(m)}(k) \;-\; (k-m) \tag{3.10}$$

$$W_i^{(m)} \triangleq \max_{k\geq m} d_i^{(m)}(k) \tag{3.11}$$

Notice that for $m=0$, the definition of $W_i^{(m)}$ bares a strong resemblance to the definition of waste from the previous subsection, with the exception that $b_i(k)$ plays the role of $\mathcal{N}(r,k)$. This observation is not surprising, since $b_i(k)$ in the omission model is analogous to $\mathcal{N}(r,k)$ in the crash model. Once again, we define the last round, ℓ^{lw}

in which $d_i^{(m)}(k)$ reaches its maximal value as the round in which *the local waste is reached*, or more formally, $\ell^{\mathrm{lw}} = max\{\arg\max_{k\geq m} d_i^{(m)}(k)\}$. The following lemma captures the connection between local waste and common knowledge.

Lemma 3.16 Let i be a nonfaulty process, let r be a run of FIP. If $(r,m) \models D_N\varphi$, then $(r\ ,\ m+(t+1-b_i(m)-W_i^{(m)})) \models C_N\varphi$.

Proof: Define $\kappa \triangleq m+(t+1-b_i(m)-W_i^{(m)})$. Since r is a run of a full-information protocol, let us assume without loss of generality that i is running ConCon.[5] Thus at m process i computes $\mathsf{horizon}_i(m)$. Define ℓ^{lw} as the round in which *the local waste was reached*. From Equations 3.9- 3.11 we have that $W_i^{(m)} = (b_i(\ell^{\mathrm{lw}})-b_i(m))-(\ell^{\mathrm{lw}}-m)$. At the end of round ℓ^{lw}, process i computes the horizon according to ConCon, and we have that:

$$\begin{aligned}
\mathsf{horizon}_i(\ell^{\mathrm{lw}}) &= \ell^{\mathrm{lw}}+t+1-b_i(\ell^{\mathrm{lw}}) = \qquad (3.12)\\
&= \ell^{\mathrm{lw}}+t+1-b_i(\ell^{\mathrm{lw}})\ +b_i(m)-b_i(m)+m-m =\\
&=\ [m+t+1-b_i(m)]\ -\ [(b_i(\ell^{\mathrm{lw}})\ -b_i(m))-(\ell^{\mathrm{lw}}-m)] =\\
&=\ [m+t+1-b_i(m)]\ -\ W_i^{(m)}\ \triangleq \kappa
\end{aligned}$$

Since $\mathsf{horizon}_i(\ell^{\mathrm{lw}}) = \kappa$, from Proposition 3.6(a) we have that $crit_i(\kappa) \geq \ell^{\mathrm{lw}}$. Without loss of generality, assume $crit_i(\kappa) = \ell' \geq \ell^{\mathrm{lw}}$. Thus $M_i[\kappa] = \mathsf{V}_{G_i(\ell')}(\ell')$. Now since $(r,m) \models D_N\varphi$ by the assumption, and since $N \subset G_i(\ell')$, we necessarily have $\varphi \in \mathsf{V}_{G_i(\ell')}(\ell')$, and thus $\varphi \in M_i[\kappa]$. From Proposition 3.9 we have that $(r,\kappa) \models C_N\varphi$, and we are done.

■

[5]We require that i will run ConCon is order to be able to define $\mathsf{horizon}_i(\cdot)$, however running ConCon is not a necessary condition for the correctness of the lemma

Lemma 3.16 presents the strong connection between *local waste* and common knowledge, by showing that facts about time m become common knowledge $(t - b_i(m)) + 1 - W_i^{(m)}$ rounds after m. For the special case of $m = 0$, it is interesting to compare our analysis in this subsection regarding the local waste, to the analysis in [3]. We first present the following lemma, which argues that *waste* is indeed a special case of *local waste*.

Lemma 3.17 Let i be a nonfaulty process, let R be a system in the crash failure model, i.e., $fm =$ crash, and let $r \in R$ be a run of FIP. Moreover, let ℓ^{w} and ℓ^{lw} be the rounds in which the waste $W(r)$ and the local waste $W_i^{(0)}$ are reached, respectively. Then:

(i) $\ell^{\mathrm{w}} = \ell^{\mathrm{lw}}$

(ii) $W_i^{(0)} = W(r)$

Proof: For (i), assume by way of contradiction that $\ell^{\mathrm{w}} \neq \ell^{\mathrm{lw}}$. Notice that since ℓ^{w} is the round in which the waste is reached, round $\ell^{\mathrm{w}} + 1$ must be clean, which means that no failure is discovered by the active processes. Notice that $G_i(\ell^{\mathrm{w}}) \subseteq A(r, \ell^{\mathrm{w}})$, since $G_i(\ell^{\mathrm{w}})$ does not include processes whose faulty behavior was discovered by i in round $\ell^{\mathrm{w}} + 1$. Notice also that $G_i(\ell^{\mathrm{w}}) \supseteq A(r, \ell^{\mathrm{w}})$, since if there is a process $j \in A(r, \ell^{\mathrm{w}}) \setminus G_i(\ell^{\mathrm{w}})$, then j's faulty behavior must be discovered by the nonfaulty process i in round $\ell^{\mathrm{w}} + 1$, which not possible since $\ell^{\mathrm{w}} + 1$ is a clean round. Thus we have that $G_i(\ell^{\mathrm{w}}) = A(r, \ell^{\mathrm{w}})$. Notice that using a similar argument $G_i(\ell^{\mathrm{lw}}) = A(r, \ell^{\mathrm{lw}})$ also holds. Since $\mathcal{N}(r, \ell^{\mathrm{w}})$ is computed according to the knowledge of the set $A(r, \ell^{\mathrm{w}})$, and $b_i(\ell^{\mathrm{w}} + 1)$ is based on the knowledge of $G_i(\ell^{\mathrm{w}})$, it follows that $b_i(\ell^{\mathrm{w}}) = \mathcal{N}(r, \ell^{\mathrm{w}})$. We thus have that $d(r, \ell^{\mathrm{w}}) = d_i^{(0)}(\ell^{\mathrm{w}})$. Similarly, we also have $b_i(\ell^{\mathrm{lw}}) = \mathcal{N}(r, \ell^{\mathrm{lw}})$, and thus $d(r, \ell^{\mathrm{lw}}) = d_i^{(0)}(\ell^{\mathrm{lw}})$. Now assume $\ell^{\mathrm{w}} > \ell^{\mathrm{lw}}$, and thus $d(r, \ell^{\mathrm{w}}) \geq d(r, \ell^{\mathrm{lw}})$. It follows that $d_i^{(0)}(\ell^{\mathrm{w}}) \geq d_i^{(0)}(\ell^{\mathrm{lw}})$, which contradicts the fact that ℓ^{lw} is the round in which the local waste is reached. Assume, on the other hand that $\ell^{\mathrm{w}} < \ell^{\mathrm{lw}}$, and thus $d_i^{(0)}(\ell^{\mathrm{w}}) \leq d_i^{(0)}(\ell^{\mathrm{lw}})$, which means that $d(r, \ell^{\mathrm{w}}) \geq d(r, \ell^{\mathrm{lw}})$, contradicting the fact that ℓ^{w} is the round in which the waste is reached. Thus $\ell^{\mathrm{w}} = \ell^{\mathrm{lw}}$.

For (ii), since $\ell^{\mathrm{w}} = \ell^{\mathrm{lw}}$, and we have shown that also $d(r,\ell^{\mathrm{w}}) = d_i^{(0)}(\ell^{\mathrm{w}})$, by the definition of $W_i^{(0)}$ and $W(r)$ it follows that $W_i^{(0)} = W(r)$, and we are done.

■

Lemma 3.17 shows that our analysis of the local waste is a direct enhancement of Dwork and Moses' analysis of *waste* in the crash model. The result in Lemma 3.16 both extends the previous analysis to the omission failure model, and generalizes the discussion from agreeing about the initial configuration to agreeing on facts about any later time m. By using the results of CONCON from the previous sections, we were able to prove Lemma 3.16, which is a direct generalization of Lemma 3.15.

Following the proof of Lemma 3.16, it is interesting to observe that $\kappa = \mathsf{horizon}_i(m) - W_i^{(m)}$, and thus an immediate corollary is:

Corollary 3.18 Let i be a nonfaulty process, let r be a run of FIP, and let φ be a fact about time m. If $(r,m) \models D\varphi$, then $(r\ ,\ \mathsf{horizon}_i(m) - W_i^{(m)})) \models C\varphi$.

Our proof of Lemma 3.16 made use of CONCON, however, Corollary 3.18 presents an even stronger connection between local waste and CONCON. Indeed, we defined $\mathsf{horizon}_i(m)$ as process i's upper bound on the *destination* of m, and showed that in fact i's estimation for m's destination may be *improved* compared to $\mathsf{horizon}_i(m)$. By Corollary 3.18 we realize that this *improvement* is precisely $W_i^{(m)}$.

The following lemma will complete the picture regarding the connection between local waste and CONCON.

Lemma 3.19 Let i be a nonfaulty process, let $m \geq 0$, and let r be a run of FIP. Furthermore, let ℓ^{lw} be the round in which the local waste, $W_i^{(m)}$ is reached. Then:

(a) $\ell^{\mathrm{lw}} = crit_i(\mathsf{horizon}_i(m) - W_i^{(m)})$

(b) $dest_i(m) = \mathsf{horizon}_i(m) - W_i^{(m)}$

Proof: Define $\kappa \triangleq \mathsf{horizon}_i(m) - W_i^{(m)}$. The proof of Lemma 3.16 showed us that $crit_i(\kappa) \geq m$. In order to show that $\ell^{\text{lw}} = crit_i(\kappa)$, let us assume by way of contradiction that $\ell^{\text{lw}} < crit_i(\kappa)$, i.e., that $crit_i(\kappa) = \ell' > \ell^{\text{lw}}$. Thus by our analysis of ConCon, it is easy to see that $\mathsf{horizon}_i(\ell') = \kappa$. By the proof of Lemma 3.16, we have that also $\mathsf{horizon}_i(\ell^{\text{lw}}) = \kappa$, and thus:

$$\begin{aligned} \mathsf{horizon}_i(\ell^{\text{lw}}) &= \mathsf{horizon}_i(\ell') \qquad (3.13)\\ \ell^{\text{lw}} + t + 1 - b_i(\ell^{\text{lw}}) &= \ \ell' + t + 1 - b_i(\ell') \\ b_i(\ell^{\text{lw}}) - \ell^{\text{lw}} &= \ b_i(\ell') - \ell^{\text{lw}} \end{aligned}$$

Now also notice that:

$$\begin{aligned} d_i^{(m)}(\ell') &= \mathcal{N}_i^{(m)}(\ell') - (\ell' - m) = (b_i(\ell') - b_i(m)) - (\ell' - m) \qquad (3.14)\\ &= b_i(\ell') - \ell' \ \ + m - b_i(m) = b_i(\ell^{\text{lw}}) - \ell^{\text{lw}} \ \ + m - b_i(m) \\ &= (b_i(\ell^{\text{lw}}) - b_i(m)) - (\ell^{\text{lw}} - m) = d_i^{(m)}(\ell^{\text{lw}}) \end{aligned}$$

And thus we have $d_i^{(m)}(\ell^{\text{lw}}) = d_i^{(m)}(\ell')$, which contradicts our definition of ℓ^{lw}, as the latest round at which the waste is reached, and we are done with part (a).

Recall that $dest_i(m) = \kappa$ exactly if $crit_i(\kappa) \geq m$, and $crit_i(\kappa - 1) < m$. The proof of Lemma 3.16 showed us that $crit_i(\kappa) \geq m$. In order to prove that $crit_i(\kappa - 1) < m$, let us assume by way of contradiction that $crit_i(\kappa - 1) \triangleq m' \geq m$. Thus $\mathsf{horizon}_i(m') = \kappa - 1$, and recall that $\ell^{\text{lw}} = crit_i(\kappa)$.

$$\begin{aligned}\kappa - 1 = \mathsf{horizon}_i(m') &= \mathsf{horizon}_i(\ell^{\mathrm{lw}}) - 1 \qquad (3.15)\\ m' + t + 1 - b_i(m') &= \ell^{\mathrm{lw}} + t + 1 - b_i(\ell^{\mathrm{lw}}) - 1\\ b_i(m') - m' &= b_i(\ell^{\mathrm{lw}}) - \ell^{\mathrm{lw}} + 1\end{aligned}$$

And if we observe $d_i^{(m)}(m')$, we find that:

$$\begin{aligned}d_i^{(m)}(m') &= (b_i(m') - b_i(m)) - (m' - m) \qquad (3.16)\\ &= b_i(m') - m' \;\; + \;\; m - b_i(m)\\ &= b_i(\ell^{\mathrm{lw}}) - \ell^{\mathrm{lw}} + 1 \;\; + \;\; m - b_i(m)\\ &= (b_i(\ell^{\mathrm{lw}}) - b_i(m)) - (\ell^{\mathrm{lw}} - m) + 1 = d_i^{(m)}(\ell^{\mathrm{lw}}) + 1\end{aligned}$$

And we obtain that $d_i^{(m)}(m') = d_i^{(m)}(\ell^{\mathrm{lw}}) + 1$, which contradicts the assumption that ℓ^{lw} is the round in which the value of $d_i^{(m)}(\cdot)$ is maximal, and we are done with (b) as well.

■

By Lemma 3.19(b) we have that the i's horizon and i's local waste at m uniquely determine m's destination, which is precisely $\mathsf{horizon}_i(m) - W_i^{(m)}$. Moreover, Lemma 3.19(a) shows that at $\mathsf{horizon}_i(m) - W_i^{(m)}$ all facts about the first ℓ^{lw} rounds become common knowledge.

3.4 Clean Rounds Revisited

In our description of clean rounds in Section 3.1, we said that up to this work clean rounds were discussed in the context of the crash model. After having presented CONCON, we extend the definition of clean rounds to the omission model. We say that a round k is clean if $b_i(k-1) = b_i(k)$ for all nonfaulty processes i. Thus, it is not the discovery of a failure in a round that makes it dirty; rather, a round is dirty if, for some process z, it is the first round in which z's failure is reported by a process that is trusted by some nonfaulty process.

Intuitively, it is straightforward from the definition of horizon that our computation of $\mathsf{horizon}_i(m) = m + (t+1) - b_i(m)$ guarantees that there exists at least one round k, such that $m < k \leq \mathsf{horizon}_i(m)$, in which $b_i(k-1) = b_i(k)$. This fact does not guarantee a clean round, however, the following lemma implies that indeed a clean round is guaranteed to occur in this interval.

Lemma 3.20 Let i be a nonfaulty process. Then for all $k \geq 0$ if $crit_i(k) = m \neq -1$ then $m+1$ is a clean round.

Proof: Assume by way of contradiction that for some nonfaulty j we have $b_j(m) \neq b_j(m+1)$. By definition of B_j, we have that b_j is a nondecreasing function of m, and thus $b_j(m) < b_j(m+1)$. Since both i and j are nonfaulty, we have by Theorem 3.8 that $crit_i(k) = crit_j(k) = m$. It follows that $\mathsf{horizon}_j(m) = k$. Since $b_j(m) < b_j(m+1)$, it follows by the definition of horizon that $\mathsf{horizon}_j(m) \geq \mathsf{horizon}_j(m+1)$. By Proposition 3.6(a) $crit_j(k) \geq m+1$, which contradicts the fact that $crit_j(k) = m$, and we are done.

■

Thus the round following a critical time must be a clean round. In fact, this observation is not surprising following Section 3.3, since by Lemma 3.19 and by the fact that $\mathsf{horizon}_i(m) = k$ we have that $W_i^{(m)} = 0$, and thus m is the round in which $W_i^{(m)}$ is reached. It is immediate from the definition of $W_i^{(m)}$ that the next round, $m+1$, must be clean.

Another interesting property of clean rounds in our context is presented below.

Lemma 3.21 Let $m+1$ be a clean round. Then $B_i(m) \cup G_i(m) = \mathbb{P}$ for every nonfaulty process i.

Proof: Assume by way of contradiction that for some nonfaulty i there exists an m and a process j such that $m+1$ is a clean round and $j \notin G_i(m) \cup B_i(m)$. Since i is nonfaulty, we have $i \in G_i(m+1)$, and since i knows about j's faulty behavior at time $m+1$, we have $j \in B_i(m+1)$, while we assumed $j \notin B_i(m)$. Thus it must be the case that $b_i(m+1) > b_i(m)$, which is in contradiction to the assumption that $m+1$ is a clean round.

■

Notice that Lemmas 3.20 and 3.21 imply that if m is a critical time of some k, then $B_i(m) \cup G_i(m)$. Intuitively, Moses and Tuttle's fixed-point construction (Figure 3.6) proposed a similar claim, however their definitions of the sets of faulty and nonfaulty processes, the sets F and S are a bit different than our definitions of good and bad processes.

Finally, we conclude this section by presenting the following lemma:

Lemma 3.22 Let φ be a fact about the first $k-1$ rounds, let i be a nonfaulty process in r and assume that $S = G_i(k-1)$. If $(r, k-1) \models D_S(\varphi)$ and round k of r is clean, then $(r,k) \models E_N\varphi$.

Proof: Assume that round k is clean, and that $(r, k-1) \models D_S(\varphi)$. We shall prove that $(r,k) \models E_N\varphi$ by showing that for all nonfaulty processes, j, we have that $(r,k) \models K_j\varphi$.

Assume that j is nonfaulty. We claim that $B_i(k-1) = B_j(k-1)$. Assume by way of contradiction that $B_i(k-1) \neq B_j(k-1)$. Without loss of generality, assume that there exists a process $z \in B_i(k-1) \setminus B_j(k-1)$. Thus, at time k process i knows about z's failure. Since i is nonfaulty, we have that $i \in G_j(k)$, and thus $z \in B_j(k)$. It follows that $B_j(k-1) \neq B_j(k)$, which contradicts the assumption that k is a clean round. Thus we have that $B_i(k-1) = B_j(k-1)$. Since round k is clean, Lemma 3.21 implies that we also have $S = G_i(k-1) = G_j(k-1)$. It follows that $(r, k-1) \models D_S(\varphi)$. Notice that by the definition of $G_j(k-1)$, all messages from processes in $G_j(k-1)$ are delivered to j successfully in round k, and since we assume a FIP, we must have that $(r,k) \models K_j\varphi$, and we are done.

■

One of the key results by Dwork and Moses [3], which is presented in Theorem 3.1, is that following a clean round every fact that was previously distributed knowledge, becomes known to all the nonfaulty processes. By Lemma 3.22, we see that the same result holds in the omission model, in the context of *clean rounds* as presented in this section. It follows that our new definition of *clean rounds* plays essentially the same role in the omission model as the original definition of clean rounds played in the crash model; after a clean round all nonfaulty processes share a consistent view of the system.

3.5 A Run of ConCon

In this section we discuss some interesting properties of the ConCon protocol, of the horizon, and of the sets of good and bad processes. We discuss these properties by presenting an example of a run, r of ConCon. In our example $n = 16$, $t = 6$, and we assume that processes i and j are nonfaulty.

The horizon Revisited

Intuitively, we described $\mathsf{horizon}_i(k)$ as process i's estimation of when facts about time k will join the core. Figure 3.7 gives further intuition about the horizon, by showing $\mathsf{horizon}_i$ and $\mathsf{horizon}_j$ as a function of k for two nonfaulty processes, i and j in r. Let us point out some observations about r, which are illustrated in Figure 3.7. These observations follow from the properties of the horizon which were described in Section 3.1.

- No processes are initially known to be faulty, i.e., $b_i(0) = 0$, and thus $\mathsf{horizon}_i(0) = t+1 = 7$.
- The horizon increases by at most one in every round.
- Horizons at different processes are not necessarily the same, e.g., $\mathsf{horizon}_i(9) = 14$, while $\mathsf{horizon}_j(9) = 15$ (however, recall that the correct processes' horizons are identical at a critical time).

- The graph shows the function $f(k) = k$ as a reference, since in a way it can be thought of as an "asymptote" of the horizon. If the system reaches a point (r,k) at which $b_i(k) = t$, i.e., all faulty processes have been discovered, then for all $\ell > k$ we will have $\mathsf{horizon}_i(\ell) = \ell + 1$. Indeed, in our example we can see that at any time $\ell > 16$, we have $\mathsf{horizon}_i(\ell) = \ell + 1$.

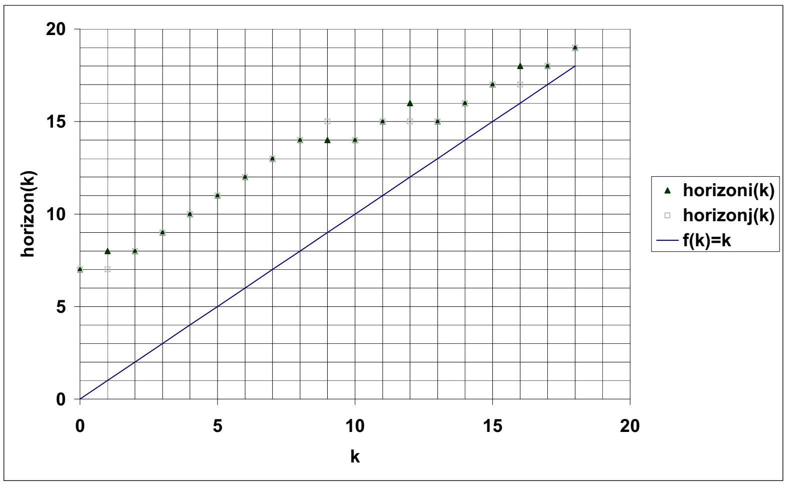


Figure 3.7: The Horizon vs. k in a run r.

Critical Times Revisited

In Figure 3.8 we can see a graph of the critical time, $crit_i(k)$ as a function of k for r. Notice the following observations:

- There are times which are not critical times, e.g., 12 is not the critical time of any k.

- If $\mathsf{horizon}_i(k) \neq \mathsf{horizon}_j(k)$ then k cannot be a critical time. This follows from Theorem 3.6, since if $k = crit_i(\ell)$ of some time ℓ, then it must be so for all non-faulty processes i. In our example, $\mathsf{horizon}_i(12) \neq \mathsf{horizon}_j(12)$, and indeed 12 is not a critical time.

- If $crit_i(k) = m$, then we can check in Figure 3.7 and see that indeed $\mathsf{horizon}_i(m) = k$. For example, Figure 3.8 shows that $crit_i(10) = 4$, and in Figure 3.7 we can see that $\mathsf{horizon}_i(4) = 10$.

- Much like the graph of the horizon, we show $f(k) = k$ as a reference, and similarly, it can be thought of as an "asymptote" for $crit_i(k)$, with $crit_i(\ell) = \ell - 1$ when $b_i = t$.

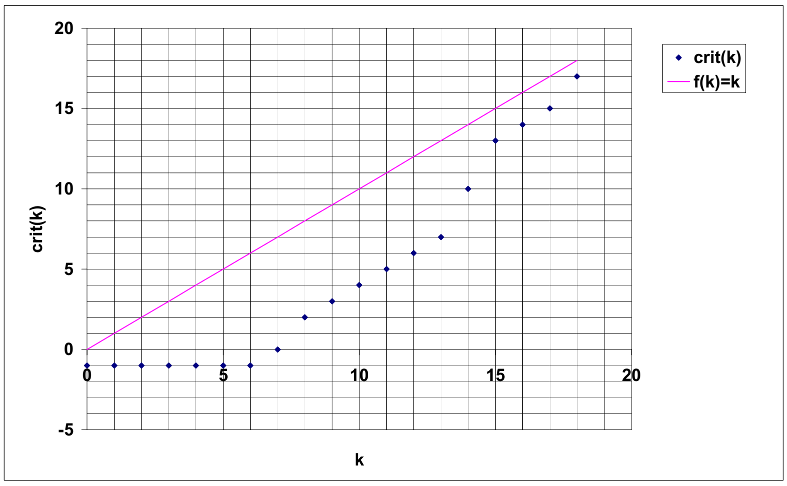


Figure 3.8: Critical time vs. k in a run r.

Good and Bad Processes Revisited

Our subtle definitions of the good and bad process are the basis of the horizon computation, and are thus a key notion in ConCon. As discussed in Section 3.1, our definition of $G_i(k)$ uses i's knowledge at time $k+1$, enabling i to receive information about round k from all processes in $G_i(k)$, and thus i's computation of $B_i(k)$ is based on the knowledge of *all* the good processes $G_i(k)$. The following properties are straightforward from the definitions of G and B, and can be seen quite vividly in Figure 3.9:

- Notice that $G_i(k)$ and $B_i(k)$ are distinct sets, i.e., $G_i(k) \cap B_i(k) = \emptyset$ for all k.

- B_i is a monotonously nondecreasing set, i.e., $B_i(k) \subseteq B_i(k+1)$, and G_i is a monotonously nonincreasing set, i.e., $G_i(k) \supseteq G_i(k+1)$.

- Another interesting property of B and G, which is a corollary of Lemma 3.21, is that if m is a critical time of some k, i.e., $m = crit_i(k)$, then we have $B_i(m) \cup G_i(m) = \mathbb{P}$. For example, recall from the previous subsection that $crit_i(10) = 4$, and indeed we can see in Figure 3.9 that $B_i(4) \cup G_i(4) = \mathbb{P}$.

- By Lemma 3.20 it follows that round 5 must be clean, and indeed we can see that $b_i(4) = b_i(5)$.

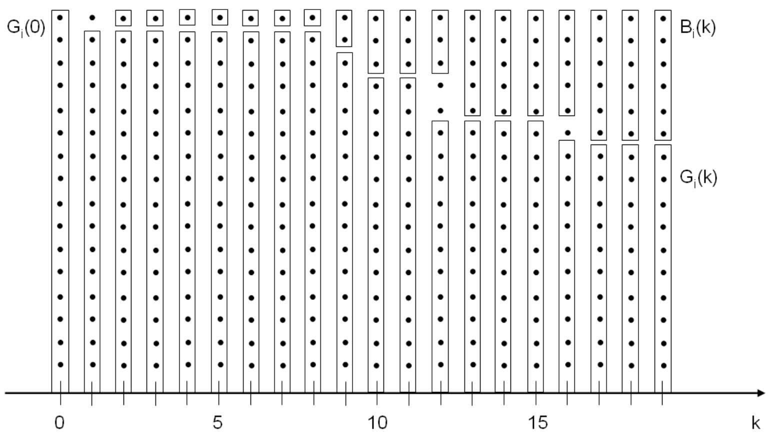


Figure 3.9: $G_i(k)$ and $B_i(k)$ in r.

Local Waste and Destination

Figure 3.10 shows $dest_i(k)$ and $\mathsf{horizon}_i(k)$ as a function of k. The following observations may be seen from the graph:

- It is possible for several points, m, m' to have the same destination. For example, $dest_i(9) = dest_i(10) = 14$. Recall that for this reason the destination is not defined as $crit_i^{-1}$.

- Process i's horizon at m is an estimation of $dest_i(m)$. As we can see in Figure 3.10, indeed $dest_i(m)$ and $\mathsf{horizon}_i(m)$ are equivalent most of the time. By Lemma 3.19 we have that the connection between the horizon and the destination is $dest_i(m) = \mathsf{horizon}_i(m) - W_i^{(m)}$, and thus the destination is different than the horizon at times m in which $W_i^{(m)} > 0$. In our particular example, the only such time m is $m = 12$.

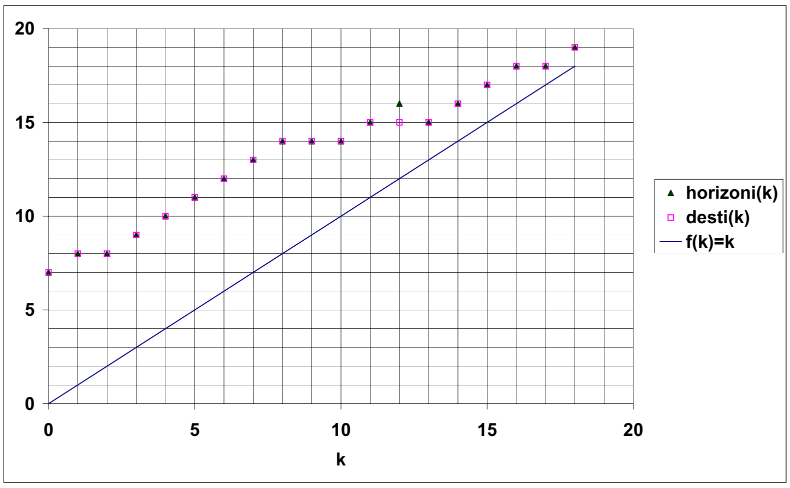


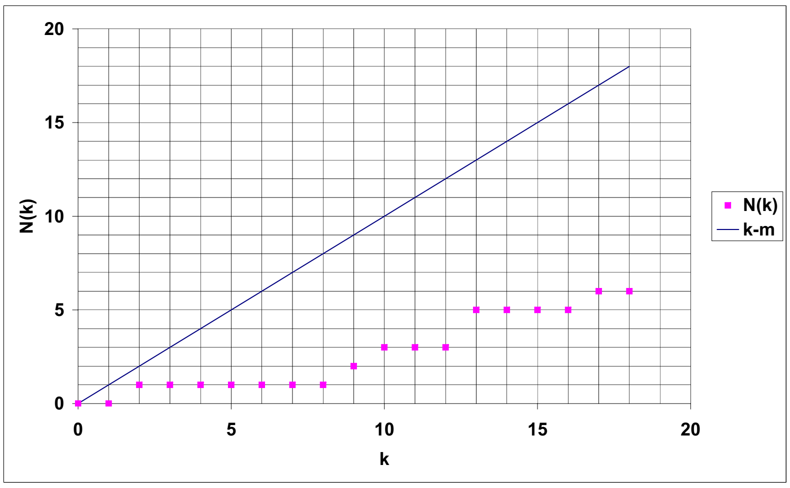
Figure 3.10: $dest_i(k)$ and $\mathsf{horizon}_i(k)$ in r.

Figure 3.11 illustrates the computation of the local waste, $W_i^{(m)}$. The first graph, 3.11(a), presents $\mathcal{N}_i^{(0)}(k)$, and the second, 3.11(b) shows $\mathcal{N}_i^{(12)}(k)$. Observe that:

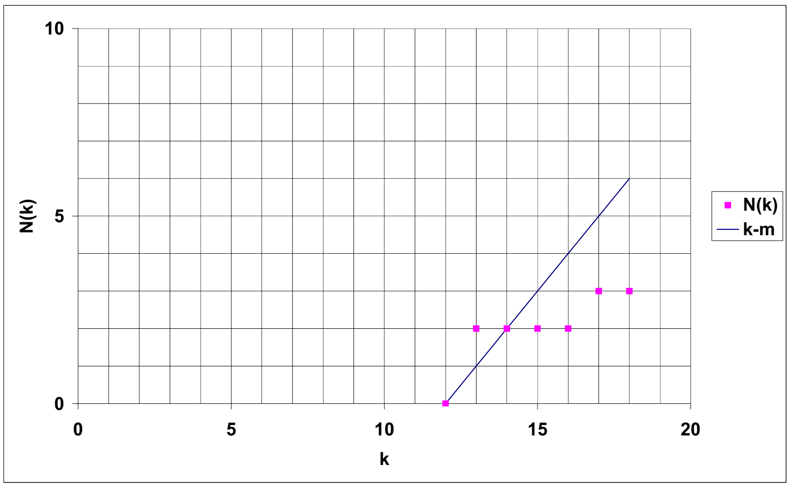
- Since $\mathcal{N}_i^{(0)}(k) = b_i(k)$, 3.11(a) in fact shows the number of bad processes as a function of k.

- Notice that in our example $W_i^{(0)} = 0$, since no failures are discovered "fast enough" after time 0. On the other hand, as we can see in 3.11(b) since $b_i(k)$ increases by 2 in round 13, we have that $W_i^{(12)} = 1$

- In both graphs we present $k - m$ as a reference. Recall that $d_i^{(m)}(k) = \mathcal{N}_i^{(m)}(k) - (k - m)$, and that the waste is reached when the difference, $d_i^{(m)}(k)$, is maximal. In the graphs, $d_i^{(m)}(k)$ is positive precisely when $\mathcal{N}_i^{(m)}(k)$ is over the $k-m$ line. Thus when $\mathcal{N}_i^{(m)}(k)$ reaches its highest point over the $k-m$ line, the waste is reached. In our example for $m=12$, the only time at which $\mathcal{N}_i^{(12)}(k)$ is over the line is at $k=13$, which is when the waste is reached.



(a) $\mathcal{N}_i^{(0)}(k)$ vs. k for $m = 0$.



(b) $\mathcal{N}_i^{(12)}(k)$ vs. k for $m = 12$.

Figure 3.11: Local Waste

Chapter 4

Uniform Continuous Consensus

4.1 The UCC Problem

The continuous consensus problem specifies constraints only on the cores of nonfaulty processes, guaranteeing nothing about faulty ones. Observe, however, that failures in our models are not considered malicious—there is no lying and faulty behavior is closely related to crashing or communication malfunction. It is thus natural to consider a stronger version of the problem, which we call *uniform continuous consensus* (*UCC*). The specification of UCC is similar to that of CC (see Section 3.1), except that Accuracy and Consistency are required to hold for arbitrary processes and not just for nonfaulty ones. Completeness, however, is still restricted to events that are known to nonfaulty processes: If an event $e \in \mathcal{E}$ is known to a *nonfaulty* process j at any point, then $e \in M_i[k]$ must hold (for *all* processes i, of course) at some time k. One can expect a solution to UCC to be similar in spirit to ConCon, because the failures we consider, both in the crash and the (sending) omission models, only affect the ability of a process to *send* messages, so that even faulty processes receive all incoming messages. In this chapter we consider how the ConCon protocol can be modified to obtain UniConCon, an optimal protocol for UCC.

As shown in [2, 5], simultaneously consistent behavior by all participants is closely related to (standard) common knowledge, that is, the traditional notion equivalent to an infinite conjunction of "everyone knows" (while in the previous chapter our analysis referred to common knowledge among the nonfaulty processes).

Common knowledge among *all* processes (see Section 2.6) and UCC are related in the same way as C_N and continuous consensus are, and a result analogous to Proposition 3.9 holds. Neiger and Tuttle [10] show that in our settings (crash and omission failure models) the two notions of common knowledge coincide:

Theorem 4.1 (Neiger and Tuttle) *$R \models (C\varphi \equiv C_N\varphi)$ for every formula φ and* FIP *system R.*

Intuitively, the claim of Theorem 4.1 can be explained by the fact that failures in these models affect only the ability of a process to send messages; even faulty processes receive all incoming messages.

A natural question in light of Theorem 4.1 is whether CONCON itself solves the UCC problem. Unfortunately, it does not. The problem with using CONCON is that a faulty process x might know of failures at time k but not tell the nonfaulty processes. If $x \in G_x(k)$, then these failures will be counted in $b_x(k)$ and will therefore play a role in determining $\mathsf{horizon}_x(k)$, but if x is silent from round $k+1$ on, for example, then these failures will not affect the calculations of other processes. An inconsistency between x's core and those of the nonfaulty processes will arise as a consequence.

A protocol for UCC

We now present a variant of CONCON, called UNICONCON, that solves the UCC problem. The UNICONCON protocol is based on the following intuition. Whether or not a process x is faulty, it is still guaranteed in our models to receive all messages that are sent to it. Moreover, if $g \in G_x(k)$, then no nonfaulty process has discovered that g is faulty by time $k-1$. The information available to g at time $k-2$ has thus been transmitted to the (truly) nonfaulty processes in round $k-1$, and they are guaranteed to relay it to x (and to all other processes) in round k. Hence, a possibly faulty process x can simulate in round k what a nonfaulty process g would compute two rounds earlier. As a result, it is possible to design a uniform protocol that mimics the behavior of CONCON as long as the distance to the horizon is three rounds or more. Special care needs to be taken when the distance to the horizon becomes two rounds or one.

In this section we use $i, j, x, z, g \in \mathbb{P}$ to denote processes in our system, such that i and j are nonfaulty processes, x and z are *potentially* faulty processes, and g is a *good* process.[1]

The UNICONCON protocol is given in Figure 4.1. We distinguish the values of $Latest_i$, $crit_i$ and M_i computed in CONCON from the corresponding values computed in UNICONCON by adding the superscript u to instances of the latter. On line 2 of the protocol, the process x chooses an arbitrary member g of $G_x(k-1)$ whose computation in round $k-2$ of CONCON process x will simulate in round k.[2] Notice that $\mathsf{horizon}_g(k-3)$ is available to x in round k because x's local state has a copy of $\mathsf{V}_g(k-2)$ (since otherwise $g \notin G_x(k-1)$). Lines 4, 5 and 6 of UNICONCON play the same role as lines 3, 4 and 5 of CONCON respectively. On lines 3 and 4 of UNICONCON process x simulates what a nonfaulty process would have computed two rounds earlier. Line 5 computes the critical time, distinguishing three distinct cases: there are special tests for whether the critical time is one or two rounds back; when neither is the case, the process uses the critical time obtained by simulating the critical round that a seemingly nonfaulty process would compute in CONCON. Finally, line 6 computes the core in much the same way as in CONCON.

The correctness claim for UNICONCON is summarized by the following theorem:

Theorem 4.2 *Fix a run r in a* FIP *system R. Let i be a nonfaulty process in r, let x be an arbitrary process, and let $k \geq 1$. Then $M^{\mathsf{u}}_x[k] = M_i[k]$.*

The proof of Theorem 4.2 as well as intermediate lemmas used in its proof are presented in Appendix A.2. An immediate corollary of Theorems 4.1 and 4.2 is

Corollary 4.3 UNICONCON is an optimum protocol for Uniform Continuous Consensus.

[1]Good here stands simply for the fact that $g \in G_x(k)$ for some process x at a certain time k of interest.
[2]A process x can choose itself as g as long as $x \in G_x(k-1)$, so that x does not know that it is faulty.

```
UNICONCON($x$)

0   $Latest^{\mathsf{u}}_x[\ell] \leftarrow -1$ for all $\ell \geq 1$

    for every round $k \geq 1$ do
1       send local state and receive messages according to FIP
2       $g \leftarrow$ arbitrary member of $G_x(k-1)$
        if $k \geq 3$ then
3           compute $G_g(k-3)$, $B_g(k-3)$ and $\mathsf{horizon}_g(k-3)$
                                 ▷ $x$ simulates $g'$s behavior in CONCON :
4           $Latest^{\mathsf{u}}_x[\mathsf{horizon}_g(k-3)] \leftarrow k-3$
        endif

5       $c^{\mathsf{u}} \leftarrow \begin{cases} k-1 & \text{if } \mathsf{horizon}_x(k-1)=k \\ k-2 & \text{if } \mathsf{horizon}_g(k-2)=k \wedge \mathsf{horizon}_x(k-1)\neq k \\ Latest^{\mathsf{u}}_x[k] & \text{otherwise} \end{cases}$

6       $M^{\mathsf{u}}_x[k] \leftarrow \begin{cases} \mathcal{E}(\lambda) & \text{if } c^{\mathsf{u}}=-1 \\ \mathcal{E}(\mathsf{V}_{G_x(k-1)}(k-1)) & \text{if } c^{\mathsf{u}}=k-1\geq 0 \\ \mathcal{E}(\mathsf{V}_{G_g(c^{\mathsf{u}})}(c^{\mathsf{u}})) & \text{otherwise} \end{cases}$
    endfor
```

Figure 4.1: Process x's computation in UNICONCON.

Chapter 5

Conclusion

In the present study we have presented and analyzed the continuous consensus (CC) problem, which generalizes simultaneously consistent action in fault-prone synchronous systems. Using a knowledge-based approach, a solution to the CC problem called CONCON was presented, as well as its uniform variant, UNICONCON. These protocols are both *simple* and *optimal*. Continuous consensus is closely related to common knowledge. Moreover, the optimal solution to the CC problem is equivalent to computing all the facts that are common knowledge. The definition of the CC problem sheds a new light on the study of simultaneously consistent actions, and allows for a simpler and more intuitive analysis.

5.1 Summary of Results

Continuous Consensus and CONCON. A continuous consensus service requires each process i to maintain at every time k an up-to-date core $M_i[k]$ of information about the past, so that the cores at all correct processes are guaranteed to be identical. A solution to the CC problem in the crash and omission failure models called **CONCON** was presented. A striking aspect of the solution is its simplicity: At every round, each process updates a single value based on a straightforward computation. Moreover, while the solution is stated in the context of the full-information protocol, it can be implemented in a more efficient manner.

Optimality and Connection to Common Knowledge. Continuous consensus is shown to be closely related to the problem of computing what is common knowledge at any given point. It is shown that the contents of the core $M_i[k]$ of any CC protocol is common knowledge among the nonfaulty processes at time k. Moreover, it is shown that the core computed in CONCON contains precisely all the facts that are common knowledge at time k, which the derives **optimality** of CONCON: the core produced by CONCON is the largest possible core at any given time. Moreover, the cores of all correct protocols for continuous consensus are subsets of CONCON's core.

Uniform Continuous Consensus. The *uniform* variant of the continuous consensus (UCC) problem requires that *all* processes, both faulty and nonfaulty, will maintain the same core at all times. Our solution to UCC in the crash and omission models, called UNICONCON, enables a potentially faulty process x to compute in every round k the *same core* as a nonfaulty process i would produce in CONCON, i.e., $M_i[k] = M^{\mathsf{u}}_x[k]$. Process x performs this computation by *simulating* what a nonfaulty process i would have computed according to CONCON. Interestingly, the solution to UCC **does not incur any degradation** in the information contained in the core of shared information. UCC is related to common knowledge among *all* processes in the same way that CC is related to common knowledge among the nonfaulty processes. The optimality of UNICONCON follows from $M_i[k] = M^{\mathsf{u}}_x[k]$, and from the equivalence in [10].

Clean Rounds. This work bridges a gap between the analysis of common knowledge with crash failures in [3] and that for omission failures in [4]. In the case of crash failures, clean rounds are rounds in which no new failures are discovered. However, prior to this work no analogous definition of clean rounds was found in the context of the omission model. The CONCON and UNICONCON protocols presented in this work, do, however, suggest a natural generalization of clean rounds to the omission model. We define round k to be *clean* in this case if $b_i(k-1) = b_i(k-2)$ holds for every nonfaulty process i. We show that if $crit_i(m) \neq -1$ then the identity of $crit_i(m)$, as well as the fact that $crit_i(m)+1$ is a clean round, become common knowledge at time m. Moreover, it is shown that a clean round enables all “good” processes to send their messages successfully, allowing the nonfaulty processes to have a consistent view of the system at the end of this round. This key property of clean rounds plays a very similar role in [3] and in the analysis in the present work.

Waste and Local Waste. The term *local waste* is defined as a direct generalization of the term *waste* which was defined in [3]. Waste was defined in [3] in the context of the crash model, as a key tool in the analysis of the question when facts about the initial state of the system became common knowledge. Local waste extends this work, and allows for the analysis of the time at which facts **about time** m become common knowledge. Furthermore, using the analysis of ConCon, we show that the properties of waste and local waste in the crash model apply to the omission model as well.

“Good” and “Bad” Processes. Our algorithms shed an interesting light on the distinction between evidence supplied by processes that are known to be faulty and ones that are not. Recall that failures in the models we considered are benign. No process ever deviates from the protocol by sending incorrect messages. Thus, every piece of information obtained from a process can be trusted. Nevertheless, the computation of the horizon by process i in round k, and thus ultimately the times at which common knowledge is obtained, depends only on the set of *bad* processes, $B_i(k-1)$. Thus, despite the fact that information from faulty processes is correct, this central computation considers only failures reported by the *good* processes, i.e., the potentially nonfaulty processes. This distinction seems an essential aspect of the evolution of common knowledge over time in these models.

5.2 Future Work

Our analysis has hitherto been restricted to the crash and sending omission models. However, the continuous consensus problem is amenable to study and analysis in harsher failure models as well. An extension to the current work [11] (work in progress) considers the CC problem in the generalized omission and the authenticated Byzantine failure models. In the generalized model a faulty process may either omit to send messages or fail to receive them, and thus a message failure does not uniquely identify a faulty process. In the authenticated Byzantine model processes may lie, possibly without ever being discovered as liars. Thus, in both these models it is not possible to define *good* or *bad* processes solely based on the message loss in the system, as we did in Chapter 3, which makes the analysis more subtle.

Despite these challenges, it is possible to prove an analogous result to Proposition 3.9 for these more complex failure models, i.e., to show that the core, $M_i[k]$, is common knowledge at time k. However, finding a solution to the CC problem which coincides with *all* the events that are common knowledge (as we did in Chapter 3 for the simpler models) may prove problematic in these models. For example, testing for common knowledge in the generalized omission model was shown in [4] to be an NP-hard computation. Thus, an *optimum* solution for CC in this model is NP-hard as well. Furthermore, in the Byzantine model we may expect an optimum solution to be even more computationally complex. When we go beyond sending omissions, it is worthwhile to seek *tractable* solutions to the CC problem that are not necessarily *optimal*.

The authenticated Byzantine model [12] assumes that although faulty processes may be “liars”, they cannot alter any relayed information. This assumption is enforced by using an authentication scheme: all messages sent in the system are authenticated by unforgeable signatures. In the full-information protocol in this model, in every round each process sends a *signed* message encoding all the information it knows. More specifically, a process i signs and relays every piece of information it receives. The nature of the model enables us to monitor events e of the form “process i claims that p” for some process i and proposition $p \in \Phi_{\mathcal{E}}$. Once i creates a signed message containing e, no other process can forge it.

A protocol called ACC (short for *authenticated continuous consensus*) solves the CC problem in the authenticated Byzantine model, under the assumption that $n > 2t$. The protocol is fairly simple: When process i receives a message containing an event e signed by a sequence of $t+1$ distinct processes, it inserts e into its core. Intuitively, the fact that $t+1$ different processes have signed and relayed e before it reached process i guarantees that at least one of the signing processes is nonfaulty. This nonfaulty process will have forwarded the message to all nonfaulty processes, which, in turn, are able to sign and forward it. This guarantees that all nonfaulty processes simultaneously receive a message regarding e with $t+1$ signatures.

A variant of the ACC protocol is ACCD (short for ACC with **d** signatures), in which we assume $n > t$, rather than $n > 2t$. In ACCD, if process i receives at time k a message containing d signatures by distinct processes about an event e for some $d \leq t+1$, it inserts e into $M_i[\ell]$. The time $\ell = k+t+1-d$ bares some resemblance to the idea of the horizon in Chapter 3. Roughly, it is the time at which e with $t+1$ signatures would be delivered to all processes if there were at least $t+1$ nonfaulty processes in the system. It can be shown that if a nonfaulty process receives at time k a message bearing d signatures regarding an event e, then at time $\ell = k+t+1-d$, the event e will simultaneously appear in the core of all nonfaulty processes. In a sense this protocol is *early stopping* compared to ACC, since it requires i to process the event e at time k, but e does not require any further attention by i in later rounds.

The generalized omission model, in which processes may fail by either omitting to send messages, or failing to receive them, is more complex than the sending omission model, and yet simpler than the authenticated Byzantine model. A solution to the CC problem in this model may be reached by either of the solutions we have seen for the authenticated Byzantine model. However, some measures may be taken to improve any of these protocols, by estimating the number and the identity of the faulty processes in the run according to the omitted messages. This task is, of course, not as simple as in the sending omission model, since for example, the fact that process z failed to send a message to process x may imply that either of these processes is faulty. Information about failures in this model can be extracted by keeping track of a *conflict graph* among processes based on the omitted messages. For example, if a process z has conflicts with more than t others, then it must be faulty. (The minimal vertex covers of the conflict graph yield more detailed information about failures — see, e.g., [4]). Clearly, different information about conflicts is known to different processes at any given time. A key observation is that if we monitor information about conflicts in the core, then it is possible to use information about failures simultaneously and consistently by the different processes. Once the core contains information determining that a process is faulty, this process can be ignored, and the rest can act as if the bound t on the number of failures is reduced by 1. This allows the nonfaulty processes to consistently ”shift gears” as a result of failures appearing in the core.

The continuous consensus problem in the harsher failure models has a rich mathematical and promises to provide exciting challenges and insights.

Bibliography

[1] T. Mizrahi and Y. Moses, “Continuous consensus via common knowledge,” in *Theoretical Aspects of Rationality and Knowledge: Proc. Tenth Conference*, pp. 236–252, 2005.

[2] J. Y. Halpern and Y. Moses, “Knowledge and common knowledge in a distributed environment,” *Journal of the ACM*, vol. 37, no. 3, pp. 549–587, 1990.

[3] C. Dwork and Y. Moses, “Knowledge and common knowledge in a Byzantine environment: crash failures,” *Information and Computation*, vol. 88, no. 2, pp. 156–186, 1990.

[4] Y. Moses and M. R. Tuttle, “Programming simultaneous actions using common knowledge,” *Algorithmica*, vol. 3, pp. 121–169, 1988.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press, 1995.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[7] L. Lamport, “The part-time parliament,” *ACM Trans. on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[8] B. A. Coan, D. Dolev, C. Dwork, and L. J. Stockmeyer, “The distributed firing squad problem.,” *SIAM J. Comput.*, vol. 18, no. 5, pp. 990–1012, 1989.

[9] B. Coan, “A communication-efficient canonical form for fault-tolerant distributed protocols,” in *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pp. 63–72, 1986.

[10] G. Neiger and M. R. Tuttle, “Common knowledge and consistent simultaneous coordination,” *Distributed Computing*, vol. 6, no. 3, pp. 181–192, 1993.

[11] T. Mizrahi and Y. Moses, “Byzantine continuous consensus,” *work in progress*, 2006.

[12] M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[13] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, “Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement,” vol. 97, pp. 205–233, Apr. 1992.

Appendix A

Detailed Proofs

A.1 Optimality of CONCON Revisited

In Theorem 3.10 we showed that $M_i^{\mathsf{P}}[m] \subseteq M_i^{\mathsf{C}}[m]$ holds for a FIP P. In Lemma A.2 we shall show that the same argument holds for an arbitrary protocol. Intuitively, since a full information protocol requires that all processes send all of the information available to them in every round, one would expect this protocol to give each process as much information about events in the system as any protocol could. Thus we expect that the core of a FIP will include at least as much information as that of any other protocol given the exact same environment. We start by quoting the following lemma from [4], which serves us in our proof.

Lemma A.1 (Moses and Tuttle) Let $\varphi \in \Phi_{\mathcal{E}}$, let r be a run of a full-information-protocol P, and let r' be a run of an arbitrary protocol $\mathcal{P}$, with the exact same execution graph as r, i.e., $\mathsf{G}^r = \mathsf{G}^{r'}$. If $(R_{\mathcal{P}}, r', m) \models C_N\varphi$ then $(R_{\mathsf{P}}, r, m) \models C_N\varphi$.

The following lemma is a corollary of Theorem 3.10, which refers to an arbitrary protocol, $\mathcal{P}$, rather than a FIP protocol.

Lemma A.2 Let r' be a run of an arbitrary protocol $\mathcal{P}$ that solves continuous consensus. Then $M_i^{\mathcal{P}}[m] \subseteq M_i^{\mathsf{C}}[m]$ for all m.

Proof: Define a run r of a FIP P such that $\mathsf{G}^r = \mathsf{G}^{r'}$. We have to show that for any primitive fact φ, if $\varphi \in M_i^{\mathcal{P}}[m]$, then $\varphi \in M_i^{\mathsf{C}}[m]$. Assume $\varphi \in M_i^{\mathcal{P}}[m]$. By Proposition 3.9 we have that the contents of $M_i^{\mathcal{P}}[m]$ is common knowledge, and thus in particular we have that $(r',m) \models C\varphi$. Then by Lemma A.1 we have that also $(r,m) \models C\varphi$, which implies that φ appears in $\hat{\mathsf{V}}_i[r,m]$. The proof of Theorem 3.10 showed us that $\hat{\mathsf{V}}_i[r,m] \subseteq M_i^{\mathsf{C}}[m]$ for every full-information-protocol P, i.e., that all the primitive facts that are common knowledge in r appear in the core, and thus we have $\varphi \in M_i^{\mathsf{C}}[m]$, and we are done.

■

A.2 Correctness proofs for UNICONCON

In this section we prove Theorem 4.2. The first part of the proof is broken down into a number of claims. In the technical development below, keep in mind that the value of every $\mathsf{horizon}_j(m)$ is a function of the FIP execution and the values of j and m, but not of the consensus protocol being followed (whether CONCON or UniConCon).

Lemma A.3 For all $x \in \mathbb{P}$, $m \geq 0$ and $g, g' \in G_x(m+2)$, if $\mathsf{horizon}_g(m) \neq \mathsf{horizon}_{g'}(m)$, then
$\mathsf{horizon}_g(m+1) \leq \min\limits_{i \in \{g,g'\}} \mathsf{horizon}_i(m)$.

Proof: Since $\mathsf{horizon}_g(m) \neq \mathsf{horizon}_{g'}(m)$, we have that either $\mathsf{horizon}_g(m) < \mathsf{horizon}_{g'}(m)$ or $\mathsf{horizon}_g(m) > \mathsf{horizon}_{g'}(m)$. We consider each case separately. First assume that $\mathsf{horizon}_g(m) < \mathsf{horizon}_{g'}(m)$. From the definition of $\mathsf{horizon}$ we have that $b_g(m) > b_{g'}(m)$. It follows from the definitions of $B_g(m)$ and $B_{g'}(m)$ that there exists a process $z \in G_g(m)$ such that $z \notin G_{g'}(m)$. Since $g, g' \in G_x(m+2)$, process g receives a message from g' in round $m+2$, since otherwise g would tell x in round $m+3$ that g' has failed to send a message, and we would have $g' \notin G_x(m+2)$. It follows that g learns of z's failure from g' in round $m+2$, so $z \in B_g(m+1)$. It follows that at the end of round $m+2$ we have that $b_g(m+1) \geq b_g(m)+1$, and we obtain $\mathsf{horizon}_g(m+1) \leq \mathsf{horizon}_g(m)$. Since $b_g(m) > b_{g'}(m)$, and since $b_g(m)$ is a nondecreasing function of m, we also have that $b_g(m+1) \geq b_{g'}(m)+1$, so $\mathsf{horizon}_g(m+1) \leq \mathsf{horizon}_{g'}(m)$.

Now assume that $\mathsf{horizon}_g(m) > \mathsf{horizon}_{g'}(m)$. Again, from the definition of horizon, we have that $b_g(m) < b_{g'}(m)$. It follows from the definitions of $B_g(m)$ and $B_{g'}(m)$ that there exists a process $z \in G_{g'}(m)$ such that $z \notin G_g(m)$. Once again, since $g, g' \in G_x(m+2)$, in particular g receives a message from g' in all rounds up to $m+2$, and thus $g' \in G_g(m+1)$. Since $B_g(m+1)$ is based on the distributed knowledge of processes in $G_g(m+1)$ at time $m+1$, in particular $B_g(m+1)$ includes all processes known by g' at time $m+1$ to be bad. Thus we have $B_{g'}(m) \subseteq B_g(m+1)$. Since $g \in G_x(m+2)$, it must be the case that $g \in G_g(m+1)$, since otherwise x would learn of g being faulty in round $m+3$. It follows that z's failure is distributed knowledge at time $m+1$ among the processes in $G_g(m+1)$, and thus by the definition of $B_g(m+1)$, we have that $z \in B_g(m+1)$. Since $B_{g'}(m) \subseteq B_g(m+1)$ and $z \in B_g(m+1)$, it follows that $b_g(m+1) \geq b_{g'}(m)+1$, and thus $\mathsf{horizon}_g(m+1) \leq \mathsf{horizon}_{g'}(m)$. Since $\mathsf{horizon}_{g'}(m) < \mathsf{horizon}_g(m)$ by the assumption, we have $\mathsf{horizon}_g(m+1) \leq \mathsf{horizon}_g(m)$, and we are done.

■

The next two lemmas provide the formal justification for the choice of critical time in the first two cases of line 5 of the UNICONCON protocol, where $crit_x^{\mathsf{u}}$ is not chosen according to the value of $Latest_x^{\mathsf{u}}[k]$. The first captures the fact that once the horizon is one round away for at least one process, all t faulty processes are known to the nonfaulty processes, and the identity of the faulty processes becomes common knowledge.

Lemma A.4 If r is a run, $x, z \in \mathbb{P}$, and $k \geq 1$, then:

(a) if $\mathsf{horizon}_x(k-1) = k$ then $G_z(k-1) = G_x(k-1)$;

(b) $\mathsf{horizon}_x(k-1) = k$ iff $\mathsf{horizon}_z(k-1) = k$.

Proof: By definition of horizon, we have that $\mathsf{horizon}_x(k-1) = k+t-b_x(k-1)$. Hence, if $\mathsf{horizon}_x(k-1) = k$ then $b_x(k-1) = t$. Let $G^x = G_x(k-1)$ and $B^x = B_x(k-1)$. By the definition of $B_x(k-1)$ and the fact that t is an upper bound on the number of faulty processes, it follows that

$$(R,r,k-1) \models D_{G^x}(\text{the set of faulty processes is } B^x)$$

and

$$(R,r,k-1) \models D_{G^x}(\text{the set of nonfaulty processes is } G^x).$$

Every set $G_z(m)$ is guaranteed to contain all nonfaulty processes in r. Since $G^x = G_x(k-1)$ is the set of nonfaulty processes, it follows that $G_z(k-1) \supseteq G_x(k-1) = G^x$. As a result, the two facts above are distributed knowledge among the processes in $G_z(k-1)$ as well. Since $G_z(k-1) \cap B_z(k-1) = \emptyset$, we conclude that $B_z(k-1) = B_x(k-1)$ and $G_z(k-1) = G_x(k-1)$. The latter yields part (a) of the claim. By the former, we have that $b_z(k-1) = t$ and so $\mathsf{horizon}_z(k-1) = k$. This establishes the only-if direction of the claim in part (b). Switching the roles of x and z in the proof yields the other direction of (b) and we are done.

■

Lemma A.5 If r is a run, i,x and g are processes such that i is nonfaulty in r, and $g \in G_x(k-1)$, $k \geq 1$, and $\mathsf{horizon}_x(k-1) \neq k$, then:

(a) if $\mathsf{horizon}_g(k-2) = k$ then $G_g(k-2) = G_i(k-2)$;

(b) $\mathsf{horizon}_g(k-2) = k$ iff $\mathsf{horizon}_i(k-2) = k$.

Proof of A.5(a): Since i is nonfaulty, we have that $i,g \in G_x(k-1)$, and thus the round k messages of both g and i are received by x. Assume, by way of contradiction, that $G_g(k-2) \neq G_i(k-2)$. Thus, there are two distinct cases:

(1) $\mathbf{G_g(k-2) \setminus G_i(k-2) \neq \emptyset}$: Let $z \in G_g(k-2) \setminus G_i(k-2)$. By the definition of $G_g(k-2)$, since $z \in G_g(k-2)$, we have that $z \notin B_g(k-2)$. Recall that $B_x(k-1)$ is

computed according to the distributed knowledge of processes in $G_x(k-1)$. Since $i,g \in G_x(k-1)$, we have that $B_x(k-1)$ includes all processes known to either i or g at time $k-1$ to be bad. Thus $B_g(k-2) \subseteq B_x(k-1)$ and $z \in B_x(k-1)$. Now since $z \notin B_g(k-2)$, we have that $B_x(k-1) = B_g(k-2) \cup \{z\}$. Since $\mathsf{horizon}_g(k-2) = k$ by the assumption, we have $\mathsf{horizon}_g(k-2) = (k-2) + t+1 - b_g(k-2) = k$, and thus $b_g(k-2) = t-1$. Hence we have that $b_x(k-1) = b_g(k-2)+1 = (t-1)+1 = t$. It now follows that $\mathsf{horizon}_x(k-1) = (k-1)+t+1-b_x(k-1) = k$, contradicting the assumption that $\mathsf{horizon}_x(k-1) \neq k$.

(2) $\mathbf{G_i(k-2) \setminus G_g(k-2) \neq \emptyset}$: Let $z \in G_i(k-2) \setminus G_g(k-2)$. In this case, too, we distinguish two distinct cases. If $z \notin B_g(k-2)$, then this case is very similar to (1), and thus by using essentially the same argument, again we reach a contradiction. On the other hand, assume $z \in B_g(k-2)$. Thus z's faulty behavior is distributed knowledge among the processes in $G_g(k-2)$, and thus at time $k-2$ some process $z' \in G_g(k-2)$ knows that z is faulty. It is easy to see that z' fails to send its messages to i in round $k-1$, since had it successfully sent to i, process i would have known that z is faulty, causing $z \notin G_i(k-2)$. Since z' fails to send to i in round $k-1$, we have $z' \notin G_i(k-2)$. Since $z' \in G_g(k-2)$, by the definition of $G_g(k-2)$ we have that $z' \notin B_g(k-2)$. Again, as in (1), Since $i,g \in G_x(k-1)$, we have that $B_x(k-1)$ includes all processes known to either i or g at time $k-1$ to be bad. Thus $B_x(k-1) = B_g(k-2) \cup \{z'\}$, and again we have $b_x(k-1) = t$, which implies that $\mathsf{horizon}_x(k-1) = (k-1)+t+1-b_x(k-1) = k$, contradicting the assumption that $\mathsf{horizon}_x(k-1) \neq k$.

Proof of A.5(b): For part (b), our proof consists of two parts:

(1) We first assume that $\mathsf{horizon}_g(k-2) = k$ and show that $\mathsf{horizon}_i(k-2) = \mathsf{horizon}_g(k-2)$. Assume, by way of contradiction, that $\mathsf{horizon}_i(k-2) \neq \mathsf{horizon}_g(k-2)$, i.e., either $\mathsf{horizon}_i(k-2) < \mathsf{horizon}_g(k-2)$ or $\mathsf{horizon}_i(k-2) > \mathsf{horizon}_g(k-2)$. We shall handle each of the two cases separately.

$\textbf{horizon}_i(k-2) < \textbf{horizon}_g(k-2)$**:** By our assumption for part (b) we have that $\mathsf{horizon}_g(k-2) = k$, and thus if $\mathsf{horizon}_i(k-2) < \mathsf{horizon}_g(k-2)$, then it must be the case that $\mathsf{horizon}_i(k-2) \leq k-1$. By definition of horizon, and since the number of bad processes is bounded by t, we have $\mathsf{horizon}_i(k-2) = (k-2) + t + 1 - b_i(k-2) \geq (k-2) + t + 1 - t = k-2$. Since both $\mathsf{horizon}_i(k-2) \leq k-1$ and $\mathsf{horizon}_i(k-2) \geq k-1$, we have that $\mathsf{horizon}_i(k-2) = k-1$. Since by definition $\mathsf{horizon}_i(k-2) = (k-2) + t + 1 - b_i(k-2)$, we have that $b_i(k-2) = t$. From the definition of $B_i(k-2)$, we have that $B_i(k-2) \subseteq B_i(k-1)$, and thus $b_i(k-2) \leq b_i(k-1)$. It follows that $b_i(k-1) = t$, and thus $\mathsf{horizon}_i(k-1) = k$. By Lemma A.4(b) we obtain that $\mathsf{horizon}_x(k-1) = k$, which contradicts the assumption that $\mathsf{horizon}_x(k-1) \neq k$.

$\textbf{horizon}_i(k-2) > \textbf{horizon}_g(k-2)$**:** By the definition of horizon, it follows that $b_i(k-2) < b_g(k-2)$. It then follows that $B_i(k-2) \neq B_g(k-2)$, and thus from the definitions of $B_i(k-2)$ and $B_g(k-2)$ we have that $G_i(k-2) \neq G_g(k-2)$, although $\mathsf{horizon}_g(k-2) = k$ by the assumption, which contradicts part (a). We have shown that if $\mathsf{horizon}_g(k-2) = k$ then $\mathsf{horizon}_i(k-2) = k$.

(2) For the second half of the proof, we assume that $\mathsf{horizon}_i(k-2) = k$, and we have to show that $\mathsf{horizon}_i(k-2) = \mathsf{horizon}_g(k-2)$. Again, we assume by way of contradiction that $\mathsf{horizon}_i(k-2) \neq \mathsf{horizon}_g(k-2)$, and distinguish the two possible cases.

$\textbf{horizon}_i(k-2) > \textbf{horizon}_g(k-2)$**:** The proof is similar to the *first* half of (1).

$\textbf{horizon}_i(k-2) < \textbf{horizon}_g(k-2)$**:** By the assumption we have that $\mathsf{horizon}_i(k-2) = k$, and thus it is easy to see from the definition of horizon we have that $b_i(k-2) = t-1$. Since $\mathsf{horizon}_i(k-2) < \mathsf{horizon}_g(k-2)$, by the definition of horizon we have that $b_i(k-2) > b_g(k-2)$. Hence there exists a process $z \in B_i(k-2) \setminus$

$B_g(k-2)$. Thus z's faulty behavior is distributed knowledge among the processes in $G_i(k-2)$, and thus at time $k-2$ some process $z' \in G_i(k-2)$ knows that z is faulty. Since $z \notin B_g(k-2)$, it follows that $z' \notin G_g(k-2)$, since otherwise the faulty behavior of z would be distributed knowledge among $G_g(k-2)$, causing $z \in B_g(k-2)$. Since $z' \in G_i(k-2)$, by the definition of $G_i(k-2)$ we have that $z' \notin B_i(k-2)$. Since $i, g \in G_x(k-1)$, we have that $B_x(k-1)$ includes all processes known to either i or g at time $k-1$ to be bad. As we have seen, $z' \notin B_i(k-2)$, and thus $B_x(k-1) = B_g(k-2) \cup \{z'\}$. It follows that $b_x(k-1) = b_i(k-2)+1 = (t-1)+1 = t$, which implies that $\mathsf{horizon}_x(k-1) = (k-1)+t+1-b_x(k-1) = k$, contradicting the assumption that $\mathsf{horizon}_x(k-1) \neq k$.

■

We are now ready to prove that the critical time $crit^{\mathsf{u}}_x(k)$ chosen in UNICONCON by an arbitrary process x is the same as the corresponding time $crit_i(k)$ that is chosen in the same FIP execution by a nonfaulty process in the CONCON protocol.

Lemma A.6 Let i be a nonfaulty process in r, let x be an arbitrary process, let $k \geq 1$, and let $g \in G_x(k-1)$. If $\mathsf{horizon}_x(k-1) \neq k$ and $\mathsf{horizon}_g(k-2) \neq k$, then $crit^{\mathsf{u}}_x(k) = crit_i(k)$.

Proof: Let $m = crit_i(k)$. By lines 3 and 4 of CONCON, it follows that $k = \mathsf{horizon}_i(m)$. By Lemma A.5(b), $\mathsf{horizon}_{g'}(k-2) = k$ iff $\mathsf{horizon}_i(k-2) = k$ for all $g' \in G_x(k-1)$. Hence, since $\mathsf{horizon}_x(k-1) \neq k$ and $\mathsf{horizon}_g(k-2) \neq k$, we have that $crit^{\mathsf{u}}_x(k)$ is assigned $Latest^{\mathsf{u}}_x[k]$ in the third case of line 5 in UNICONCON. By line 5 of the protocol, we thus have that $crit^{\mathsf{u}}_x(k) \leq k-3$. Also notice that $\mathsf{horizon}_i(k-1) \neq k$ and $\mathsf{horizon}_i(k-2) \neq k$ by Lemmas A.4(b) and A.5(b) respectively, and thus $crit_i(k) \leq k-3$.

We claim that $crit^{\mathsf{u}}_x(k) = crit_i(k)$. Assume, by way of contradiction, that $crit^{\mathsf{u}}_x(k) \neq crit_i(k)$. We consider two distinct cases.

- First, suppose that $crit^{\mathsf{u}}_x(k) < crit_i(k)$. We claim that $\mathsf{horizon}_g(m) = \mathsf{horizon}_i(m)$ for all $g \in G_x(m+2)$. Assume, again by way of contradiction that $\mathsf{horizon}_g(m) \neq$

$\mathsf{horizon}_i(m)$. Then we have by Lemma A.3 that $\mathsf{horizon}_i(m+1) \leq \mathsf{horizon}_i(m)$. By assumption, $m = crit_i(k)$, and in particular $\mathsf{horizon}_i(m) = k$. It follows that $\mathsf{horizon}_i(m+1) \leq k$. By Proposition 3.6(a) we have that $crit_i(k) \geq m+1$, which contradicts the assumption that $m \triangleq crit_i(k)$. We have proved that $\mathsf{horizon}_g(m) = k$ for all $g \in G_x(m+2)$, and thus in round $m+3$ when process x executes line 4 of UNICONCON, it assigns $Latest^{\mathsf{u}}_x[k]$ the value m. Notice that $Latest^{\mathsf{u}}_x[k]$ may be updated again on line 4 of the protocol at a later round $\ell > m+3$, but it can then be assigned only values $\ell - 3 > m$. It follows that $Latest^{\mathsf{u}}_x[k] \geq m$. Recall that we have shown that $crit_i(k) \leq k-3$, which implies that $k \geq m+3$. Since $Latest^{\mathsf{u}}_x[k] \geq m$ holds at time $m+3$ and at any later time, it follows that $crit^{\mathsf{u}}_x(k) \geq m = crit_i(k)$, contradicting the assumption that $crit^{\mathsf{u}}_x(k) < crit_i(k)$.

- Now suppose that $crit^{\mathsf{u}}_x(k) > crit_i(k)$. Let $crit^{\mathsf{u}}_x(k) = m^{\mathsf{u}} > -1$. Thus by lines 4 and 5 of UNICONCON for some $g \in G_x(m^{\mathsf{u}}+2)$ we have that $\mathsf{horizon}_g(m^{\mathsf{u}}) = k$. We shall prove that $\mathsf{horizon}_g(m^{\mathsf{u}}) = \mathsf{horizon}_i(m^{\mathsf{u}})$. Assume by way of contradiction that $\mathsf{horizon}_g(m^{\mathsf{u}}) \neq \mathsf{horizon}_i(m^{\mathsf{u}})$ then by Lemma A.3 we have $\mathsf{horizon}_i(m^{\mathsf{u}}+1) \leq \mathsf{horizon}_g(m^{\mathsf{u}}) = k$. Thus by Proposition 3.6(a), we have that $crit^{\mathsf{u}}_x(k) \geq m^{\mathsf{u}}+1$, which is in contradiction to the fact that $crit^{\mathsf{u}}_x(k) = m^{\mathsf{u}}$. We have shown that $\mathsf{horizon}_g(m^{\mathsf{u}}) = \mathsf{horizon}_i(m^{\mathsf{u}})$, i.e., that $\mathsf{horizon}_i(m^{\mathsf{u}}) = k$. By Proposition 3.6(b) we obtain $crit_i(k) \geq m^{\mathsf{u}} = crit^{\mathsf{u}}_x(k)$, contradicting the assumption that $crit^{\mathsf{u}}_x(k) > crit_i(k)$, and we are done.

■

We can now finally prove the correctness of UNICONCON:

Proof of Theorem 4.2: We first argue that process x has access to all of the data necessary to carry out the actions specified in every line of UNICONCON. Since knowledge is defined with respect to the fixed system of runs of the FIP, process x can determine the identity of the members of $G_x(k-1)$ at time k. Hence, x can perform the choice of $g \in G_x(k-1)$ on line 2. Moreover, observe that for every $g' \in G_x(k-1)$, at time k process x has a copy of $\mathsf{V}_{g'}(k')$ for all $k' \leq k-1$. It follows that x can compute $\mathsf{horizon}_g(k-3)$ used on line 4, as well as $\mathsf{horizon}_g(k-2)$ used in the middle case of line 5. Finally, it can compute $\mathcal{E}(\mathsf{V}_{G_x(k-1)}(k-1))$ and $\mathcal{E}(\mathsf{V}_{G_g(c^{\mathsf{u}})}(c^{\mathsf{u}}))$ when $c^{\mathsf{u}} < k-1$, and thus is able to perform the commands on line 6.

It remains to show that $crit^{\mathsf{u}}_x(k) = crit_i(k)$ and that $G_i(c) = G_g(c)$ for all $g \in G_x(k-1)$. From line 6 in UNICONCON and line 5 in CONCON, it will then follow that $M^{\mathsf{u}}_x[k] = M_i[k]$. Notice that line 5 of UNICONCON assigns one of three values to $crit^{\mathsf{u}}_x(k)$. Thus in our proof we handle the three possible assignments on line 5 as three distinct cases. Theses cases are indeed distinct since each of the first two cases in the assignment on line 5 implies that the value of $crit^{\mathsf{u}}_x(k)$ is $k-1$ or $k-2$ respectively, while the third case on line 5 assigns $Latest^{\mathsf{u}}_x[k]$ to $crit^{\mathsf{u}}_x(k)$. Notice that $Latest^{\mathsf{u}}_x[k]$ is last updated in some round $m \leq k$ on line 4 of UNICONCON, and thus the value assigned to $Latest^{\mathsf{u}}_x[k]$ was $m-3$. Thus $Latest^{\mathsf{u}}_x[k] = m-3 \leq k-3$, and thus in the third case of line 5 we necessarily have $crit^{\mathsf{u}}_x(k) \leq k-3$.

$crit^{\mathsf{u}}_x(k) = k-1$: By line 5 of UNICONCON we have that $\mathsf{horizon}_x(k-1) = k$. Thus, by Lemma A.4(b) we also have that $\mathsf{horizon}_i(k-1) = k$, and hence $crit_i(k) = crit^{\mathsf{u}}_x(k) = k-1$. By Lemma A.4(a) we have that $G_z(k-1) = G_i(k-1)$ for all $z \in \mathbb{P}$, and in particular $G_x(k-1) = G_i(k-1)$. It follows that $M^{\mathsf{u}}_x[k] = M_i[k]$.

$crit^{\mathsf{u}}_x(k) = k-2$: Since $crit^{\mathsf{u}}_x(k) = k-2$, by line 5 of UNICONCON we have that $\mathsf{horizon}_g(k-2) = k$ for some $g \in G_x(k-1)$, and also that $\mathsf{horizon}_x(k-1) \neq k$. By Lemma A.5(b) we have that $\mathsf{horizon}_i(k-2) = k$, and by Lemma A.4(b) we have that $\mathsf{horizon}_i(k-1) \neq k$. Thus, in round $k-1$ on line 3 of CONCON process i assigns $k-2$ to $Latest_i[k]$, and $Latest_i[k]$ is not updated in round k. It follows that $crit_i(k) = k-2$. By Lemma A.5(a) we have that $G_g(k-2) = G_i(k-2)$, and we again obtain that $M^{\mathsf{u}}_x[k] = M_i[k]$.

$\boldsymbol{crit^{\sqcup}_x(k) < k-2}$**:** This case applies only if $k \geq 2$. By Line 5 of UNICONCON, in this case we have that $crit^{\sqcup}_x(k) = Latest^{\sqcup}_x[k]$. Moreover, by Lemma A.6 we have that $crit^{\sqcup}_x(k) = crit_i(k)$. Let $c \triangleq crit^{\sqcup}_x(k) = crit_i(k)$. It remains to show that $G_i(c) = G_g(c)$. Set $g \in G_x(k-1)$. Assume by way of contradiction that $G_i(c) \neq G_g(c)$. First we claim that $b_i(c+1) > b_i(c)$. Notice that in round $c+2$ both i and g receive each other's messages, since $g \in G_x(c+2)$, and i is nonfaulty. If there exists a $p \in G_i(c)$ such that $p \notin G_g(c)$, then process i learns of p's failure from g in round $c+2$, and moreover, $p \in B_i(c+1)$, and thus $b_i(c+1) > b_i(c)$. On the other hand, if there exists a $p \notin G_i(c)$ such that $p \in G_g(c)$, then it must be the case that i learns of p's faulty behavior in round $c+1$, and thus $p \notin B_i(c)$. However, $p \in B_i(c+1)$, and thus again $b_i(c+1) > b_i(c)$. Thus, in all cases $b_i(c+1) > b_i(c)$, and by the definition of $\mathsf{horizon}$ we have that $\mathsf{horizon}_i(c+1) \leq \mathsf{horizon}_i(c)$. Thus, by the definition of c, we have that $\mathsf{horizon}_i(c+1) \leq k$. Now, from Proposition 3.6(a) we obtain that $crit_i(k) \geq c+1$, i.e., that $c \geq c+1$, which contradicts our assumption, and we are done with the third case.

$\blacksquare$