Distributed Computing (2022) 35:123–143
https://doi.org/10.1007/s00446-021-00417-3

Unbeatable consensus

Armando Castañeda[1] · **Yannai A. Gonczarowski**[2] · **Yoram Moses**[3]

Received: 19 September 2019 / Accepted: 20 December 2021 / Published online: 12 January 2022

Abstract
The *unbeatability* of a consensus protocol, introduced by Halpern et al. (SIAM J Comput 31:838–865, 2001), is a stronger notion of optimality than the accepted notion of early stopping protocols. Using a novel knowledge-based analysis, this paper derives the first explicit unbeatable consensus protocols in the literature, for the standard synchronous message-passing model with crash failures. These protocols strictly dominate the best kno-wn protocols for uniform and for Nonuniform Consensus, in some cases improving on them by a large margin. The analysis provides a new understanding of the logical structure of consensus, and of the distinction between uniform and nonuniform Consensus. All protocols presented in this paper have very concise descriptions, and are shown to be efficiently implementable.

Keywords Consensus · Uniform consensus · Optimality · Knowledge

1 Introduction

Following [18], we say that a protocol P is a *worst-case optimal* solution to a decision task S in a given model if it solves S, and decisions in P are always taken no later than the *worst-case* lower bound for decisions in this problem, in that model. Here we consider standard synchronous message-passing models with n processes and an *a priori* given bound of at most $\boldsymbol{t} < n$ crash failures per run; it will be convenient to denote the number of *actual* failures in a given run by $\boldsymbol{f}$. Processes proceed in a sequence of synchronous rounds. The very first consensus protocols were worst-case optimal, deciding and stopping at the end of exactly $\boldsymbol{t}+1$ rounds in all runs [8,22]. It was soon realized, however, that they could be strictly improved upon by *early stopping* protocols [7], which are guaranteed to halt in $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds. Such protocols are also worst-case optimal, but can often decide and stop much faster than the original ones. This paper presents a number of consensus protocols that are not only worst-case optimal and early stopping, but furthermore cannot be strictly improved upon, and are thus optimal in a much stronger sense.

✉ Armando Castañeda
armando.castaneda@im.unam.mx

Yannai A. Gonczarowski
yannai@gonch.name

Yoram Moses
moses@ee.technion.ac.il

[1] Instituto de Matemáticas, UNAM, Mexico City, Mexico

[2] Department of Economics and Department of Computer Science, Harvard University, Cambridge, USA

[3] Electrical Engineering Department, Technion, Haifa, Israel

1.1 Unbeatability

In benign failure models it is typically possible to define the behaviour of the environment in a manner that is independent of the protocol, in terms of a pair $\alpha = (\mathbf{v}, \mathsf{F})$ consisting of a vector $\mathbf{v}$ of initial values and a failure pattern F that specifies when and how processes crash (a formal definition is given in Sect. 3). A **failure model** $\mathcal{F}$ is identified with a set of (possible) failure patterns. For ease of exposition, we will think of such a pair $\alpha = (\mathbf{v}, \mathsf{F})$ as a particular *adversary*. In a synchronous environment, a deterministic protocol P and an adversary α uniquely define a *run* $r = P[\alpha]$. With this terminology, we can compare the performance of different *decision* protocols (later we deal with *stopping* ones) that solve a particular task in a given *context* $\gamma = (\mathbb{V}^n, \mathcal{F})$, where $\mathbb{V}^n$ is a set of possible vectors of initial values and $\mathcal{F}$ is a failure model. A decision protocol Q *dominates* a protocol P in γ, denoted by $Q \preceq_{\boldsymbol{\gamma}} P$ if, for all adversaries $\alpha = (\mathbf{v}, \mathsf{F})$ in the context γ (namely, $\mathbf{v} \in \mathbb{V}^n$ and $\mathsf{F} \in \mathcal{F}$) and every process i, if i decides in $P[\alpha]$ at time m_i, then i decides in $Q[\alpha]$ at some time $m_i' \leq m_i$. Moreover, we say that Q *strictly dominates* P if $Q \preceq_\gamma P$ and $P \not\preceq_\gamma Q$; i.e., Q dominates P and for some

adversary α in γ there exists a process i that decides in $Q[\alpha]$ *strictly before* it does so in $P[\alpha]$.

In the crash failure model, the early-stopping protocols of [7] strictly dominate the original protocols of [22], in which decisions are always performed at time $\boldsymbol{t}+1$. Nevertheless, these early stopping protocols may not be optimal solutions to consensus. Following [18], we call a protocol P solving a decision task S in a context γ *all-case optimal* if it dominates every protocol P' that solves S in γ. Dwork and Moses presented all-case optimal solutions to the *simultaneous* variant of consensus [10], in which all decisions are required to be made at the same time. For the standard *eventual* variant of consensus, in which decisions are not required to occur simultaneously, Moses and Tuttle showed that no all-case optimal solution exists [21]. Consequently, Halpern, Moses and Waarts in [17] initiated the study of a natural notion of optimality that is achievable by eventual consensus protocols:

Definition 1 (Unbeatability [17][1]) A protocol P is an *unbeatable* solution to a decision task S in a context γ if P solves S in γ and no protocol Q solving S in γ strictly dominates P.

In [17], Halpern, Moses and Waarts presented a two-step transformation that, given a binary consensus protocol P defines an unbeatable protocol Q_P that dominates P. This transformation and the resulting protocols are based on a notion of *continual* common knowledge that is computable, but not efficiently: in the resulting protocol, each process executes exponential time (PSPACE) local computations in every round. Their transformation is not applied in [17] to an actual protocol. As an example of an unbeatable protocol, they present a particular protocol, called $P0_{\text{opt}}$, and argue that it is unbeatable in the crash failure model. Unfortunately, as we will show, $P0_{\text{opt}}$ is in fact beatable. This does not refute the general analysis and transformation defined in [17]; they remain correct. Rather, the fault is in an unsound step in the proof of optimality of $P0_{\text{opt}}$ (Theorem 6.2 of [17]), in which an inductive step is not explicitly detailed, and does not hold.

1.2 Contributions

The main contributions of this paper are:

1. A knowledge-based analysis is applied to the classic binary consensus problem, and is shown to yield unbeatable solutions that are optimal in a much stronger sense than all previous solutions. Considerably simpler and more intuitive than the framework used in [17], it illustrates how the knowledge-based approach can yield a structured methodology for the derivation of efficient protocols.
2. Opt_0, the first explicit unbeatable protocol for *nonuniform* consensus (only correct processes are required to reach agreement) is presented. It is computationally efficient, and its unbeatability is established by way of a succinct proof. Moreover, Opt_0 is shown to strictly dominate the $P0_{\text{opt}}$ protocol from [17], proving that the latter is in fact beatable.
3. An analysis of *uniform* consensus (in which all decisions, including those made by processes that later crash, must be in agreement) gives rise to $u-Opt_0$, the first explicit unbeatable protocol for Uniform Consensus. The analysis used in the design of $u-Opt_0$ sheds light on the inherent difference and similarities between the uniform and nonuniform variants of consensus in this model.
4. We identify the notion of a *hidden path* as being crucial to decide in the consensus task. If a process identifies that no hidden path exists, then it can decide. In the fastest early-stopping protocols preceeding this work, a process decides after the first round in which it does not detect a new failure. Our unbeatable protocols, in which a process decides when it detects that no hidden path exists, can in some cases stop after a small constant number of rounds, compared to $\boldsymbol{t}+1$ rounds required by the best early stopping protocols in the literature.

For ease of exposition and analysis, all of our protocols are full-information, i.e., every process sends all the information it is aware of, in every round. However, in fact, they can all be efficiently implemented in such a way that the total number of bits that any given process sends to any other process throughout an execution is $O(n \log n)$, in the worst case.

Similarly, for simplicity, we focus first on *decision* protocols, namely, protocols that only specify when a process decides. Thus, formally, in our algorithms correct processes run forever. However, in all our solutions processes can safely stop in at most $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds where $\boldsymbol{f}$ denotes the actual number of failures in an execution.

1.3 Roadmap

The rest of the paper is structured as follows. Section 2 first sketches the intuition behind our knowledge-based analysis for deriving unbeatable consensus protocols. Section 3 reviews the definitions of the synchronous crash-failure model and of knowledge in this model. Section 4 presents Opt_0, our unbeatable consensus protocol, proves its unbeatability, and shows that it beats the protocol $P0_{\text{opt}}$ of [17]. Section 5 studies Uniform Consensus, and derives $u-Opt_0$, an unbeatable protocol in which processes decide

[1] All-case optimal protocols are called "*optimal in all runs*" in [10]. They are termed "*optim**um***" in [17], while unbeatable protocols are simply called "*optim**al***" there. We prefer the term *unbeatable* because "optimal" is used very broadly, and inconsistently, in the literature.

in at most $\min(f+2, t+1)$ rounds. Then, Sect. 6 shows that all our protocol can be efficiently implemented and that in each of them a process can stop in at most $\min(f+2, t+1)$ rounds. Section 7 introduces an alternative notion of unbeatability that is incomparable to the usual one, and shows that all our algorithms are unbeatable with respect to this notion as well. Finally, Sect. 8 concludes with a final discussion. For ease of exposition, some proofs are presented in the Appendix.

2 Intuition

We start with a discussion of the intuition behind our unbeatable consensus protocols. In the standard version of binary consensus, every process i starts with an initial value $v_i \in \{0, 1\}$, and the following properties must hold in every run r: (Nonuniform) *Consensus*

- *Decision:* Every correct process must decide on some value,
- *Validity:* If a correct process decides v, then v is the input of at least one process, and
- *Agreement:* All correct processes decide on the same value.

The connection between knowledge and distributed computing was proposed in [16] and has been used in the analysis of a variety of problems, including consensus (see [11] for more details and references). In this paper, we employ simpler techniques to perform a more direct knowledge-based analysis. Our approach is based on a simple principle recently formulated by Moses in [20], called the *knowledge of preconditions* principle (**K***o***P**), which captures an essential connection between knowledge and action in distributed and multi-agent systems. Roughly speaking, the **K***o***P** principle says that if C is a necessary condition for an action α to be performed by process i, then $K_i(C)$—i knowing C—is a necessary condition for i performing α. E.g., it is not enough for a client to have positive credit in order to receive cash from an ATM; the ATM must *know* that the client has positive credit.

Problem specifications typically state or imply a variety of necessary conditions. In the crash failure model studied in this paper, informally, we will say that a process is *active* at time m in a given run, if it does not crash before time m. For $v \in \{0, 1\}$, we denote by $\mathsf{decide}_i(v)$ the action of i deciding v, and use $\bar{v}$ as shorthand for $1 - v$.

Lemma 1 *Consensus implies the following necessary conditions for* $\mathsf{decide}_i(v)$ *in the crash failure model:*

(a) "at least one processes had initial value v*" (we denote this by* $\exists v$*), and*

(b) "no currently active process has decided, or is currently deciding, $\bar{v}$*" (denoted by* $\mathsf{none\_decided}(\bar{v})$*).*

Both parts follow from observing that if i decides v at a point where either (a) or (b) does not hold, then the execution can be extended to a run in which i is correct (does not crash), as well as another j for (b), and this run violates *Validity* for (a) or *Agreement* for (b).

Given Lemma 1, **K***o***P** implies that both $K_i \exists v$ and $K_i \mathsf{none\_decided}(\bar{v})$ are also necessary conditions for the action $\mathsf{decide}_i(v)$ to be performed. In this paper, we will explore how this insight can be exploited in order to design efficient consensus protocols. Indeed, our first unbeatable protocol will be one in which, roughly speaking, the rule for $\mathsf{decide}_i(0)$ will be $K_i \exists 0$, and the rule for $\mathsf{decide}_i(1)$ will be $K_i \mathsf{none\_decided}(0)$. As we will show, if the rule for $\mathsf{decide}_i(0)$ is $K_i \exists 0$, then $\mathsf{none\_decided}(0)$ reduces to the fact $\mathsf{not\_known}(\exists 0)$, which is true at a given time if $K_j \exists 0$ holds for no currently-active process j. Thus, $K_i \mathsf{none\_decided}(0)$—our candidate rule for deciding 1—then becomes $K_i \mathsf{not\_known}(\exists 0)$. While $K_i \exists 0$ involves the knowledge a process has about initial values, $K_i \mathsf{not\_known}(\exists 0)$ is concerned with i's knowledge about the knowledge of others.

Converting the above description into an actual protocol essentially amounts to providing concrete tests for when these knowledge conditions hold. It is straightforward to show (and quite intuitive) that in a full-information protocol $K_i \exists 0$ holds exactly if there is a message chain from some process j whose initial value is 0, to process i. To determine that $K_i \mathsf{not\_known}(\exists 0)$, a process must have proof that no such chain, leading to any process i' can exist at the current time. Our technical analysis identifies a notion of a *hidden path* with respect to i at a time m, which implies that a message chain could potentially be communicating a value unbeknownst to i at time m. It is shown that hidden paths are key to evaluating whether $K_i \mathsf{not\_known}(\exists 0)$ holds. We will review the formal definition of knowledge in the next section, in order to turn this into a rigorous condition.

It turns out that hidden paths are key to obtaining additional unbeatable protocols in the crash failure model. We present an unbeatable protocol for the *uniform* variant of consensus [3,9,15,19,23,24] in which we replace the *Agreement* condition of consensus by:

> *Uniform Agreement:* The processes that decide in a given run must all decide on the same value.

This forces correct processes and faulty ones to decide in a consistent manner. Requiring uniformity makes sense only in a setting where failures are benign, and all processes that decide do so according to the protocol. Uniformity may be desirable when elements outside the system can observe deci-

sions, as in distributed databases when decisions correspond to commitments to values.

3 Preliminary Definitions

3.1 Model of Computation

Our model of computation is the standard synchronous message-passing model with benign crash failures. A system has $n \geq 2$ processes, denoted $\mathsf{Procs} = \{1, \ldots, n\}$. Each pair of processes is connected by a two-way communication link, and each message is tagged with the identity of the sender. Processes 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 synchronous *rounds*, with round m taking place between time $m-1$ and time m. Each process starts in some *initial state* at time 0, usually with an *input value* of some kind. In every round, each process first performs a local computation, and performs local actions (e.g. deciding on a value), then it sends a set of messages to other processes, and finally receives messages sent to it by other processes during the same round. We consider the local computations and sending actions of round m as being performed at time $m-1$, and the messages are received at time m.

A faulty process fails by *crashing* in some round $m \geq 1$. It behaves correctly in the first $m-1$ rounds and sends no messages from round $m+1$ on. During its crashing round m, the process can crash at any point during that round, hence it may succeed in sending only a subset of the messages in round m; the subset might be empty (e.g. if the process crashes during its local computation). At most $1 \leq \boldsymbol{t} \leq n-1$ processes fail in any given execution. The actual number of failures in a given execution is denoted $0 \leq \boldsymbol{f} \leq \boldsymbol{t}$.

It is convenient to consider the state and behaviour of processes at different (process,time) nodes, where a *node* is a pair $\langle i, m\rangle$ referring to process i at time m. A *failure pattern* is a tuple $(\mathsf{F}, \mathsf{fail})$ that describes how processes fail in an execution. It has a layered graph F whose vertices are nodes $\langle i, m\rangle$ for $i \in \mathsf{Procs}$ and $m \geq 0$. Such a vertex denotes process i and time m. An edge has the form $(\langle i, m-1\rangle, \langle j, m\rangle)$ and it denotes the fact that a message sent by i to j in round m would be delivered successfully. The function fail indicates the exact time a faulty process crashes and whether it completes all its local actions before crashing (hence possibly making a decision). More specifically, for a faulty process i, $\mathsf{fail}(i)$ is a pair (m, b) meaning that i crashes at time m, and it either executes all its local actions before crashing, when $b = \mathsf{true}$, or executes no local action at all before crashing, when $b = \mathsf{false}$. For every correct process i, $\mathsf{fail}(i) = \perp$. F and fail must be consistent: if $\mathsf{fail}(i) = (m, b)$, then in F, i sends all its messages in the first $m-1$ rounds, sends some of its messages in round m and sends no messages in the next rounds.

Let $\mathsf{Crash}(\boldsymbol{t})$ denote the set of failure patterns in which all failures are crash failures, and no more than $\boldsymbol{t}$ crash failures occur. An *input vector* describes the initial values that the processes receive in an execution. The only inputs we consider are initial values that processes obtain at time 0. An input vector is thus a tuple $\mathbf{v} = (v_1, \ldots, v_n)$ where v_j is the input to (or, alternatively, the initial value of) process j. We think of the input vector and the failure pattern as being determined by an external scheduler, and thus a pair $\alpha = (\mathbf{v}, (\mathsf{F}, \mathsf{fail}))$ is called an *adversary*.

A *protocol* describes what messages a process sends and what decisions it takes (if it takes a decision), as a deterministic function of its local state at the start of a round and the messages received in the previous round. We assume that a protocol P has access to the values of n and $\boldsymbol{t}$, typically passed to P as parameters.

A *run* is a description of an infinite behaviour of the system. Given a run r and a time m, we denote by $r_i(m)$ the *local state* of process i at time m in r, namely, the state of the process after receiving all messages sent to it in round m, and performing the local computation (which may include taking a decision) and sending its messages (if there are any) corresponding to round $m+1$; if $m = 0$, the process receives a single message with its input in a fictitious round 0. The *global state* at time m is defined to be $r(m) = \langle r_1(m), r_2(m), \ldots, r_n(m)\rangle$. A protocol P and an adversary α uniquely determine a run, and we write $r = P[\alpha]$.

Since we restrict our attention to benign failure mo-dels and focus on decision times and solvability in this paper, it is sufficient to consider *full-information* protocols [4] (*fip*'s for short), defined below. Informally speaking, every process in a full-information protocol sends all information it has seen so far, in every round. There is a convenient way to consider such protocols in our setting. With an adversary $\alpha = (\mathbf{v}, (\mathsf{F}, \mathsf{fail}))$, we associate a *communication graph* $\mathcal{G}_\alpha$, consisting of the graph F extended by labelling the initial nodes $\langle j, 0\rangle$ with the initial states v_j according to $\mathbf{v}$. Every node $\langle i, m\rangle$ is associated with a subgraph $\mathcal{G}_\alpha(i, m)$ of $\mathcal{G}_\alpha$, which we think of as i's *view* at $\langle i, m\rangle$. Intuitively, this graph will represent all nodes $\langle j, \ell\rangle$ from which $\langle i, m\rangle$ has heard, and the initial values it has seen. Formally, $\mathcal{G}_\alpha(i, m)$ is defined by induction on m. $\mathcal{G}_\alpha(i, 0)$ consists of the node $\langle i, 0\rangle$, labelled by the initial value v_i. Assume that $\mathcal{G}_\alpha(1, m-1), \ldots, \mathcal{G}_\alpha(n, m-1)$ have been defined, and let $J \subseteq \mathsf{Procs}$ be the set of processes j such that $j = i$ or $e_j = (\langle j, m-1\rangle, \langle i, m\rangle)$ is an edge of F. Namely, J consists of the processes i receives messages from at the end of round m. Then, $\mathcal{G}_\alpha(i, m)$ consists of the node $\langle i, m\rangle$, the union of all graphs $\mathcal{G}_\alpha(j, m-1)$ with $j \in J$, and the edges $e_j = (\langle j, m-1\rangle, \langle i, m\rangle)$ for all $j \in J$. We say that $\langle j, \ell\rangle$ is *seen* by $\langle i, m\rangle$ if $\langle j, \ell\rangle$ is a node of $\mathcal{G}_\alpha(i, m)$.

Note that this occurs exactly if the failure pattern F allows a (Lamport) message chain from $\langle j, \ell \rangle$ to $\langle i, m \rangle$.

A *full-information* protocol P is one in which at every node $\langle i, m \rangle$ of a run $r = P[\alpha]$ with adversary $\alpha = (\mathbf{v}, (\mathsf{F}, \mathsf{fail}))$, the process i constructs $\mathcal{G}_\alpha(i, m)$ after receiving its round m nodes, and then sends $\mathcal{G}_\alpha(i, m)$ to all other processes in round $m + 1$. Moreover, the protocol P specifies what decisions i should take (if there is one) at $\langle i, m \rangle$ based on $\mathcal{G}_\alpha(i, m)$; such decision is taken in round $m + 1$. Full-information protocols thus differ only in the decisions taken at the nodes. Let $\mathsf{d}(i, m)$ be status of i's decision at time m, either '$\perp$' if it is undecided, or a concrete value 'v'. We assume that decisions are irrevocable; namely, once set to a non-$\perp$ value, the decision of a process is never changed: if $\mathsf{d}(i, m) \neq \perp$, then $\mathsf{d}(i, m') = \mathsf{d}(i, m)$, for every $m' > m$. For the smallest time m such that $\mathsf{d}(i, m) \neq \perp$, we must have that, according to α, either i does not crash at time m (i.e., $\mathsf{fail}(i) = \perp$) or it crashes at time m but it completes all its local actions before crashing (i.e., $\mathsf{fail}(i) = (m, \mathsf{true})$). Thus, in a run $r = P[\alpha]$, we define $r_i(m) = \langle \mathsf{d}(i, m), \mathcal{G}_\alpha(i, m) \rangle$ if i does not crash at time m, while if i crashes at time $m' \leq m$, $r_i(m) = \langle \mathsf{d}(i, m'), \mathcal{G}_\alpha(i, m'), \frown \rangle$ where $\frown$ indicates that i is crashed at time m'.

Finally, we formally define the notion of a process being active used in Sect. 2: given an adversary $\alpha = (\mathbf{v}, (\mathsf{F}, \mathsf{fail}))$, a process i is *active* at time m in the run $r = P[\alpha]$ of a protocol P, if $\mathsf{fail}(i)$ is $\perp$ or (m, true).

3.2 Knowledge

Our construction of unbeatable protocols will be assisted and guided by a knowledge-based analysis, in the spirit of [11, 16]. Runs are dynamic objects, changing from one time point to the next. E.g., at one point process i may be undecided, while at the next it may decide on a value. Similarly, the set of initial values that i knows about, or has seen, may change over time. In general, whether a process "knows" something at a given point can depend on what is true in other runs in which the process has the same information, i.e., runs that are *indistinguishable* to i. We will therefore consider the truth of facts at *points* (r, m)—time m in run r, with respect to a set of runs R, which we call a *system*. We will be interested in systems of the form $R_P = R(P, \gamma)$ where P is a protocol and $\gamma = \gamma(\mathrm{V}^n, \mathcal{F})$ is a *context*, namely, the set of all adversaries that assign initial values from V and failures according to $\mathcal{F}$.

We will write $(R, r, m) \models A$ to state that fact A holds, or is satisfied, at (r, m) in the system R. The truth of some facts can be defined directly. For example, we define the fact $\exists \mathrm{v}$ to hold at the point (r, m) in R if some process has initial value v in $(r, 0)$. We say that (*satisfaction of*) a fact A is *well-defined in* R if for every point (r, m) with $r \in R$ and $m \geq 0$, it can be determined whether or not $(R, r, m) \models A$ from the global state $r(m)$. Fact $\exists \mathrm{v}$ is thus well defined. Moreover, any boolean combination of well-defined facts is also well defined. We will write $K_i A$ to denote that *process i knows A*, and define:

Definition 2 (Knowledge) Suppose that A is well defined in R. Define that $(R, r, m) \models K_i A$ if and only if $(R, r', m) \models A$ holds for all $r' \in R$ with $r_i(m) = r'_i(m)$.

Thus, if A is well defined in R then Definition 2 makes $K_i A$ well defined in R. Note that what a process knows or does not know depends on its local state. The definition can then be applied recursively, to define the truth of $K_j K_i A$ etc.

Knowledge has been used to study a variety of problems in distributed computing. We will make use of the following fundamental connection between knowledge and actions in distributed systems. A fact A is a *necessary condition* for process i performing action σ (e.g. deciding on an output value) in R if $(R, r, m) \models A$ whenever i performs σ at a point (r, m) of R.

Theorem 1 *(Knowledge of Preconditions [20]) Let $R_P = R(P, \gamma)$ be the set of runs of a deterministic protocol P. If A is a necessary condition for i performing σ in R_P, then so is $K_i A$.*

4 Unbeatable consensus

We start with the (nonuniform) version of consensus defined in Sect. 2, and consider the crash failure context $\gamma^t_{\mathrm{cr}} = \langle \mathrm{V}^n, \mathsf{Crash}(t) \rangle$, where $\mathrm{V} = \{0, 1\}$. Every protocol P in this setting determines a system $R_P = R(P, \gamma^t_{\mathrm{cr}})$.

4.1 Deriving an unbeatable protocol

Recall that Lemma 1 in Sect. 2 establishes necessary conditions for decision in consensus (Appendix A contains a proof of the lemma). Based on Lemma 1 and Theorem 1, we obtain:

Lemma 2 *Let P be a consensus protocol for γ^t_{cr} and let $R_P = R(P, \gamma^t_{\mathrm{cr}})$. Then both $K_i \exists \mathrm{v}$ and $K_i \mathsf{none\_decided}(\bar{\mathrm{v}})$ are necessary conditions for $\mathsf{decide}_i(\mathrm{v})$ in R_P.*

An analysis of knowledge for *fips* in the crash failure model was first performed by Dwork and Moses in [10]. The following result is an immediate consequence of that analysis. Under the full-information protocol, processes that survive until time $t + 1$ obtain the same knowledge about $\exists \mathrm{v}$:

Lemma 3 *(Dwork and Moses [10]) Let P be a fip in γ^t_{cr} and let $r \in R_P = R(P, \gamma^t_{\mathrm{cr}})$. For all processes i, j, $(R_P, r, t + 1) \models K_i \exists \mathrm{v}$ iff $(R_P, r, t + 1) \models K_j \exists \mathrm{v}$.*

While Lemma 3 is given and proved in [10], for completeness we reprove it in Appendix A using the notation and machinery of this paper.

Of course, a process that does not know $\exists 0$ must itself have an initial value of 1. Hence, based on Lemma 3, it is natural to design a *fip*-based consensus protocol that performs $\mathsf{decide}_i(0)$ at time $t+1$ if $K_i\exists 0$, and otherwise performs $\mathsf{decide}_i(1)$. (In the very first consensus protocols, all decisions are performed at time $t+1$ [22].) Indeed, one can use Lemma 3 to obtain a strictly better protocol, in which decisions on 0 are performed sooner:

Protocol P_0 (for an undecided process i at time m):

if	$K_i\exists 0$	*then* $\mathsf{decide}_i(0)$
elseif	$m = t+1$	*then* $\mathsf{decide}_i(1)$

Notice that in a *fip* consensus protocol, it is only necessary to describe the rules for $\mathsf{decide}_i(0)$ and $\mathsf{decide}_i(1)$, since in every round a process sends its complete local state to all processes. Since $K_i\exists 0$ is a necessary condition for the action $\mathsf{decide}_i(0)$, the protocol P_0 decides on 0 as soon as any consensus protocol can. In the early 80's Dolev suggested a closely related protocol B (standing for *Beep*) for γ^t_{cr}, in which processes decide 0 and broadcast the existence of a 0 when they see a 0, and decide 1 at $t+1$ otherwise [6]; for all adversaries, it performs the same decisions at the same times as P_0. Halpern, Moses and Waarts show in [17] that for every consensus protocol P in γ^t_{cr} there is an unbeatable consensus protocol Q dominating P. Our immediate goal is to obtain an unbeatable consensus protocol dominating P_0. To this end, we make use of the following.

Lemma 4 *Let Q be a consensus protocol. If $Q \preceq P_0$, then $\mathsf{decide}_i(0)$ is performed in Q exactly when $K_i\exists 0$ first holds, for every process i.*

Proof Assume that $Q \preceq P_0$ solves consensus; w.l.o.g., Q is a *fip* as well. We prove the claim for all processes i and adversaries α, by induction on the time m at which $K_i\exists 0$ first holds in $Q[\alpha]$ (and, equally, in $P_0[\alpha]$).

Base ($m = 0$): As i decides 0 at time 0 in $P_0[\alpha]$, by Lemma 2 we have $K_i\exists 0$ at time 0 in $P_0[\alpha]$ (and so also in $Q[\alpha]$). Since process i at time 0 knows no initial value but its own, it follows that i is assigned an initial value of 0 by α. Hence, $K_i\exists 1$ does *not* hold at time 0. By Lemma 2, i therefore does not decide 1 at time 0 in $Q[\alpha]$. Since i decides at time 0 in $P_0[\alpha]$, it must decide at time 0 in $Q[\alpha]$ as well, and so decides 0, as required.

Inductive step ($m > 0$): Assume that the claim holds for all times $< m$. Recall that m is the first time at which $K_i\exists 0$ holds. In a fip, this can only happen if $K_i\exists 0$ does not hold at time $m' < m$ and i receives at time m a message with a 0 from some process j that is active at time $m-1$. Thus, $K_j\exists 0$ holds at time $m-1$, and by the induction hypothesis, j decides 0 when $K_j\exists 0$ first holds in $Q[\alpha]$ — denote this time by m'; as $K_j\exists 0$ holds at time $m-1$, we have $m' \leq m-1$. Observe that in γ^t_{cr}, if i receives a message from j in round m, then i cannot know that j is faulty at time m; more precisely, denoting by β the adversary that never crashes i nor j at all, and that otherwise agrees with α (this is a legal adversary, as is specifies no more than t crash failures), we have in the run $r' = Q[\beta]$ that *1)* $r'_i(m) = r_i(m)$, *2)* $r'_j(m') = r_j(m')$, and *3)* neither i nor j fail. Since Q satisfies *Agreement*, i cannot decide 1 during $Q[\beta]$, and therefore cannot decide 1 at or before time m during $Q[\alpha]$. Moreover, by Lemma 2, $K_i\exists 0$ is a precondition for process i deciding 0, and so i cannot decide 0 before time m during $Q[\alpha]$. Since Q dominates P_0, we have that i must decide by time m in $Q[\alpha]$, and therefore it decides 0 at m in $Q[\alpha]$. □

We define $(R, r, m) \models \mathsf{not\_known}(\exists 0)$ to hold iff $(R, r, m) \not\models K_j\exists 0$ holds for every process j. [2] We can now formalize the discussion in Sect. 2, showing that if decisions on 0 are performed precisely when $K_i\exists 0$ first holds, then $\mathsf{none\_decided}(0)$ reduces to $\mathsf{not\_known}(\exists 0)$. The proof of Lemma 5 (in Appendix A) is fairly immediate: If $(R_P, r, m) \not\models K_i\mathsf{not\_known}(\exists 0)$ then there is a run r' of R_P such that both $r_i(m) = r'_i(m)$ and $(R_P, r', m) \models K_j\exists 0$ for some correct process j; therefore, process j decides 0 in r'. The other direction follows directly from the decision rule for 0.

Lemma 5 *Let P be a fip in which $\mathsf{decide}_i(0)$ is performed exactly when $K_i\exists 0$ first holds, and let $R_P = R(P, \gamma^t_{\mathrm{cr}})$. Then, for every process i, it is the case that*

$$(R_P, r, m) \models K_i\mathsf{none\_decided}(0)$$

iff

$$(R_P, r, m) \models K_i\mathsf{not\_known}(\exists 0),$$

for all $r \in R_P$ and $m \geq 0$.

We can now define a *fip* consensus protocol in which 0 is decided as soon as its necessary condition $K_i\exists 0$ holds, and 1 is decided as soon as possible, given the rule for deciding 0:

Protocol Opt_0 (for an undecided process i at time m):

if	$K_i\exists 0$	*then* $\mathsf{decide}_i(0)$
elseif	$K_i\mathsf{not\_known}(\exists 0)$	*then* $\mathsf{decide}_i(1)$

We can show that Opt_0 solves consensus and is, indeed, an unbeatable protocol.

[2] Notice that if j crashes before time m in r then $(R, r, m) \not\models K_j\exists 0$ is guaranteed, since the crashed local state ☹ for j appears, in particular, in runs in which $\exists 0$ is not true since all initial values are 1.

Lemma 6 *Let P be a fip in γ_{cr}^{t} and let $r \in R_P = R(P, \gamma_{\mathrm{cr}}^{t})$. For every process i, it is the case that if*

$$(R_P, r, t+1) \not\models K_i \exists 0,$$

then

$$(R_P, r, t+1) \models K_i \mathsf{not\_known}(\exists 0).$$

Proof By Lemma 3, we have that $\neg K_i \exists 0$ at time $t+1$ implies $\mathsf{not\_known}(\exists 0)$ at that time; by definition of knowledge, we therefore have that $K_i(\neg K_i \exists 0 \wedge m = t+1)$ implies $K_i \mathsf{not\_known}(\exists 0)$. Since every process i knows both the clock value and the bound t, we therefore have that $K_i(\neg K_i \exists 0)$ at time $t+1$ implies that $K_i \mathsf{not\_known}(\exists 0)$ at that time. Finally, by the definition of knowledge we have that $K_i(\neg K_i \exists 0)$ holds iff $\neg K_i \exists 0$ holds, and the proof is complete. □

Theorem 2 *Opt_0 solves consensus in γ_{cr}^{t}.*

Proof Consider any run r of Opt_0 and let i be a nonfaulty process.

Decision: By definition of Opt_0, for any process that is active at time $t+1$, if i has not decided 0 by that time, we have $\neg K_i \exists 0$ at that time. Therefore, by Lemma 6, we have that $K_i \mathsf{not\_known}(\exists 0)$ at that time and so i decides upon 1 if it is undecided. Therefore, all processes that are active at time $t+1$, and in particular all nonfaulty processes, decide by that time at the latest, and in particular decide at some point throughout the run, as required.

Henceforth, let m be the decision time of i and let v be the value upon which i decides.

Validity: If $v = 0$, then $K_i \exists 0$ at m; thus, $\exists 0$ as required. Otherwise, $K_i \exists 0$ does not hold at m; therefore, $v_i = 1$, and so $\exists 1$ as required.

Agreement: It is enough to show that if $v = 1$, then no correct process ever decides 0 in the current run. Indeed, if any nonfaulty process j decided 0 at some time $m' < m$, then i would have received a message with a 0 from j at $m'+1 \leq m$, and so we would have $K_i \exists 0$ at m. To complete the proof, it is enough to show that no process decides 0 at any time $m' \geq m$; this follows by an easy inductive argument, using the fact that $\mathsf{not\_known}(\exists 0)$ at any time m'' implies $\mathsf{not\_known}(\exists 0)$ at $m''+1$. □

Theorem 3 *Opt_0 is an unbeatable consensus protocol in γ_{cr}^{t}.*

Proof Correctness is shown in Theorem 2. We show that for every consensus protocol $Q \preceq Opt_0$, we also have $Opt_0 \preceq Q$. Let, therefore, Q be a consensus protocol s.t. $Q \preceq Opt_0$; w.l.o.g., Q is a fip.

We first claim that $Opt_0 \preceq P_0$. Indeed, whenever P_0 decides upon 0, so does Opt_0; let therefore i be a process deciding upon 1 in P_0; by definition of P_0, this decision is made at time $m = t+1$, and furthermore, $\neg K_i \exists 0$ at that time. By Lemma 6, we therefore have that $K_i \mathsf{not\_known}(\exists 0)$ at time, and so, i decides upon 1 in Opt_0 at that time if it has not already decided.

By transitivity of domination, we thus have that $Q \preceq P_0$. By Lemma 4, we therefore have that $\mathsf{decide}_i(0)$ is performed in Q exactly when $K_i \exists 0$ first holds; therefore, no decision on 0 is made in Q before Opt_0. By Lemmas 2 and 5, we therefore have that $K_i \mathsf{not\_known}(\exists 0)$ is a necessary condition for $\mathsf{decide}_i(1)$ in $R_Q = R(Q, \gamma_{\mathrm{cr}}^{t})$. Therefore, no decision on 1 is made in Q before Opt_0. Therefore $Opt_0 \preceq Q$, as required, and the proof is complete. □

4.2 Testing for knowing that nobody knows

Opt_0 is not a standard protocol, because its actions depend on tests for process i's knowledge. (It is a *knowled-ge-based program* in the sense of [11].) In order to turn it into a standard protocol, we need to replace these by explicit tests on the processes' local states. The rule for $\mathsf{decide}_i(0)$ is easy to implement. By Lemma 3(a), $K_i \exists 0$ holds exactly if i's local state contains a time 0 node that is labelled with value 0. In the second case, the rule $K_i \mathsf{not\_known}(\exists 0)$ for performing $\mathsf{decide}_i(1)$ holds when i knows that no active process knows $\exists 0$, and we now characterize when this is true. A central role in our analysis will be played by process i's knowledge about the contents of various nodes in the communication graph. Recall that local states $r_i(m)$ in *fip*'s are communication graphs of the form $\mathcal{G}_\alpha(i, m)$; we abuse notation and write $\theta \in r_i(m)$ (respectively, $(\theta, \theta') \in r_i(m)$) if θ is a node of $\mathcal{G}_\alpha(i, m) = r_i(m)$ (respectively, if (θ, θ') is an edge of $\mathcal{G}_\alpha(i, m) = r_i(m)$); in this case, we say that θ is *seen* by $\langle i, m \rangle$. We now make the following definition:

Definition 3 (Revealed) Let $r \in R_P = R(P, \gamma_{\mathrm{cr}}^{t})$ for a *fip* protocol P. We say that *node $\langle j', m' \rangle$ is revealed to $\langle i, m \rangle$ in r if either (1) $\langle j', m' \rangle \in r_i(m)$, or (2) for some process i' such that $\langle i', m' \rangle \in r_i(m)$ it is the case that $\big(\langle j', m'-1 \rangle, \langle i', m' \rangle\big) \notin r_i(m)$*. We say that *time m' is revealed to $\langle i, m \rangle$ in r* if $\langle j', m' \rangle$ is revealed to $\langle i, m \rangle$ for all processes j'.

Intuitively, if node $\langle j', m' \rangle$ is revealed to $\langle i, m \rangle$ then i has proof at time m that $\langle j', m' \rangle$ can not carry information that is not known at $\langle i, m \rangle$ but may be known at another node $\langle j, m \rangle$ at the same time. This is because either i sees $\langle j', m' \rangle$ at that point—this is part (1)—or i has proof that j' crashed before time m', and so its state there was ☹, and j' did not send any messages at or after time m'. It is very simple and straightforward from the definition to determine which nodes are revealed to $\langle i, m \rangle$, based on $r_i(m) = \mathcal{G}_\alpha(i, m)$.

Observe that if a node $\langle j', m' \rangle$ is revealed to $\langle i, m \rangle$, then i knows at m what message could have been sent at $\langle j', m' \rangle$: If $\langle j', m' \rangle \in r_i(m)$ then $r_{j'}(m')$ is a subgraph of $r_i(m)$, while if $\big(\langle j', m' - 1 \rangle, \langle i', m' \rangle\big) \notin r_i(m)$ for some node $\langle i', m' \rangle \in r_i(m)$, then j' crashed before time m' in r, and so it sends no messages at time m'. Whether and when a node $\langle j', m' \rangle$ is revealed to i depends crucially on the failure pattern. If i receives a message from j' in round $m' + 1$, then $\langle j', m' \rangle$ is immediately revealed to $\langle i, m' + 1 \rangle$. If this message is not received by $\langle i, m' + 1 \rangle$, then $\langle j', m' + 1 \rangle$ — the successor of $\langle j', m' \rangle$ — becomes revealed (as being crashed, i.e., in state ☹) to $\langle i, m' + 1 \rangle$. But in general $\langle j', m' \rangle$ can be revealed to i at a much later time than $m' + 1$, (A simple instance of this is when $K_i \exists 0$ first becomes true at a time $m > 1$; in this case a node $\langle j, 0 \rangle$ with $v_j = 0$ is first revealed to i at time m.)

Suppose that some time $k \leq m$ is revealed to $\langle i, m \rangle$. Then, in a precise sense, process i at time m has access to all of the information that existed in the system at time k (in the hands of processes that had not crashed by then). In particular, if this information does not record an initial value of 0, then nobody can know $\exists 0$ at or after time k. In particular, i has proof that nobody knows $\exists 0$ at time m. We now formalize this intuition and show that revealed nodes can be used to determine when a process can know $\mathsf{not\_known}(\exists 0)$.

Lemma 7 *Let P be a fip and let $r \in R_P = R(P, \gamma^t_{\mathrm{cr}})$. For every node $\langle i, m \rangle$, it is the case that $(R_P, r, m) \models K_i \mathsf{not\_known}(\exists 0)$ exactly if both (1) $(R_P, r, m) \not\models K_i \exists 0$ and (2) some time $k \leq m$ is revealed to $\langle i, m \rangle$ in r.*

Proof We first claim that $(R_P, r, m) \models \mathsf{not\_known}(\exists 0)$ iff for every $0 \leq k \leq m$, there exists a process j_k s.t. $K_{j_k} \exists 0$ at time k in r — we call such $j_0, \ldots, j_m$ a ***0-path*** for time m in r; the proof is similar to (and simpler than) that of Lemma 20 in Appendix A, and is left to the reader.

Assume first that some time $k \leq m$ is revealed to $\langle i, m \rangle$ in r. As $(R_P, r, m) \not\models K_i \exists 0$, we thus have that no time-k node j satisfies $K_j \exists 0$; therefore, no 0-path exists for time m in r, and so $(R_P, r, m) \models \mathsf{not\_known}(\exists 0)$. Note that by definition of knowledge, time k is revealed to $\langle i, m \rangle$ in r iff $(R_P, r, m) \models K_i(\text{time } k \text{is revealed to } \langle i, m \rangle \text{ in the current run})$. Therefore, we have that time k being revealed to $\langle i, m \rangle$ implies not only $(R_P, r, m) \models \mathsf{not\_known}(\exists 0)$, but also $(R_P, r, m) \models K_i \mathsf{not\_known}(\exists 0)$, as required.

Assume now that no time $k \leq m$ is revealed to $\langle i, m \rangle$, i.e., that for every $k \leq m$, there exists a time-k node $\langle j_k, k \rangle$ that is not revealed to $\langle i, m \rangle$ in r—in Subsect. 4.4, we call such $j_0, \ldots, j_m$ a *hidden path* w.r.t. $\langle i, m \rangle$ in r. We construct a run $r' \in R_P$ s.t. $r'_i(m) = r_i(m)$, in which $j_0, \ldots, j_m$ constitutes a 0-path for time m—see Fig. 1 in Subsect. 4.4. The adversary in r' meets the following conditions, and otherwise coincides with that of r:

- $v_{j_0} = 0$.
- For every $k < m$, the node $\langle j_k, k \rangle$ crashes, successfully sending a message solely to $\langle j_{k+1}, k + 1 \rangle$.
- $\langle j_m, m \rangle$ is nonfaulty.

It is easy to verify that $r'_i(m) = r_i(m)$, that no more crashes occur in r' than in r, and that $j_0, \ldots, j_m$ indeed is a 0-path for time m in r. As $(R_P, r', m) \not\models \mathsf{not\_known}(\exists 0)$, and as $r'_i(m) = r_i(m)$, we therefore have that $(R_P, r, m) \not\models K_i \mathsf{not\_known}(\exists 0)$, as required. □

Based on Lemma 7, we obtain a standard unbea-table consensus protocol for γ^t_{cr} that implements Opt_0:

Protocol Opt_0^{std} (for an undecided proc. i at time m):

if i has seen a time-0 node with initial value 0
  then $\mathsf{decide}_i(0)$
elseif some time $k \leq m$ is revealed to $\langle i, m \rangle$
  then $\mathsf{decide}_i(1)$

In addition to facilitating an efficient implementation, the formulation of Opt_0^{std} also makes the worst-case decision time of Opt_0^{std} and Opt_0 apparent.

Lemma 8 *In Opt_0^{std} (and thus also Opt_0), all decisions are made by time $\boldsymbol{f} + 1$ at the latest.*

Proof Let i be an undecided node at time m in Opt_0^{std}; it is enough to show that $m \leq \boldsymbol{f}$. As i is undecided, by definition of Opt_0^{std}, for every $0 \leq k < m$, there exists a process j_k s.t. $\langle j_k, k \rangle$ is not revealed to $\langle i, m \rangle$. We first note that all of the nodes j_k are faulty; indeed, as $\langle j_k, k \rangle$ is not revealed to $\langle i, m \rangle$, and as $k < m$, we have that $\langle i, k + 1 \rangle$ receives no message from $\langle j_k, k \rangle$. We further note that all j_k are distinct; indeed, for every $k < k' < m$, we have (once again since $\langle j_k, k \rangle$ is not revealed to $\langle i, m \rangle$) that $\big(\langle j_k, k' - 1 \rangle, \langle i, k' \rangle\big) \notin r_i(m)$ while $\langle i, k' \rangle \in r_i(m)$, and so by definition $\langle j_k, k' \rangle$ is revealed to $\langle i, m \rangle$. We conclude that $j_0, \ldots, j_{m-1}$ are m distinct faulty nodes, and so $m \leq \boldsymbol{f}$ and the proof is complete. □

4.3 Comparing Opt_0 to previous protocols

It is interesting to compare Opt_0 with efficient early-stopping consensus protocols [3,7,13,17]. Let's say that *the sender set repeats at* $\langle i, m \rangle$ in run r if i hears from the same set of processes in rounds $m - 1$ and m. If this happens then, for every $\langle j, m - 1 \rangle \notin r_i(m)$, we are guaranteed that $(\langle j, m - 2 \rangle, \langle i, m - 1 \rangle) \notin r_i(m)$. Thus, all nodes at time $(m - 1)$ are revealed to $\langle i, m \rangle$. Indeed, in a run in which $\boldsymbol{f}$ failures actually occur, the sender set will repeat for every correct process by time $\boldsymbol{f} + 1$ at the latest. Efficient early stopping protocols typically decide when the sender set repeats.

Indeed, the protocol $P0_{\text{opt}}$ that was claimed by [17] to be unbeatable does so as well, with a slight optimization. Writing $\forall 1$ to stand for "all initial values are 1", $P0_{\text{opt}}$ is described as follows:

Protocol $P0_{\text{opt}}$ (for an undec. proc. i at time m) [17] :

if $K_i \exists 0$
  then $\mathsf{decide}_i(0)$
elseif $K_i \forall 1$ or ($m \geq 2$ and the sender set repeats)
  then $\mathsf{decide}_i(1)$

Protocols Opt_0 and $P0_{\text{opt}}$ differ only in the rule for deciding 1. But Opt_0 strictly beats $P0_{\text{opt}}$, and sometimes by a wide margin. If $\boldsymbol{t} = \Omega(n)$ then it can decide faster by a *ratio* of $\Omega(n)$. Indeed, we can show:

Lemma 9 *If* $3 \leq \boldsymbol{t} \leq n-2$, *then* Opt_0 *strictly dominates* $P0_{\text{opt}}$. *Moreover, there exists an adversary for which* $\mathsf{decide}_i(1)$ *is performed after 3 rounds in* Opt_0, *and after* $\boldsymbol{t}+1$ *rounds in* $P0_{\text{opt}}$.

Proof First notice that Opt_0 dominates $P0_{\text{opt}}$, since $K_i \forall 1$ is true iff time 0 is revealed to i, and if i's sender set repeats in round m, then time $m-1$ is revealed to i at time m. Hence, for every adversary, processes decide in Opt_0 at least as soon as they do in $P0_{\text{opt}}$. We now show an adversary for which the decisions are made strictly earlier in Opt_0 than in $P0_{\text{opt}}$; moreover, this adversary meets the conditions of the second clause of the Lemma 9.

Denote the processes by $\mathsf{Procs} = \{1, 2, \ldots, n\}$. Let α be defined as follows. All initial values in α are 1. In round 1, only process 1 fails, and it is silent: it crashes without sending any messages. In round 2 two processes crash—process 2 and process 3, with process 2 sending only to process n, and process 3 sending to everyone except process n. No process fails in round 3, and, in each of the rounds $m = 4, \ldots, \boldsymbol{t}$, process m crashes without sending any messages. Since precisely $\boldsymbol{t}$ processes fail in α we have that $\alpha \in \mathsf{Crash}(\boldsymbol{t})$.

Observe that in $fip[\alpha]$ no correct process ever knows process 1's initial value. In addition, for every correct process, the first round in which the sender set repeats is round $\boldsymbol{t}+1$. Indeed, every correct process other than n fails to hear from process m for the first time in round m, for $m = 1, \ldots, \boldsymbol{t}$, while process n differs slightly, in that it fails to hear from process 3 in round 2 and from process 2 in round 3. Therefore, in $P0_{\text{opt}}[\alpha]$ all correct processes decide 1 at time $\boldsymbol{t}+1$, since round $\boldsymbol{t}+1$ is the first one in which their sender set repeats; no process decides any earlier. Now let us consider when a process i that is correct according to α decides in Opt_0. By definition, i receives messages in round 3 from both $\langle n, 2\rangle$ and $\langle n-1, 2\rangle$. Together, these contain the information about nodes $\langle 2, 1\rangle, \langle 3, 1\rangle, \ldots, \langle n, 1\rangle$. Moreover, node $\langle 1, 1\rangle$ is revealed to $\langle i, 3\rangle$ as well (as being crashed), since the edge $(\langle 1, 0\rangle, \langle i, 1\rangle)$ is absent from i's view at $\langle i, 3\rangle$. It follows that time 1 is revealed to $\langle i, 3\rangle$, and so i decides 1 at time 3, after 3 rounds, as claimed. Since $3 < 4 \leq \boldsymbol{t}+1$, we have that when the adversary is α, decisions in Opt_0 occur strictly earlier than in $P0_{\text{opt}}$, and we are done. □

4.4 Hidden paths and agreement

It is instructive to examine the proof of Lemma 7 and consider when an active process i is undecided at $\langle i, m\rangle$ in Opt_0. This occurs if both $\neg K_i \exists 0$ and, in addition, for every $k = 0, \ldots, m$ there is at least one node $\langle j_k, k\rangle$ that is not revealed to $\langle i, m\rangle$. We call the sequence of nodes $\langle j_0, 0\rangle, \ldots, \langle j_m, m\rangle$ a *hidden path w.r.t.* $\boldsymbol{\langle i, m\rangle}$. Such a hidden path implies that all processes $j_0, \ldots, j_m$ have crashed. Roughly speaking, $\exists 0$ could be relayed along such a hidden path without i knowing it (see Fig. 1). More formally, its existence means that there is a run, indistinguishable at $\langle i, m\rangle$ from the current one, in which $v_{j_0} = 0$ and this fact is sent from each j_k to j_{k+1} in every round $k+1 \leq m$. In that run process j_m is active at time m and $K_{j_m} \exists 0$, and that is why $K_i \mathsf{not\_known}(\exists 0)$ *does not* hold. Hidden paths are implicit in many lower bound proofs for consensus in the crash failure model [7,10], but they have never before been captured formally.

[3]. [4] Clearly, hidden paths can relay more than just the existence of a value of 0. In a protocol in which some view can prove that the state is univalent in the sense of Fischer, Lynch and Paterson [12], a hidden path from a potentially pivotal state can keep processes from deciding on the complement value. Our analysis in the remainder of the paper provides additional cases in which unbeatable consensus is obtained when hidden paths can be ruled out.

5 Unbeatable uniform consensus

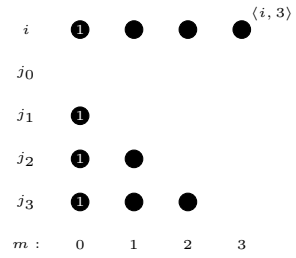
Recall that in the *uniform* version of consensus, it is required that any two processes that decide must decide on the same value, even if one (or both) of them crash soon after deciding.

5.1 Deriving an unbeatable protocol

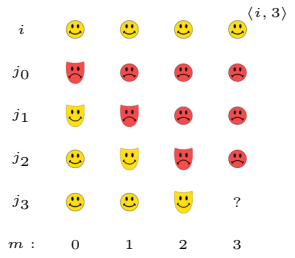
Under crash failures, a process generally does not know whether or not it is correct. Indeed, as long as it has not

[3] For simplicity, in this example every node seen by $\langle i, 3\rangle$ is also seen by all other nodes in the view of $\langle i, 3\rangle$. In other words, there exists no node $\langle j, m'\rangle$ that is in state 😈 according to the information held by $\langle i, 3\rangle$, i.e., both $\langle j, m'\rangle$ is seen by $\langle i, 3\rangle$, and i has indirectly learnt by time 3 that j has in fact crashed at m'.

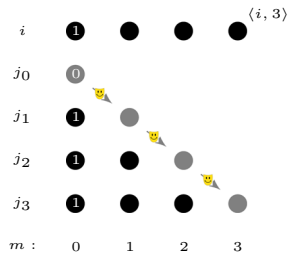
[4] In this run, the state of both $\langle j_0, 0\rangle$ and $\langle j_1, 1\rangle$, according to the information held by $\langle j_3, 3\rangle$, is 😈, as defined in Footnote 3.



(a) All nodes seen (directly or indirectly) by $\langle i, 3\rangle$. The initial value is shown for all seen time-0 nodes. Notably, both $\neg K_i \exists 0$ and $\neg K_i \neg \exists 0$ hold at time $m = 3$.



(b) The state of each node, according to the information held by $\langle i, 3\rangle$: 🙂=seen by all; 🙂=seen, may have crashed; 😟=revealed, seen by none; 😈=hidden: may have been seen by others.[3]



(c) A run that is possible according to the information held by $\langle i, 3\rangle$;[4] in this run, $K_{j_3} \exists 0$ holds at time $m = 3$. Therefore, $\neg K_i \mathsf{not\_known}(\exists 0)$ at time $m = 3$. $\langle i, 3\rangle$ is therefore undecided in OPT_0.

Fig. 1 A hidden path $\langle j_0, 0\rangle, \ldots, \langle j_3, 3\rangle$ w.r.t. $\langle i, 3\rangle$ implies $\neg K_i \mathsf{not\_known}(\exists 0)$ at 3

seen $\boldsymbol{t}$ failures, the process may (for all it knows) crash in the future. As a result, while $K_i \exists 0$ is a necessary condition for $\mathsf{decide}_i(0)$ as before, it cannot be a *sufficient* condition for decision in any Uniform Consensus protocol. This is because, with this rule, a process starting with 0 immediately decides 0, and may immediately crash. If all other processes have initial value 1, all other decisions can only be on 1. Of course, $K_i \exists 0$ is still a necessary condition for deciding 0, but it is *not* sufficient. Denote by $\exists \mathsf{correct}(\mathrm{v})$ the fact "some *correct* process knows $\exists \mathrm{v}$". We show the following:

Lemma 10 *$K_i \exists \mathsf{correct}(\mathrm{v})$ is a necessary condition for i deciding v in any protocol solving uniform consensus.*

Proof Let r be a run of a Uniform Consensus protocol P, such that $(R_P, r, m) \not\models K_i \exists \mathsf{correct}(\mathrm{v})$. Thus, there exists a run $r' \in P[\alpha']$ such that $r_i(m) = r'_i(m)$ and $(R_P, r', m) \not\models \exists \mathsf{correct}(\mathrm{v})$. Consider the adversary β that agrees with α' up to time m, and in which all active but faulty processes at (r', m) crash at time m without sending any messages. $\beta \in \gamma^{\boldsymbol{t}}_{\mathrm{cr}}$ because it has a legal input vector (identical to α'), and at most $\boldsymbol{t}$ crash failures, as it has the same set of faulty processes as $\alpha' \in \gamma^{\boldsymbol{t}}_{\mathrm{cr}}$. It follows that $r'' = P[\beta]$ is a run of P. Since β agrees with α' on the first m rounds, we have that $r''_i(m) = r'_i(m)$. Nonetheless, no correct process will ever know $\exists \mathrm{v}$ in r'', and thus by *Validity* no correct process ever decides v in r''. By decision, all correct processes thus decide not on v. By *Uniform Agreement*, and as $\boldsymbol{t} < n$ (i.e., there are correct processes), i cannot decide on v in r'', and thus, as $r''_i(m) = r'_i(m) = r_i(m)$, it cannot decide on v in r at m. □

Definition 4 For a given run r, we denote by $F\langle i, m\rangle$ the number of failures known to $\langle i, m\rangle$, i.e., the number of processes $j \neq i$ from which i does not receive a message in round m.

As the next lemma shows, there is a direct and simple way to test whether $K_i \exists \mathsf{correct}(\mathrm{v})$ holds, based on $r_i(m)$:

Lemma 11 *Let $r \in R_P = R(P, \gamma^{\boldsymbol{t}}_{\mathrm{cr}})$, let i be an active process at time m in r, and let $\boldsymbol{d} \triangleq F\langle i, m\rangle$. Then $(R_P, r, m) \models K_i \exists \mathsf{correct}(\mathrm{v})$ iff at least one of*

(a) $m > 0$ and $(R_P, r, m-1) \models K_i \exists \mathrm{v}$, or
(b) $(R_P, r, m) \models K_i(K_j \exists \mathrm{v} \mathit{held at time}\ m-1)$ holds for at least $(\boldsymbol{t}-\boldsymbol{d})$ distinct processes $j \neq i$.

Proof It is straightforward to see that each of conditions (a) and (b) implies $K_i \exists \mathsf{correct}(\mathrm{v})$ (Condition (a): as $\langle i, m-1\rangle$ is seen at m by all correct processes; condition (b): as the number of distinct processes knowing $\exists 0$, *including i itself*, is greater than the maximum number of active processes that can yet fail). If neither condition holds, then i considers it possible that only incorrect processes know $\exists \mathrm{v}$, and that they all immediately fail (i at time m before sending any messages, and the others — immediately after sending the last message seen by i). Consequently, no correct process would ever know $\exists \mathrm{v}$ in r □

By Lemma 3, at time $\boldsymbol{t}+1$ the conditions $K_i \exists \mathrm{v}$ and $K_i \exists \mathsf{correct}(\mathrm{v})$ are equivalent. As in the case of (nonuniform) consensus, we note that if $K_i \exists 0$ (equivalently, $K_i \exists \mathsf{correct}(0)$) does not hold at time $\boldsymbol{t}+1$, then it never will. We thus phrase the following *beatable* algorithm, analogous to P_0 in Sect. 4, for Uniform Consensus; in this protocol, $K_i \exists \mathsf{correct}(0)$ (the necessary condition for deciding 0 in Uniform Consensus) replaces $K_i \exists 0$ (the necessary condition in Nonuniform Consensus) as the decision rule for 0. The decision rule for 1 remains the same.

Protocol $u\text{-}P_0$ (for an undecided proc. i at time m):

if	$K_i \exists\mathsf{correct}(0)$	*then* $\mathsf{decide}_i(0)$
elseif	$m = \boldsymbol{t}+1$	*then* $\mathsf{decide}_i(1)$

Following a similar line of reasoning to that leading to Opt_0, we obtain an unbeatable Uniform Consensus protocol:

Protocol $u - Opt_0$ (for an undecided proc. i at time m):

if $K_i \exists\mathsf{correct}(0)$
  then $\mathsf{decide}_i(0)$
elseif $\neg K_i \exists 0$ and some time $k \leq m$ is revealed to $\langle i, m\rangle$
  then $\mathsf{decide}_i(1)$

Recall that whether $K_i \exists\mathsf{correct}(0)$ holds can be che-cked efficiently via the characterization in Lemma 11. For ease of exposition, the correctness of $u - Opt_0$ in Theorem 4 is shown in Appendix B.

Theorem 4 *$u - Opt_0$ solves Uniform Consensus in $\gamma^{\boldsymbol{t}}_{\mathrm{cr}}$. Furthermore,*

- *If $\boldsymbol{f} \geq \boldsymbol{t} - 1$, then all decisions are made by time $\boldsymbol{f}+1$ at the latest.*
- *Otherwise, all decisions are made by time $\boldsymbol{f}+2$ at the latest.*

Hidden paths again play a central role in the correctness and unbeatability proof of $u - Opt_0$. Indeed, as in the construction of Opt_0 from P_0 in Sect. 4, the construction of $u - Opt_0$ from $u-P_0$ involves some decisions on 1 being moved earlier in time, by means of the last condition, checking the absence of a hidden path. (Decisions on 0 cannot be moved any earlier, as they are taken as soon as the necessary condition for deciding 0 holds.) Observe that the need to obtain $K_i \exists\mathsf{correct}(v)$ rather than $K_i \exists v$ for deciding v concisely captures the essential distinction between Uniform Consensus and Nonuniform Consensus. The fact that the same condition—the existence of a hidden path—keeps a process i from knowing that no active j can know $K_j \exists\mathsf{correct}(v)$, as well as keeping i from knowing that no j knows $K_j \exists v$, explains why the bounds for both problems, and their typical solutions, are similar.

5.2 Unbeatability of $u - Opt_0$

Proving the unbeatability of $u - Opt_0$ is more challenging than proving it for Opt_0 in Sect. 4. Intuitively, this is because the fact that an initial value of 0 is known by a nonfaulty process does not imply that some process has already decided on 0. As a result, the possibility of dominating $u - Opt_0$ by switching 0 decisions to 1 decisions needs to be explicitly rejected. This is done by employing reachability arguments essentially establishing the existence of the continual common knowledge conditions of [17].

More formally, as with P_0 in the case of Nonuniform Consensus, by analyzing decisions in protocols dominating $u-P_0$, we show that no Uniform Consensus protocol can dominate $u - Opt_0$. Lemmas 12 and 13 below give sufficient conditions for deciding 0 in any Uniform Consensus protocol dominating $u-P_0$. As mentioned above, the analysis is considerably more subtle for Uniform Consensus, because the analogue of Lemma 4 is not true: receiving a message with value 0 in a protocol dominating $u-P_0$ does not imply that the sender has decided 0.

Remark 1 (No decision at time 0) Let Q be a protocol that solves Uniform Consensus. Since $\boldsymbol{t} > 0$, no process decides at time 0 in any run of Q.

Lemma 12 *(Decision at time 1) Let $Q \preceq u-P_0$ be a protocol that solves Uniform Consensus and let $r = r[\alpha]$ be a run of Q. Moreover, assume that i is active at time 1 and $v_i = 0$ in r. If one of the following conditions holds in r, then $\langle i, 1\rangle$ decides 0.*

1. *There exists a process $j \neq i$ s.t. $v_j = 0$ and $\langle j, 0\rangle$ is seen by $\langle i, 1\rangle$.*
2. *$\boldsymbol{t} > 1$ and $F\langle i, 1\rangle < \boldsymbol{t}$.*

Proof Sketch: By domination i must decide at $\langle i, 1\rangle$, but why must it decide on the value 0? For Part 1, we argue by descending induction over the number of messages of 0-value nodes seen by $\langle i, 1\rangle$. The base case is by validity. The induction step proceeds by cases. Case I: If, for some process k, the node $\langle k, 0\rangle$ is hidden from $\langle i, 1\rangle$, then i considers it possible that k started with value 0 and sent a message to j but not to i. So, by the induction hypothesis, process j decides on 0, and hence by Uniform Agreement so must i. Case II: If there is no process k for which the node $\langle k, 0\rangle$ is hidden from $\langle i, 1\rangle$, then it is possible that some node $\langle k', 0\rangle$ is hidden from $\langle j, 1\rangle$. Therefore, process i considers it possible that j is as in Case I (and considers it possible that i sees that k' started out with 0 after all). By Case I, process j must decide 0, and therefore so should i.

Finally, for Part 2, the condition guarantees that i considers it possible that some other process does not see some node $\langle j, 0\rangle$ nor will it ever see $\langle i, 1\rangle$ but will forever consider it possible that nonetheless i saw $\langle j, 0\rangle$ with value 0 and therefore by Part 1 decided on 0 at time one before crashing. Beyond a sketch, we now have:

Proof For both parts, we first note that by Lemma 11 and by definition of $u-P_0$, i decides 0 at time 1 in $u-P_0[\alpha]$. As $Q \preceq u-P_0$, we thus have that i must decide upon some value in r by time 1. Since process i does not decide at $(r, 0)$ by Remark 1, it decides at $(r, 1)$.

We now show Part 1 by induction on $n-|Z_i^0|$, where

$$Z_i^0 \triangleq \{k \mid v_k = 0 \ \&\ \langle k,0\rangle \text{is seen by} \langle i,1\rangle.\}$$

Since $\langle j,0\rangle$ is seen by $\langle i,1\rangle$ and i is active at time 1, we have that $i, j \in Z_i^0$, and so $1 < |Z_i^0| \leq n$.

Base: $|Z_i^0| = n$. In this case, all initial values are 0, and so by *Validity* i decides 0 at $(r,1)$.

Inductive step: Let $1 < \ell < n$, let $|Z_i^0| = \ell$, and assume inductively that Part 1 holds for all runs r' in which $|Z_i^0| = \ell + 1$. We reason by cases.

1. There exists a process k s.t. $\langle k,0\rangle$ is hidden from $\langle i,1\rangle$. We consider a run r' of Q in which $v_k = 0$, i cannot distinguish r' from r at time 1, every node that is seen by $\langle i,1\rangle$ is also seen by $\langle j,1\rangle$, and finaly $\langle k,0\rangle$ is seen by $\langle j,1\rangle$. Note that this is possible since $\langle k,0\rangle$ is hidden from $\langle i,1\rangle$. Formally, r' is a run of Q such that *1)* $r'_i(1) = r_i(1)$, *2)* j is active at $(r',1)$, *3)* k has initial value 0 and crashes in round 1 in r', and finally *4)* $Z_j^0 = Z_i^0 \cup \{k\}$ in r'. (Note that by definition, Z_i^0 is the same in both r and r'.) By the induction hypothesis (switching the roles of i and j), j decides 0 at $(r',1)$, and therefore by *Uniform Agreement*, i cannot decide 1 at $(r',1)$, and therefore it does not decide 1 at $(r,1)$. Thus, i decides 0 at $(r,1)$.
2. Otherwise, $\langle k,0\rangle$ is seen by $\langle i,1\rangle$ for all processes k. As $|Z_i^0| < n$, there exists a process $k \notin Z_i^0$ with initial value 1 (in particular, $k \notin \{i,j\}$). Hence, as $\boldsymbol{t} > 0$, there exists a run r' of Q, s.t. *1)* $r'_i(1) = r_i(1)$, *2)* j is active at $(r',1)$, *3)* $\langle k,0\rangle$ is hidden from $\langle j,1\rangle$ in r', and *4)* $Z_j^0 = Z_i^0$ in r'. (Once again, Z_i^0 has the same value in both r and r'.) By Case I (switching the roles of i and j), j decides 0 at $(r',1)$, and therefore by *Uniform Agreement*, i cannot decide 1 at $(r',1)$, and therefore it does not decide 1 at $(r,1)$. Thus, i decides 0 at $(r,1)$.

We move on to prove Part 2 zeros. Since $F\langle i,1\rangle < \boldsymbol{t}$ by hypothesis, there exists a process $k \neq i$ s.t. $\langle k,0\rangle$ is seen by $\langle i,1\rangle$. As $n > \boldsymbol{t} > 1$, we have that $n > 2$ and so there exists a process $j \notin \{i,k\}$; if $F\langle i,1\rangle > 0$, then we pick j s.t. $\langle j,0\rangle$ is hidden from $\langle i,1\rangle$. Since $\boldsymbol{t} > 1$ (for the case in which $F\langle i,1\rangle = 0$ and $\langle j,0\rangle$ is seen by $\langle i,1\rangle$) and since $\boldsymbol{t} > F\langle i,1\rangle$ (for the case in which $\langle j,0\rangle$ is hidden from $\langle i,1\rangle$), there exists a run r' of Q, s.t. *1)* $r'_i(1) = r_i(1)$, *2)* k never fails in r', *3)* j fails at $(r',0)$ before sending any messages except perhaps to i, and *4)* i fails at $(r',1)$, immediately after deciding but before sending any messages. Thus, there exists a run r'' of Q, s.t. *1)* $r''_k(m') = r'_k(m')$ *for all* m', *2)* k never fails in r'', *3)* i and j both have initial value 0 in r'', *4)* j fails at $(r'',0)$ while successfully sending a message only to i (and therefore $j \in Z_i^0$ in r''), and *5)* i fails at $(r'',1)$, immediately after deciding but before sending out any messages. By Part 1, i decides 0 at $(r'',1)$, and therefore k can never decide 1 during r'', and therefore neither during r'. As k never fails during r', by *Decision* it must thus decide 0 at some point during r'. Therefore, by *Uniform Agreement*, i cannot decide 1 at $(r',1)$, and thus it does not decide 1 at $(r,1)$. It follows that i decides 0 at $(r,1)$, as claimed. □

Lemma 13 *(Decision at times later than 1) Let $Q \preceq u-P_0$ be a protocol that solves Uniform Consensus, let $r = Q[\alpha]$ be a run of Q and let $m > 0$. Let i be a process s.t. (a) $K_i\exists 0$ holds at time m for the first time in r, s.t. (b) $K_i\exists\mathsf{correct}(0)$ holds at time $m+1$ for the first time in r, and s.t. (c) i is active at $(r,m+1)$. If either of the following hold in r, then i decides 0 at $(r,m+1)$.*

1. *All of the following hold.*
   - $F\langle i,m+1\rangle < \boldsymbol{t}$.
   - *There exists a process z s.t. $K_z\exists 0$ holds at time $m-1$, s.t. $\langle z,m-1\rangle$ is seen by $\langle i,m\rangle$, but s.t. $\langle z,m\rangle$ is not seen by $\langle i,m+1\rangle$,*
   - *There exists a process $j \neq i$ s.t. $\langle j,m\rangle$ is seen by $\langle i,m+1\rangle$ and $\langle z,m-1\rangle$ is seen by $\langle j,m\rangle$.*
2. $F\langle i,m+1\rangle < \boldsymbol{t}-1$.

Proof Sketch: The question is the same one as in Lemma 12, but for times later than 1. The proof is more involved, and uses reachability considerations of higher order. In other words, it uses many more intermediate runs until we "end up" with a run where we know for certain what some process may decide. This then tells us what processes in all these intermediate runs must decide, by Uniform Agreement for every pair of adjacent runs. By domination, i must decide at time $m+1$, but why must it decide on a 0 value? We prove this by induction on m.

Let z be a process that $\langle i,m\rangle$ "hears about a 0 from" (so $\langle i,m\rangle$ sees $\langle z,m-1\rangle$ and hears from it about a zero). We first consider the case in which we are either in Part 1 (so the process j is defined by assumption), or we're in Part 2 and there's some other process other than i that sees $\langle z,m-1\rangle$ at time m and is seen by $\langle i,m+1\rangle$—call this process j. So either way, we have a process called j with certain properties. We will prove the induction step for the current value of m by (an inner) induction on the number of processes that are hidden from $\langle i,m+1\rangle$ or are seen by $\langle i,m+1\rangle$ (in other words, are not known by $\langle i,m+1\rangle$ to have failed before time m) but have not seen $\langle z,m-1\rangle$. This is the most technical part of the proof. It involves bookkeeping of the various assumptions and massaging them to show that the analogous assumptions are met when applying the (inner or outer) induction hypotheses.

In a nutshell, if there are no such processes (the "inner induction" base case) then we show that we are clearly in

Part 2 and show that by the induction hypothesis of the outer induction—or by Lemma 12 if $m = 1$—when applied to z, that z decides 0 at time m. If there are such processes (the "inner induction" step), then somewhat like in the proof of Lemma 12, we show that one of following two cases must hold. Case I: process i considers it possible that j at time $m+1$ "is like i" only seeing one additional agent that is hidden from i, so by the induction hypothesis of the inner induction (since the number of processes hidden from j is smaller), process j might decide 0 at time $m+1$, and therefore by Uniform Agreement so must process i. Case II: Process i considers it possible that j meets the conditions of Case I, i.e., that j considers it possible that the induction hypothesis of the inner induction can be applied to i at time $m+1$. Thus, i considers it possible that j may decide 0 at $m+1$, and therefore again by Uniform Agreement so must i.

The final case to consider is that we are in Part 2 and there's no process at time m other than i that sees $\langle z, m-1\rangle$ and is seen by $\langle i, m+1\rangle$. In this case (again, with appropriate bookkeeping), process i considers it possible that it will fail immediately after deciding but before sending any messages, while some other process, say k, will not fail, and that k will consider it possible that before failing i satisfied the conditions of the previous case of the proof (the one with a well-defined process j with certain properties) and that i therefore decided 0 just before failing. Under this scenario, when k decides (even if this is much later), it cannot decide on 1 by Uniform Agreement. But since it does not fail, it must decide at some time and will decide on 0 when doing so. Thus, since i must decide now (by domination), and since regardless of what i decides on, if i immediately fails then k may consider it possible that i decided on 0 before failing, process i must really decide on 0.

Proof We prove the lemma by induction on m, with the base and the step sharing the same proof. Both parts are proven together, highlighting local differences in reasoning for the different parts as needed. For Part 2, we denote by z an arbitrary process s.t. $K_z\exists 0$ holds at time $m-1$ and s.t. $\langle z, m-1\rangle$ is seen by $\langle i, m\rangle$. As $m > 0$, such a process must exist for i to know $\exists 0$ at time m for the first time. (Nonetheless, unlike when proving Part 1, it is not guaranteed when proving this part that $\langle z, m\rangle$ is not seen by $\langle i, m+1\rangle$.)

We first note that by Lemma 10 and by definition of $u-P_0$, i decides 0 at time $m+1$ in $u-P_0[\alpha]$. As $Q \preceq u-P_0$, we thus have that i must decide upon some value in r by time $m+1$. By Lemma 10, the precondition for deciding 0 is not met by i at (r, m). Therefore, it is enough to show that i does not decide 1 before or at time $m+1$ in r in order to show that i decides 0 at $(r, m+1)$.

Let $Z_i^{z,m}$ be the set of processes k s.t. $\langle k, m\rangle$ is seen by $\langle i, m+1\rangle$ in r and s.t. $\langle z, m-1\rangle$ is seen by $\langle k, m\rangle$ in r. (By definition, $i \in Z_i^{z,m}$.) Let C_i be the set of all processes k s.t. $\langle k, m\rangle$ is either seen by, or is hidden from $\langle i, m+1\rangle$ (i.e., the set of nodes that $\langle i, m+1\rangle$ does not know to be inactive at time m). Note that by definition, $Z_i^{z,m} \subseteq C_i$. We first consider the case that $Z_i^{z,m} \supsetneq \{i\}$, and prove the induction for the given value of m under this assumption, by induction on $|C_i \setminus Z_i^{z,m}|$.

Base: $Z_i^{z,m} = C_i$. In this case, $\langle i, m+1\rangle$ does not know that z fails at time m–1 . Thus, $z \in C_i$ and therefore $z \in Z_i^{z,m}$. It follows that $\langle z, m\rangle$ is seen by $\langle i, m+1\rangle$ and therefore the second condition of Part 1 does not hold. Hence, the condition of Part 2 must hold: $F\langle i, m+1\rangle < t-1$. Furthermore, z must be active at time m. We now argue that z decides 0 at (r, m), which completes the proof of the base case, as by *Uniform Agreement* i can never decide 1 during r. We reason by cases; for both cases, note that since $\langle z, m\rangle$ is seen by $\langle i, m+1\rangle$, we have that $F\langle z, m\rangle \leq F\langle i, m+1\rangle < \boldsymbol{t}-1$.

- If $m = 1$: As $K_z\exists 0$ at time $m-1 = 0$, z has initial value 0. As $F\langle z, m\rangle < t-1$, we have that $\boldsymbol{t} > 1$. By Part 2 of Lemma 12 (for $i = z$), we thus have that z decides 0 at $(r, 1) = (r, m)$.
- Otherwise, $m > 1$. In this case, as $\langle z, m-2\rangle$ is seen by $\langle i, m-1\rangle$, and as $K_i\exists 0$ holds at time m for the first time, we have that $K_z\exists 0$ holds at time $m-1$ for the first time. Similarly, as $\langle z, m-1\rangle$ is seen by $\langle i, m\rangle$, and as $K_i\exists\mathsf{correct}(0)$ does not hold at time m, we have that $K_z\exists\mathsf{correct}(0)$ does not hold at time $m-1$. By Part 2 of the m-induction hypothesis (for $i = z$), process z decides 0 at (r, m).

Inductive step: Let $\{i\} \subsetneq Z_i^{z,m} \subsetneq C_i$, and assume that the claim holds whenever $Z_i^{z,m}$ is of larger size. For Part 1, note that $j \in Z_i^{z,m}$, for j as defined in the conditions for that part; for Part 2, let $j \in Z_i^{z,m}$ be arbitrary. Analogously to the proof of the induction step in the proof of Part 1 of Lemma 12, we reason by cases. For the time being, assume that the conditions of Part 2 hold, i.e., that $F\langle i, m+1\rangle < \boldsymbol{t}$–1.

I. There exists a process $k \in C_i$ s.t. $\langle k, m\rangle$ is hidden from $\langle i, m+1\rangle$. We consider a run r' of Q that i cannot distinguish from the run r, up to time $m+1$ (hence $Z_i^{z,m}$ and C_i are the same sets in both r and r'), and for each $k' \in Z_i^{z,m} \cup \{k\}$, the node $\langle z, m-1\rangle$ is seen by $\langle k', m\rangle$, which in turn is seen by $\langle j, m+1\rangle$ (and consequently $Z_j^{z,m} = Z_i^{z,m} \cup \{k\}$ in r'); note that this is possible because $\langle k, m\rangle$ is hidden from $\langle i, m+1\rangle$ and z can fail at time $m-1$ after sending a message to all processes in $Z_i^{z,m} \cup \{k\}$. More precisely, r' is a run of Q s.t. *1)* $r'_i(m+1) = r_i(m+1)$, *2)* j is active at $(r', m+1)$, *3)* $\langle z, m-1\rangle$ is seen by $\langle k, m\rangle$ in r', and *4)* $Z_j^{z,m} = Z_i^{z,m} \cup \{k\}$ and $C_j = C_i$ in r'. We note that $F\langle j, m+1\rangle = F\langle i, m+1\rangle - 1$ in r', and that by

definition $F\langle i, m+1\rangle$ is the same in both r and r'. We also observe that $r'_j(m) = r_j(m)$ (as $r'_i(m+1) = r_i(m+1)$). By the inductive hypothesis for $Z_j^{z,m}$ (i.e., for j w.r.t. z at time m), process j decides 0 at $(r', m+1)$, and therefore by *Uniform Agreement*, i cannot decide 1 in r', and therefore it cannot decide 1 before or at $m+1$ in r', and the proof is complete.

II Otherwise, for each process $k \in C_i$, $\langle k, m\rangle$ is seen by $\langle i, m+1\rangle$. As $Z_i^{z,m} \subsetneq C_i$, there exists a process $k \neq i$ s.t. $\langle k, m\rangle$ is seen by $i, m+1$ but s.t. $\langle z, m-1\rangle$ is not seen by $\langle k, m\rangle$ (thus $k \neq j$). Let us consider a run r' of Q that i cannot distinguish from the run r up to time $m+1$ (hence $Z_i^{z,m}$ and C_i are the same sets in both r and r'), in which process k crashes at time m after sending a message to i but before sending a message to j. Namely, $\langle k, m\rangle$ is seen by $\langle i, m+1\rangle$ but unseen by $\langle j, m+1\rangle$ (k can crash in r' because $F\langle i, m+1\rangle < \boldsymbol{t}$ in r). Finally, for each $k' \in C_i$, the node $\langle k', m\rangle$ is seen by $\langle j, m+1\rangle$ in r' (thus $C_j = C_i$ in r'; note that k' might crash after sending a message to i at time m), and if $k' \in Z_i^{z,m}$, then the node $\langle z, m-1\rangle$ is seen by $\langle k', m\rangle$ (and consequently $Z_j^{z,m} = Z_i^{z,m}$ in r'). Formally, r' is a run of Q s.t. *1)* $r'_i(m+1) = r_i(m+1)$, *2)* process j is active at $(r', m+1)$, *3)* $\langle k, m\rangle$ is hidden from $\langle j, m+1\rangle$ in r', and *4)* $Z_j^{z,m} = Z_i^{z,m}$ and $C_j \supseteq C_i$ in r'. (Once again, $Z_i^{z,m}$ and C_i have the same values in both r and r'.) We note that $F\langle j, m+1\rangle = F\langle i, m+1\rangle + 1$ in r', and that once more, by definition, $F\langle i, m+1\rangle$ is the same in both r and r'. Again, we have that $r'_j(m) = r_j(m)$. By Case I (for $i = j$), and since Case I uses the inductive hypothesis for $Z_j^{z,m}$ with one less failure, we conclude that j decides 0 at $(r', m+1)$. Therefore, by *Uniform Agreement*, process i cannot decide 1 at $(r', m+1)$, and thus it cannot decide 1 before or at $m+1$ in r, and the proof is complete.

To show that the $Z_i^{z,m}$-induction step also holds under the conditions of Part 1, we observe that since $\langle z, m\rangle$ is not seen by i,m+1 in this case, the amount of invocations of Case II (which uses Case I with one additional known failure) before reaching the $Z_i^{z,m}$-induction base is strictly smaller than that of Case I (which uses the $Z_i^{z,m}$-induction hypothesis with one less known failure), and therefore the $Z_i^{z,m}$-induction base is reached with less known failures, i.e., with less than $\boldsymbol{t}-1$ known failures, i.e., the conditions of Part 2 hold at that point.

Finally, we consider the case in which $Z_i^{z,m} = \{i\}$. As any j as in Part 1 satisfies $j \in Z_i^{z,m}$, we have that the conditions of Part 2 hold, i.e., $F\langle i, m+1\rangle < t-1$. Furthermore, we have that $\langle z, m\rangle$ is not seen by $\langle i, m+1\rangle$ (otherwise, $z \in Z_i^{z,m}$). As $F\langle i, m+1\rangle < t-1 < n-2$, there exist two distinct processes $j, k \neq i$ that are not known to $\langle i, m+1\rangle$ to fail (and thus i, j, k, z are distinct). Thus, $\langle j, m\rangle$ and $\langle k, m\rangle$ are seen by $\langle i, m+1\rangle$.

By definition of j, k, there exists a run r' of Q, s.t. *1)* $r'_i(m+1) = r_i(m+1)$, *2)* k never fails in r', *3)* j fails at (r', m) before sending any messages, *4)* i fails at $(r', m+1)$, immediately after deciding but before sending any messages, and *5)* the faulty processes in r' are those known by $\langle i, m\rangle$ to fail in r, and in addition i and j. We note that by definition, $F\langle i, m+1\rangle$ is the same in r and r', even though the number of failures in r' is $F\langle i, m+1\rangle + 2$. We notice that there exists a run r'' of Q, s.t. *1)* $r''_k(m') = r'_k(m')$ for all m', *2)* k never fails in r'', *3)* $\langle z, m-1\rangle$ is seen by both $\langle i, m\rangle$ and $\langle j, m\rangle$ in r'', *4)* j fails at (r'', m) while successfully sending a message only to i (and therefore both $j \in Z_i^{z,m}$ and $F\langle i, m+1\rangle < \boldsymbol{t}-1$ in r''), and *5)* i fails at $(r'', m+1)$, immediately after deciding but before sending out any messages. By the proof for the case in which $Z_i^{z,m} \supsetneq \{i\}$ ($j \in Z_i^{z,m}$), i decides 0 at $(r'', m+1)$, and therefore k can never decide 1 during r'', and therefore neither during r'. As k never fails during r', by *Decision* it must thus decide 0 at some point during r'. Therefore, by *Uniform Agreement*, i cannot decide 1 before or at $m+1$ in r', and thus it does not decide 1 before or at $m+1$ in r, and the proof is complete. □

Now that we have established when processes must decide 0 in any protocol dominating $u-P_0$, we can deduce when processes cannot decide in any such protocol.

Lemma 14 *(No earlier decisions when $K_i\exists 0$) Let $Q \preceq u-P_0$ be a protocol that solves Uniform Consensus, let r be a run of Q, let m be a time, and let i be a process. If at time m in r we have $K_i\exists 0$, but $\neg K_i\exists\mathsf{correct}(0)$, then i does not decide at (r, m).*

Proof If $m = 0$, by Lemma 11 and since $\neg K_i\exists\mathsf{correct}(0)$ at $m = 0$ (even though $K_i\exists 0$), we have $\boldsymbol{t} > 0$. Thus, by Remark 1, i does not decide at (r, m). Assume henceforth, therefore, that $m > 0$.

As $\neg K_i\exists\mathsf{correct}(0)$, we have that $\neg K_i\exists 0$ holds at time $m-1$, by Lemma 11. Thus, there exists a process z s.t. $K_z\exists 0$ at m–1, and $\langle z, m-1\rangle$ is seen by $\langle i, m\rangle$. In turn, by Lemma 11, we have that $F\langle i, m\rangle < \boldsymbol{t}-1$. There exists a run r' of Q, s.t. *1)* $r'_i(m) = r_i(m)$, and *2)* the faulty processes in r' are those known by $\langle i, m\rangle$ to fail in r. We henceforth reason about r'. By definition of r', $F\langle i, m+1\rangle = F\langle i, m\rangle < \boldsymbol{t}-1$ (by definition, the value of $F\langle i, m\rangle$ is the same in both r and r'). Thus, by Part 2 of Lemma 13, i decides 0 at $(r', m+1)$, and hence i does not decide at (r', m), and therefore neither does it decide at (r, m). □

Lemma 15 *(No earlier decisions when $\neg K_i\exists 0$) Let $Q \preceq u-P_0$ be a protocol that solves Uniform Consensus, let r be a run of Q, let m be a time, and let i be a process. If there exists a hidden path w.r.t. $\langle i, m\rangle$ in r, and if at time m in r we have $\neg K_i\exists 0$, then i does not decide at (r, m).*

Proof sketch: We start by considering the more intuitive case of $F\langle i,m\rangle < t$. We first note that i considers possible at time m a run r' in which a 0 value "trickled down" the hidden path to a process z that is last in the path (and failed). Moreover, in r' this value was seen by a process j that never fails, and process i is about to fail in this round. By previous lemmas, it follows that in r' process j must decide on some value before or at time $m+1$. It is therefore enough to show that j does not decide on 1 up to time $m+1$ in this run, because it must then decide on 0. Now we note that since j sees $\langle z, m-1\rangle$, process j considers possible at time $m+1$ in r' a run r'' in which z in fact fails only at time m. By previous lemmas it follows that in r'' process z decides 0 at time m, so j cannot decide 1 in r''. Since j cannot distinguish between r' and r'' up to time $m+1$, it follows that j cannot decide 1 up to time $m+1$ in r', as desired.

The second case of $F\langle i,m\rangle = t$ is more technical, since the previous lemmas utilized in the first case all require $F\langle\cdot,\cdot\rangle < t$ for the process we are reasoning about at the time we are considering. That is, all of the lemmas require having an extra "possible failure" from the point of view of that process. The proof is therefore more involved and goes through a longer sequence of "runs considered possible" until, by careful bookkeeping, a run in which such a condition holds is reached, and the previous lemmas can be used.

Proof As $\neg K_i\exists 0$ at time m, then by *Validity*, i does not decide 0 at (r,m) (see also Lemma 2). Thus, it is enough to show that i does not decide 1 at (r,m) in order to complete the proof. If $m=0$, then by Remark 1, i does not decide 1 at (r,m) either. Assume henceforth, therefore, that $m>0$.

As there exists a hidden path w.r.t. $\langle i,m\rangle$, there exist a process $z\neq i$ such that $\langle z,m-1\rangle$ is hidden from $\langle i,m\rangle$. We claim that there is a process $j\neq i$ for which $\langle j,m-1\rangle$ is seen by $\langle i,m\rangle$. If $n=2$, then there cannot exist a hidden path w.r.t. $\langle i,m\rangle$ because time 0 is revealed to i at time 1, regardless of failures of the other process; thus, necessarily $n\geq 3$. Now, if $\forall j\neq i$, it is the case that $\langle j,m-1\rangle$ is not seen by $\langle i,m\rangle$, then time $m-1$ is revealed to $\langle j,m-1\rangle$, which contradicts the existence of a hidden path w.r.t. $\langle i,m\rangle$.

We first consider the case in which $F\langle i,m\rangle < \boldsymbol{t}$. In this case, there exists a run $r'=Q[\beta]$ of Q, s.t. all of the following hold in r':

- $r'_i(m)=r_i(m)$.
- z is the unique process that knows $\exists 0$ at $m-1$, and knows so then for the first time, either having initial value 0 (if $m=1$) or (as explained in the Nonuniform Consensus section) seeing only a single node that knows $\exists 0$ at $m-2$ (if $m>1$).
- z fails at $(r',m-1)$, successfully sending messages to all nodes except for i.
- The faulty processes in r' are those known by $\langle i,m\rangle$ to fail in r, and in addition i, which fails at time m without sending out any messages. In particular, j never fails.

We henceforth reason about r'. First, we note that $\langle j,m+1\rangle$ does not know that z fails at $m-1$ (as opposed to at m). As $\langle j,m\rangle$ sees $\langle z,m-1\rangle$, as $K_z\exists 0$ at $m-1$, and as j never fails, by Lemma 11 we have that $K_j\exists\mathsf{correct}(0)$ at $(r',m+1)$. Thus, j decides at time $m+1$ in $u-P_0[\beta]$, and so j must decide before or at $m+1$ in r'. As $r_i(m)=r'_i(m)$, then by *Uniform Agreement* it is enough to show that j does not decide 1 up to time $m+1$ in r' in order to complete the proof.

There exists a run r'' of Q, s.t. *1)* $r''_j(m+1)=r'_j(m+1)$, and *2)* the only difference between r'' and r' up to time m is that in r'', z fails only at time m, after deciding but without sending a message to j. By *Uniform Agreement*, it is enough to show that z decides 0 at (r'',m) in order to complete the proof.

We henceforth reason about r''. As z does not know at m that neither z nor i fail, we have $F\langle z,m-1\rangle \leq F\langle z,m\rangle < \boldsymbol{t}-1$. Thus, $\boldsymbol{t}>1$. If $m=1$, we therefore have by Part 2 of Lemma 12 that z decides 0 at (r'',m). Otherwise, $m>1$. As $K_z\exists 0$ at $m-1$ for the first time, as $\langle z,m-1\rangle$ sees only one node at $m-1$ that knows $\exists 0$, and as $F\langle z,m\rangle < \boldsymbol{t}-1$, by Lemma 11 we have $\neg K_z\exists\mathsf{correct}(0)$ at $m-1$. Thus, by Part 2 of Lemma 13 (for $i=z$), z decides 0 at (r'',m). In both of these cases, the claim holds.

We now consider the case in which $F\langle i,m\rangle = \boldsymbol{t}$. There exists a run $\hat{r}=Q[\beta]$ of Q, s.t. all of the following hold:

- $\hat{r}_i(m)=r_i(m)$.
- All processes k s.t. $\langle k,m-1\rangle$ is hidden from $\langle i,m\rangle$ (including $k=z$) know $\exists 0$ at $(\hat{r},m-1)$, either having initial value 0 (if $m=1$) or all seeing only a single node that knows $\exists 0$ at $m-2$ (and which fails at time $m-2$ without being seen by $\langle i,m\rangle$) — denote this node by z'.
- All such processes fail at time $m-1$, successfully sending messages to all nodes except for i.
- The faulty processes failing in $\hat{r}$ are those known by $\langle i,m\rangle$ to fail in r. In particular, there are $\boldsymbol{t}$ such processes.

We henceforth reason about $\hat{r}$. Note that i and j never fail in $\hat{r}$. Moreover, for each k s.t. $\langle k,m-1\rangle$ is hidden from $\langle i,m\rangle$, it is the case that $\langle k,m-2\rangle$ is seen by $\langle i,m-1\rangle$ (for otherwise $\langle k,m-1\rangle$ would not be hidden from $\langle i,m\rangle$). Thus, $F\langle i,m-1\rangle \leq F\langle j,m\rangle$ (equality can actually be shown to hold here, but we do not need it). As the number of nodes at $m-1$ knowing $\exists 0$ that are seen by $\langle j,m\rangle$ equals $F\langle i,m\rangle - F\langle i,m-1\rangle \geq \boldsymbol{t} - F\langle j,m\rangle$ (by the above remark, equality holds here as well), we have by Lemma 11 that $K_j\exists\mathsf{correct}(0)$ at m, and therefore j decides at time m in $u-P_0[\beta]$; thus, it must decide before or at m in $\hat{r}$. As

$r_i(m) = \hat{r}_i(m)$, by *Uniform Agreement* it is enough to show that j does not decide 1 up to time m in $\hat{r}$ in order to complete the proof.

We proceed with an argument similar in a sense to those of Part 1 of Lemma 12 and the inner induction in the proof of Lemma 13.

As $\langle z, m-1\rangle$ is seen by $\langle j, m\rangle$, there exists a run r'' of Q, s.t. *1)* $r''_j(m) = \hat{r}_j(m)$, and *2)* the only difference between r'' and $\hat{r}$ up to time m is that in r'', z never fails, but rather i fails at $m-1$ after sending a message to j but without sending a message to z. We note that there are $\boldsymbol{t}$ processes failing throughout r''. We henceforth reason about r''. If $m = 1$, then z has initial value 0 and if $m > 1$, then $\langle z, m-1\rangle$ sees $\langle z', m-2\rangle$; either way, by Lemma 11, $K_z\exists\mathsf{correct}(0)$ at (r'', m) and therefore z must decide before or at time m. Thus, it is enough to show that z does not decide 1 up to time m in r'' in order to complete the proof.

As $\langle i, m-1\rangle$ is not seen by $\langle z, m\rangle$, there exists a run $r^{(3)}$ of Q, s.t. *1)* $r^{(3)}_z(m) = r''_z(m)$, and *2)* the only difference between $r^{(3)}$ and r'' up to time m is that in $r^{(3)}$, $\langle i, m-1\rangle$ sees $\langle z', m-2\rangle$ (or, if $m = 1$, then the difference is that i has initial value 0); we note that $\langle i, m-1\rangle$ is still seen by $\langle j, m\rangle$. We note that there are $\boldsymbol{t}$ processes failing throughout $r^{(3)}$. Observe that the number of nodes at $m-1$ knowing $\exists 0$ that are seen by $\langle j, m\rangle$ in $r^{(3)}$ is greater than in each of r and $\hat{r}$ (between which j at m cannot distinguish), however $F\langle j, m\rangle$ remains the same in all three runs $\hat{r}$, r'' and $r^{(3)}$; thus, $K_j\exists\mathsf{correct}(0)$ at m in $r^{(3)}$ as well, and therefore j must decide before or at time m in $r^{(3)}$. Thus, it is enough to show that j does not decide 1 up to time m in $r^{(3)}$ in order to complete the proof. We henceforth reason about $r^{(3)}$.

As $\langle i, m-1\rangle$ is seen by $\langle j, m\rangle$, there exists a run $r^{(4)}$ of Q, s.t. *1)* $r^{(4)}_j(m) = r^{(3)}_j(m)$, and *2)* the only difference between $r^{(4)}$ and $r^{(3)}$ up to time m is that in $r^{(4)}$, i does not fail (and is thus seen by $\langle z, m\rangle$). We note that there are $\boldsymbol{t}-1$ processes failing throughout $r^{(4)}$, and thus in particular $F\langle z, m\rangle < \boldsymbol{t}$. If $m = 1$, then by Part 1 of Lemma 12 (for $i = z$ and $j = i$), z decides 0 in $(r^{(4)}, m)$. Otherwise, i.e., if $m > 1$, by Part 1 of Lemma 13 (for $i = z$, $z = z'$, and $j = i$), z decides 0 in $(r^{(4)}, m)$. Either way, the proof is complete. □

Directly from Lemmas 14 and 15, we deduce sufficient conditions for unbeatability of Uniform Consensus protocols dominating $u-P_0$; these conditions also become necessary if it can be shown that there exists some Uniform Consensus protocol dominating $u-P_0$ that meets them, as we indeed show momentarily for $u - Opt_0$.

Lemma 16 *A protocol $Q \preceq u-P_0$ that solves Uniform Consensus and in which a node $\langle i, m\rangle$ decides whenever any of the following hold at m, is an unbeatable Uniform Consensus protocol.*

- $K_i\exists\mathsf{correct}(0)$.
- *No hidden path w.r.t. $\langle i, m\rangle$ exists, and* $\neg K_i\exists 0$.

By Lemma 16, we have that if $u - Opt_0$ solves Uniform Consensus, then it does so in an unbeatable fashion.

Lemma 17 $u - Opt_0 \preceq u-P_0$.

Proof As explained above, at time $\boldsymbol{t}+1$ no hidden paths exist (see the proof of Lemma 8), and furthermore, by Lemma 3 we have at time $\boldsymbol{t}+1$ that $K_i\exists 0$ iff $K_i\exists\mathsf{correct}(0)$. The claim therefore holds by definition of $u - Opt_0$ and $u-P_0$. □

The unbeatability of $u - Opt_0$ directly follows from Theorem 4 and Lemma 16:

Theorem 5 *$u - Opt_0$ is an unbeatable Uniform Consensus protocol in γ^t_{cr}.*

5.3 Comparing $u - Opt_0$ to Previous Protocols

The fastest early-stopping protocol for Uniform Consensus in the literature, opt-EDAUC of [3] (a similar algorithm is in [9]), also stops in $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds at the latest. Similarly to Lemma 9, not only does $u - Opt_0$ strictly dominate opt-EDAUC, but furthermore, there are adversaries against which $u - Opt_0$ decides in 1 round, while opt-EDAUC decides in $\boldsymbol{t}+1$ rounds:

Lemma 18 *If $2 \le \boldsymbol{t} \le n-2$, then $u - Opt_0$ strictly dominates the opt-EDAUC protocol of [3]. Moreover, there exists an adversary for which $\mathsf{decide}_i(1)$ is performed after 1 round in $u - Opt_0$, and after $\boldsymbol{t}+1$ rounds in opt-EDAUC.*

Proof The proof has a similar structure to that of Lemma 9. opt-EDAUC decides either one round after the sender set repeats, or at time $\boldsymbol{t}+1$. As argued in the proof of Lemma 9, when the sender set repeats there is a round k all of whose nodes are revealed; if they don't contain evidence of an initial value of 0, then $u - Opt_0$ decides at round k. Otherwise, by Lemma 11(a) a correct process will know $\exists\mathsf{correct}(0)$ and decide one round later, and if this occurs at time $m = \boldsymbol{t}+1$, then by Lemma 11(b) it decides at time $\boldsymbol{t}+1$. An adversary β on which $u - Opt_0$ beats opt-EDAUC with the claimed margins is a simplified version of the adversary α defined in the proof of Lemma 9. Denote the processes by $\mathsf{Procs} = \{1, 2, \ldots, n\}$. All initial values in β are 0. In round 1, two processes crash—process 1 and process 2, with process 1 sending only to process n and nobody else, and process 2 sending to everyone except process n. No process fails in round 2, and in each of the rounds $m = 3, \ldots, \boldsymbol{t}$, process m crashes without sending any messages. Since precisely $\boldsymbol{t}$ processes fail in β we have that $\beta \in \mathsf{Crash}(\boldsymbol{t})$. For $3 \le m \le \boldsymbol{t}$, every correct process fails to hear from process m in round m for the first time. Every correct process $i \neq n$ fails to hear from process 1 in round 1 and from process 2 in round 2,

while process n fails to hear from 2 in round 1 and from process 1 in round 2. In the protocol opt-EDAUC of [3], no process decides before its sender set repeats, and thus all decisions are taken at time $\boldsymbol{t}+1$ when the adversary is β. In $u-Opt_0$, every correct process i sees $n-1 \geq \boldsymbol{t}+1$ values of 0 in the first round. By Lemma 11(b) it follows that $K_i \exists \mathsf{correct}(0)$ holds at time 1, the rule for $\mathsf{decide}_i(0)$ in $u-Opt_0$ is satisfied, and process i decides 0 at time 1. □

6 Efficient implementations and stopping times

In our analysis, all protocols are full-information, however, in fact, they can all be implemented in such a way that any process sends any other process a total of $O(n \log n)$ bits throughout any execution.

Moses and Tuttle in [21] show how to implement full-information protocols in the crash failure model with linear-size messages. In our case, a further improvement is possible, since decisions in all of the protocols depend only on the identity of hidden nodes and on the vector of initial values.

In a straightforward implementation, we can have a process i in each round report only the new information it learned in the previous round, or a *heartbeat* in case no new information was learned. More precisely, process i reports "`value(`j`) = v`" once for every j whose initial value it discovers, and "`failed_at(`j`) =` ℓ" once where ℓ is the earliest failure round it knows for j. In addition, it should send an "`I'm_alive`" message in every round in which it has nothing to report.

We argue that in every execution of such an implementation each process i sends a total of $O(n \log n)$ bits to any other process:

1. `I'm_alive` messages. Since all our protocols decide in at most $\boldsymbol{f}+1$ or $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds, Lemma 8 and Theorem 4, i sends `I'm_alive` fewer than $\boldsymbol{f}+2$ times. Thus, i sends $O(\boldsymbol{f})$ bits for `I'm_alive` messages as these messages are constant-size.
2. `value` messages. Clearly a process i sends at most one `value` message for every (faulty or correct) j. Since encoding j's ID requires $\log n$ bits and j's initial value requires one bit, we have that i sends $O(n \log n)$ bits for `value` messages.
3. `failed_at` messages. First we observe that i sends at most two `failed_at` messages for every faulty j: when j crashes, say at round m, processes might have uncertainty if j crashes at round m or $m+1$ (due to the failure pattern) but certainly it is not possible that a process sees j crashes before of after. Since encoding j's ID along with a failure round number $m \leq \boldsymbol{f}+2$ requires $O(\log n)$ bits, we obtain that a process i sends a total of $O(\boldsymbol{f} \log n)$ bits for `failed_at` messages.

Finally, we note that in our efficient implementation processes can perform the same decision tests in protocols Opt_0 and $u-Opt_0$, since decisions depend only on the times at which crashed nodes fail and on initial values.

Theorem 6 *For each of the protocols Opt_0 and $u-Opt_0$ there is a protocol with identical decision times for all adversaries, in which every process sends $O(n \log n)$ bits overall to each other process in any given run.*

We now consider the question when a process that has decided on a value can stop, thereby sending no further messages and terminating the protocol.

First consider the protocol Opt_0. Consider any execution and a process i deciding at some round. If i's decision is 0, it could be the case that i is correct in the execution and is the only one knowing the existence of a 0, hence it cannot terminate because all other correct processes would decide 1. Thus, it has to go one more round in order to communicate its decision to all correct processes. Interestingly, if i's decision is 1, i knows that there is no active process that knows the existence of a 0, hence it can terminate at the very same round as there is no way a correct process can decide 0 in any subsequent round. Therefore, in Opt_0, every process can terminate in at most $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds.

Now for protocol $u-Opt_0$, consider any execution and a process i deciding at some time m. In this case, regardless of its decision, i can stop at time m. The reason is that both deciding rules ensure that all active processes have the same information: the rule for deciding 1 is the same as in Opt_0, while the rule for 0 guarantees that at least a single process knows a 0, which implies that all active processes know there is a 0 preventing them to decide 1. It follows that processes can terminate no later than time

$\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds in Opt_0.

Theorem 7 *In protocols Opt_0 and $u-Opt_0$, processes can terminate in at most $\min(\boldsymbol{f}+2, \boldsymbol{t}+1)$ rounds in every execution.*

7 Alternative notions of unbeatability

It is possible to consider variations on the notion of unbeatability. One could, for example, compare runs in terms of the time at which the last correct process decides. We call the corresponding notion *last-decider unbeatability*. [5] This neither implies nor is implied by the notion of unbeatability

[5] This notion was suggested to us by Michael Schapira; we thank him for the insight.

studied so far in this paper. None of the consensus protocols in the literature is last-decider unbeatable. In fact, all of our protocols are also last-decider unbeatable:

Theorem 8 *The protocol Opt_0 is also last-decider unbeatable for consensus, while $u - Opt_0$ is last-decider unbeatable for uniform consensus.*

We note that Lemmas 9 and 18 show that our protocols beat the previously-known best ones by a large margin w.r.t. last-decider unbeatability as well.

Definition 5 (Last-decider domination and unbeatability) A decision protocol Q *last-decider dominates* a protocol P in γ, denoted by $Q \preceq_\gamma^{l.d.} P$, if for all adversaries α, if i the last decision in $P[\alpha]$ is at time m_i, then all decisions in $Q[\alpha]$ are taken before or at m_i. Moreover, we say that Q *strictly last-decider dominates* P if $Q \preceq_\gamma^{l.d.} P$ and $P \not\preceq_\gamma^{l.d.} Q$. I.e., if for some $\alpha \in \gamma$ the last decision in $Q[\alpha]$ is *strictly before* the last decision in $P[\alpha]$.

A protocol P is a *last-decider unbeatable* solution to a decision task S in a context γ if P solves S in γ and no protocol Q solving S in γ strictly last-decider dominates P.

Remark 2
- If $Q \preceq_\gamma P$, then $Q \preceq_\gamma^{l.d.} P$. (But not the other way around.)
- None of the above forms of strict domination implies the other.
- None of the above forms of unbeatability implies the other.

As the remark states, last-decider domination does not imply domination in the sense of this paper (on which our proofs is based). Nonetheless, the specific property of protocols dominating Opt_0 and $u - Opt_0$, which we use to prove that these protocols are unbeatable, holds also for protocols that only last-decider dominate these protocols.

Lemma 19 *1. Let $Q \preceq_\gamma^{l.d.} P_0$ satisfy Decision. If $K_i \exists 0$ at m in a run $r = Q[\alpha]$ of Q, then i decides in r no later than at m.*

2. Let $Q \preceq_\gamma^{l.d.} u - P_0$ satisfy Decision. If $K_i \exists \mathsf{correct}(0)$ at m in a run $r = Q[\alpha]$ of Q, then i decides in r no later than at m.

The main idea in the proof of Lemma 19 is to show that i considers it possible that all other active processes also know the fact stated in that part, and so they must all decide by the current time in the corresponding run of the dominated protocol. Hence, the last decision in that run is made in the current time; thus, by last-decider domination, i must decide. The proof of Lemma 19 appears in Appendix C.

As already explained, Theorem 8 directly follows from Lemma 19, and from the proofs of Theorems 3, 4 and 5.

8 Final discussion

8.1 More related work

In line of this work, [1] applies the notion of unbeatability to the well-known *k-set agreement* problem, a generalization of consensus in which the number of decisions is at most k (thus consensus is 1-set agreement). In the fault-tolerant synchronous message setting, that paper uses a similar knowledge-based analysis to obtain an unbeatable algorithm for nonuniform k-set agreement and an algorithm for uniform k-set agreement that strictly dominates all known solutions to this problem.

The Knowledge of Precondition Principle has also been helpful for obtaining better solutions to other agreement problems or understanding the power of models of computation. In [14], this principle is instrumental in obtaining communication efficient algorithms in synchronous settings by exposing communication patterns that enable communication through silence, which leads to a message-optimal protocol for the *atomic commitment* problem. Furthermore, with the use of the Knowledge of Precondition Principle, it is shown in [5] that the *bounded communication model without clocks*, a model introduced in that paper in which processes do not have clocks, but do have bounds on the duration of events and of communication, is more powerful than the fully asynchronous message-passing model.

8.2 Conclusions

Unbeatability is a natural optimality criterion for distributed protocols. It formalizes the intuition that a given protocol cannot be strictly improved upon, which is significantly stronger than saying that it is worst-case optimal, or even early stopping. Using a knowledge-based analysis guided by the Knowledge of Precondition Principle, we have obtained the first explicit unbeatable protocols for Nonuniform and Uniform Consensus. All of the protocols that we have presented have a very concise and intuitive description, and are efficiently implementable; thus, unbeatability is attainable at a modest price. Crucially, our unbeatable protocols can decide much faster than previously known solutions to the same problems (see [2] for a more detailed comparison with previous Uniform Consensus solutions). Finally, although we have focused on the binary version of consensus, our analysis can be easily extended to the *multivalued* version of the problem.

Acknowledgements Armando Castañeda was supported by PAPIIT projects IA102417 and IN108720. Yannai Gonczarowski was supported in part by ISF grant 230/10, by the Google Inter-university center for Electronic Markets and Auctions, by the European Research Council under the European Community's Seventh Framework Programme

(FP7/2007-2013) / ERC grant agreement no. [249159] and by an Adams Fellowship of the Israeli Academy of Sciences and Humanities. Yoram Moses is the Israel Pollak Academic chair at the Technion; his work was supported in part by ISF grants 1520/11 and 2061/19.

A Additional Proofs of Sect. 4

Proof of Lemma 1 Let P be a consensus protocol and let $R_P = R(P, \gamma^t_{\mathrm{cr}})$. Let $v \in \mathbb{V}$, let $r \in R_P$ and let $\langle i, m\rangle$ be a node s.t. i decides on v at time m in r.

We commence by proving (a). Assume for contradiction that no process has initial value v in r. By definition of γ^t_{cr}, there exists a run r' of P, s.t. *1)* $r'_i(m) = r_i(m)$, *2)* i does not fail in r', and *3)* The initial values in r' are the same as in r. As $r'_i(m) = r_i(m)$, we have that i decides on v at time m in r' as well. As the initial values in r' are the same as in r, we have that no process has initial value v in r'. As i does not fail in r', we therefore have that *Validity* does not hold regarding the decision of i in r'—a contradiction.

We move on to proving (b). Assume for contradiction that some process j decides $\bar{v}$ at some time $m' \leq m$ in r, and that j is active at m in r. Once again by definition of γ^t_{cr}, there exists a run r' of P, s.t. *1)* $r'_i(m) = r_i(m)$, *2)* $r'_j(m') = r_j(m')$, and *3)* neither i nor j fail in r'. As $r'_i(m) = r_i(m)$, we have that i decides on v at time m in r' as well; as $r'_j(m') = r_j(m')$, we have that j decides on $\bar{v}$ at time m' in r' as well. As neither i not j fail in r', we therefore have that *Agreement* does not hold in r' — a contradiction. □

The proof of Lemma 3 is assisted by Definition 6 and Lemma 20:

Definition 6 Let P be a protocol in γ^t_{cr} and let $r \in R_P = R(P, \gamma^t_{\mathrm{cr}})$. Let $v \in \mathbb{V}$ and let $\langle i, m\rangle$ be a node. We say that *there is a v-chain for $\langle i, m\rangle$ in the run r* if, for some $d \leq m$, there is a sequence $j_0, j_1, \ldots, j_d = i$ of distinct processes, such that $v_{j_0} = v$ and for all $1 \leq k \leq d$, the process j_k receives a message from j_{k-1} at time k in r.

Lemma 20 *Let P be a fip in γ^t_{cr} and let $r \in R_P = R(P, \gamma^t_{\mathrm{cr}})$. Then, for every processes i and time $m \geq 0$, it is the case that $(R_P, r, m) \models K_i \exists 0$ iff there is a 0-chain for $\langle i, m\rangle$ in r.*

Proof For the first direction, assume that there is a 0-chain $j_0, \ldots, j_d = i$ for $\langle i, m\rangle$ in r. It is easy to show by induction that $K_{j_k} \exists 0$ at k in r for every k; therefore, $K_i \exists 0$ at d in r, and since P is a fip, $K_i \exists 0$ at m in r, as required. We prove the second direction for all i by induction on m.

Base ($m = 0$): Since process i at time 0 knows no initial value but its own, we have that $v_i = 0$ and so i (with $d = 0$) is a 0-chain as required.

Inductive step ($m > 0$): In a fip, $K_i \exists 0$ at m implies that either $K_i \exists 0$ at $m-1$ or $K_j \exists 0$ at $m-1$ for some $j \neq i$ that successfully sends a message at time $m-1$ to j. If $K_i \exists 0$ at $m-1$, then by the induction hypothesis there exists a 0-chain for $\langle i, m-1\rangle$ in r, and by definition this is also a 0-chain for $\langle i, m\rangle$ in r. It remains to consider the case in which $K_i \exists 0$ does not hold at $m-1$; therefore, $K_j \exists 0$ at $m-1$ for some j that successfully sends a message at time $m-1$ to j. By the induction hypothesis, there exists a 0-chain $j_0, \ldots, j_d = j$ for $\langle j, m-1\rangle$. We first claim that i does not appear in that chain; indeed, if $j_{d'} = i$ for some $d' < d$, then by definition $j_0, \ldots, j_{d'}$ would be a 0-chain for $\langle i, m-1\rangle$, and by the previous direction we would have $K_i \exists 0$ at $m-1$ in r. We now claim that $d = m-1$; indeed, if $d < m-1$, then $j_0, \ldots, j_d$ would be a 0-chain for $\langle j, d\rangle$, and so we would have $K_j \exists 0$ at $d < m-1$. As j is active at all times earlier than $m-1$, we would have that $\langle j, d\rangle$ successfully sends a message to i, and so $K_i \exists 0$ at $d+1 \leq m-1$; as P is a fip, we would therefore have that $K_i \exists 0$ at $m-1$ — a contradiction. As i does not appear in $j_0, \ldots, j_d$, and as $d = m-1$, by definition $j_0, \ldots, d_j, i$ is a 0-chain for i, as required. □

Proof of Lemma 3 Assume that $(R_P, r, \boldsymbol{t}+1) \models K_i \exists v$. By Lemma 20, there exists a 0-chain $j_0, \ldots, j_d$ for $\langle i, \boldsymbol{t}+1\rangle$. If j appears in $j_0, \ldots, j_d$, then by Lemma 20 we are done; assume, therefore, that j does not appear in $j_0, \ldots, j_d$. If $d < \boldsymbol{t}+1$, then since i successfully sends all messages at times earlier than $\boldsymbol{t}+1$, we have that $j_0, \ldots, j_d, j$ is a 0-chain for $\langle j, \boldsymbol{t}+1\rangle$; therefore, by Lemma 20, $K_j \exists v$ at $\boldsymbol{t}+1$, as required. Otherwise, $d = \boldsymbol{t}+1$, and so, as $j_0, \ldots, j_{d-1}$ are $\boldsymbol{t}+1$ distinct processes, there exists $0 \leq d' \leq d-1$ s.t. $j_{d'}$ is nonfaulty throughout r. Therefore, $j_0, \ldots, j_{d'}, j$ is a 0-chain for $\langle j, \boldsymbol{t}+1\rangle$, as required. □

Proof of Lemma 5 $\Longrightarrow$: Assume that $(R_P, r, m) \not\models K_i$ not_known$(\exists 0)$. Therefore, by definition of K_i, there exists a run $r' \in R_P$ s.t. *1)* $r'_i(m) = r_i(m)$, and *2)* $(R_P, r', m) \not\models$ not_known$(\exists 0)$. As $(R_P, r', m) \not\models$ not_known$(\exists 0)$, there exists a process j s.t. $K_j \exists 0$ holds at m in r' (and j is active at m in r'). By definition, $K_j \exists 0$ first holds at or before time m in r', and so j decides 0 before or at time m in r'; therefore, $(R_P, r', m) \not\models$ none_decided(0). As $r'_i(m) = r_i(m)$, we therefore have $(R_P, r, m) \not\models K_i$none_decided$(0)$, as required.

$\Longleftarrow$: We will show that $(R_P, r, m) \models$ not_known$(\exists 0)$ implies that $(R_P, r, m) \models$ none_decided(0); by definition of knowledge, it will then follow that $(R_P, r, m) \models K_i$not_known$(\exists 0)$ implies $(R_P, r, m) \models K_i$none_decided(0). Assume, therefore, that $(R_P, r, m) \models$ not_known$(\exists 0)$, and let j be a process that is active at time m in r. As not_known$(\exists 0)$ at m in r, we have that $K_j \exists 0$ does not hold at m in r. As P is a fip, we have that neither does $K_j \exists 0$ hold at any time prior to m in r. By definition, therefore j does not decide 0 before or at m in r, as required. □

B Proof of Theorem 4

Decision: In some run of $u - Opt_0$, let i be a process and let m be a time s.t. i is active at m but has not decided until m, inclusive. Let $\tilde{m} \leq m$ be the latest time not later than m s.t. a hidden path exists w.r.t. $\langle i, \tilde{m}\rangle$. We claim that as i is undecided at m, we have $\tilde{m} \geq m-1$; indeed, otherwise, by i being undecided at $\tilde{m}+1$ despite the absence of a hidden path w.r.t. $\langle i, \tilde{m}+1\rangle$, we would have $K_i \exists 0$ at $\tilde{m}+1$, and so, by Lemma 11, we would have $K_i \exists \mathsf{correct}(0)$ at $\tilde{m}+2 \leq m$—a contradiction to i being undecided at m.

In Definition 4, for a node $\langle i, m\rangle$, we denote by $F\langle i, m\rangle \in \{0, \ldots, t\}$ the number of failures known to $\langle i, m\rangle$, i.e., the number of processes $j \neq i$ from which i does not receive a message at time m.

As a hidden path exists w.r.t. $\langle i, \tilde{m}\rangle$, we have, as in the proof of Lemma 8, that $\tilde{m} \leq \boldsymbol{f}$; in fact, the same proof shows the even stronger claim $\tilde{m} \leq F\langle i, \tilde{m}\rangle$ — we we will later return to this inequality. As $\tilde{m} \leq \boldsymbol{f}$, we therefore have that $m \leq \tilde{m}+1 \leq \boldsymbol{f}+1$. We thus have that every process that is active at time $\boldsymbol{f}+2$, decides by this time at the latest.

Before moving on to show *Validity* and *Uniform Agreement*, we first complete the analysis of stopping times. Assume that $m = \boldsymbol{f}+1$. (i is still a process that is active but undecided at m.) As $\boldsymbol{f} = m-1 \leq \tilde{m} \leq F\langle i, \tilde{m}\rangle \leq F\langle i, m\rangle \leq \boldsymbol{f}$, we have that both $\tilde{m} = m-1$ and $F\langle i, m\rangle = \boldsymbol{f}$. As $\tilde{m} = m-1$, we have that no hidden path exists w.r.t. $\langle i, m\rangle$. As i is undecided at m, we thus have, by definition of $u - Opt_0$ and the fact that a time $\leq m$ has been revealed to $\langle i, m\rangle$, that $K_i \exists 0$ while $\neg K_i \exists \mathsf{correct}(0)$ at m. We therefore have that $K_i \exists 0$ at m for the first time. Therefore, as $m > \tilde{m} \geq 0$, there exists a process j such that $K_j \exists 0$ at $m-1$ and s.t. $\langle j, m-1\rangle$ is seen by $\langle i, m\rangle$. Thus, by Lemma 11 and since $\neg K_i \exists \mathsf{correct}(0)$, we have $F\langle i, m\rangle < \boldsymbol{t}-1$, and so $\boldsymbol{f} = F\langle i, m\rangle < \boldsymbol{t}-1$.

We thus have that if $\boldsymbol{f} = \boldsymbol{t}-1$, then every process that completes round $\boldsymbol{f}+2$ decides by time $\boldsymbol{f}+1$ at the latest.

We move on to show *Validity* and *Uniform Agreement*. Henceforth, let i be a (possibly faulty) process that decides in some run of $u - Opt_0$, let m be the decision time of i, and let v be the value upon which i decides.

Validity: If $v = 0$, then by definition $K_i \exists \mathsf{correct}(0)$ at m, and so $K_i \exists 0$ at m, and in particular $\exists 0$. If $v = 1$, then by definition $\neg K_i \exists 0$, and so the initial value of i is 1, and so $\exists 1$. Either way, we have $\exists v$ as required.

Uniform Agreement: If $v = 0$ then i decides 0 at the first time m such that $K_i \exists \mathsf{correct}(0)$ holds. From Lemma 11 we get that $m \geq 1$ and for every process j that is active at time m, it holds that $K_j \exists 0$ at time m, at the latest. Therefore, no process decides 1. We now show that if $v = 1$, then 0 is never decided upon in the current run. For the rest of this proof we assume, therefore, that $v = 1$; therefore, by definition of $u - Opt_0$, we have that both $\neg K_i \exists 0$ and no hidden path exists w.r.t. $\langle i, m\rangle$. By Lemma 7, we thus have that $K_i \mathsf{not\_known}(\exists 0)$ at m, and in particular $\mathsf{not\_known}(\exists 0)$ at m. By a trivial induction, as in the proof of Theorem 2, we have that $\mathsf{not\_known}(\exists 0)$ at every time later than m. In particular, we have that no correct process ever learns of an initial value of 0 (as $\mathsf{not\_known}(\exists 0)$ would never hold from that point on), and so $\exists \mathsf{correct}(0)$ never holds; therefore, $K_j \exists \mathsf{correct}(0)$ never holds for any j, and so by definition of $u - Opt_0$ no process ever decides upon 0, and the proof is complete.

C Proof of Lemma 19

We first prove Part 1; If $m = 0$, then there exists a run $r' = Q[\beta]$ of Q, s.t. *1)* $r'_i(0) = r_i(0)$, *2)* in r' all initial values are 0, and *3)* i never fails in r'. Hence, in $P_0[\beta]$ all decisions are taken at time $m = 0$, and therefore so is the last decision. Therefore, the last decision in r' must be taken at time 0. As i never fails in r', by *Decision* it must decide at some point during this run, and therefore must decide at 0 in r'. As $r_i(0) = r'_i(0)$, i decides at 0 in r as well, as required.

If $m > 0$, then there exists a process j s.t. $K_j \exists 0$ at $m-1$ in r and $\langle j, m-1\rangle$ is seen by $\langle i, m\rangle$. Thus, there exists a run $r' = Q[\beta]$ of Q, s.t. *1)* $r'_i(m) = r_i(m)$, and *2)* i and j never fail in r'. Thus, all processes that are active at m in r' see $\langle j, m-1\rangle$ in r' and therefore know $\exists 0$ in r'. Hence, in $P_0[\beta]$ all decisions are taken by time m, and therefore so is the last decision. Therefore, the last decision in r' must be taken no later than at time m. As i never fails in r', by *Decision* it must decide at some point during this run, and therefore must decide by m in r'. As $r_i(m) = r'_i(m)$, i decides by m in r as well, as required.

We now prove Part 2. If $m = 0$, then by Lemma 11, $\boldsymbol{t} = 0$. There exists a run $r' = Q[\beta]$ of Q, s.t. *1)* $r'_i(0) = r_i(0)$, and *2)* in r' all initial values are 0. Therefore, as $\boldsymbol{t} = 0$, we have by Lemma 11 that all processes know $\exists \mathsf{correct}(0)$ at $m = 0$ in r'. Hence, in $u - P_0[\beta]$ all decisions are taken at time $m = 0$, and therefore so is the last decision. Therefore, the last decision in r' must be taken at time 0 as well. Since $\boldsymbol{t} = 0$, i never fails in r', and so by *Decision* it must decide at some point during this run, and therefore must decide at 0 in r'. As $r_i(0) = r'_i(0)$, i decides at 0 in r as well, as required.

If $m > 0$, then there exists a process j s.t. $K_j \exists 0$ at $m-1$ in r and $\langle j, m-1\rangle$ is seen by $\langle i, m\rangle$ in r. Furthermore, as $\boldsymbol{t} < n$, there exists a set of processes I s.t. *1)* $i, j \notin I$, *2)* $|I| = t - F\langle i, m\rangle - 1$, and *3)* $\langle k, m-1\rangle$ is seen by $\langle i, m\rangle$ for every $k \in I$. Thus, there exists a run $r' = Q[\beta]$ of Q, s.t. *1)* $r'_i(m) = r_i(m)$, *2)* i and j never fail in r', *3)* all of I fail in r' at $m-1$, successfully sending messages only to i, and *4)* every process at $m-1$ in r' that is not seen by $\langle i, m\rangle$, is not seen by any other process at m as well. We henceforth reason about

r'. Every process $k \neq j$ that is active at m sees $\langle j, m-1\rangle$ and furthermore satisfies $F\langle k, m\rangle \geq F\langle i, m\rangle + |I| = \boldsymbol{t} - 1$. Thus, by Lemma 11, $K_k \exists \mathsf{correct}(0)$ at m, and thus k decides at time m in $u-P_0[\beta]$. Additionally, as $K_j \exists 0$ at $m-1$, by Lemma 11 $K_j \exists \mathsf{correct}(0)$ at m, and thus j decides at time m in $u-P_0[\beta]$. Hence, in $u-P_0[\beta]$ all decisions are taken by time m, and therefore so is the last decision. Therefore, the last decision in r' must be taken no later than at time m. As i never fails in r', by *Decision* it must decide at some point during this run, and therefore must decide by m in r'. As $r_i(m) = r'_i(m)$, i decides by m in r as well, as required.

References

1. A. Castañeda, Y. A. Gonczarowski, and Y. Moses. Unbeatable set consensus via topological and combinatorial reasoning. In: proceedings of the 2016 ACM symposium on principles of distributed computing, PODC 2016, Chicago, IL, USA, July 25-28, pp. 107–116, (2016)
2. A. Castañeda, Y. Moses, M. Raynal, and M. Roy. Early decision and stopping in synchronous consensus: a predicate-based guided tour. In: proceedings 5th international conference networked systems, NETYS 2017, Marrakech, Morocco, May 17-19, pp. 206–221, (2017)
3. Charron-Bost, B., Schiper, A.: Uniform consensus is harder than consensus. J. Algorithms **51**(1), 15–37 (2004)
4. B. Coan. A communication-efficient canonical form for fault-tolerant distributed protocols. In: proceedings 5th ACM symposium on principles of distributed computing, pp. 63–72, (1986)
5. A. Dan, R. Manohar, and Y. Moses. On using time without clocks via zigzag causality. In: proceedings of the ACM symposium on principles of distributed computing, PODC 2017, Washington, DC, USA, July 25-27, pp. 241–250, (2017)
6. D. Dolev. Beep protocols (personal communication)
7. Dolev, D., Reischuk, R., Strong, H.R.: Early stopping in Byzantine agreement. J. ACM **34**(7), 720–741 (1990)
8. D. Dolev and H. R. Strong. Requirements for agreement in a distributed system. In: H. J. Schneider, editor, Distributed data bases, pp. 115–129. North-Holland, (1982)
9. Dutta, P., Guerraoui, R., Pochon, B.: The time-complexity of local decision in distributed agreement. SIAM J. Comput. **37**(3), 722–756 (2007)
10. Dwork, C., Moses, Y.: Knowledge and common knowledge in a Byzantine environment: crash failures. Inf. Comput. **88**(2), 156–186 (1990)
11. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, London (2003)
12. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty processor. J. ACM **32**(2), 374–382 (1985)
13. Gafni, E., Guerraoui, R., Pochon, B.: The complexity of early deciding set agreement. SIAM J. Comput. **40**(1), 63–78 (2011)
14. Goren, G., Moses, Y.: A preliminary version appeared in PODC 2018. Silence. J. ACM **67**(1), 1–26 (2020)
15. V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In: fault-tolerant distributed computing, pp. 201–208, (1986)
16. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. A preliminary version appeared in PODC, 1984. J. ACM **37**(3), 549–587 (1990)
17. Halpern, J.Y., Moses, Y., Waarts, O.: A characterization of eventual byzantine agreement. SIAM J. Comput. **31**(3), 838–865 (2001)
18. M. Herlihy, Y. Moses, and M. R. Tuttle. Transforming worst-case optimal solutions for simultaneous tasks into all-case optimal solutions. In PODC, pp. 231–238, (2011)
19. Keidar, I., Rajsbaum, S.: A simple proof of the uniform consensus synchronous lower bound. Inf. Process. Lett. **85**(1), 47–52 (2003)
20. Y. Moses. Relating knowledge and coordinated action: the knowledge of preconditions principle. In: proceedings fifteenth conference on theoretical aspects of rationality and knowledge, TARK 2015, Carnegie Mellon University, Pittsburgh, USA, June 4-6, 2015., pp. 231–245, (2015)
21. Moses, Y., Tuttle, M.R.: Programming simultaneous actions using common knowledge. Algorithmica **3**, 121–169 (1988)
22. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
23. M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In: SPAA, pp. 302–310. ACM Press, (2004)
24. Wang, X., Teo, Y.M., Cao, J.: A bivalency proof of the lower bound for uniform consensus. Inf. Process. Lett. **96**(5), 167–174 (2005)