

# Lower Bounds for Asynchronous Consensus

Leslie Lamport

28 July 2004

Revised 20 August 2005

Minor corrections made 19 January 2006

MSR-TR-2004-72

Microsoft Research  
Microsoft Corporation  
One Microsoft Way  
Redmond, WA 98052

<http://www.research.microsoft.com>

### **Abstract**

Impossibility results and best-case lower bounds are proved for the number of message delays and the number of processes required to reach agreement in an asynchronous consensus algorithm that tolerates non-Byzantine failures. General algorithms exist that achieve these lower bounds in the normal case, when the response time of non-faulty processes and the transmission delay of messages they send to one another are bounded. Our theorems allow algorithms to do better in certain exceptional cases, and such algorithms are presented. Two of these exceptional algorithms may be of practical interest.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Traditional Consensus . . . . .	2
1.2	Agents . . . . .	3
1.3	Results . . . . .	5
1.4	Proofs . . . . .	6
<b>2</b>	<b>Theorems</b>	<b>7</b>
2.1	Scenarios and Algorithms . . . . .	7
2.2	The Lower Bound on Acceptors . . . . .	10
2.3	Fast Learning . . . . .	11
2.4	Collision-Fast Learning . . . . .	15
2.5	Hyperfast Learning . . . . .	19
<b>3</b>	<b>Algorithms</b>	<b>20</b>
3.1	The Lower Bound on Acceptors . . . . .	20
3.2	Fast, Collision-Fast, and Hyperfast Learning . . . . .	21
3.2.1	Fast Learning . . . . .	21
3.2.2	Collision-Fast Learning . . . . .	22
3.2.3	Hyperfast Learning . . . . .	25
<b>4</b>	<b>Conclusion</b>	<b>25</b>
	<b>References</b>	<b>26</b>
	<b>Appendix</b>	<b>27</b>
<b>A</b>	<b>Proofs</b>	<b>27</b>
A.1	The Accepting Lemma . . . . .	29
A.2	The Acceptor Lower Bound Theorem . . . . .	30
A.3	The Fast Accepting Lemma . . . . .	32
A.4	The Fast Learning Theorem . . . . .	35
A.5	The Collision-Fast Learning Theorem . . . . .	41
A.6	The Hyperfast Learning Theorem . . . . .	49
<b>B</b>	<b>Formal Statements of the Results</b>	<b>52</b>

# 1 Introduction

In an asynchronous system, how many processors are needed to achieve consensus in the presence of  $f$  (non-Byzantine) faults? And how fast can they achieve it? The answers are known:  $2f + 1$  processors and two message delays [2]. These are correct answers, but to the wrong questions. The traditional definition of consensus hides one message that occurs in most applications—namely, the initial message from a client to the processes implementing the consensus algorithm.

Here, we answer the questions *How many processes?* and *How many message delays?* for a more pertinent definition of consensus. For an  $n$ -process algorithm that can reach consensus even if  $f$  processes fail, the approximate answers are:

- Consensus is possible only if  $n > 2f$ .
- In the absence of conflicting requests from different clients, consensus can be achieved in two message delays despite the failure of  $e$  processes, for  $0 \leq e \leq f$ , only if  $n > 2e + f$ .
- Consensus cannot be achieved in fewer than two message delays, nor can it be guaranteed in two message delays if there are conflicting user requests.

Each of these answers is false in certain exceptional cases. For example, consensus can be achieved in two message delays in the absence of failure despite conflict if the only clients are the processes that execute the consensus algorithm.

Our exact answers to these questions consist of several precisely stated and rigorously proved lower-bound results. Algorithms exist that show the bounds to be tight. For the normal cases, those algorithms appear elsewhere. We provide algorithms for the exceptional cases. Two of those algorithms may be useful in practice.

We assume only non-Byzantine (omission) failures. This means that a process can fail by stopping, not by performing incorrect actions; and a message can be lost or delivered multiple times, but cannot be (undetected) corrupted.

Approximate versions of extensions to most of these results were previously announced [10]. The extensions treated Byzantine as well as non-Byzantine failures. We would assert that precise statements of the extended versions and their proofs will appear, but experience has shown the foolhardiness of such a prediction.

## 1.1 Traditional Consensus

We begin by examining the traditional consensus problem. One assumes a collection of  $n$  processes, each of which can propose a value. The problem is to find an algorithm for choosing a value subject to the following conditions, where  $f$  is the number of faults to be tolerated.

**Nontriviality** Only a proposed value may be chosen.

**Consistency** Only a single value may be chosen.

**Progress** If at least  $n - f$  processes are nonfaulty, then a value must be chosen and must be learned by every nonfaulty process.

The well-known result of Fischer, Lynch, and Paterson (FLP) [5] implies that an asynchronous algorithm cannot solve this problem in the presence of even a single fault. But most real systems are synchronous at least most of the time, meaning that nonfaulty processes can usually perform actions and communicate with one another in a bounded length of time. An asynchronous consensus algorithm is required to maintain nontriviality and consistency in the presence of any non-Byzantine failures; progress need be guaranteed only under the assumption that the system is eventually synchronous for a long enough period of time [4].

The cost of consensus we consider is the number of message delays between the proposal of a value and the learning of a value. If local computation were instantaneous and messages were received exactly one second after they were sent, then this would be the number of seconds between the first proposal of a value and the learning of a value by every non-faulty process. The general definition of the number of message delays in an asynchronous system is fairly obvious [6]. The FLP result implies that the worst-case cost for achieving consensus is unbounded. Fortunately, it is the best-case cost that is interesting. A good algorithm will achieve the best-case cost in the normal case of synchronous behavior.

The cost of a consensus algorithm is important only when a system executes multiple instances of the algorithm. If a system executed the algorithm only a few times—for example, during start-up—the algorithm’s efficiency wouldn’t matter. In the state-machine approach [6], a sequence of instances of a consensus algorithm are used to choose the sequence of commands.

Perhaps the most widely-known consensus algorithm is the Paxos algorithm [3, 8, 9]. It has a preliminary section consisting of the election of a leader and the leader’s execution of phase 1 for a new ballot number. This

phase is executed only when a leader fails, and it can be executed simultaneously for all instances of the algorithm. The cost of the preliminary section can therefore be ignored. The significant cost of the Paxos algorithm is that of phase 2. In that phase, the leader proposes a value by sending phase 2a messages with that value to all the processes, which respond with phase 2b messages. The leader learns that the value has been chosen when it receives phase 2b messages from a majority of the processes. Hence, if a majority of processes are nonfaulty, then the leader learns the chosen value in two message delays. By having each process send its phase 2b message to every other process, every nonfaulty process can also learn the chosen value in two message delays. Paxos is therefore optimal for solving the traditional consensus problem. But it is not necessarily optimal for implementing a system that uses consensus.

## 1.2 Agents

In the state-machine approach, a set of servers execute a sequence of instances of a consensus algorithm to choose a sequence of client commands. In Paxos, a client sends its command to the leader, and the leader proposes that command in the next instance of the Paxos consensus algorithm. By considering only the cost of the consensus algorithm, we are ignoring the message sent by the client to the leader. In other applications of consensus as well, the proposed values need not be generated by the processes that choose a value. So, instead of defining consensus in terms of a single set of processes, we define it more generally in terms of three fixed sets of agents:

**Proposers** A proposer can propose a value.

**Acceptors** The acceptors cooperate to choose a value.

**Learners** A learner can learn what value has been chosen.

These sets need not be disjoint. For example, an agent might be both a proposer and an acceptor. The traditional statement of consensus corresponds to the case in which these three sets are equal. This situation represented the class of process-control systems that originally inspired the consensus problem [14]. However, it is not typical of applications of consensus in asynchronous systems. For example, a state-machine implementation that can tolerate the failure of  $f$  computers needs only  $f + 1$  copies of the machine's state. However,  $2f + 1$  computers are required to achieve consensus on the state-machine commands. A fault-tolerant state-machine implementation

requires  $2f + 1$  computers to act as acceptors, but only the  $f + 1$  of them that maintain the state-machine's state have to be learners.

By choosing what computers play what roles, we can make tradeoffs between time and message complexity. For example, by letting clients as well as servers be learners, we can reduce the number of message delays between when the client issues a command and when it receives a response. However, this reduction comes at the cost of extra messages.

We now generalize the consensus problem to a system of agents performing these three roles. The generalizations of nontriviality and consistency are obvious:

**Nontriviality** Only a proposed value may be learned.

**Consistency** Any two values that are learned must be equal.

The generalization of progress is less obvious. We want our definition of consensus to apply to client/server systems, in which clients are not necessarily reliable. For example, a client might issue a command and then “disappear”. Since proposer and learner are roles that might be assigned to a client, we cannot require that they not fail. We can make reliability assumptions only about acceptors. Let  $n$  be the total number of acceptors, which we assume to be finite. A consensus algorithm is said to be  $f$ -fault tolerant iff only  $n - f$  nonfaulty acceptors are needed to ensure that a value is chosen. (Remember that a consensus algorithm must maintain nontriviality and consistency despite any number of failures;  $f$ -fault tolerance means only that it must make progress if no more than  $f$  acceptors have failed.) However, no value can be chosen if none is proposed. Moreover, we cannot expect the acceptors to find out about a proposal if the proposer failed immediately after issuing it. So, we can require a value to be chosen only if there is a proposal issued by a nonfaulty proposer. We naturally require only nonfaulty learners to learn. This all leads to the following requirement for a consensus algorithm that tolerates  $f$  faults:

**Progress** For any proposer  $p$  and learner  $l$ , if  $p$ ,  $l$ , and  $n - f$  acceptors are nonfaulty and  $p$  proposes a value, then  $l$  must learn a value.

As in the traditional consensus problem, progress can be guaranteed only under some synchrony assumption. Lower bounds applying only to purely asynchronous algorithms that guarantee progress would therefore be vacuous. However, we prove lower bounds for any algorithm that *can* make progress. These bounds apply *a fortiori* to any algorithm that, under some other assumption such as eventual synchrony [4], *must* make progress.

### 1.3 Results

We now sketch our lower-bound results for asynchronous consensus algorithms. The exposition here is very informal and not completely accurate. It is only an intuitive introduction to our results, which are stated more precisely in Section 2. Our first two theorems were announced previously without proof [10].

Consensus becomes trivial if there is only one proposer or one learner. (In either case, a proposer can simply send its proposal directly to a learner, which learns the first proposal it receives.) So, we assume that there are at least two proposers and two learners.

A *quorum* is a set  $Q$  consisting of enough acceptors to choose a value. Our first result is the following theorem about the size of quorums. Its anomalous case, as well as the exceptional cases of other theorems, are described in the precise statements of the theorems in Section 2.

**Theorem (Acceptor Lower Bound)** For any consistent asynchronous algorithm with  $n$  acceptors, if any set of  $n - f$  acceptors is a quorum then  $n > 2f$ , except in one anomalous case with exactly three agents in which  $n = 2$  and  $f = 1$ .

For Paxos and most of the asynchronous consensus algorithms that have been proposed, a quorum consists of any majority of the servers. It is therefore easy to see that the acceptor lower bound is tight for the normal case. Section 3.1 gives an algorithm that works in the anomalous case.

An execution of a consensus algorithm is said to be *fast-learning* iff there is at most a two-message delay between the proposal of a value and the learning of the value. The FLP result implies that an asynchronous consensus algorithm must have non-terminating executions, so no algorithm can guarantee learning in any fixed number of message delays. The best we can hope for a fast-learning asynchronous algorithm is that it is fast for certain “good” executions, in which messages are delivered in a timely fashion. Our lower bound for fast learning assumes only that fast learning is possible.

It isn’t hard to find an algorithm that allows fast learning for a single proposer. In particular, the Paxos algorithm allows fast learning of proposals issued by the leader. We therefore consider algorithms that are fast for at least two proposers. We define a set of acceptors to be *fast-accepting* for a proposer  $p$  if it allows  $p$ ’s proposals to be learned quickly by any learner. (Having a hypothesis that *allows* rather than *requires* fast learning makes our lower-bound result stronger.) For an algorithm with  $n$  acceptors to



tolerate  $f$  faults, every set of  $n - f$  acceptors should be a quorum. However, we do not require every set of  $n - f$  acceptors to be fast-accepting. Instead, we consider algorithms that are fast-learning when there are at most  $e$  faulty acceptors, where  $0 \leq e \leq f$ . If  $e < f$ , such an algorithm may slow down when  $e + 1$  failures have occurred. Our second result is:

**Theorem (Fast Learning)** For any asynchronous consensus algorithm with  $n$  acceptors, and any  $e$  and  $f$  with  $e \leq f$  and  $f > 0$ , if any set of  $n - f$  acceptors is a quorum and there are two proposers for which any set of  $n - e$  acceptors is fast-accepting, then  $n > 2e + f$ , except in one special case.

Fast Paxos, a variant of Paxos, achieves this lower bound [12]. The basic idea behind Fast Paxos was originally observed by Brasileiro et al. [1]. The Generic Broadcast algorithm of Pedone and Schiper [13] is a fast-learning algorithm for the case  $e = f$ .

The bound for the normal case is therefore tight. The special case is described in Section 2.3, and Section 3.2.1 gives a fast-learning algorithm for it.

Two message delays is a best-case lower bound on learning for a general algorithm. It is achieved by Fast Paxos only in the absence of collisions, where a collision occurs when two proposers concurrently propose different values. The Collision-Fast Learning Theorem stated in Section 2.4 asserts that no general consensus algorithm can be fast-learning in the presence of collisions. However, there are two potentially useful cases in which fast learning is possible despite collisions. Algorithms for those cases are given in Section 3.2.2.

We have stated that two message delays is a lower bound on the time to reach consensus. That is not quite true. We define hyperfast learning to mean learning within one message delay. It is not hard to see that no algorithm can provide hyperfast learning for two different proposers. However, there are certain special cases in which hyperfast learning for a single proposer is possible. The Hyperfast Learning theorem of Section 2.5 enumerates those cases, and Section 3.2.3 provides the corresponding hyperfast-learning algorithms.

## 1.4 Proofs

It is not hard to write convincing informal proofs of our theorems. Nor is it hard to write such proofs for incorrect versions of the theorems. Since the exceptional cases of the theorems lead to algorithms that may be interesting,

we feel it is important to ensure that we have correctly characterized those cases. We would have no confidence in the correctness of our results had we not written very detailed proofs. Readers should be skeptical of results such as ours that are not accompanied by rigorous proofs. However, such proofs may not provide an intuitive understanding of why the results are true. We therefore give proof sketches in the main text and rigorous proofs in Section A of the appendix. Although the proofs in the appendix are long and somewhat tedious, their hierarchical structure makes them easy to check.

## 2 Theorems

We have tried to make our definitions of an asynchronous consensus algorithm and of fast learning as general as possible. Indeed, they allow round-based algorithms in which values are proposed only in the first round and any message sent in one round is either lost or received by the next round. However, we cannot rule out the possibility that our definitions are too restrictive and are not satisfied by algorithms that are, in a practical sense, asynchronous or fast-learning. It is therefore important that we state our results very precisely, making it clear exactly what kinds of algorithms they show to be impossible. In this section, our definitions and theorems are stated rigorously but informally. Section B of the appendix expresses them formally in TLA<sup>+</sup> [11].

We are assuming that the set of acceptors is finite. As we remarked above, consensus is trivial with a single proposer or a single learner. Consensus is also trivial if only a single value may be proposed. So, we assume once and for all:

**Assumption (Agent)** The set of acceptors is finite, and there are at least two proposers and two learners.

**Assumption (Value)** There are at least two proposable values.

We let  $n$  be the number of acceptors, and we define an *agent* to be a proposer, a learner, or an acceptor.

### 2.1 Scenarios and Algorithms

To state theorems about consensus algorithms, we must define what an algorithm is. We represent a possible execution of an algorithm by a set of

events. An algorithm is then described by a set of sets of events, representing all its possible partial executions. We now define this precisely.

We begin by defining what an event is. We assume that the events performed by a single agent are totally ordered. (Events performed concurrently by a single agent can be ordered arbitrarily.) So an event  $e$  specifies an agent  $e_{agent}$  and a positive integer  $e_{num}$ , indicating that  $e$  is the  $e_{num}^{\text{th}}$  event performed by  $e_{agent}$ . An event can be performed either spontaneously or upon receipt of a message. For a message-receiving event  $e$ , we let  $e_{rcvd}$  be a triple  $\langle m, a, i \rangle$ , indicating that the event was triggered by the receipt of a message  $m$  sent by the  $i^{\text{th}}$  event of agent  $a$ . For simplicity, we assume that each event  $e$  sends exactly one message  $e_{msg}$ , which can be received by any agent (including itself). The sending of a possibly empty set  $\mathcal{M}$  of messages can be modeled by letting  $e_{msg}$  equal  $\mathcal{M}$  and having an event that receives  $\mathcal{M}$  ignore any of its elements not meant for the receiver. Since we are concerned with when learning occurs and not with termination, we don't care if an agent ever stops sending messages.

We now define a scenario to be a set of events that could conceivably be generated by a single (possibly partial) execution of some algorithm. But first, for any set  $S$  of events, we define the precedence relation  $\preceq_S$  on  $S$  to be the transitive closure of the relation  $\rightarrow$  such that  $d \rightarrow e$  iff either (i)  $d_{agent} = e_{agent}$  and  $d_{num} \leq e_{num}$  or (ii)  $e$  is a message-receiving event such that  $e_{rcvd} = \langle d_{msg}, d_{agent}, d_{num} \rangle$ . (This is the reflexive form of the usual precedence relation for events in a distributed system [6].) A scenario is then defined as follows.

**Definition (Scenario)** A *scenario*  $S$  is a set of events such that

- For any agent  $a$ , the set of events in  $S$  performed by  $a$  consists of  $k_a$  events numbered from 1 through  $k_a$ , for some natural number  $k_a$ .
- For every message-receiving event  $e$  in  $S$ , there exists an event  $d$  in  $S$  different from  $e$  such that  $e_{rcvd} = \langle d_{msg}, d_{agent}, d_{num} \rangle$ .
- $\preceq_S$  is a partial order on  $S$ .

Since  $\preceq_S$  is defined to be transitively closed, the last requirement asserts that  $\preceq_S$  has no cycles, meaning that  $d \preceq_S e$  and  $e \preceq_S d$  imply  $d = e$ , for all  $d$  and  $e$  in  $S$ . The relation  $\preceq_S$  describes causality,  $d \preceq_S e$  holding for  $d \neq e$  iff it is possible for event  $d$  to causally influence event  $e$ . A cycle therefore cannot occur if  $S$  represents a possible partial execution of an algorithm.

A *prefix* of a scenario  $T$  consists of a set of events in  $T$  that precede all other events in  $T$ . The precise definition is:

**Definition (Prefix)** A subset  $S$  of a scenario  $T$  is a *prefix* of  $T$ , written  $S \sqsubseteq T$ , iff for any events  $d$  in  $T$  and  $e$  in  $S$ , if  $d \preceq_T e$  then  $d$  is in  $S$ .

It is easy to see that any prefix of a scenario is also a scenario.

An *algorithm* is defined to be any non-empty set of scenarios. Our results apply to asynchronous algorithms. However, our definition of an asynchronous algorithm is very weak, allowing algorithms that assume a great deal of synchrony. Our basic assumption is that whether or not the algorithm can perform an event  $e$  may depend only on events that causally precede  $e$ . For example, our definition includes algorithms that assume agents have perfectly synchronized clocks and point-to-point communication links that deliver messages with a known delay, but that allow agents and communication links to fail at any time. Our precise definition is as follows, where  $Agents(S)$  is the set of all agents that perform events in the set  $S$  of events, and  $A \setminus B$  is the subset of the set  $A$  consisting of all elements not in the set  $B$ .

**Definition (Asynchronous Algorithm)** An *asynchronous algorithm*  $Alg$  is a set of scenarios such that:

A1. Every prefix of a scenario in  $Alg$  is in  $Alg$ .

[The occurrence of an event  $d$  in the prefix cannot depend on whether an event  $e$  not in the prefix occurs, because  $e$  cannot causally effect  $d$ .]

A2. If  $T$  and  $U$  are scenarios of  $Alg$  and  $S$  is a prefix of both  $T$  and  $U$  such that  $Agents(T \setminus S)$  and  $Agents(U \setminus S)$  are disjoint sets, then  $T \cup U$  is a scenario of  $Alg$ .

[The assumptions about  $S$ ,  $T$ , and  $U$  imply that  $T \cup U$  is a scenario. The sets of events  $T \setminus S$  and  $U \setminus S$  both represent continuations of  $S$  that are allowed by the algorithm. Each continuation is performed by agents that have no way of knowing what the other's agents are doing. Hence the algorithm must allow both continuations to be performed, producing the execution described by  $T \cup U$ .]

Letting  $S$  be the empty scenario (the one containing no events), we deduce from A1 and A2 the following:

**Lemma (Scenario Union)** For any scenarios  $T$  and  $U$  of an asynchronous algorithm  $Alg$  such that  $Agents(T)$  and  $Agents(U)$  are disjoint,  $T \cup U$  is also a scenario of  $Alg$ .

We now assume that there are proposing and learning events. A *proposing event*  $e$  is one in which proposer  $e_{agent}$  proposes a value  $e_{proposed}$ . A *learning*

event  $e$  is one in which learner  $e_{agent}$  learns a value  $e_{learned}$ . Nontriviality and consistency are defined by:

**Definition (Nontriviality)** An algorithm  $Alg$  is *nontrivial* iff, for every scenario  $S$  in  $Alg$  and any learning event  $e$  in  $S$ , there is a proposing event  $d$  in  $S$  with  $d_{proposed} = e_{learned}$ .

**Definition (Consistency)** An algorithm  $Alg$  is *consistent* iff, for every scenario  $S$  in  $Alg$ , if  $d$  and  $e$  are learning events in  $S$ , then  $d_{learned} = e_{learned}$ .

We define a *consensus algorithm* to be an algorithm that is nontrivial and consistent. (Since our lower-bound results are about what *can* happen rather than what *must* happen, we do not require a progress property.)

## 2.2 The Lower Bound on Acceptors

A quorum is a set of acceptors that is large enough to choose a value, regardless of what steps of the algorithm have been performed so far. We formally define quorums in Section 2.3. Here, we define an *accepting set* to be one large enough that it can choose a value starting *ab initio*. A quorum is therefore an accepting set.

**Definition (Accepting Set)** A set  $Q$  of acceptors is *accepting for* a proposer  $p$  in algorithm  $Alg$  iff for every value  $v$  and learner  $l$ , there is a scenario  $S$  of  $Alg$  with  $Agents(S) \subseteq \{p, l\} \cup Q$  such that  $S$  has an event in which  $p$  proposes  $v$  and an event in which  $l$  learns  $v$ .

Our Acceptor Lower Bound Theorem assumes that any set with at least some minimum number of acceptors is an accepting set. Algorithms such as Paxos allow more general accepting sets, so the following result is of interest. It implies that for any algorithm that works with arbitrary sets of proposers and acceptors, no two accepting sets can be disjoint.

**Lemma (Accepting)** For any consistent asynchronous algorithm  $Alg$ , any proposers  $p_1$  and  $p_2$ , any learners  $l_1$  and  $l_2$ , and any sets  $Q_1$  and  $Q_2$  of acceptors such that each  $Q_i$  is an accepting set for  $p_i$ , the sets  $\{p_1, l_1\} \cup Q_1$  and  $\{p_2, l_2\} \cup Q_2$  are not disjoint.

**Proof Sketch** Choose two different proposable values  $v_1$  and  $v_2$ . For each  $i$ , since  $Q_i$  is an accepting set for  $p_i$ , there exists a scenario  $S_i$  of  $Alg$  with agents  $\{p_i, l_i\} \cup Q_i$  in which  $l_i$  learns  $v_i$ . If the  $\{p_i, l_i\} \cup Q_i$  were disjoint,

then the Scenario Union Lemma (Section 2.1) would imply that  $S_1 \cup S_2$  is a scenario, contradicting consistency.  $\square$

Our first lower bound is a direct corollary of the Accepting Lemma.

**Theorem (Acceptor Lower Bound)** For any natural number  $f$  with  $f \leq n$  and any consistent asynchronous algorithm  $Alg$ , if any set of  $n - f$  acceptors is an accepting set in  $Alg$  for every proposer, then either

- a.  $n > 2f$ , or
- b.  $f = 1$  and there are three agents  $a_1$ ,  $a_2$ , and  $a_3$  such that  $\{a_1, a_2\}$  is the set of acceptors,  $\{a_1, a_3\}$  is the set of proposers, and  $\{a_2, a_3\}$  is the set of learners.

**Proof Sketch** We assume that  $n \leq 2f$  and every set of  $n - f$  acceptors is an accepting set for every proposer, and we obtain a contradiction to the Accepting Lemma except if case b holds. One can show that  $n \leq 2f$  implies  $n - f \leq \lfloor n/2 \rfloor$ . Therefore, there exist disjoint sets  $Q_1$  and  $Q_2$  that each contain  $n - f$  acceptors and are thus accepting sets for every proposer.

The Agent and Value Assumptions imply that there are proposers  $p_1$  and  $p_2$  and learners  $l_1$  and  $l_2$  such that the sets  $\{p_1, l_1\}$  and  $\{p_2, l_2\}$  are disjoint. Since  $Q_1$  and  $Q_2$  are disjoint, we get the required contradiction to the Accepting Lemma if we show the existence of such  $p_i$  and  $l_i$  with  $\{p_1, l_1\}$  and  $Q_2$  disjoint and  $\{p_2, l_2\}$  and  $Q_1$  disjoint. This is trivial to do if the  $p_i$  and  $l_i$  can be chosen not to be acceptors, and it is not hard to do if  $n \geq 4$ . Showing that such  $p_i$  and  $l_i$  exist when  $n < 4$ , except in case b, requires a complicated case analysis.  $\square$

Section 3.1 below gives an algorithm for the anomalous case b. This case is of no practical interest, being just a weird consequence of defining consensus in terms of proposers, learners, and acceptors.

## 2.3 Fast Learning

Before stating our results about fast learning, we explain what it means. We define a *source* of a scenario  $S$  to be a minimal event in the ordering  $\preceq_S$ , which is an event  $e$  in  $S$  such that  $d \preceq_S e$  implies  $d = e$ , for any event  $d$  in  $S$ . We define the *depth* of an event in a scenario to be the number of message delays before the execution of that event:

**Definition (Event Depth)** The *depth* of an event  $e$  in a scenario  $S$  equals 0 if  $e$  is a source of  $S$ , otherwise it equals the maximum of (i) the depths of all events  $d$  with  $d_{agent} = e_{agent}$  and  $d_{num} < e_{num}$  and (ii) if  $e$  is

an event that receives a message sent by event  $d$ , then 1 plus the depth of  $d$ .

The depth of a scenario is defined to be the maximum of the depths of its events.

As we observed above, no algorithm can always achieve fast learning. However, we are interested only in whether an algorithm *allows* fast learning. We define a set  $M$  of acceptors to be *fast-accepting for* a proposer  $p$  if any learner can learn a proposal of  $p$  in two message delays by communicating only with  $p$  and the acceptors in  $M$ . The nonobvious aspect of the definition is that we require  $p$ 's proposal to be the learning scenario's only source event. This requirement may seem too restrictive because it rules out algorithms like Paxos that can perform preliminary actions before proposals are issued. However, we can consider such an algorithm to begin after the execution of those preliminary actions. What we rule out are algorithms that "cheat" by restricting what proposals can be learned fast. For example, Paxos can be modified to be fast-learning for a single value that is nondeterministically selected in advance by the leader. This algorithm has a scenario in which any proposed value is learned quickly—namely, one in which the leader happened to select that value in advance. We are not ruling out "secondary" proposals issued in response to a message from some other agent. We simply do not consider the learning of a value proposed in this way to be fast.

**Definition (Fast Accepting)** A set  $M$  of acceptors is *fast-accepting for* a proposer  $p$  in algorithm  $Alg$  iff for every proposable value  $v$  and learner  $l$ , there is a scenario  $S$  of  $Alg$  with  $Agents(S) \subseteq \{p, l\} \cup M$  such that  $S$  has depth at most 2, has as its only source an event in which  $p$  proposes  $v$ , and contains an event in which  $l$  learns  $v$ .

Our lower bound on fast learning assumes that a quorum is not just an accepting set, but that it is able to choose a value regardless of what steps of the algorithm have been performed so far. Stated in terms of learners, a quorum  $Q$  should be able to tell any learner  $l$  what value is chosen. Since nontriviality implies that a value can be chosen only if one is proposed, some proposer  $p$  may be needed to propose a value for  $l$  to learn.

**Definition (Quorum)** A set  $Q$  of acceptors is a *quorum* for an algorithm  $Alg$  iff it is an accepting set in  $Alg$  for every proposer and, for every proposer  $p$ , learner  $l$ , and scenario  $S$  of  $Alg$ , there exists a scenario  $T$  of  $Alg$  such that (i)  $S$  is a prefix of  $T$ , (ii)  $Agents(T \setminus S) \subseteq \{p, l\} \cup Q$ , and (iii)  $T$  contains a learning event of  $l$ .

The lower bound on fast learning is proved with the following obscure technical lemma. The best way to understand the lemma is to consider what it means for a general algorithm that works with any sets of proposers and learners. In this case, we can assume that proposers, learners, and acceptors are disjoint sets of agents, and that there are at least three learners and three acceptors. The lemma then implies that the intersection any quorum with any two sets that are fast-accepting for different proposers must be non-empty. This is precisely the condition on quorums and fast-accepting sets required by the Fast Paxos algorithm.

**Lemma (Fast Accepting)** For any consistent asynchronous algorithm  $Alg$ , if there exist proposers  $p_1, p_2$ , and  $p_q$ , learners  $l_1, l_2$  and  $l_q$ , fast-accepting sets  $M_1$  for  $p_1$  and  $M_2$  for  $p_2$  in  $Alg$ , and a quorum  $Q$  for  $Alg$  such that

- $p_1 \neq p_2$
- $p_1 \notin M_2$  and  $p_2 \notin M_1$
- $l_1 \notin \{p_2, p_q, l_q\} \cup (M_2 \setminus M_1) \cup Q$
- $l_2 \notin \{p_1, p_q, l_q\} \cup (M_1 \setminus M_2) \cup Q$
- $\{p_q, l_q\} \cap M_1 \cap M_2$  is empty

then  $M_1 \cap M_2 \cap Q$  is nonempty.

**Proof Sketch** We assume that  $M_1 \cap M_2 \cap Q$  is empty and obtain a contradiction. Let  $v_1$  and  $v_2$  be two different proposable values. Since each  $M_i$  is a fast-accepting set for  $p_i$ , there exists a scenario  $T_i$  executed by the agents in  $\{p_i, l_i\} \cup M_i$  in which  $l_i$  learns  $v_i$  within two message delays. Scenario  $T_i$  begins with  $p_i$  proposing  $v_i$  and sending “round 1” messages to  $l_i$  and the acceptors in  $M_i$ ; the acceptors in  $M_i$  can then send “round 2” messages to  $l_i$ . Upon receipt of the rounds 1 and 2 messages sent to it,  $l_i$  learns  $v_i$ . We can assume that only these round 1 and 2 messages are received in  $T_i$ , any other messages sent having been lost.

We will obtain a contradiction by constructing an initial scenario that can be completed to either  $T_1$  or  $T_2$  by agents not in  $\{p_q, l_q\} \cup Q$ . Since  $Q$  is a quorum, the agents in  $\{p_q, l_q\} \cup Q$  must be able to complete this initial scenario to one in which  $l_q$  learns a value. However, whatever value  $l_q$  learns, agents not in  $\{p_q, l_q\} \cup Q$  could execute the remaining actions of one of the  $T_i$  in which the value  $v_i$  that is learned is different from the value learned by  $l_q$ . This contradicts the assumption that the algorithm is consistent.



*Remark:* An algorithm that preserves consistency cannot allow a scenario to contain all the events of both  $T_1$  and  $T_2$ . It prevents this by not allowing acceptors in  $M_1 \cap M_2$  to perform events from both scenarios. It is those acceptors that can determine whether  $T_1$  or  $T_2$  occurred. In the fast-learning scenarios  $T_i$ , the acceptors in  $M_1 \cap M_2$  have no time to guarantee that other acceptors know in which scenario they are participating before the messages are sent that allow  $l_i$  to learn  $v_i$ . The assumption that  $M_1 \cap M_2 \cap Q$  is empty means that none of the acceptors that know whether  $T_1$  or  $T_2$  occurred are in  $Q$ . *End of Remark*

To construct the necessary initial scenario, let  $U_i$  be the prefix of  $T_i$  consisting of all its events except the learning event of  $l_i$  and the events of acceptors in  $M_1 \cap M_2$ . A careful analysis shows that the hypotheses imply that  $U_1$  and  $U_2$  are performed by disjoint sets of agents, so the Scenario Union Lemma implies that  $U_1 \cup U_2$  is a scenario of the algorithm. The hypotheses also imply that  $U_1 \cup U_2$  can be completed to a scenario containing all the actions of either  $T_1$  or  $T_2$  by adding events performed by agents not in  $\{p_q, l_q\} \cup Q$ . Hence,  $U_1 \cup U_2$  is the initial scenario that provides the required contradiction.  $\square$

The Paxos algorithm is fast-accepting for a single proposer—namely, the initial leader. Our lower bound on fast learning is for an algorithm that is fast-accepting for at least two different proposers. Its hypothesis asserts that every set of  $n - e$  acceptors is fast-accepting and every set of  $n - f$  acceptors is a quorum. By the Fast-Accepting Lemma, we expect this to be possible only for values of  $n$ ,  $e$ , and  $f$  satisfying the following condition: every two sets of  $n - e$  acceptors and every set of  $n - f$  acceptors have a nonempty intersection. Simple reasoning about finite sets shows that this condition is equivalent to  $n > 2e + f$ , which is our basic lower bound. However, there is one weird case in which this bound does not apply.

To rule out other uninteresting special cases, we assume  $e \leq f$  even though that assumption is not necessary to prove our basic lower bound. (In fact, Fast Paxos works for arbitrary  $e$  and  $f$  with  $n > 2e + f$ .) However, the case  $f < e$ , where the algorithm allows fast progress with fewer processes than are required to ensure eventual progress, seems to be of no practical interest. We also rule out the uninteresting case of  $f = 0$  (no fault tolerance). Fast learning is easily achieved in this case by letting one particular acceptor choose the value.

**Theorem (Fast Learning)** For any natural numbers  $e$  and  $f$  with  $f > 0$  and  $e \leq f \leq n$ , and for any asynchronous consensus algorithm  $Alg$ , if every set of  $n - f$  acceptors is a quorum for  $Alg$  and every set of  $n - e$

acceptors is fast-accepting in  $Alg$  for two distinct proposers  $p_1$  and  $p_2$ , then  $n > 2e + f$  or the set of learners equals  $\{p_1, p_2\}$ .

**Proof Sketch** We assume that the conclusion of the theorem is false and obtain a contradiction. The obvious approach is to construct proposers  $p_1$ ,  $p_2$ , and  $p_q$ , learners  $l_1$ ,  $l_2$ , and  $l_q$ , and sets  $M_1$ ,  $M_2$ , and  $Q$  of acceptors that contradict the Fast Accepting Lemma. The assumption that the conclusion is false implies  $n \leq 2e + f$ , from which we deduce the existence of the sets  $M_1$  and  $M_2$  with  $n - e$  acceptors each and the set  $Q$  with  $n - f$  acceptors such that  $M_1 \cap M_2 \cap Q$  empty. A careful analysis shows that we can also find the necessary proposers and learners if  $n > 2f$ . By the Acceptor Lower Bound Theorem, this yields the desired contradiction except in the anomalous three-agent case b of that theorem. We obtain a contradiction in this anomalous case by constructing the same sort of scenarios used in the proof of the Fast Accepting Lemma. (We could strengthen the lemma to handle the anomalous case as well, but it seems easier to consider that case separately rather than complicating the proof of the lemma.)  $\square$

## 2.4 Collision-Fast Learning

As explained above, Fast Paxos is fast-learning only in the absence of collisions. This is also true of the Generic Broadcast algorithm of Pedone and Schiper [13]. We now consider collision-fast algorithms, which are ones that are fast despite collisions. We show that such algorithms are possible only in certain special cases.

Our lower bound result for fast learning assumed only the existence of certain fast-learning scenarios. However, any algorithm that permits fast learning allows scenarios that are fast-learning in the presence of collision—namely, scenarios in which the messages sent by one of the proposers reaches the acceptors before messages sent by the other proposers. What we want to show is the impossibility of an algorithm that can always guarantee fast learning despite collisions. But this is trivial because the FLP result shows that no algorithm can always guarantee learning. So, what kind of interesting impossibility result can we find?

A useful asynchronous consensus algorithm works in the “normal” case, in which faulty processes send no messages, and all messages sent by non-faulty processes are delivered quickly, before any timeouts can occur. We define a collision-fast learning algorithm to be one in which fast learning always occurs in the normal case. Hence, we want the definition of normality to be as restrictive as possible, since that strengthens the definition of collision-fast and therefore makes our impossibility result stronger.

**Definition (Normal Scenario)** A scenario  $S$  is *normal* iff it satisfies the following properties:

- The only sources of  $S$  are proposal events.  
[As in the definition of fast accepting, this rules out cheating.]
- The message sent by any single event is not received twice by the same agent.  
[This allows an agent to resend the same message and have both copies received by another agent.]
- Every non-source event is a message-receiving event.  
[Except for initial proposals, each event is triggered by the receipt of a message and thus not by a timeout.]
- If  $d1$  and  $d2$  are events in  $S$  with  $d1_{agent} = d2_{agent}$  and  $d1 \preceq_S d2$ , and  $e2$  is an event in  $S$  that receives the message sent by  $d2$ , then there exists an event  $e1$  in  $S$  with  $e1_{agent} = e2_{agent}$  and  $e1 \preceq_S e2$  such that  $e1$  receives the message sent by  $d1$   
[Messages sent from any agent  $a$  to any agent  $b$  are delivered in FIFO order, with no gaps.]
- If  $d$  and  $e$  are events in  $S$  and  $e$  receives the message sent by  $d$ , then the depth of  $e$  in  $S$  equals 1 plus the depth of  $d$  in  $S$ .  
[Messages are delivered in an order consistent with a round-based algorithm, in which all messages sent in one round are delivered before those sent in the next round.]

We say that an agent  $a$  is *complete to depth  $\delta$*  in a normal scenario iff  $a$  performs all events of depth  $\delta$  or less that it possibly could. The precise definition is:

**Definition (Complete to Depth)** An agent  $a$  is *complete to depth  $\delta$*  in a scenario  $S$  iff either  $\delta = 0$  or every agent in  $Agents(S)$  is complete to depth  $\delta - 1$  and  $a$  receives every message sent by an event in  $S$  of depth less than  $\delta$ .

**Definition (Collision-Fast Accepting)** A set  $M$  of acceptors is *collision-fast* in algorithm  $Alg$  for a set  $P$  of proposers iff for every nonempty subset  $\{p_1, \dots, p_k\}$  of  $P$  with the  $p_i$  all distinct:

- For any proposable values  $v_1, \dots, v_k$  there is a scenario  $\{e_1, \dots, e_k\}$  in  $Alg$  such that each  $e_i$  is a spontaneous event (a source) in which  $p_i$  proposes  $v_i$ .

- For every learner  $l$  and every normal scenario  $S$  of  $Alg$  with  $Agents(S) = \{l, p_1, \dots, p_k\} \cup M$  that contains  $\{e_1, \dots, e_k\}$  as a prefix, if  $l$  is complete to depth 2 in  $S$ , then  $l$  learns a value in  $S$ .

Our definition of an asynchronous algorithm is satisfied by an algorithm that can control the precise order in which messages sent by different agents will be delivered. For example, it is satisfied by an algorithm in which an agent that receives a message from proposer  $p_1$  without having received a message from proposer  $p_2$  can conclude that  $p_2$  has not issued a proposal. To prove our impossibility result for collision-fast learning, we need to require that messages sent by different agents in events of the same depth can be delivered in arbitrary order to other agents. This requirement is expressed by the following definition.

**Definition (Independent Delivery)** An algorithm  $Alg$  has *independent delivery* iff for any normal scenario  $S$  in  $Alg$ , any event  $e$  in  $S$ , and any agent  $a$ , if

- For every event  $d$  in  $S$  performed by  $e_{agent}$  that precedes  $e$ , there is an event in  $S$  in which  $a$  receives the message sent by  $d$ .  
[ $a$  has received every previous message sent by  $e_{agent}$ .]
- There is no event in  $S$  in which  $a$  receives the message sent by event  $e$ .  
[ $a$  has not yet received the message sent by  $e$ .]
- For every event  $d$  in  $S$ , if the depth of  $d$  in  $S$  is less than the depth of  $e$  in  $S$ , then there is an event in  $S$  in which  $a$  receives the message sent by  $d$ .  
[Having  $a$  now receive the message sent by  $e$  is consistent with a round-based algorithm.]

then there is an event  $c$  performed by  $a$  that receives the message sent by  $e$  such that  $S \cup \{c\}$  is a scenario of  $Alg$ .

[ $a$  can now receive the message sent by  $e$ .]

In an algorithm with independent delivery, each agent can send messages to itself that are delivered in arbitrary order relative to messages sent by other agents. This appears strange, since we would expect an agent to be able to control when a message it sends to itself arrives. Such a message can be helpful only in special architectures where sending it influences the arrival order of other messages—for example, by clearing some message queue. It is this type of dependent delivery order that we are ruling out. It is not

hard to see that any result that holds for independent delivery also holds if agents do not send messages to themselves, since not sending a message is equivalent to ignoring the message when it arrives. However, the details of the proof become a bit simpler if the same assumptions are made for all possible messages, including ones sent by an agent to itself.

**Theorem (Collision-Fast Learning)** For any natural numbers  $e$  and  $f$ , with  $e \leq f \leq n$  and  $f > 0$ , and any asynchronous consensus algorithm  $Alg$  with independent delivery, if every set of  $n - f$  acceptors is a quorum for  $Alg$  and there are two distinct proposers  $p_1$  and  $p_2$  such that every set of  $n - e$  acceptors is collision-fast accepting for  $\{p_1, p_2\}$  in  $Alg$ , then  $e = 0$  and

- a.  $f = 1$ , every learner is an acceptor, and at least one acceptor is not a learner, or
- b.  $p_1$  or  $p_2$  (or both) is an acceptor.

**Proof Sketch** The proof is by contradiction. Let  $a_1, \dots, a_n$  be the acceptors, and define a sequence  $S_0, \dots, S_n$  of scenarios as follows. In each  $S_j$ , let  $p_1$  propose  $v_1$  and  $p_2$  propose  $v_2$ , where  $v_1 \neq v_2$ . The messages generated by these two proposal events are received by  $p_1, p_2$ , and all the acceptors. In  $S_j$ , acceptors  $a_1, \dots, a_j$  receive the message from  $p_1$  then the message from  $p_2$ , while acceptors  $a_{j+1}, \dots, a_n$  receive those messages in the opposite order.

Let  $l$  be a learner. Adding events performed by  $l$ , we extend  $S_j$  to a normal scenario  $T_j(l)$  in which  $l$  is complete to depth 2, so it must learn either  $v_1$  or  $v_2$ . In  $T_0(l)$ , all of  $p_2$ 's messages arrive before  $p_1$ 's, so  $T_0(l)$  contains a prefix in which  $p_1$  performs no events, only  $p_2$  issues a proposal, and  $l$  is complete to depth 2. Learner  $l$  must learn  $v_2$  in that prefix and hence in  $T_0(l)$ . Similarly,  $l$  must learn  $v_1$  in  $T_n(l)$ . Thus, there is some  $k$  with  $0 < k \leq n$  such that  $l$  learns  $v_2$  in  $T_{k-1}(l)$  and  $v_1$  in  $T_k(l)$ . Scenarios  $T_{k-1}(l)$  and  $T_k(l)$  differ only in the events performed by  $a_k$  and  $l$ . If scenario  $T_{k-1}(l)$  or  $T_k(l)$  occurs and  $a_k$  and  $l$  both fail, then there is no way for the remaining agents to know if  $l$  learned  $v_1$  or  $v_2$ , so it is impossible to ensure consistency if any other learner learns a value. This contradicts the hypothesis that any  $n - f$  acceptors form a quorum, where  $f > 0$ .

The proof sketched here breaks down if (i)  $l$  is an acceptor, (ii)  $p_1$  or  $p_2$  is an acceptor, or (iii)  $l$  equals  $p_1$  or  $p_2$ . (Only if  $e = 0$  does the negation of the theorem's conclusion, which is assumed in a proof by contradiction, imply that (ii) is false.) The proof in the appendix handles these three cases.  $\square$

Collision-fast algorithms for both exceptional cases are given below in Section 3.2.2. These algorithms may be of practical interest.

## 2.5 Hyperfast Learning

For completeness, we consider hyperfast learning, which is learning a value in one message delay. With hyperfast learning, a learner must learn a value after receiving messages only from a proposer. (If the proposal event is the only source, then messages from other agents are received only after at least two message delays.)

**Definition (Hyperfast Accepting)** An algorithm  $Alg$  is *hyperfast-accepting* for a proposer  $p$  iff for every proposable value  $v$  and learner  $l$ , there is a scenario  $S$  of  $Alg$  such that  $S$  has depth at most 1, has as its only source an event in which  $p$  proposes  $v$ , and contains an event in which  $l$  learns  $v$ .

The following result shows that hyperfast learning is impossible except in very restricted special cases.

**Theorem (Hyperfast Learning)** A consistent asynchronous algorithm  $Alg$  cannot be hyperfast-accepting for two different proposers. For any integer  $f$  with  $0 < f \leq n$ , if every set of  $n - f$  acceptors is a quorum for  $Alg$  and  $Alg$  is hyperfast-accepting for a proposer  $p$ , then

1.  $f = 1$ ,
2.  $p$  is an acceptor that is not a learner, and
3. For every learner  $l$ , either  $l$  is an acceptor or  $\{p, l\}$  is the set of proposers.

**Proof Sketch** Hyperfast accepting for a proposer  $q$  implies that, for any learner  $l$  and value  $v$ , there is a scenario in which  $q$  proposes  $v$  and sends a message to  $l$ , which receives the message and thereupon learns  $v$ . The set of agents in this scenario is  $\{q, l\}$ .

Suppose the algorithm is hyperfast accepting for two different proposers  $p_1$  and  $p_2$ . The Agent Assumption implies that there are learners  $l_1$  and  $l_2$  with  $\{p_1, l_1\}$  and  $\{p_2, l_2\}$  disjoint. We can then show that consistency is violated by applying the Scenario Union Lemma to two scenarios whose set of agents are  $\{p_1, l_1\}$  and  $\{p_2, l_2\}$ , in which  $l_1$  and  $l_2$  learn different values. This shows that hyperfast accepting is impossible for two different proposers.

We now sketch a proof of the rest of the theorem. If  $Alg$  is hyperfast accepting for  $p$ , then for any learner  $l_1$  and value  $v_1$  there is a scenario with set of agents  $\{p, l_1\}$  in which  $l_1$  learns  $v_1$ . Suppose the set of all agents other than  $p$  and  $l_1$  contains a proposer  $p_2$ , a learner  $l_2$ , and a quorum  $Q$ . We can then construct a scenario with agents  $\{p_2, l_2\} \cup Q$  in which  $l_2$  learns a value  $v_2 \neq v_1$ . Applying the Scenario Union Lemma then shows that consistency is violated. To complete the proof of the theorem, we must show that such agents  $p_2$  and  $l_2$  and quorum  $Q$  exist unless the theorem's three conclusions hold. This is not hard, but requires care in checking the details.  $\square$

It is quite easy to see that the analog of fast and hyperfast learning with zero message delays is never possible.

### 3 Algorithms

Our theorems assert that consensus algorithms do not exist except in certain cases. We now describe algorithms for those cases, showing that the theorems are as strong as possible. The theorems are covered in the same order as in Section 2. Paxos and Fast Paxos provide algorithms for the normal cases. Among the exceptional cases, only the ones for the Collision-Fast Learning Theorem yield potentially useful algorithms; they appear in Section 3.2.2. Since the theorems are our main results, we just sketch the algorithms and their correctness proofs. All the algorithms have a parameter  $f$  such that any set of  $n - f$  acceptors is a quorum.

#### 3.1 The Lower Bound on Acceptors

The Paxos algorithm [9] achieves the lower bound of  $n = 2f + 1$  asserted by the main case of the Acceptor Lower Bound Theorem, for arbitrary sets of learners and proposers. We now exhibit a consensus algorithm for the anomalous case of the theorem, with  $f = 1$  and three agents  $a_1$ ,  $a_2$ , and  $a_3$  such that  $\{a_1, a_2\}$  is the set of acceptors,  $\{a_1, a_3\}$  is the set of proposers,  $\{a_2, a_3\}$  is the set of learners, and both  $\{a_1\}$  and  $\{a_2\}$  are quorums. The existence of such an algorithm is actually an artifact of our definitions. We show that the requirements are satisfied by any 1-fault tolerant consensus algorithm  $Alg$  that works for  $n = 3$ , such as Paxos, in which  $a_1$ ,  $a_2$ , and  $a_3$  all act as acceptors.

One-fault tolerance means that any two of the three agents form a quorum for  $Alg$ . In other words, any set containing a proposer, a learner, and two agents can make progress. Since the nontriviality and consistency re-

quirements do not depend on the set of quorums, to show that *Alg* satisfies the requirements of the theorem’s anomalous case, it suffices to show that  $\{a_1\}$  and  $\{a_2\}$  are quorums. This requires showing that any set consisting of a proposer, a learner, and either  $a_1$  or  $a_2$  can make progress. But a simple enumeration of all possible cases shows that any such set contains two agents, so *Alg* guarantees that progress can be made.

## 3.2 Fast, Collision-Fast, and Hyperfast Learning

We now describe several algorithms for fast and hyperfast learning. All but one of these algorithms is described in terms of a fast round that may be followed by a slow round. An agent enters the slow round either spontaneously by a timeout or when it receives a slow-round message from another agent. Once it enters the slow round, the agent stops participating in the fast round and ignores any fast-round messages it may receive.

The slow round can use any consensus algorithm, such as Paxos, that ensures progress if a proposer, a learner, and  $n - f$  acceptors are nonfaulty. The round first determines either (a) that some single value  $v$  might have been learned in the fast round, or (b) no value was learned in the first round. In case (a), only the value  $v$  may be proposed to the consensus algorithm; in case (b), any value may be proposed.

Our algorithms for fast and collision-fast learning have a parameter  $e$  such that every set of  $n - e$  acceptors is a fast or collision-fast accepting set. They require that, in the fast round, every nonfaulty learner learns a value within 2 message delays if  $n - e$  acceptors are nonfaulty and, for ordinary fast learning, if only a single proposer issues a proposal.

### 3.2.1 Fast Learning

The Fast Learning Theorem asserts that if an algorithm is fast-accepting for at least two different proposers  $p_1$  and  $p_2$ , then either (a)  $n > 2e + f$  or (b) the set of learners equals  $\{p_1, p_2\}$ .

Fast Paxos is an  $f$ -fault tolerant consensus algorithm that works for arbitrary sets of proposers and learners. Under the same synchrony assumption as ordinary Paxos, it ensures fast learning in the absence of collision if  $n > 2f$  and  $n > 2e + f$ . By the Acceptor Lower Bound Theorem, there can exist a fast-learning algorithm with  $n \leq 2f$  for case (a) only in the anomalous three-agent case of that theorem with  $n = 2$ ,  $f = 1$ , and  $e = 0$ . A collision-fast, and hence fast, algorithm for this case is given in Section 3.2.2 below.

Here we describe only a fast-learning algorithm that works for  $e = f$  and



$n > 2f$  in case (b), when  $\{p_1, p_2\}$  is the set of learners. In the fast round, each  $p_i$  can send a proposal to the acceptors iff it has not already learned a value. An acceptor votes for a proposed value in the fast round and sends its vote to the  $p_i$  iff it has not already voted. A learner learns a value  $v$  in the fast round iff it receives at least  $n - f$  fast-round votes for  $v$  and it has not proposed any value other than  $v$ .

The algorithm is fast accepting because a proposed value is learned in 2 message delays if it is the only one proposed,  $n - f$  acceptors are nonfaulty, and no timeout occurs during the fast round. Consistency is maintained in the fast round because  $n > 2f$  implies that two different values cannot both receive  $n - f$  votes. We now show that progress is ensured if a learner, a proposer, and  $n - f$  acceptors are nonfaulty.

Suppose  $p_i$ , a learner, and  $n - f$  acceptors are nonfaulty. Then in the slow round, those agents can determine that exactly one of the following occurred in the fast round:

- $p_i$  learned  $v$ .
- $p_i$  did not learn a value, at least one of the  $n - f$  acceptors voted for a value  $v$ , and
  - $v$  was *not* proposed by  $p_i$  (so it was proposed by the other proposer), or
  - $v$  was proposed by  $p_i$  and none of the acceptors voted for a value other than  $v$ .
- None of the  $n - f$  acceptors voted.

In the first two cases,  $v$  was proposed and no value other than  $v$  could have been learned in the fast round, so  $v$  can be proposed in the slow round. In the third case, no value could have been learned in the fast round, so any value can be proposed in the slow round. Hence, a value can be proposed in the slow-round consensus algorithm, ensuring progress.

### 3.2.2 Collision-Fast Learning

We now give two collision-fast algorithms for  $e = 0$  and  $n > 2f$ :

- A. An algorithm that is collision-fast learning for any set of proposers when  $f = 1$  and every acceptor except one is a learner.
- B. An algorithm, in which every acceptor is a proposer, that is collision-fast for a set of proposers consisting of the acceptors together with at most one additional proposer.

We can always reduce the set of proposers or learners by simply not letting some proposers propose and ignoring what some learners learn. Therefore, these are the most general algorithms whose existence is not ruled out by the Collision-Fast Learning Theorem.

These two algorithms are of more than theoretical interest. The  $e = 0$  case is important because, if a system is automatically reconfigured to remove any server that fails, then all servers are nonfaulty most of the time. Algorithm A is interesting because the most common applications of asynchronous consensus will probably tolerate one fault by using three servers, only two of which maintain the system state and thus need to be learners. Algorithm B handles the traditional case in which each agent is a proposer, a learner, and an acceptor.

**Algorithm A** Algorithm A is a simple variant of the Paxos algorithm for  $f = 1$ . Recall how Paxos works in this case [9]. It begins with a leader chosen for ballot 0. Proposers send proposed values to the leader. When the leader receives the first such value, it sends it in a ballot-0 phase 2a message to each acceptor. Upon receiving the phase 2a message, an acceptor then sends the value to every learner in a ballot-0 phase 2b message. A learner learns a value if it receives  $n - f$  ballot-0 phase 2b messages with the value.

We explain Algorithm A first for the case  $n = 3$ . We let the leader for ballot 0 be the acceptor that is not a learner, so the learners are the other two acceptors. The leader sends its phase 2a and phase 2b messages together. Upon receiving the leader's phase 2a/b message, each other acceptor knows about two phase 2b messages for the value—the leader's and its own—so it learns the value. (Recall that  $n = 3$  and  $f = 1$ .) In fact, the learner does not actually have to send its phase 2b message. In the absence of failures or timeouts, the proposal is then learned within two message delays. The algorithm is thus collision-fast for any set of proposers.

To generalize the algorithm to an arbitrary  $n > 2$ , we observe that Paxos works with any set of quorums, subject only to the requirement that any two quorums have an acceptor in common. A learner learns a value when it receives phase 2b messages with the same ballot number from all acceptors in a quorum. We let the leader be the acceptor that is not a learner, and we let a quorum consist of either (i) any set containing the leader and at least one other acceptor or (ii) the set of all acceptors other than the leader. This set of quorums satisfies the requirement that any two quorums have an acceptor in common. Since  $f = 1$ , any  $n - f$  acceptors form a quorum. As in the  $n = 3$  case, a learner learns a value when it receives the ballot-

0 phase 2a/b message from the leader, because it and the leader form a quorum.

**Algorithm B** Let  $p_1, \dots, p_n$  be the acceptors, which are also proposers, and let  $p_0$  be an additional proposer. These agents perform the following actions in the fast round.

- If it has not already sent a fast-round message, a proposer  $p_i$  can propose a value  $v$  by sending the message  $\langle p_i, v \rangle$  to all acceptors other than itself and, if it is an acceptor, sending the message  $\langle p_i : \langle p_i, v \rangle \rangle$  to all learners.
- Upon receipt of the message  $\langle p_j, v \rangle$ , an acceptor  $p_i$  may send the message  $\langle p_i : \langle p_j, v \rangle \rangle$  to all learners iff it has not already sent a message  $\langle p_i : \langle p_k, v' \rangle \rangle$  with  $k > j$ .

A learner learns the value  $v$  if, for some  $i$ , it has received the message  $\langle p_j : \langle p_i, v \rangle \rangle$  from every acceptor  $p_j$ .

For each  $i$ , if  $p_i$  proposed a value in the fast round, let  $v_i$  be that value. Suppose some  $p_i$  did propose a value in the fast round, and let  $k$  be the largest such  $i$ . We prove consistency of the fast round by showing that  $v_k$  is the only value that can be learned. This is obvious if  $k = 0$ , since  $v_0$  is then the only value proposed. If  $k > 0$ , then it is implied by the following assertion, which is easily shown to be an invariant of the algorithm, for any  $i > 0$ :

If process  $i$  has proposed a value, then it has not sent any message  $\langle p_i : \langle p_j, v \rangle \rangle$  with  $j < i$ .

If  $p_k$  and all the acceptors are non-faulty and no timeout occurs during the fast round, then every acceptor  $i$  sends the message  $\langle p_i : \langle p_k, v_k \rangle \rangle$ , so every non-faulty learner learns  $v_k$  in two message delays. Hence, the algorithm is collision-fast accepting for  $\{p_0, \dots, p_n\}$ .

To ensure progress in the slow round, any set of  $n - f$  acceptors must be able to determine either (a) a proposed value  $v_j$  such that no value except  $v_j$  could have been learned in the fast round, or (b) that no value was learned in the fast round. For  $n - f > 0$  (so a quorum contains at least one acceptor), this is easy to do because, for any acceptor  $p_i$ :

- (a) If  $j$  is the largest integer such that  $p_i$  sent a  $\langle p_i : \langle p_j : v_j \rangle \rangle$  message, then only  $v_j$  could have been learned in the fast round.
- (b) If  $p_i$  sent no  $\langle p_i : \langle p_j : v_j \rangle \rangle$  message, then no value was learned in the fast round.

**The Anomalous Three-Agent Case** Algorithms A and B both work and are equivalent for the anomalous three-agent case of the Acceptor Lower Bound Theorem. Algorithm A is fast-learning despite the failure of the non-acceptor agent because fast learning requires only that the leader be nonfaulty. For this set of agents, the fast round of Algorithm B is equivalent to the initial part of Algorithm A in which proposal messages and ballot-0 phase 2a/b messages are sent.

### 3.2.3 Hyperfast Learning

The Hyperfast Learning Theorem permits hyperfast learning only with  $f = 1$  and only for a single proposer  $p$  such that (i)  $p$  is an acceptor that is not a learner and (ii) either (a) every learner is an acceptor, or (b) every learner but one is an acceptor and the other learner is the only proposer besides  $p$ . We show that a hyperfast learning algorithm exists in this case.

The definition of hyperfast learning pretty much determines the fast round. Proposer  $p$  can propose at most one value by sending a message to every learner, and a learner learns that value upon receiving the message. Hyperfast learning and consistency of the fast round are obvious. To ensure progress in the slow round, any proposer, learner, and set of  $n - 1$  acceptors must be able to determine if  $p$  proposed a value that could have been learned in the fast round. They can do this because conditions (i) and (ii) imply that either  $p$  is the proposer,  $p$  is one of the  $n - 1$  acceptors, or the proposer and the  $n - 1$  acceptors include all learners.

## 4 Conclusion

In a fault-tolerant asynchronous consensus algorithm, how many message delays must occur between when a value is proposed and when a value is learned? We have shown that the answer is 3, 2, or 1, depending on how many acceptor processes are used to choose the value, how many of them may fail, exactly who the proposers and learners are, and whether or not proposals are issued concurrently. For an algorithm that uses  $n$  acceptors, can always make progress if  $n - f$  of them are non-faulty, and works with arbitrary proposers and learners, the answers are:

- Learning is possible in 3 message delays iff  $n > 2f$ .
- Learning is possible in 2 message delays when  $n - e$  acceptors are non-faulty, for  $0 \leq e \leq f$  and  $f > 0$ , iff  $n > 2e + f$ . However, concurrent proposals can prevent an algorithm from achieving such fast learning.

- Learning is impossible in 1 message delay.

We have presented algorithms showing that each of these results is false for special choices of proposers and/or learners. Our two algorithms that permit learning in 2 message delays despite concurrent proposals may have practical applications.

The consensus problem is important, so we want our results to be unambiguous and correct. Our presentation has therefore been rigorous, and the proofs and formal exposition in the appendix are rather long and tedious. We do not know any easier way to avoid errors.

## References

- [1] Francisco Brasileiro, Fabíola Greve, Achour Mostefaoui, and Michel Raynal. Consensus in one communication step. In V. Malyskin, editor, *Parallel Computing Technologies (6th International Conference, PaCT 2001)*, volume 2127 of *Lecture Notes in Computer Science*, pages 42–50. Springer-Verlag, 2001.
- [2] Bernadette Charron-Bost and André Schiper. Uniform consensus is harder than consensus (extended abstract). Technical Report DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland, May 2000.
- [3] Roberto De Prisco, Butler Lampson, and Nancy Lynch. Revisiting the PAXOS algorithm. *Theoretical Computer Science*, 243:35–91, 2000.
- [4] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
- [5] Michael J. Fischer, Nancy Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
- [6] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [7] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, August-September 1995.
- [8] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

- [9] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column)*, 32(4):18–25, December 2001.
- [10] Leslie Lamport. Lower bounds for asynchronous consensus. In André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 22–23. Springer, 2003.
- [11] Leslie Lamport. *Specifying Systems*. Addison-Wesley, Boston, 2003.
- [12] Leslie Lamport. Fast paxos. Technical Report MSR-TR-2005-112, Microsoft Research, July 2005.
- [13] Fernando Pedone and André Schiper. Handling message semantics with generic broadcast. *Distributed Computing*, 15(2):97–107, 2002.
- [14] J. Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1254, October 1978.

## Appendix

### A Proofs

We now provide rigorous proofs of the results presented in Section 2. The results are not hard to prove under the assumption that the sets of proposers, acceptors, and learners are pairwise disjoint. Difficulties are caused by the exceptional cases that arise when those sets are not pairwise disjoint. Since exceptional cases may permit interesting algorithms, we feel that it is important to make sure that we have identified all of them. In other words, we want to make sure that our theorems are not just “approximately correct”.

The only practical way we know of checking the correctness of a theorem is by writing its proof in the hierarchically structured style explained in [7]. A structured proof consists of a sequence of statements and their proofs; each of those proofs is either a structured proof or an ordinary paragraph-style proof. The  $j^{\text{th}}$  step in the current level  $i$  proof is numbered  $\langle i \rangle j$ . The proof statement “ $\langle i \rangle j$ . Q.E.D.” denotes the current goal—that is, the level  $i - 1$  statement being proved by this step. A proof statement

ASSUME:  $A$     PROVE:  $P$

asserts that  $P$  can be proved under assumption  $A$ , and that  $A$  is assumed in the lower-level steps that prove  $P$ . Each paragraph-style “leaf” proof explicitly names each assumption and prior proof step required to prove the statement. However,  $n$ ,  $f$ , and  $e$  are taken to be natural numbers without mentioning the assumptions implying that they are.

We recommend reading the proofs hierarchically, from the top level down. To read the proof of a long level  $i$  step, first read the level  $i + 1$  statements that form its proof, together with the proof of the final Q.E.D. step (which is usually a short paragraph). The proof of the level  $i + 1$  steps can then be read in any order.

The steps of the proof are written in informal mathematics. They could be expressed as TLA<sup>+</sup> formulas using the definitions in Section B below, except that:

- TLA<sup>+</sup> has no construct corresponding to an ASSUME/PROOF step. A PROOF clause can be expressed as a TLA<sup>+</sup> formula, and an ASSUME clause as a collection of CONSTANT declarations and a formula.
- A step of the form “Choose an  $x$  satisfying  $P(x)$ ” or “Let  $x$  satisfy  $P(x)$ ” is expressed as the definition  $x \triangleq \text{CHOOSE } x : P(x)$  and the assertion  $P(x)$ , which is verified by proving  $\exists x : P(x)$ .

Each proof is preceded by an informal sketch and an explanation of any notation introduced for the proof. The following notation is used in more than one proof.

- $\mathcal{A}$  is the set of all acceptors.
- For  $i$  equal to 1 or 2, we define  $\neg i$  by  $\neg 1 = 2$  and  $\neg 2 = 1$ .
- $\{x \in C : P(x)\}$  is the subset of the set  $C$  consisting of all elements  $x$  that satisfy the predicate  $P(x)$ .
- We say that the set of agents is *anomalous* iff there are three agents  $a_1$ ,  $a_2$ , and  $a_3$  such that  $\{a_1, a_2\}$  is the set of acceptors,  $\{a_1, a_3\}$  is the set of proposers, and  $\{a_2, a_3\}$  is the set of learners.  
[Case b of the Acceptor Lower Bound Theorem asserts that the set of agents is anomalous.]
- An itemized list of assertions denotes their conjunction if the items are labeled with numbers; it denotes their disjunction if they are labeled with letters. (In the latter case, “or”s are written explicitly.)

- $i \dots j$  is the set of all integers  $k$  with  $i \leq k \leq j$ , for any natural numbers  $i$  and  $j$ .

## A.1 The Accepting Lemma

**Lemma (Accepting)** For any consistent asynchronous algorithm  $Alg$ , any proposers  $p_1$  and  $p_2$ , any learners  $l_1$  and  $l_2$ , and any sets  $Q_1$  and  $Q_2$  of acceptors such that each  $Q_i$  is an accepting set for  $p_i$ , the sets  $\{p_1, l_1\} \cup Q_1$  and  $\{p_2, l_2\} \cup Q_2$  are not disjoint.

The proof sketch of Section 2.2 is quite rigorous. We provide a hierarchically structured version here mainly as an introduction to the proof style.

### Proof

ASSUME: 1.  $Alg$  is a consistent asynchronous algorithm.

2.  $p_1$  and  $p_2$  are proposers,  $l_1$  and  $l_2$  are learners, and  $Q_1$  and  $Q_2$  are sets of acceptors such that  $Q_i$  is an accepting set for  $p_i$ , for each  $i = 1, 2$ .

3.  $\{p_1, l_1\} \cup Q_1$  and  $\{p_2, l_2\} \cup Q_2$  are disjoint.

PROVE: FALSE

$\langle 1 \rangle 1$ . Choose proposable values  $v_1$  and  $v_2$  and scenarios  $S_1$  and  $S_2$  such that:

1.  $v_1 \neq v_2$

2.  $S_i$  is a scenario of  $Alg$  with  $Agents(S_i) \subseteq \{p_i, l_i\} \cup Q_i$ , for  $i = 1, 2$ .

3.  $S_i$  has an event in which  $l_i$  learns  $v_i$ , for  $i = 1, 2$ .

PROOF: Values  $v_1$  and  $v_2$  exist by the Value Assumption. Scenarios  $S_i$  satisfying conditions 2 and 3 exist by assumption 2 and the definition of an accepting set.

$\langle 1 \rangle 2$ .  $S_1 \cup S_2$  is a scenario of  $Alg$ .

PROOF: This follows from  $\langle 1 \rangle 1.2$ , assumption 3, and the Scenario Union Lemma (page 9 of Section 2.1).

$\langle 1 \rangle 3$ . Q.E.D.

PROOF: Step  $\langle 1 \rangle 1.2$  implies that each learner  $l_i$  learns  $v_i$  in  $S_1 \cup S_2$ , which is a scenario of  $Alg$  by  $\langle 1 \rangle 2$ . By  $\langle 1 \rangle 1.1$ , this contradicts assumption 1 (the consistency of  $Alg$ ).



## A.2 The Acceptor Lower Bound Theorem

**Theorem (Acceptor Lower Bound)** For any natural number  $f$  with  $f \leq n$  and any consistent asynchronous algorithm  $Alg$ , if any set of  $n - f$  acceptors is an accepting set in  $Alg$  for every proposer, then either

- a.  $n > 2f$ , or
- b.  $f = 1$  and the set of agents is anomalous.

We show that if  $n \leq 2f$ , then except in the case of the anomalous set of agents, there exist accepting sets, proposers, and learners that contradict the Accepting Lemma. The proof is easy if  $n \geq 4$ , but rather complicated if  $n < 4$ . Fortunately the proof for  $n < 4$  can be checked with a computer by exhaustive enumeration. We do this by having the TLC model checker verify the truth of a TLA<sup>+</sup> formula, so the reader can determine directly the correspondence between what the computer verified and what we claim has been proved. We feel that it is much less likely for an undetected error in TLC to let it verify an incorrect formula than it would be for us to make an undetected error in a hand proof.

### Proof

ASSUME: 1.  $Alg$  is a consistent asynchronous algorithm.

2.  $f$  is a natural number with  $f \leq n$  and  $n \leq 2f$ .

3. Every set of  $n - f$  acceptors is an accepting set for every proposer.

PROVE:  $f = 1$  and there exist  $a_1, a_2$ , and  $a_3$  such that  $\{a_1, a_2\}$  is the set of acceptors,  $\{a_1, a_3\}$  is the set of proposers, and  $\{a_2, a_3\}$  is the set of learners.

⟨1⟩1. Choose proposers  $p_1$  and  $p_2$  and learners  $l_1$  and  $l_2$  such that  $\{p_1, l_1\}$  and  $\{p_2, l_2\}$  are disjoint.

PROOF: The existence of the  $p_i$  and  $l_i$  follows from the Agent Assumption.

⟨1⟩2. CASE:  $n \geq 4$

⟨2⟩1. Choose acceptors  $a_1, \dots, a_n$  such that:

1.  $\mathcal{A} = \{a_1, \dots, a_n\}$
2.  $\{p_1, l_1\} \cap \mathcal{A} \subseteq \{a_1, a_2\}$
3.  $\{p_2, l_2\} \cap \mathcal{A} \subseteq \{a_{n-1}, a_n\}$

PROOF: It follows from step ⟨1⟩1, the level ⟨1⟩ case assumption ( $n \geq 4$ ), and the definition of  $n$  (the cardinality of  $\mathcal{A}$ ) that we can number the acceptors in this way.

⟨2⟩2.  $\{p_1, l_1\} \cup \{a_1, \dots, a_{n-f}\}$  and  $\{p_2, l_2\} \cup \{a_{f+1}, \dots, a_n\}$  are disjoint.

⟨3⟩1.  $\{a_1, \dots, a_{n-f}\}$  and  $\{a_{f+1}, \dots, a_n\}$  are disjoint.

PROOF: Assumption 2 ( $n \leq 2f$ ) implies  $n - f < f + 1$ . This implies the disjointness of  $\{a_1, \dots, a_{n-f}\}$  and  $\{a_{f+1}, \dots, a_n\}$ , since ⟨2⟩1.1 and the definition of  $n$  imply that the  $a_i$  are all distinct.

⟨3⟩2. 1.  $\{p_1, l_1\}$  and  $\{a_{f+1}, \dots, a_n\}$  are disjoint.

2.  $\{p_2, l_2\}$  and  $\{a_1, \dots, a_{n-f}\}$  are disjoint.

PROOF: The level ⟨1⟩ case assumption ( $n \geq 4$ ) and assumption 2 ( $n \leq 2f$ ) imply  $f \geq 2$ . Since this implies  $f + 1 > 2$ , part 1 follows from ⟨2⟩1.2 and ⟨3⟩1. Since  $f \geq 2$  implies  $n - f < n - 1$ , part 2 follows from ⟨2⟩1.3 and ⟨3⟩1.

⟨3⟩3. Q.E.D.

PROOF: ⟨2⟩2 follows from ⟨1⟩1, ⟨3⟩1, and ⟨3⟩2.

⟨2⟩3. 1.  $\{a_1, \dots, a_{n-f}\}$  is an accepting set for  $p_1$  in *Alg*.

2.  $\{a_{f+1}, \dots, a_n\}$  is an accepting set for  $p_2$  in *Alg*.

PROOF: By assumption 3 and ⟨1⟩1 (the  $p_i$  are proposers).

⟨2⟩4. Q.E.D.

PROOF: Steps ⟨1⟩1 (the  $p_i$  are proposers and the  $l_i$  are learners), ⟨2⟩2, ⟨2⟩3, and assumption 1 contradict the Accepting Lemma.

⟨1⟩3. CASE:  $n \leq 3$  and the set of agents is not anomalous.

PROOF: It suffices to show that, if  $n \leq 3$ , then except in the anomalous case with three agents, we can choose proposers  $p_1$  and  $p_2$ , learners  $l_1$  and  $l_2$ , and sets  $Q_1$  and  $Q_2$  each containing  $n - f$  acceptors such that  $\{p_1, l_1\} \cup Q_1$  and  $\{p_2, l_2\} \cup Q_2$  are disjoint. By the Agent Assumption, it suffices to show this when  $\mathcal{A}$  is the set  $\{1, \dots, n\}$  and there exist two distinct proposers and two distinct learners in the set  $\{1, \dots, n + 4\}$ . The existence of the  $p_i$ ,  $l_i$ , and  $Q_i$  is therefore asserted by the TLA<sup>+</sup> formula of Figure 1. The validity of this formula has been verified with the TLC model checker.

⟨1⟩4. Q.E.D.

PROOF: Cases ⟨1⟩2 and ⟨1⟩3 are exhaustive.

$$\begin{aligned}
& \forall n \in 0..3 : \\
& \quad \forall f \in 0..n : \\
& \quad \quad n \leq 2 * f \Rightarrow \\
& \quad \quad \forall pp1, pp2, ll1, ll2 \in 1..(n+4) : \\
& \quad \quad \quad (pp1 \neq pp2) \wedge (ll1 \neq ll2) \Rightarrow \\
& \quad \quad \quad \quad \forall \exists p1 \in \{pp1, pp2\} : \\
& \quad \quad \quad \quad \quad \exists p2 \in \{pp1, pp2\} \setminus \{p1\} : \\
& \quad \quad \quad \quad \quad \quad \exists ll \in \{ll1, ll2\} : \\
& \quad \quad \quad \quad \quad \quad \quad \exists l2 \in \{ll1, ll2\} \setminus \{ll\} : \\
& \quad \quad \quad \quad \quad \quad \quad \quad \exists Q1, Q2 \in \text{SUBSET}(1..n) : \\
& \quad \quad \quad \quad \quad \quad \quad \quad \wedge \text{Cardinality}(Q1) = n - f \\
& \quad \quad \quad \quad \quad \quad \quad \quad \wedge \text{Cardinality}(Q2) = n - f \\
& \quad \quad \quad \quad \quad \quad \quad \quad \wedge (\{p1, ll\} \cup Q1) \cap (\{p2, l2\} \cup Q2) = \{\} \\
& \quad \quad \quad \vee \wedge n = 2 \\
& \quad \quad \quad \quad \wedge f = 1 \\
& \quad \quad \quad \quad \wedge \exists a1, a2 \in \{1, 2\} : \\
& \quad \quad \quad \quad \quad \exists lp \in \{pp1, pp2\} \cap \{ll1, ll2\} : \\
& \quad \quad \quad \quad \quad \quad \wedge \{pp1, pp2\} = \{a1, lp\} \\
& \quad \quad \quad \quad \quad \quad \wedge \{ll1, ll2\} = \{a2, lp\} \\
& \quad \quad \quad \quad \quad \quad \wedge lp \notin \{a1, a2\}
\end{aligned}$$

Figure 1: TLA<sup>+</sup> formula used in the proof of the Acceptor Lower Bound Theorem.

### A.3 The Fast Accepting Lemma

**Lemma (Fast Accepting)** For any consistent asynchronous algorithm *Alg*, if there exist proposers  $p_1, p_2$ , and  $p_q$ , learners  $l_1, l_2$  and  $l_q$ , fast-accepting sets  $M_1$  for  $p_1$  and  $M_2$  for  $p_2$  in *Alg*, and a quorum  $Q$  for *Alg* such that

- $p_1 \neq p_2$
- $p_1 \notin M_2$  and  $p_2 \notin M_1$
- $l_1 \notin \{p_2, p_q, l_q\} \cup (M_2 \setminus M_1) \cup Q$
- $l_2 \notin \{p_1, p_q, l_q\} \cup (M_1 \setminus M_2) \cup Q$
- $\{p_q, l_q\} \cap M_1 \cap M_2$  is empty

then  $M_1 \cap M_2 \cap Q$  is nonempty.

The proof follows the proof sketch in Section 2.3. Recall that  $\neg 1 = 2$  and  $\neg 2 = 1$ .

## Proof

- ASSUME: 1.  $Alg$  is a consistent asynchronous algorithm.  
 2.  $p_1, p_2$ , and  $p_q$  are proposers,  $l_1, l_2$ , and  $l_q$  are learners, and  $M_1, M_2$ , and  $Q$  are sets of acceptors.  
 3.  $M_i$  is a fast-accepting set for  $p_i$  in  $Alg$ , for  $i = 1, 2$ .  
 4.  $Q$  is a quorum for  $Alg$ .  
 5.  $p_1 \neq p_2$   
 6.  $p_i \notin M_{-i}$ , for  $i = 1, 2$ .  
 7.  $l_i \notin \{p_{-i}, p_q, l_q\} \cup (M_{-i} \setminus M_i) \cup Q$ , for  $i = 1, 2$ .  
 8.  $\{p_q, l_q\} \cap M_1 \cap M_2$  is empty.  
 9.  $M_1 \cap M_2 \cap Q$  is empty.

PROVE: FALSE

- \langle 1 \rangle 1. For  $i = 1, 2$ , choose proposable values  $v_i$  and a scenario  $S_i$  such that:
1.  $v_1 \neq v_2$ .
  2.  $S_i$  is in  $Alg$ .
  3.  $Agents(S_i) \subseteq \{p_i, l_i\} \cup M_i$ .
  4.  $S_i$  has depth at most 2.
  5. The only source of  $S_i$  is an event in which  $p_i$  proposes  $v_i$ .
  6.  $S_i$  contains an event  $e_i$  in which  $l_i$  learns  $v_i$ .

PROOF: The existence of the  $v_i$  follows from the Value Assumption. The existence of the  $S_i$  follows from assumptions 2 ( $p_i$  a proposer and  $l_i$  a learner) and 3 ( $M_i$  fast accepting for  $p_i$ ).

DEFINITION  $T_i \triangleq \{e \in S_i : e \preceq_{S_i} e_i\}$   
 $U_i \triangleq \{e \in T_i : (Depth(e, T_i) \leq 1) \wedge (e_{agent} \in \{p_i\} \cup (M_i \setminus M_{-i}))\}$

for  $i = 1, 2$ , where  $Depth(e, T_i)$  is the depth of event  $e$  in  $T_i$ .

- \langle 1 \rangle 2. For  $i = 1, 2$ :
1.  $U_i \sqsubseteq T_i \sqsubseteq S_i$
  2.  $T_i$  and  $U_i$  are scenarios in  $Alg$ .

PROOF:  $T_i$  is clearly a prefix of  $S_i$ . By \langle 1 \rangle 1.5, the only depth 0 events of  $S_i$  are performed by  $p_i$ , which implies that  $U_i$  is a prefix of  $T_i$ , proving part 1. Part 2 follows from part 1 by \langle 1 \rangle 1.2 and assumption 1 ( $Alg$  asynchronous).

- \langle 1 \rangle 3.  $U_1 \cup U_2$  is in  $Alg$ .

PROOF: The definition of  $U_i$  implies  $Agents(U_i) \subseteq \{p_i\} \cup (M_i \setminus M_{-i})$ . Assumption 6 therefore implies that  $Agents(U_1)$  and  $Agents(U_2)$  are

disjoint. Step ⟨1⟩2.2, assumption 1 (*Alg* asynchronous), and the Scenario Union Lemma then imply that  $U_1 \cup U_2$  is in *Alg*.

⟨1⟩4. For  $i = 1, 2$ :

1.  $l_i$  learns  $v_i$  in  $T_i$ .
2.  $Agents(U_{\neg i})$  and  $Agents(T_i)$  are disjoint.

PROOF: Part 1 follows from the definition of  $T_i$  and ⟨1⟩1.6. By ⟨1⟩1.3, ⟨1⟩2.1, and the definition of  $U_i$ , to prove part 2 it suffices to show that  $\{p_i, l_i\} \cup M_i$  and  $\{p_{\neg i}\} \cup (M_{\neg i} \setminus M_i)$  are disjoint. This follows from:

- $p_i \neq p_{\neg i}$  by assumption 5.
- $p_i \notin (M_{\neg i} \setminus M_i)$  by assumption 6.
- $l_i \notin \{p_{\neg i}\} \cup (M_{\neg i} \setminus M_i)$  by assumption 7.
- $M_i$  is disjoint from  $\{p_{\neg i}\}$  by assumption 6.
- $M_i$  is disjoint from  $M_{\neg i} \setminus M_i$  by definition of set difference.

⟨1⟩5. Choose a scenario  $V$  such that:

1.  $V$  is in *Alg*.
2.  $(U_1 \cup U_2) \sqsubseteq V$ .
3.  $Agents(V \setminus (U_1 \cup U_2)) \subseteq \{p_q, l_q\} \cup Q$
4.  $V$  contains a learning event  $eq$  performed by  $l_q$ .

PROOF: The existence of  $V$  follows from ⟨1⟩3, assumption 2 ( $p_q$  a proposer and  $l_q$  a learner), and assumption 4 ( $Q$  a quorum).

⟨1⟩6.  $T_i \cup V$  is a scenario of *Alg*, for  $i = 1, 2$ .

⟨2⟩1.  $T_i \cup U_{\neg i}$  is in *Alg*.

PROOF: By ⟨1⟩2.2, ⟨1⟩4.2, assumption 1 (*Alg* asynchronous) and the Scenario Union Lemma.

⟨2⟩2.  $U_1 \cup U_2 \sqsubseteq T_i \cup U_{\neg i}$

PROOF: Since  $T_i$  is a scenario (by ⟨1⟩2.1), an event  $e$  in  $T_i \cup U_{\neg i}$  precedes an event  $f$  in  $T_i$  iff  $e$  is in  $T_i$ , so ⟨1⟩2.1 ( $U_i \sqsubseteq T_i$ ) implies that  $U_1 \cup U_2$  is a prefix of  $T_i \cup U_{\neg i}$ .

⟨2⟩3.  $Agents((T_i \cup U_{\neg i}) \setminus (U_1 \cup U_2)) \subseteq \{l_i\} \cup (M_1 \cap M_2)$

PROOF: ⟨1⟩2.1 ( $U_i \sqsubseteq T_i$ ) and ⟨1⟩4.2 imply  $(T_i \cup U_{\neg i}) \setminus (U_1 \cup U_2)$  equals  $T_i \setminus U_i$ . Step ⟨1⟩1.3 asserts that  $Agents(T_i) \subseteq \{p_i, l_i\} \cup M_i$ , and ⟨1⟩1.4 and the definition of  $T_i$  imply that the only events in  $T_i$  of depth greater than 1 are performed by  $l_i$ . Hence the definitions of  $T_i$  and  $U_i$  imply  $Agents(T_i \setminus U_i) \subseteq \{l_i\} \cup (M_1 \cap M_2)$ , since  $M_i \setminus (M_i \setminus M_{\neg i}) = M_1 \cap M_2$ .

(2)4.  $Agents(V \setminus (U_1 \cup U_2))$  and  $Agents((T_i \cup U_{\neg i}) \setminus (U_1 \cup U_2))$  are disjoint.

PROOF: By (1)5.3 and (2)3, we must prove that  $\{l_i\} \cup (M_1 \cap M_2)$  and  $\{p_q, l_q\} \cup Q$  are disjoint. This follows from:

- $l_i \notin \{p_q, l_q\} \cup Q$  by assumption 7.
- $(M_1 \cap M_2)$  and  $\{p_q, l_q\}$  are disjoint by assumption 8.
- $M_1 \cap M_2$  and  $Q$  are disjoint by assumption 9.

(2)5. Q.E.D.

PROOF: (1)6 follows from (1)5.1, (1)5.2, (2)1, (2)2, (2)4, assumption 1, and part A2 of the definition of an asynchronous algorithm, substituting  $T \leftarrow V$ ,  $U \leftarrow T_i \cup U_{\neg i}$ , and  $S \leftarrow U_1 \cup U_2$ .

(1)7. Q.E.D.

PROOF: (1)4.1, (1)5.4, (1)6, and assumption 1 (*Alg* consistent) imply that  $l_q$  learns  $v_i$  in event  $eq$ , for  $i = 1, 2$ . This is impossible by (1)1.1.

#### A.4 The Fast Learning Theorem

**Theorem (Fast Learning)** For any natural numbers  $e$  and  $f$  with  $f > 0$  and  $e \leq f \leq n$  and for any asynchronous consensus algorithm *Alg*, if every set of  $n - f$  acceptors is a quorum for *Alg* and every set of  $n - e$  acceptors is fast-accepting in *Alg* for two distinct proposers  $p_1$  and  $p_2$ , then  $n > 2e + f$  or the set of learners equals  $\{p_1, p_2\}$ .

The proof fills in the details missing from the proof sketch in Section 2.3.

#### Proof

ASSUME: 1. *Alg* is a consistent asynchronous Algorithm.

2.  $e$  and  $f$  are natural numbers with

1.  $e \leq f \leq n$

2.  $0 < f$

3.  $n \leq 2e + f$

3. Every set of  $n - f$  acceptors is a quorum for *Alg*.

4.  $p_1$  and  $p_2$  are proposers with  $p_1 \neq p_2$ .

5. Every set of  $n - e$  acceptors is fast-accepting for  $p_1$  and  $p_2$ .

6. The set of learners does not equal  $\{p_1, p_2\}$ .

PROVE: FALSE

(1)1. CASE: The set of agents is not anomalous.

(2)1. 1.  $n > 2f$

2.  $n > 2e$
3.  $e > 0$
4.  $n - e > 1$

PROOF: Part 1 follows from assumptions 1 and 3, the level  $\langle 1 \rangle$  case assumption, and the Acceptor Lower Bound Theorem. Part 2 follows from part 1 and assumption 2.1. Part 3 follows from

$$\begin{aligned} f &< n && \text{[by part 1]} \\ &\leq 2e + f && \text{[assumption 2.3]} \end{aligned}$$

Part 4 follows from

$$\begin{aligned} n - e &> 2f - e && \text{[by part 1]} \\ &\geq 1 + (f - e) && \text{[since } f \geq 1 \text{ by assumption 2.2]} \\ &\geq 1 && \text{[since } (f - e) \geq 0 \text{ by assumption 2.1]} \end{aligned}$$

$\langle 2 \rangle 2$ . Choose learners  $l_1$  and  $l_q$  such that

1.  $l_1 \neq l_q$
2.  $l_1 \notin \{p_1, p_2\}$
3. If there is a learner in  $\mathcal{A} \setminus \{p_1, p_2\}$ , then  $l_1 \in \mathcal{A}$ .

PROOF:  $l_1$  and  $l_q$  exist by the Agent Assumption and assumption 6.

$\langle 2 \rangle 3$ . Choose acceptors  $a_1, \dots, a_n$  such that

1.  $\mathcal{A} = \{a_1, \dots, a_n\}$
2. IF  $l_1 \in \mathcal{A}$  THEN  $l_1 = a_{n-e}$
3. IF  $p_1 \in \mathcal{A}$  THEN  $p_1 = a_1$
4. IF  $p_2 \in \mathcal{A}$  THEN  $p_2 = a_n$
5. IF  $(l_q \in \mathcal{A}) \wedge (l_q \notin \{p_1, p_2\})$   
THEN IF  $p_1 \notin \mathcal{A}$  THEN  $l_q = a_1$   
ELSE IF  $p_2 \notin \mathcal{A}$  THEN  $l_q = a_n$   
ELSE  $l_q = a_2$

$\langle 3 \rangle 1$ . CASE: a.  $l_q$  is not an acceptor, or  
b.  $l_q \in \{p_1, p_2\}$ , or  
c.  $p_1$  or  $p_2$  is not an acceptor.

PROOF: In each of these three cases, parts 2–5 of  $\langle 2 \rangle 3$  constrain at most the choices of  $a_1$ ,  $a_{n-e}$ , and  $a_n$ . By assumption 4 ( $p_1 \neq p_2$ ),  $\langle 2 \rangle 2.1$ , and  $\langle 2 \rangle 2.2$ , no two of these acceptors are constrained to equal the same agent. Hence, the constraints can be satisfied because  $\langle 2 \rangle 1.3$  and  $\langle 2 \rangle 1.4$  imply that 1,  $n - e$ , and  $n$  are three distinct integers.

$\langle 3 \rangle 2$ . CASE: 1.  $l_q, p_1$ , and  $p_2$  are all acceptors, and  
2.  $l_q \notin \{p_1, p_2\}$

$\langle 4 \rangle 1$ . CASE:  $l_1$  is not an acceptor.

PROOF: In this case,  $\langle 2 \rangle 3$  is satisfied with  $p_1 = a_1$ ,  $l_q = a_2$ , and

$p_2 = a_n$ , since  $2 < n$  by the level ⟨3⟩ case assumption and assumption 4 ( $p_1 \neq p_2$ ).

⟨4⟩2. CASE:  $l_1$  is an acceptor.

⟨5⟩1.  $n \geq 4$

PROOF: By the levels ⟨3⟩ and ⟨4⟩ case assumptions and ⟨2⟩2.

⟨5⟩2.  $n - e > 2$

⟨6⟩1. CASE:  $e > 1$

PROOF: In this case, ⟨5⟩2 follows from ⟨2⟩1.2, which implies  $n - e > e$ .

⟨6⟩2. CASE:  $e = 1$

PROOF: In this case, ⟨5⟩2 follows from ⟨5⟩1.

⟨6⟩3. Q.E.D.

PROOF: Cases ⟨6⟩1 and ⟨6⟩2 are exhaustive by ⟨2⟩1.3.

⟨5⟩3. Q.E.D.

PROOF: ⟨5⟩2 and ⟨2⟩1.3 imply  $2 < n - e < n$ , so by ⟨2⟩2 and the levels ⟨3⟩ and ⟨4⟩ case assumptions (which imply that  $l_q, l_1, p_1$ , and  $p_2$  are all distinct), we can satisfy ⟨2⟩3 by numbering the acceptors so that  $p_1 = a_1, l_q = a_2, l_1 = a_{n-e}$  and  $p_2 = a_n$ .

⟨4⟩3. Q.E.D.

PROOF: Cases ⟨4⟩1 and ⟨4⟩2 are exhaustive.

⟨3⟩3. Q.E.D.

PROOF: Cases ⟨3⟩1 and ⟨3⟩2 are exhaustive (by propositional logic).

DEFINITION  $M_1 \triangleq \{a_1, \dots, a_{n-e}\}$   
 $M_2 \triangleq \{a_{e+1}, \dots, a_n\}$

⟨2⟩4. Let  $Q$  be a subset of  $\{a_1, \dots, a_e\} \cup \{a_{n-e+1}, \dots, a_n\}$  containing  $n - f$  elements.

PROOF: ⟨2⟩1.2 (which implies  $n - e + 1 > e$ ) and ⟨2⟩3.1 (which implies that the  $e_i$  are all distinct) show that  $\{a_1, \dots, a_e\} \cup \{a_{n-e+1}, \dots, a_n\}$  contains  $2e$  elements, and assumption 2.3 implies  $n - f \leq 2e$ . Hence such a  $Q$  exists.

DEFINITION  $l_2 \triangleq l_1$  and  $p_q \triangleq p_1$

⟨2⟩5. 1.  $p_i \notin M_{-i}$ , for  $i = 1, 2$ .

2.  $l_i \notin \{p_{-i}, p_q, l_q\} \cup (M_{-i} \setminus M_i) \cup Q$ , for  $i = 1, 2$ .

3.  $\{p_q, l_q\} \cap M_1 \cap M_2$  is empty.

4.  $M_1 \cap M_2 \cap Q$  is empty.

⟨3⟩1.  $e + 1 \leq n - e < n - e + 1$



PROOF:  $e + 1 \leq n - e$  by  $\langle 2 \rangle 1.2$ .

$\langle 3 \rangle 2$ .  $M_1 \cap M_2 = \{a_{e+1}, \dots, a_{n-e}\}$

PROOF: By definition of  $M_1$  and  $M_2$ , since  $\langle 2 \rangle 1.2$  implies  $e+1 \leq n-e$  and  $\langle 2 \rangle 3.1$  implies that the  $a_i$  are distinct.

$\langle 3 \rangle 3$ .  $M_1 \cap M_2 \cap Q$  is empty.

PROOF: By  $\langle 3 \rangle 2$  and  $\langle 2 \rangle 4$ , since  $\langle 2 \rangle 3.1$  implies that the  $a_i$  are distinct.

$\langle 3 \rangle 4$ .  $p_i \notin M_{-i}$ , for  $i = 1, 2$ .

PROOF: Since  $e > 0$  by  $\langle 2 \rangle 1.3$ , we have

1.  $1 < e + 1$ , so  $p_1 \notin M_2$  by  $\langle 2 \rangle 3.3$ .

2.  $n - e < n$ , so  $p_2 \notin M_1$  by  $\langle 2 \rangle 3.4$ .

$\langle 3 \rangle 5$ .  $l_i \notin \{p_{-i}, p_q, l_q\} \cup (M_{-i} \setminus M_i) \cup Q$ , for  $i = 1, 2$ .

PROOF:

1.  $l_i \notin \{p_{-i}, p_q, l_q\}$  by  $\langle 2 \rangle 2.1$ ,  $\langle 2 \rangle 2.2$ , and the definitions of  $l_i$  and  $p_q$ .

2.  $l_i \notin (M_{-i} \setminus M_i) \cup Q$  by  $\langle 2 \rangle 3.2$  and  $\langle 3 \rangle 3$ , since  $\langle 3 \rangle 2$  implies that  $a_{n-e} \in M_1 \cap M_2$ .

$\langle 3 \rangle 6$ .  $\{p_q, l_q\} \cap M_1 \cap M_2$  is empty.

$\langle 4 \rangle 1$ .  $p_q \notin M_1 \cap M_2$

PROOF: By  $\langle 2 \rangle 3.3$  and  $\langle 3 \rangle 2$ , since  $\langle 2 \rangle 1.3$  implies  $1 < e + 1$ , and  $p_q$  is defined to equal  $p_1$ .

$\langle 4 \rangle 2$ .  $l_q \notin M_1 \cap M_2$

$\langle 5 \rangle 1$ . CASE:  $l_q = a_2$

$\langle 6 \rangle 1$ .  $n \geq 4$

PROOF: The level  $\langle 5 \rangle$  case assumption and  $\langle 2 \rangle 3$  imply that the IF clause of  $\langle 2 \rangle 3.5$  is true and that  $p_1$  and  $p_2$  are acceptors. Step  $\langle 2 \rangle 2$  and assumption 4 ( $p_1 \neq p_2$ ) then imply that  $p_1$ ,  $p_2$ ,  $l_1$ , and  $l_q$  are all distinct acceptors, so  $n \geq 4$ .

$\langle 6 \rangle 2$ .  $n < 4e$

PROOF: This follows from

$$2n \leq 4e + 2f \quad [\text{by assumption 2.3}]$$

$$< 4e + n \quad [\text{by } \langle 2 \rangle 1.1]$$

$\langle 6 \rangle 3$ . Q.E.D.

PROOF:  $\langle 6 \rangle 1$  and  $\langle 6 \rangle 2$  imply  $e + 1 > 2$ . Step  $\langle 4 \rangle 2$  then follows from  $\langle 3 \rangle 2$  and the level  $\langle 5 \rangle$  case assumption.

$\langle 5 \rangle 2$ . CASE:  $l_q \neq a_2$

PROOF: In this case, if  $l_q$  is an acceptor, then  $\langle 2 \rangle 3$  implies that it equals  $a_1$  or  $a_n$ . Step  $\langle 4 \rangle 2$  then follows from  $\langle 3 \rangle 2$  because  $\langle 2 \rangle 1.3$  implies  $1 < e + 1$  and  $n - e < n$ .

⟨5⟩3. Q.E.D.

PROOF: Cases ⟨5⟩1 and ⟨5⟩2 are exhaustive.

⟨4⟩3. Q.E.D.

PROOF: ⟨3⟩6 follows from ⟨4⟩1 and ⟨4⟩2.

⟨3⟩7. Q.E.D.

PROOF: ⟨2⟩5 follows from ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, and ⟨3⟩3.

⟨2⟩6. Q.E.D.

PROOF:  $M_1$  and  $M_2$  are both fast-accepting sets for  $p_1$  and  $p_2$  in  $Alg$  by assumption 5, and  $Q$  is a quorum for  $Alg$  by assumption 3. The other hypotheses of the Fast-Accepting Lemma are asserted by assumption 4, ⟨2⟩5.1, ⟨2⟩5.2, and ⟨2⟩5.3. The conclusion of the Fast-Accepting Lemma and ⟨2⟩5.4 provide the desired contradiction.

⟨1⟩2. CASE: The set of agents is anomalous.

⟨2⟩1. Let  $a_1$ ,  $a_2$ , and  $a_3$  be three distinct agents such that  $\mathcal{A} = \{a_1, a_2\}$ ,  $\{a_1, a_3\}$  is the set of proposers, and  $\{a_2, a_3\}$  is the set of learners.

PROOF: The  $a_i$  exist by the level ⟨1⟩ case assumption.

DEFINITION  $q_1 \triangleq a_1$  and  $q_2 \triangleq a_3$

⟨2⟩2.  $n = 2$ ,  $f = 1$ , and  $e > 0$ .

PROOF: ⟨2⟩1 implies  $n = 2$ , and assumption 1 ( $Alg$  consistent and asynchronous), assumption 2.2 ( $f > 0$ ), assumption 3 ( $n - f$  acceptors are a quorum), and the Accepting Lower Bound Theorem imply  $f = 1$ . Assumption 2.3 ( $n \leq 2e + f$ ) then implies  $e > 0$ .

⟨2⟩3. Let  $v_1$  and  $v_2$  be proposable values with  $v_1 \neq v_2$  and, for  $i = 1, 2$ , let  $T_i$  be a scenario in  $Alg$  such that:

1. The only source event of  $T_i$  is a proposal of  $v_i$  by  $q_i$ .
2.  $Agents(T_i) = \{q_i, a_2\}$ .
3.  $T_i$  has depth at most 2.
4. There is an event  $e_i$  in  $T_i$  in which  $a_2$  learns  $v_i$ .

PROOF: The  $v_i$  exist by the Value Assumption. Step ⟨2⟩1 implies  $\{p_1, p_2\}$  equals  $\{q_1, q_2\}$ , and ⟨2⟩2 and assumption 2.1 imply  $e = 1$ , so assumption 5 implies  $\{a_2\}$  is fast-accepting for  $q_1$  and  $q_2$ . This implies the existence of the scenarios  $T_i$  in  $Alg$ .

DEFINITION For  $i = 1, 2$ :  $S_i \triangleq \{e \in T_i : e \preceq_{T_i} e_i\}$   
 $R_i \triangleq \{e \in R_i : e_{agent} = q_i\}$

⟨2⟩4. For  $i = 1, 2$ :

1.  $R_i$  and  $S_i$  are in  $Alg$ .

2.  $R_i \sqsubseteq S_i$
3.  $a_2$  learns  $v_i$  in  $S_i$ .

PROOF:  $S_i$  is a prefix of  $T_i$ , so it is in  $Alg$  by assumption 1 ( $Alg$  asynchronous). By  $\langle 2 \rangle 3.4$ ,  $a_2$  learns  $v_i$  in  $S_i$ . By  $\langle 2 \rangle 1$  (which implies  $q_i \neq a_2$ ),  $\langle 2 \rangle 3.1$ , and  $\langle 2 \rangle 3.3$ ,  $q_i$  cannot receive a message from another agent in  $S_i$ . Hence  $R_i$  is a prefix of  $S_i$  and is therefore in  $Alg$  by assumption 1.

- $\langle 2 \rangle 5$ . 1.  $R_1 \cup R_2$  is in  $Alg$
2. For  $i = 1, 2$ : 1.  $S_i \cup R_{-i}$  is in  $Alg$ .
    2.  $R_1 \cup R_2 \sqsubseteq S_i \cup R_{-i}$ .
    3.  $Agents((S_i \cup R_{-i}) \setminus (R_1 \cup R_2)) = \{a_2\}$

PROOF: Assumption 1 ( $Alg$  asynchronous),  $\langle 2 \rangle 4.1$ ,  $\langle 2 \rangle 3.2$ , the definition of  $R_{-i}$  (which implies  $Agents(R_{-i}) = \{q_{-i}\}$ ),  $\langle 2 \rangle 1$  (which implies  $q_1, q_2$ , and  $a_2$  are distinct agents), and the Scenario Union Lemma imply that  $S_i \cup R_{-i}$  is in  $Alg$ . Step  $\langle 2 \rangle 4.2$  implies  $R_1 \cup R_2$  is a prefix of  $S_i \cup R_{-i}$ , so it is in  $Alg$  by assumption 1 ( $Alg$  asynchronous). The definition of  $R_i$  and  $\langle 2 \rangle 3.2$  imply  $Agents((S_i \cup R_{-i}) \setminus (R_1 \cup R_2)) = \{a_2\}$ .

- $\langle 2 \rangle 6$ . Choose a scenario  $U$  such that
1.  $U$  is in  $Alg$ .
  2.  $R_1 \cup R_2 \sqsubseteq U$ .
  3.  $Agents(U \setminus (R_1 \cup R_2)) \subseteq \{a_1, a_3\}$
  4.  $a_3$  learns a value in  $U$ .

PROOF: Scenario  $U$  exists by  $\langle 2 \rangle 5.1$ ,  $\langle 2 \rangle 1$  ( $a_1$  a proposer and  $a_3$  a learner and acceptor),  $\langle 2 \rangle 2$  ( $f = 1$ ), and assumption 3 (quorum assumption).

- $\langle 2 \rangle 7$ .  $U \cup S_i$  is in  $Alg$ , for  $i = 1, 2$ .

PROOF: By  $\langle 2 \rangle 5.2$ ,  $\langle 2 \rangle 6$ , assumption 1, and part A2 of the definition of an asynchronous algorithm, since  $\langle 2 \rangle 6.2$  implies  $U \cup S_i = U \cup (S_i \cup R_{-i})$  and  $\langle 2 \rangle 1$  implies that  $\{a_2\}$  and  $\{a_1, a_3\}$  are disjoint.

- $\langle 2 \rangle 8$ . Q.E.D.

PROOF:  $\langle 2 \rangle 4.3$  implies that  $a_2$  learns  $v_i$  in  $U \cup S_i$ , for  $i = 1, 2$ , and  $\langle 2 \rangle 6.4$  asserts that  $a_3$  learns a value in  $U$ . Since  $v_1 \neq v_2$  by  $\langle 2 \rangle 3$ , assumption 1 (consistency) implies that  $U \cup S_1$  and  $U \cup S_2$  cannot both be in  $Alg$ , contradicting  $\langle 2 \rangle 7$ .

- $\langle 1 \rangle 3$ . Q.E.D.

PROOF: Cases  $\langle 1 \rangle 1$  and  $\langle 1 \rangle 2$  are exhaustive.

	$p_1$	$p_2$	$a_1, \dots, a_n$	$l$
0	$prop\ v_1$	$prop\ v_2$		
1	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_2 \rangle$
2				$\{\langle p_2 : p_2 \rangle, \langle a_1 : p_2 \rangle,$ $\dots, \langle a_n : p_2 \rangle, \langle l : p_2 \rangle\}$

Figure 2: Example of a tabular scenario specification.

### A.5 The Collision-Fast Learning Theorem

**Theorem (Collision-Fast Learning)** For any natural numbers  $e$  and  $f$ , with  $e \leq f \leq n$  and  $f > 0$ , and any asynchronous consensus algorithm  $Alg$  with independent delivery, if any set of  $n - f$  acceptors is a quorum for  $Alg$  and there are two distinct proposers  $p_1$  and  $p_2$  such that any set of  $n - e$  acceptors is collision-fast accepting for  $\{p_1, p_2\}$  in  $Alg$ , then  $e = 0$  and

- a.  $f = 1$ , every learner is an acceptor, and at least one acceptor is not a learner, or
- b.  $p_1$  or  $p_2$  (or both) is an acceptor.

We transform the proof sketch in Section 2.4 into a rigorous proof. The difficulty lies in handling the cases in which one or both of the  $p_i$  is an acceptor and/or there is no learner  $l$  that is neither an acceptor nor one of the  $p_i$ . We try to make the proof easier to understand by using a tabular method of specifying scenarios illustrated by the example in Figure 2. In this figure:

- $prop\ v_1$  and  $prop\ v_2$  are depth 0 events that propose the values  $v_1$  and  $v_2$ , respectively. These two events are assumed to be the same ones in all our tabular descriptions.
- Each  $\langle p_i \rangle$  is a depth 1 event that receives the message generated by  $p_i$ 's  $prop\ v_i$  event.
- Each  $\langle b : p_i \rangle$  is a depth 2 event that receives the message sent by agent  $b$ 's event  $\langle p_i \rangle$ .

	$p_1$	$p_2$	$a_1, \dots, a_n$	$l$
0	$prop\ v_1$	$prop\ v_2$		
1	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_2 \rangle$
2				$\{\langle p_2 : p_2 \rangle, \langle a_1 : p_2 \rangle,$ $\dots, \langle a_n : p_2 \rangle, \langle l : p_2 \rangle\}$

Figure 3: Another example of a tabular scenario specification.

The figure specifies any scenario whose set of agents is  $\{p_1, p_2, a_1, \dots, a_n, l\}$  such that:

- The depth 0 events consist of  $prop\ v_1$  performed by  $p_1$  and  $prop\ v_2$  performed by  $p_2$ .
- The depth 1 events of  $p_1$  consist of  $\langle p_1 \rangle$  followed by  $\langle p_2 \rangle$ ; the depth 1 events of  $p_2, a_1, \dots, a_n$  consist of  $\langle p_2 \rangle$  followed by  $\langle p_1 \rangle$ ; and  $l$  performs the single depth 1 event  $\langle p_2 \rangle$ .
- The only depth 2 events are performed by  $l$  and consist of the ones in the set  $\{\langle p_2 : p_2 \rangle, \langle a_1 : p_2 \rangle, \dots, \langle a_n : p_2 \rangle, \langle l : p_2 \rangle\}$ , performed in any order.

Figure 2 does not imply that the agents  $p_1, \dots, l$  are all different. For example, if  $p_2$  and  $a_1$  are the same agent, then that agent first performs the  $prop\ v_2$  event and then performs the events  $\langle p_2 \rangle$  and  $\langle p_1 \rangle$ . However, the picture does not specify any scenario in case certain of the agents are equal. For example,  $p_1$  and  $a_1$  cannot be the same agent, since the picture specifies that they perform the events  $\langle p_1 \rangle$  and  $\langle p_2 \rangle$  in different orders. Similarly,  $p_2$  and  $l$  cannot be the same agent because  $p_2$  performs the depth 1 event  $\langle p_1 \rangle$  and  $l$  does not.

A boxed entry in a tabular specification indicates that the entry replaces the corresponding entry for any other column that describes the same agent. For example, Figure 3 is the same as Figure 2 except for the boxed depth 1 entry of agent  $l$ . This specification allows  $p_2$  and  $l$  to be equal. The boxed entry means that, if  $l$  equals  $p_2$ , then that agent's only depth 1 event is  $\langle p_2 \rangle$ . As before, the agent's depth 0 event is  $prop\ v_2$  and its depth 2 events are the ones in  $\{\langle p_2 : p_2 \rangle, \dots, \langle a_n : p_2 \rangle\}$ . However, if the boxed entry contained the event  $\langle p_1 \rangle$ , then Figure 3 would not specify any scenario when  $l$  equals

	$p_1$	$p_2$	$a_1, \dots, a_j$	$a_{j+1}, \dots, a_n$
0	$prop\ v_1$	$prop\ v_2$		
1	$\langle p_1 \rangle$	$\langle p_2 \rangle$	$\langle p_1 \rangle$	$\langle p_2 \rangle$
	$\langle p_2 \rangle$	$\langle p_1 \rangle$	$\langle p_2 \rangle$	$\langle p_1 \rangle$

Figure 4: Tabular specification of a normal scenario  $S_j$ , for all  $j$  in  $\pi_1 \dots \pi_2$ .

$p_2$  because it would assert that this agent did not perform the depth 1 event  $\langle p_2 \rangle$  that generates the message received by its depth 2  $\langle p_2 : p_2 \rangle$  event.

### Proof

- ASSUME: 1.  $Alg$  is a nontrivial, consistent, asynchronous algorithm with independent delivery.
2.  $e$  and  $f$  are natural numbers with
1.  $e \leq f \leq n$
  2.  $0 < f$
3. Every set of  $n - f$  acceptors is a quorum for  $Alg$ .
4.  $p_1$  and  $p_2$  are proposers with  $p_1 \neq p_2$ .
5. Every set of  $n - e$  acceptors is collision-fast accepting for  $\{p_1, p_2\}$ .
6. a.  $e > 0$ , or
- b. 1. a.  $f > 1$ , or
  - b. there is a learner that is not an acceptor, or
  - c. every acceptor is a learner
2.  $p_1$  and  $p_2$  are both not acceptors

PROVE: FALSE

- $\langle 1 \rangle$ 1. Choose acceptors  $a_1, \dots, a_n$  such that
1.  $\{a_1, \dots, a_n\}$  is the set of acceptors.
  2. If  $p_1$  is an acceptor, then  $p_1 = a_1$ .
  3. If  $p_2$  is an acceptor, then  $p_2 = a_n$ .

PROOF: Assumptions 1 ( $Alg$  asynchronous and consistent), 2.2, and 3 and the Acceptor Lower Bound Theorem imply  $n \geq 2$ , so such a numbering of the acceptors exists by assumption 4 ( $p_1 \neq p_2$ ).

DEFINITION  $\pi_1 \triangleq$  IF  $p_1$  is an acceptor THEN 1 ELSE 0  
 $\pi_2 \triangleq$  IF  $p_2$  is an acceptor THEN  $n - 1$  ELSE  $n$

- ⟨1⟩2. Choose two proposable values  $v_1$  and  $v_2$  with  $v_1 \neq v_2$ , and choose a normal scenario  $S_j$  of  $Alg$  satisfying the specification of Figure 4, for all  $j$  in  $\pi_1 \dots \pi_2$ .

PROOF: The Value Assumption implies the existence of  $v_1$  and  $v_2$ . Assumption 4 ( $p_1 \neq p_2$ ), ⟨1⟩1.2, and ⟨1⟩1.3 imply that the figure specifies a normal scenario  $S_j$  for  $j = 1, \dots, n-1$ , for  $j = 0$  if  $p_1$  is not an acceptor, and for  $j = n$  if  $p_2$  is not an acceptor. Hence,  $S_j$  exists for all  $j$  in  $\pi_1 \dots \pi_2$ . Assumption 5 (collision-fast accepting) implies that  $Alg$  contains a scenario satisfying

	$p_1$	$p_2$
0	$prop\ v_1$	$prop\ v_2$

Using assumption 1 (independent delivery), a simple induction argument starting with this scenario shows that  $Alg$  contains the normal scenario  $S_j$ , for all  $j$  in  $\pi_1 \dots \pi_2$ .

- ⟨1⟩3. For each learner  $l$ , and each  $j$  in  $\pi_1 \dots \pi_2$ , choose a normal scenario  $T_j(l)$  such that:
1.  $T_j(l)$  is in  $Alg$ .
  2.  $S_j \subseteq T_j(l)$ .
  3.  $Agents(T_j(l) \setminus S_j) = \{l\}$ .
  4.  $l$  learns  $v_1$  or  $v_2$  in  $T_j(l)$ .

PROOF: Assumption 1 (independent delivery) implies that, by adding events of  $l$  to  $S_j$ , we can construct a normal scenario  $T_j(l)$  satisfying properties 1–3, in which  $l$  receives every message sent by a depth 0 or depth 1 event performed by itself or by any of the agents  $p_1, p_2, a_1, \dots, a_{n-e}$ . (This can be done even if  $l$  equals one of the agents  $p_1, \dots, a_n$ .) Let  $Z$  be the set of events in  $T_j(l)$  performed by agents in  $\{l, p_1, p_2, a_1, \dots, a_{n-e}\}$ . Then  $Z$  is a prefix of  $T_j(l)$  in which  $l$  is complete to depth 2. Assumption 5 (collision-fast accepting) implies that  $l$  learns a value in  $Z$  and hence in  $T_j(l)$ , and assumption 1 (nontriviality) implies that  $l$  can learn only  $v_1$  or  $v_2$ .

- ⟨1⟩4.  $T_j(l_1) \cup T_j(l_2)$  is a scenario in  $Alg$ , for all  $j$  in  $\pi_1 \dots \pi_2$  and all learners  $l_1$  and  $l_2$ .

PROOF: This is trivial if  $l_1 = l_2$ . If  $l_1 \neq l_2$ , it follows from parts 1–3 of ⟨1⟩3, assumption 1, and condition A2 in the definition of an asynchronous algorithm.

- ⟨1⟩5. Choose  $k$  in  $(\pi_1 + 1) \dots \pi_2$  such that each learner  $l$  learns  $v_2$  in  $T_{k-1}(l)$  and  $v_1$  in  $T_k(l)$ .

	$p_1$	$a_1, \dots, a_{\pi_2}$	$l_1$
0	$prop\ v_1$		
1	$\langle p_1 \rangle$	$\langle p_1 \rangle$	$\langle p_1 \rangle$
2			$\{\langle p_1 : p_1 \rangle, \langle a_1 : p_1 \rangle, \dots, \langle a_{\pi_2} : p_1 \rangle, \langle l_1 : p_1 \rangle\}$

Figure 5: Tabular specification of scenario  $Q$ .

(2)1. Each learner  $l$  learns  $v_{\neg i}$  in  $T_{\pi_i}(l)$ , for  $i = 1, 2$ .

PROOF: We prove this for  $i = 2$ . The proof for  $i = 1$  is essentially symmetric. (In the proof for  $i = 1$ , we replace  $a_1, \dots, a_{\pi_2}$  by  $a_{\pi_1+1}, \dots, a_n$  and  $p_1$  by  $p_2$ .)

(3)1. Let  $l_1$  be a learner different from  $p_2$ , and let  $Q$  be a scenario satisfying the specification of Figure 5 such that

1.  $Q$  is in  $Alg$ .
2.  $l_1$  learns  $v_1$  in  $Q$ .
3.  $l$  performs the events in  $\{\langle p_1 : p_1 \rangle, \langle a_1 : p_1 \rangle, \dots, \langle a_{n-e} : p_1 \rangle\}$  before any other depth 2 events.

PROOF:  $l_1$  exists by the Agent Assumption. Figure 5 specifies scenarios, which are in  $Alg$  by assumption 1 (independent delivery), since assumption 5 implies that the scenario consisting only of the  $prop\ v_1$  event is in  $Alg$ . Assumption 6 implies  $e > 0$  or  $\pi_2 = n$ , so  $\{a_1, \dots, a_{\pi_2}\}$  contains  $\{a_1, \dots, a_{n-e}\}$  and a scenario  $Q$  therefore exists that satisfies (3)1.3 and the specification of Figure 5. This scenario has a prefix  $Y$  with agent set  $\{l, p_1, a_1, \dots, a_{n-e}\}$  such that  $l$  is complete to depth 2 in  $Y$ . Assumption 5 implies  $\{a_1, \dots, a_{n-e}\}$  is a collision-fast accepting set for  $\{p_1, p_2\}$ , so  $l$  learns a value in  $Y$  and hence in  $Q$ . By assumption 1 (nontriviality),  $l$  must learn  $v_1$ .

(3)2. Let  $R$  be a scenario satisfying the specification of Figure 6 such that  $Q \sqsubseteq R$  and

1.  $R$  is in  $Alg$
2.  $l_1$  learns  $v_1$  in  $R$ .

PROOF: There exists a scenario  $R$  satisfying Figure 6 and having prefix  $Q$  because  $p_2 \neq p_1$  by assumption 4,  $p_2 \neq l_1$



	$p_1$	$p_2$	$a_1, \dots, a_{\pi_2}$	$l_1$
0	<i>prop</i> $v_1$	<i>prop</i> $v_2$		
1	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_1 \rangle$
2				$\{\langle p_1 : p_1 \rangle, \langle a_1 : p_1 \rangle,$ $\dots, \langle a_{\pi_2} : p_1 \rangle, \langle l_1 : p_1 \rangle\}$

Figure 6: Tabular specification of a scenario  $R$ .

by  $\langle 3 \rangle 1$ , and  $p_2$  is not in  $\{a_1, \dots, a_{\pi_2}\}$  by  $\langle 1 \rangle 1.3$  and the definition of  $\pi_2$ . Assumption 1 (independent delivery) and  $\langle 3 \rangle 1.1$  imply that  $R$  is in  $Alg$ , and  $\langle 3 \rangle 1.2$  implies that  $l_1$  learns  $v_1$  in  $R$ .

- $\langle 3 \rangle 3$ . Let  $J$  be the set of all depth 0 and depth 1 events in  $R$ .
1.  $J$  is a prefix of  $R$ .
  2.  $J$  is in  $Alg$ .

PROOF:  $J$  is clearly a prefix of  $R$ , so it is in  $Alg$  by Assumption 1 ( $Alg$  asynchronous) and  $\langle 3 \rangle 2.1$ .

- $\langle 3 \rangle 4$ . Let  $p$  be a proposer and  $l_2$  a learner such that  $l_1$  is not in  $\{p, l_2\}$ , and let  $K$  be a scenario in  $Alg$  such that
1.  $J$  is a prefix of  $K$ .
  2.  $Agents(K \setminus J) \subseteq \{p, l_2\} \cup (\mathcal{A} \setminus \{l_1\})$
  3.  $l_2$  learns  $v_1$  in  $K$ .

- $\langle 4 \rangle 1$ . We can choose  $p$  and  $l_2$  such that  $l_1$  is not in  $\{p, l_2\}$ .  
PROOF: By the Agent Assumption.

- $\langle 4 \rangle 2$ . Let  $K$  be a scenario in  $Alg$  such that
1.  $J$  is a prefix of  $K$ .
  2.  $Agents(K \setminus J) \subseteq \{p, l_2\} \cup (\mathcal{A} \setminus \{l_1\})$
  3.  $l_2$  learns a value in  $K$ .

PROOF: Assumptions 2.2 ( $f > 0$ ) and 3 (quorum assumption) imply that  $\mathcal{A} \setminus \{l_1\}$  contains a quorum, and  $\langle 3 \rangle 3.2$  then implies the existence of  $K$  in  $Alg$  satisfying 1–3.

- $\langle 4 \rangle 3$ .  $R \cup K$  is in  $Alg$ .

PROOF: By  $\langle 3 \rangle 3$ ,  $\langle 4 \rangle 2$ , assumption 1, and part A2 of the definition of an asynchronous algorithm, since

	$p_1$	$p_2$	$a_1, \dots, a_{k-1}$	$a_{k+1}, \dots, a_n$
0	$prop\ v_1$	$prop\ v_2$		
1	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$	$\langle p_1 \rangle$ $\langle p_2 \rangle$	$\langle p_2 \rangle$ $\langle p_1 \rangle$

Figure 7: Tabular specification of the prefix  $U$  of  $S_{k-1}$  and  $S_k$ .

$Agents(R \setminus J) = \{l_1\}$  by  $\langle 3 \rangle 2$  and  $\langle 3 \rangle 3$ , and  $\{l_1\}$  is disjoint from  $Agents(K \setminus J)$  by  $\langle 4 \rangle 2.2$  and  $\langle 4 \rangle 1$ .

$\langle 4 \rangle 4$ . Q.E.D.

PROOF:  $\langle 4 \rangle 1$  and  $\langle 4 \rangle 2$  prove all of  $\langle 3 \rangle 4$  except  $\langle 3 \rangle 4.3$ . Assumption 1 (consistency),  $\langle 4 \rangle 2.3$ ,  $\langle 3 \rangle 2.2$ , and  $\langle 4 \rangle 3$  imply  $\langle 3 \rangle 4.3$ .

$\langle 3 \rangle 5$ .  $K \cup T_{\pi_2}(l_1)$  is in  $Alg$ .

$\langle 4 \rangle 1$ .  $J \sqsubseteq T_{\pi_2}(l_1)$  and  $Agents(T_{\pi_2}(l_1) \setminus J) = \{l_1\}$ .

PROOF: Since either  $p_2$  or  $a_{\pi_2}$  equals  $a_n$ , every agent other than  $l_1$  performs the same events in  $J$  as in  $T_{\pi_2}(l_1)$ . The sequence of events performed by  $l_1$  in  $J$  is a subsequence of the events that  $l_1$  performs in  $T_{\pi_2}(l_1)$ .

$\langle 4 \rangle 2$ . Q.E.D.

PROOF:  $\langle 3 \rangle 5$  follows from  $\langle 4 \rangle 1$ ,  $\langle 3 \rangle 4$  and assumption 1 ( $Alg$  asynchronous), since  $l_1$  not in  $\{p, l_2\}$  implies that  $\{l_1\}$  is disjoint from  $\{p, l_2\} \cup (\mathcal{A} \setminus \{l_1\})$ .

$\langle 3 \rangle 6$ . Q.E.D.

PROOF:  $\langle 3 \rangle 4.3$ ,  $\langle 1 \rangle 3.4$ ,  $\langle 3 \rangle 5$ , and assumption 1 (consistency) imply that  $l_1$  learns  $v_1$  in  $T_{\pi_2}(l_1)$ . Step  $\langle 2 \rangle 1$  (for  $i = 2$ ) then follows from  $\langle 1 \rangle 4$ ,  $\langle 1 \rangle 3.4$ , and assumption 1 (consistency).

$\langle 2 \rangle 2$ . Q.E.D.

PROOF:  $\langle 2 \rangle 1$  and  $\langle 1 \rangle 3.4$  imply that, for any individual learner  $l$ , there exists a  $k$  in  $(\pi_1 + 1) \dots \pi_2$  such that  $l$  learns  $v_2$  in  $T_{k-1}(l)$  and  $v_1$  in  $T_k(l)$ . That this holds for all  $l$  (with the same  $k$ ) follows from  $\langle 1 \rangle 4$  and assumption 1 (consistency).

$\langle 1 \rangle 6$ . Q.E.D.

⟨2⟩1. For all  $k \in \pi_1 \dots \pi_2$ , let  $U$  be the prefix of  $S_k$  specified by Figure 7.

1.  $U$  is a normal scenario that is a prefix of both  $S_{k-1}$  and  $S_k$ .
2.  $U$  is in  $Alg$ .
3.  $Agents(T_{k-1}(l) \setminus U)$  and  $Agents(T_k(l) \setminus U)$  are subsets of  $\{a_k, l\}$ , for any learner  $l$ .

PROOF: The definition of  $\pi_1$  and  $\pi_2$  imply that Figure 7 specifies a normal scenario if  $\pi_1 \leq k \leq \pi_2$ . Parts 1 and 3 are then obvious. Part 2 follows from part 1 by ⟨1⟩2 and assumption 1 ( $Alg$  asynchronous).

⟨2⟩2. CASE:  $e = 0$

⟨3⟩1. Choose a learner  $l_1$  such that:

- IF  $a_k$  is a learner THEN  $l_1 = a_k$   
ELSE a.  $l_1$  is not an acceptor, or  
b.  $f > 1$

PROOF: The existence of  $l_1$  follows from the level ⟨2⟩ case assumption ( $e = 0$ ) and part b.1 of assumption 6.

⟨3⟩2. Let  $l_2$  be a learner and let  $q$  be a proposer such that  $\{q, l_2\}$  and  $\{l_1, a_k\}$  are disjoint.

PROOF: The existence of a learner  $l_2$  not in  $\{l_1, a_k\}$  follows from the Agent Assumption and ⟨3⟩1, which implies that  $l_1 = a_k$  if  $a_k$  is a learner. The level ⟨2⟩ case assumption and part b.2 of assumption 6 imply that neither  $p_1$  nor  $p_2$  is an acceptor, so neither equals  $a_k$ . By assumption 4 ( $p_1 \neq p_2$ ), at least one of the  $p_i$  is not equal to  $l_1$ , so we can choose  $q$  to be that  $p_i$ .

⟨3⟩3. Let  $V$  be a behavior such that:

1.  $V$  is in  $Alg$
2.  $U \subseteq V$
3.  $Agents(V \setminus U) \subseteq (\{q, l_2\} \cup \mathcal{A}) \setminus \{a_k, l_1\}$
4.  $l_2$  learns a value in  $V$ .

PROOF: By ⟨3⟩2,  $(\{q, l_2\} \cup \mathcal{A}) \setminus \{a_k, l_1\}$  contains  $q$  and  $l_2$ , and by assumption 2.2 and ⟨3⟩1 (which implies  $l_1 = a_k$  if  $l_1$  an acceptor), it contains at least  $n - f$  acceptors. By ⟨2⟩1.2 and assumption 3 (quorum assumption), we can choose  $V$  satisfying 1–4.

⟨3⟩4.  $V \cup T_{k-1}(l_1)$  and  $V \cup T_k(l_1)$  are in  $Alg$ .

PROOF: By ⟨2⟩1, ⟨3⟩3, assumption 1, and part A2 of the

definition of an asynchronous algorithm.

⟨3⟩5. Q.E.D.

PROOF: Step ⟨1⟩5 asserts that  $l_1$  learns  $v_2$  in  $T_{k-1}(l_1)$  and  $v_1$  in  $T_k(l_1)$ . Steps ⟨3⟩3.4 and ⟨3⟩4 then imply that  $Alg$  is not consistent, contradicting assumption 1.

⟨2⟩3. CASE:  $e > 0$

⟨3⟩1. Let  $l_1$  and  $l_2$  be two distinct learners such that  $l_2 \neq a_k$ , and let  $W$  be a normal scenario such that

1.  $W$  is in  $Alg$ .
2.  $U \subseteq W$
3.  $Agents(W \setminus U) = \{l_2\}$
4.  $l_2$  learns a value in  $W$ .

PROOF: We can choose  $l_1$  and  $l_2$  by the Agent Assumption. The level ⟨2⟩ case assumption ( $e > 0$ ) and assumption 5 imply that there is a subset  $\mathcal{B}$  of  $\mathcal{A} \setminus \{a_k\}$  that is collision-fast in  $Alg$ . for  $\{p_1, p_2\}$ . By ⟨2⟩1.2 and assumption 1 (independent delivery), we can choose a scenario  $W$  satisfying 1–3 containing a normal prefix  $X$  such that  $Agents(X) = \{l_2, p_1, p_2\} \cup \mathcal{B}$  and  $l$  is complete to depth 2 in  $X$ . By definition of collision-fast, this implies that  $l_2$  learns a value in  $X$  and hence in  $W$ .

⟨3⟩2.  $W \cup T_{k-1}(l_1)$  and  $W \cup T_k(l_1)$  are in  $Alg$ .

PROOF: By ⟨1⟩3.1, ⟨2⟩1.1 (which by ⟨1⟩3.2 implies  $U$  is a prefix of  $T_{k-1}(l_1)$  and  $T_k(l_1)$ ), ⟨2⟩1.3, ⟨3⟩1, assumption 1, and part A2 of the definition of an asynchronous algorithm, since ⟨3⟩1 implies that  $\{a_k, l_1\}$  and  $\{l_2\}$  are disjoint.

⟨3⟩3. Q.E.D.

PROOF: ⟨3⟩1.4, ⟨3⟩2, and ⟨1⟩5 imply that  $Alg$  is not consistent, contradicting assumption 1 (consistency).

⟨2⟩4. Q.E.D.

PROOF: Cases ⟨2⟩2 and ⟨2⟩3 are exhaustive.

## A.6 The Hyperfast Learning Theorem

**Theorem (Hyperfast Learning)** A consistent asynchronous algorithm  $Alg$  cannot be hyperfast-accepting for two different proposers. For any integer  $f$  with  $0 < f \leq n$ , if every set of  $n - f$  acceptors is a quorum for

$Alg$  and  $Alg$  is hyperfast-accepting for a proposer  $p$ , then

1.  $f = 1$ ,
2.  $p$  is an acceptor that is not a learner, and
3. For every learner  $l$ , either  $l$  is an acceptor or  $\{p, l\}$  is the set of proposers.

The proof follows the proof sketch in Section 2.5.

### Proof

- ASSUME: 1.  $Alg$  is a consistent asynchronous algorithm.  
 2.  $f$  is an integer with  $0 < f \leq n$ .  
 3. Every set of  $n - f$  acceptors is a quorum.  
 4.  $Alg$  is hyperfast-accepting for a proposer  $p$ .

- PROVE: 1.  $Alg$  is not hyperfast-accepting for any proposer other than  $p$ .  
 2.  $f = 1$ .  
 3.  $p$  is an acceptor and not a learner.  
 4. For every learner  $l$ , either  $l$  is an acceptor or  $\{p, l\}$  is the set of proposers.

- $\langle 1 \rangle 1$ . ASSUME: 1.  $q$  is a proposer.  
 2.  $l$  is a learner.  
 3.  $v$  is a proposable value.  
 4.  $Alg$  is hyperfast-accepting for  $q$ .

PROVE: There exists a scenario  $S$  in  $Alg$  such that  $Agents(S) = \{q, l\}$  and  $l$  learns  $v$  in  $S$ .

- $\langle 2 \rangle 1$ . Let  $T$  be a scenario in  $Alg$  such that  $T$  has depth at most 1, has as its only source event one in which  $q$  proposes  $v$ , and contains an event  $e_l$  in which  $l$  learns  $v$ .

PROOF:  $T$  exists by the level  $\langle 1 \rangle$  assumptions and the definition of hyperfast-accepting.

- $\langle 2 \rangle 2$ . Let  $S$  be the prefix of  $T$  consisting of all events  $e$  in  $T$  such that  $e \preceq_S e_l$ . Then  $Agents(S) = \{q, l\}$ .

PROOF: Since  $e_l$  has depth at most 1 and  $\langle 2 \rangle 1$  implies that the only events of  $T$  with depth 0 are performed by  $q$ , any event of  $T$  that precedes or equals  $e_l$  is performed by  $q$  or  $l$ .

- $\langle 2 \rangle 3$ . Q.E.D.

PROOF: By  $\langle 2 \rangle 1$ ,  $\langle 2 \rangle 2$ , and assumption 1 ( $Alg$  asynchronous).

⟨1⟩2. ASSUME:  $Alg$  is hyperfast-accepting for a proposer  $p_2$  different from  $p$ .

PROVE: FALSE

⟨2⟩1. Choose learners  $l_1$  and  $l_2$  such that  $\{p, l_1\}$  and  $\{p_2, l_2\}$  are disjoint.

PROOF: The Agent Assumption implies the existence of  $l_1$  and  $l_2$ .

DEFINITION  $p_1 \triangleq p$

⟨2⟩2. Choose values  $v_1$  and  $v_2$  with  $v_1 \neq v_2$  and, for each  $i = 1, 2$ , let  $S_i$  be a scenario in  $Alg$  such that  $Agents(S_i) = \{p_i, l_i\}$  and  $l_i$  learns  $v_i$  in  $S_i$ .

PROOF:  $v_1$  and  $v_2$  exist by the Value Assumption. The  $S_i$  exist by ⟨1⟩1, the level ⟨1⟩ assumption, and assumption 4 ( $Alg$  hyperfast for  $p$ ).

⟨2⟩3. Q.E.D.

PROOF: ⟨2⟩1, ⟨2⟩2, assumption 1 ( $Alg$  asynchronous), and the Scenario Union Lemma imply  $S_1 \cup S_2$  is in  $Alg$ . By ⟨2⟩2, this implies that  $Alg$  is not consistent, contradicting assumption 1.

⟨1⟩3. ASSUME: a.  $f > 1$ , or

b.  $p$  is not an acceptor, or

c.  $p$  is a learner, or

d. There exists a learner  $l$  and proposer  $q$  such that:

1.  $l$  is not an acceptor, and

2.  $q$  is not in  $\{p, l\}$

PROVE: FALSE

⟨2⟩1. Choose learners  $l$  and  $l_2$  and a proposer  $p_2$  such that

1.  $\mathcal{A} \setminus \{p, l\}$  contains a quorum.

2.  $\{p, l\}$  and  $\{p_2, l_2\}$  are disjoint.

⟨3⟩1. CASE:  $f > 1$  or  $p$  is not an acceptor.

PROOF: The Agent Assumption implies that we can choose  $l$ ,  $p_2$ , and  $l_2$  satisfying ⟨2⟩1.2. Assumption 2 ( $f > 0$ ), assumption 3 (quorum assumption), and the case assumption then implies ⟨2⟩1.1.

⟨3⟩2. CASE:  $p$  is a learner.

PROOF: In this case, assumption 2 ( $f > 0$ ) and assumption 3 (quorum assumption) imply that ⟨2⟩1.1 holds with  $l = p$ . The existence of  $p_2$  and  $l_2$  satisfying ⟨2⟩1.2 then follows from the Agent Assumption.

⟨3⟩3. CASE: 1.  $p$  is not a learner

2. There exists a learner  $l$  and proposer  $q$  such that
  1.  $l$  is not an acceptor, and
  2.  $q$  is not in  $\{p, l\}$

PROOF: Part 2.1 of the case assumption, assumption 2 ( $f > 0$ ), and assumption 3 (quorum assumption) imply  $\langle 2 \rangle 1.1$ . Parts 1 and 2.2 of the case assumption and the Agent Assumption imply that we can choose a learner  $l_2$  such that  $\langle 2 \rangle 1.2$  is satisfied with  $p_2 = q$ .

$\langle 3 \rangle 4$ . Q.E.D.

PROOF: The level  $\langle 1 \rangle$  case assumption implies that cases  $\langle 3 \rangle 1$ ,  $\langle 3 \rangle 2$ , and  $\langle 3 \rangle 3$  are exhaustive.

$\langle 2 \rangle 2$ . Let  $S$  be a scenario in  $Alg$  with  $Agents(S) \subseteq \{p_2, l_2\} \cup (\mathcal{A} \setminus \{p, l\})$  such that  $l_2$  learns a value  $v_2$  in  $S$ .

PROOF:  $S$  exists by  $\langle 2 \rangle 1.1$  and the definition of a quorum (substituting the empty scenario for  $S$  in the definition).

$\langle 2 \rangle 3$ . Let  $v_1$  be a proposable value different from  $v_2$  and let  $T$  be a scenario in  $Alg$  with  $Agents(T) = \{p, l\}$  such that  $l$  learns  $v_1$  in  $T$ .

PROOF:  $T$  exists by assumption 4 ( $Alg$  hyperfast for  $p$ ),  $\langle 2 \rangle 1$  ( $l$  a learner) and  $\langle 1 \rangle 1$ .

$\langle 2 \rangle 4$ . Q.E.D.

PROOF:  $\langle 2 \rangle 1.2$ ,  $\langle 2 \rangle 2$ ,  $\langle 2 \rangle 3$ , assumption 1 ( $Alg$  asynchronous), and the Scenario Union Lemma imply that  $S \cup T$  is in  $Alg$ . By  $\langle 2 \rangle 2$  and  $\langle 2 \rangle 3$ , this implies that  $Alg$  is not consistent, contradicting assumption 1.

$\langle 1 \rangle 4$ . Q.E.D.

PROOF: The theorem follows from  $\langle 1 \rangle 2$  and  $\langle 1 \rangle 3$  by predicate logic.

## B Formal Statements of the Results

We now formalize our definitions and results in  $TLA^+$  [11]. We do not explain  $TLA^+$  notation here, but readers familiar with simple logic and set theory should be able to understand most of the formalism. The results are expressed as definitions, assumptions, and theorems in the following  $TLA^+$  module. This is a constant module, so it makes no use of  $TLA$  (the temporal logic of actions).  $TLA^+$  serves only as a convenient language for writing formulas of ordinary first-order logic and set theory. Although the

module contains some comments, most of the explanation is provided by the informal presentation in Section 2.

---

MODULE *LowerBounds*

---

EXTENDS *Naturals, Sequences, FiniteSets*

We take the conventional approach of representing a relation  $R$  by a set of ordered pairs, where  $\langle x, y \rangle \in R$  means  $x R y$ , and we define  $TransitiveClosure(R)$  to be the transitive closure of relation  $R$ . We locally define  $Dom$  and  $Rng$  to be the domain and range of  $R$ , and we recursively define  $TC[i]$  to be  $R^{i+1}$ , the composition of  $R$  with itself  $i + 1$  times.

$$\begin{aligned}
 &TransitiveClosure(R) \triangleq \\
 &\text{LET } Dom \triangleq \{r[1] : r \in R\} \\
 &\quad Rng \triangleq \{r[2] : r \in R\} \\
 &\quad TC[i \in Nat] \triangleq \\
 &\quad\quad \text{IF } i = 0 \text{ THEN } R \\
 &\quad\quad\quad \text{ELSE } \{\langle d, e \rangle \in Dom \times Rng : \\
 &\quad\quad\quad\quad \exists c \in Dom \cap Rng : \wedge \langle d, c \rangle \in TC[i - 1] \\
 &\quad\quad\quad\quad\quad \wedge \langle c, e \rangle \in R\} \\
 &\text{IN } \text{UNION } \{TC[i] : i \in Nat\}
 \end{aligned}$$


---

We declare the parameters of the specification, which are all constants, and we state the Agent and Value assumptions. We adopt the convention of using singular nouns as names, so  $x \in S$  can be read as “ $x$  is an  $S$ ”.

CONSTANTS <i>Proposer</i> ,	The set of proposers.
<i>Acceptor</i> ,	The set of acceptors.
<i>Learner</i> ,	The set of learners.
<i>PVal</i> ,	The set of proposable values.
<i>Message</i>	The set of all possible messages.

ASSUME *Agent Assumption*  
 $\wedge IsFiniteSet(Acceptor)$   
 $\wedge \exists p1, p2 \in Proposer : p1 \neq p2$   
 $\wedge \exists l1, l2 \in Learner : l1 \neq l2$

ASSUME *Value Assumption*  
 $\exists v1, v2 \in PVal : v1 \neq v2$

We define  $n$  to be the number of acceptors and  $Agent$  to be the set of all proposers, learners, and acceptors.

$$\begin{aligned}
 n &\triangleq Cardinality(Acceptor) \\
 Agent &\triangleq Proposer \cup Learner \cup Acceptor
 \end{aligned}$$


---



We now formalize the definitions of Section 2.1. We define an event to be a record, using record notation instead of subscripts—for example, writing  $e.num$  instead of  $e_{num}$ . Each record has *proposed*, *learned*, and *rcvd* fields, using the special value *NotAVal* to indicate that no value is proposed or learned, and letting  $e.rcvd$  equal  $\langle \rangle$  if  $e$  is not a message-receiving event. We define *Event* to be the set of all events.

$$NotAVal \triangleq \text{CHOOSE } v : v \notin PVal$$

$$Event \triangleq [agent : Agent, \\ num : Nat \setminus \{0\}, \\ msg : Message, \\ proposed : PVal \cup \{NotAVal\}, \\ learned : PVal \cup \{NotAVal\}, \\ rcvd : (Message \times Agent \times (Nat \setminus \{0\})) \cup \{\langle \rangle\}]$$

We write  $\preceq_S$  as  $Pre(S)$ . We define *Scenario* to be the set of all scenarios and  $Prefix(T)$  to be the set of prefixes of a scenario  $T$ .

$$Pre(S) \triangleq \text{TransitiveClosure}(\{\langle d, e \rangle \in S \times S : \\ \vee \wedge d.agent = e.agent \\ \wedge d.num \leq e.num \\ \vee e.rcvd = \langle d.msg, d.agent, d.num \rangle\})$$

$$Scenario \triangleq \\ \{S \in \text{SUBSET } Event : \\ \forall e \in S : \\ \wedge (e.proposed \neq NotAVal) \Rightarrow (e.agent \in Proposer) \\ \wedge (e.learned \neq NotAVal) \Rightarrow (e.agent \in Learner) \\ \wedge \forall d \in S : (d.agent = e.agent) \wedge (d.num = e.num) \Rightarrow (d = e) \\ \wedge (e.num > 1) \Rightarrow \exists d \in S : \wedge d.agent = e.agent \\ \wedge d.num = e.num - 1 \\ \wedge (e.rcvd \neq \langle \rangle) \Rightarrow \exists d \in S \setminus \{e\} : e.rcvd = \langle d.msg, d.agent, d.num \rangle \\ \wedge \forall d \in S : (\langle d, e \rangle \in Pre(S)) \wedge (\langle e, d \rangle \in Pre(S)) \Rightarrow (d = e) \}$$

$$Prefix(T) \triangleq \{S \in \text{SUBSET } T : \\ \forall d \in T, e \in S : (\langle d, e \rangle \in Pre(T)) \Rightarrow (d \in S)\}$$

$$S \sqsubseteq T \triangleq S \in Prefix(T)$$

We define *Algorithm* to be the set of all algorithms and *AsynchronousAlgorithm* to be the set of all asynchronous algorithms, and we assert the Scenario Union Lemma as a theorem.

$$Algorithm \triangleq \text{SUBSET } Scenario$$

$$Agents(S) \triangleq \{e.agent : e \in S\}$$

$$AsynchronousAlgorithm \triangleq$$

$$\begin{aligned}
& \{Alg \in Algorithm : \\
& \quad \wedge \forall T \in Alg : Prefix(T) \subseteq Alg \\
& \quad \wedge \forall T, U \in Alg : \\
& \quad \quad \forall S \in Prefix(T) \cap Prefix(U) : \\
& \quad \quad \quad (Agents(T \setminus S) \cap Agents(U \setminus S) = \{\}) \Rightarrow (T \cup U \in Alg)\}
\end{aligned}$$

THEOREM Scenario Union Lemma

$$\begin{aligned}
& \forall Alg \in AsynchronousAlgorithm : \\
& \quad \forall T, U \in Alg : (Agents(T) \cap Agents(U) = \{\}) \Rightarrow (T \cup U \in Alg)
\end{aligned}$$

We now define the sets of nontrivial, consistent, and consensus algorithms.

$$\begin{aligned}
NontrivialAlgorithm & \triangleq \\
& \{Alg \in Algorithm : \\
& \quad \forall S \in Alg : \\
& \quad \quad \forall e \in S : \\
& \quad \quad \quad (e.learned \neq NotAVal) \Rightarrow (\exists d \in S : e.learned = d.proposed)\}
\end{aligned}$$

$$\begin{aligned}
ConsistentAlgorithm & \triangleq \\
& \{Alg \in Algorithm : \\
& \quad \forall S \in Alg : \\
& \quad \quad \forall d, e \in S : (d.learned \neq NotAVal) \wedge (e.learned \neq NotAVal) \\
& \quad \quad \quad \Rightarrow (d.learned = e.learned)\}
\end{aligned}$$

$$ConsensusAlgorithm \triangleq NontrivialAlgorithm \cap ConsistentAlgorithm$$

We now formalize the definitions, assumptions, and results of Section 2.2. We define  $IsAcceptingFor(Q, p, Alg)$  to mean that the set  $Q$  of acceptors is an accepting set for proposer  $p$  in algorithm  $Alg$ .

$$\begin{aligned}
IsAcceptingFor(Q, p, Alg) & \triangleq \\
& \forall v \in PVal, l \in Learner : \\
& \quad \exists S \in Alg : \wedge Agents(S) \subseteq \{p, l\} \cup Q \\
& \quad \quad \wedge \exists d, e \in S : \wedge d.proposed = v \\
& \quad \quad \quad \wedge d.agent = p \\
& \quad \quad \quad \wedge e.learned = v \\
& \quad \quad \quad \wedge e.agent = l
\end{aligned}$$

THEOREM Accepting Lemma

$$\begin{aligned}
& \forall Alg \in ConsistentAlgorithm \cap AsynchronousAlgorithm, \\
& \quad p1, p2 \in Proposer, l1, l2 \in Learner, Q1, Q2 \in SUBSET Acceptor : \\
& \quad IsAcceptingFor(Q1, p1, Alg) \wedge IsAcceptingFor(Q2, p2, Alg) \\
& \quad \Rightarrow (\{p1, l1\} \cup Q1) \cap (\{p2, l2\} \cup Q2) \neq \{\}
\end{aligned}$$

**THEOREM** Acceptor Lower Bound Theorem

$\forall f \in 0 \dots n, Alg \in ConsistentAlgorithm \cap AsynchronousAlgorithm :$

$(\forall Q \in SUBSET\ Acceptor :$

$(Cardinality(Q) = n - f) \Rightarrow \forall p \in Proposer :$

$IsAcceptingFor(Q, p, Alg))$

$\Rightarrow \forall n > 2 * f$

$\forall \wedge f = 1$

$\wedge Cardinality(Agent) = 3$

$\wedge \exists a1, a2, a3 \in Agent : \wedge Acceptor = \{a1, a2\}$

$\wedge Proposer = \{a1, a3\}$

$\wedge Learner = \{a2, a3\}$

We now formalize the definitions and results of Section 2.3. We define  $Depth(e, S)$  to be the depth of event  $e$  in scenario  $S$ . Because TLA<sup>+</sup> allows recursive definitions only of functions, not operators, we locally define  $D[d]$  to equal the depth of event  $d$  in  $S$ . We also make the following local definitions:

- $PrecedesInAgent(d)$  is the set of events in  $S$  performed by  $d.agent$  that precede  $d$ .
- $SendsTo(d)$  is the event that sends the message received by  $d$ , if  $d$  receives a message.
- $Max(X)$  is the maximum of a finite set  $X$  of numbers, or 0 if  $X$  is empty.

We define  $Sources(S)$  to be the set of all sources of scenario  $S$

$Depth(e, S) \triangleq$

LET  $PrecedesInAgent(d) \triangleq \{c \in S : \wedge c.agent = d.agent$   
 $\wedge c.num < d.num \}$

$SendsTo(d) \triangleq \text{CHOOSE } c \in S : d.rcvd = \langle c.msg, c.agent, c.num \rangle$

$Max(X) \triangleq \text{IF } X = \{\} \text{ THEN } 0$

$\text{ELSE CHOOSE } i \in X : \forall j \in X : i \geq j$

$D[d \in S] \triangleq$

$Max(\{Max(\{D[c] : c \in PrecedesInAgent(d)\}),$

$\text{IF } d.rcvd \neq \langle \rangle \text{ THEN } 1 + D[SendsTo(d)] \text{ ELSE } 0\})$

IN  $D[e]$

$Sources(S) \triangleq \{e \in S : \forall d \in S \setminus \{e\} : \langle d, e \rangle \notin Pre(S)\}$

$IsFastAcceptingFor(M, p, Alg) \triangleq$

$\forall v \in PVal, l \in Learner :$

$\exists S \in Alg : \wedge Agents(S) \subseteq \{p, l\} \cup M$

$\wedge \forall e \in S : Depth(e, S) \leq 2$

$\wedge \exists e \in S : \wedge e.proposed = v$

$\wedge e.agent = p$

$\wedge \{e\} = Sources(S)$

$$\begin{aligned} \wedge \exists e \in S : \wedge e. \text{learned} = v \\ \wedge e. \text{agent} = l \end{aligned}$$

$IsQuorum(Q, Alg) \triangleq$

$\forall p \in Proposer :$

$\wedge IsAcceptingFor(Q, p, Alg)$

$\wedge \forall l \in Learner, S \in Alg :$

$\exists T \in Alg : \wedge S \sqsubseteq T$

$\wedge Agents(T \setminus S) \subseteq \{p, l\} \cup Q$

$\wedge \exists e \in T : \wedge e. \text{agent} = l$

$\wedge e. \text{learned} \neq NotAVal$

**THEOREM** Fast Accepting Lemma

$\forall Alg \in ConsistentAlgorithm \cap AsynchronousAlgorithm,$

$p1, p2, pq \in Proposer,$

$l1, l2, lq \in Learner,$

$M1, M2, Q \in SUBSET \text{ Acceptor} :$

$\wedge IsFastAcceptingFor(M1, p1, Alg)$

$\wedge IsFastAcceptingFor(M2, p2, Alg)$

$\wedge IsQuorum(Q, Alg)$

$\wedge p1 \neq p2$

$\wedge (p1 \notin M2) \wedge (p2 \notin M1)$

$\wedge l1 \notin \{p2, pq, lq\} \cup (M2 \setminus M1) \cup Q$

$\wedge l2 \notin \{p1, pq, lq\} \cup (M1 \setminus M2) \cup Q$

$\wedge \{pq, lq\} \cap M1 \cap M2 = \{\}$

$\Rightarrow (M1 \cap M2 \cap Q \neq \{\})$

**THEOREM** Fast Learning Theorem

$\forall f \in 1 .. n :$

$\forall e \in 0 .. f :$

$\forall Alg \in ConsensusAlgorithm \cap AsynchronousAlgorithm :$

$(\forall Q \in SUBSET \text{ Acceptor} :$

$(Cardinality(Q) = n - f) \Rightarrow IsQuorum(Q, Alg))$

$\Rightarrow \forall p1, p2 \in Proposer :$

$\wedge p1 \neq p2$

$\wedge \forall M \in SUBSET \text{ Acceptor} :$

$(Cardinality(M) = n - e) \Rightarrow$

$\wedge IsFastAcceptingFor(M, p1, Alg)$

$\wedge IsFastAcceptingFor(M, p2, Alg)$

$\Rightarrow \forall n > 2 * e + f$

$\forall Learner = \{p1, p2\}$



$$\begin{aligned}
& \text{HasIndependentDelivery}(\text{Alg}) \triangleq \\
& \forall S \in \text{Alg} \cap \text{NormalScenario} : \\
& \quad \forall e \in S, a \in \text{Agent} : \\
& \quad \quad \wedge \forall d \in S : \wedge d.\text{agent} = e.\text{agent} \\
& \quad \quad \quad \wedge d.\text{num} < e.\text{num} \\
& \quad \quad \quad \Rightarrow \exists c \in S : \wedge c.\text{agent} = a \\
& \quad \quad \quad \quad \wedge c.\text{rcvd} = \langle d.\text{msg}, d.\text{agent}, d.\text{num} \rangle \\
& \quad \wedge \neg \exists d \in S : \wedge d.\text{agent} = a \\
& \quad \quad \quad \wedge d.\text{rcvd} = \langle e.\text{msg}, e.\text{agent}, e.\text{num} \rangle \\
& \quad \wedge \forall d \in S : (\text{Depth}(d, S) < \text{Depth}(e, S)) \\
& \quad \quad \quad \Rightarrow \exists c \in S : \wedge c.\text{agent} = a \\
& \quad \quad \quad \quad \wedge c.\text{rcvd} = \langle d.\text{msg}, d.\text{agent}, d.\text{num} \rangle \\
& \Rightarrow \exists c \in \text{Event} : \wedge c.\text{agent} = a \\
& \quad \quad \quad \wedge c.\text{rcvd} = \langle e.\text{msg}, e.\text{agent}, e.\text{num} \rangle \\
& \quad \quad \quad \wedge S \cup \{c\} \in \text{Alg}
\end{aligned}$$

THEOREM **Collision-Fast Learning Theorem**

$$\begin{aligned}
& \forall f \in 1 \dots n : \\
& \quad \forall e \in 0 \dots f : \\
& \quad \quad \forall \text{Alg} \in \text{ConsensusAlgorithm} \cap \text{AsynchronousAlgorithm} : \\
& \quad \quad \quad \wedge \text{HasIndependentDelivery}(\text{Alg}) \\
& \quad \quad \quad \wedge \forall Q \in \text{SUBSET } \text{Acceptor} : \\
& \quad \quad \quad \quad (\text{Cardinality}(Q) = n - f) \Rightarrow \text{IsQuorum}(Q, \text{Alg}) \\
& \quad \quad \Rightarrow \forall p1, p2 \in \text{Proposer} : \\
& \quad \quad \quad \quad \wedge p1 \neq p2 \\
& \quad \quad \quad \quad \wedge \forall M \in \text{SUBSET } \text{Acceptor} : \\
& \quad \quad \quad \quad \quad (\text{Cardinality}(M) = n - e) \\
& \quad \quad \quad \quad \quad \Rightarrow \text{IsCollisionFastAcceptingFor}(M, \{p1, p2\}, \text{Alg}) \\
& \quad \quad \Rightarrow \wedge e = 0 \\
& \quad \quad \quad \wedge \vee \wedge f = 1 \\
& \quad \quad \quad \quad \wedge \text{Learner} \subseteq \text{Acceptor} \\
& \quad \quad \quad \quad \wedge \neg(\text{Acceptor} \subseteq \text{Learner}) \\
& \quad \quad \quad \quad \vee (p1 \in \text{Acceptor}) \vee (p2 \in \text{Acceptor})
\end{aligned}$$


---

We now formalize the definitions and results of Section 2.5.

$IsHyperfastAcceptingFor(p, Alg) \triangleq$

$$\begin{aligned} & \forall v \in PVal, l \in Learner : \\ & \quad \exists S \in Alg : \wedge \forall e \in S : Depth(e, S) \leq 1 \\ & \quad \quad \wedge \exists e \in S : \wedge Sources(S) = \{e\} \\ & \quad \quad \quad \wedge e.agent = p \\ & \quad \quad \quad \wedge e.proposed = v \\ & \quad \quad \wedge \exists e \in S : \wedge e.agent = l \\ & \quad \quad \quad \wedge e.learned = v \end{aligned}$$

**THEOREM** Hyperfast Learning Theorem

$$\begin{aligned} & \forall f \in 1 \dots n : \\ & \quad \forall Alg \in ConsistentAlgorithm \cap AsynchronousAlgorithm : \\ & \quad \quad \wedge \forall p, q \in Proposer : \\ & \quad \quad \quad \wedge IsHyperfastAcceptingFor(p, Alg) \\ & \quad \quad \quad \wedge IsHyperfastAcceptingFor(q, Alg) \\ & \quad \quad \quad \Rightarrow (p = q) \\ & \quad \quad \wedge (\forall Q \in SUBSET Acceptor : (Cardinality(Q) = n - f) \\ & \quad \quad \quad \Rightarrow IsQuorum(Q, Alg)) \\ & \quad \quad \Rightarrow \forall p \in Proposer : \\ & \quad \quad \quad IsHyperfastAcceptingFor(p, Alg) \\ & \quad \quad \quad \Rightarrow \wedge f = 1 \\ & \quad \quad \quad \quad \wedge p \in Acceptor \setminus Learner \\ & \quad \quad \quad \quad \wedge \forall l \in Learner : \vee l \in Acceptor \\ & \quad \quad \quad \quad \quad \vee Proposer = \{p, l\} \end{aligned}$$