# Verifying Fault-Tolerant Distributed Algorithms In The Heard-Of Model ${ }^{*}$ 

Henri Debrat ${ }^{1}$ and Stephan Merz ${ }^{2}$<br>${ }^{1}$ Université de Lorraine \& LORIA<br>${ }^{2}$ Inria Nancy Grand-Est \& LORIA<br>Villers-lès-Nancy, France

September 13, 2023

Distributed computing is inherently based on replication, promising increased tolerance to failures of individual computing nodes or communication channels. Realizing this promise, however, involves quite subtle algorithmic mechanisms, and requires precise statements about the kinds and numbers of faults that an algorithm tolerates (such as process crashes, communication faults or corrupted values). The landmark theorem due to Fischer, Lynch, and Paterson shows that it is impossible to achieve Consensus among $N$ asynchronously communicating nodes in the presence of even a single permanent failure. Existing solutions must rely on assumptions of "partial synchrony".
Indeed, there have been numerous misunderstandings on what exactly a given algorithm is supposed to realize in what kinds of environments. Moreover, the abundance of subtly different computational models complicates comparisons between different algorithms. Charron-Bost and Schiper introduced the Heard-Of model for representing algorithms and failure assumptions in a uniform framework, simplifying comparisons between algorithms. In this contribution, we represent the Heard-Of model in Isabelle/HOL. We define two semantics of runs of algorithms with different unit of atomicity and relate these through a reduction theorem that allows us to verify algorithms in the coarse-grained semantics (where proofs are easier) and infer their correctness for the fine-grained one (which corresponds to actual executions). We instantiate the framework by verifying six Consensus algorithms that differ in the underlying algorithmic mechanisms and the kinds of faults they tolerate.

[^0]
## Contents

1 Introduction ..... 4
2 Heard-Of Algorithms ..... 5
2.1 The Consensus Problem ..... 5
2.2 A Generic Representation of Heard-Of Algorithms ..... 7
3 Reduction Theorem ..... 13
3.1 Fine-Grained Semantics ..... 14
3.2 Properties of the Fine-Grained Semantics ..... 17
3.3 From Fine-Grained to Coarse-Grained Runs ..... 24
3.4 Locally Similar Runs and Local Properties ..... 26
3.5 Consensus as a Local Property ..... 28
4 Utility Lemmas About Majorities ..... 30
5 Verification of the One-Third Rule Consensus Algorithm ..... 31
5.1 Model of the Algorithm ..... 31
5.2 Communication Predicate for One-Third Rule ..... 33
5.3 The One-Third Rule Heard-Of Machine ..... 34
5.4 Proof of Integrity ..... 34
5.5 Proof of Agreement ..... 37
5.6 Proof of Termination ..... 41
5.7 One-Third Rule Solves Consensus ..... 43
6 Verification of the Uniform Voting Consensus Algorithm ..... 44
6.1 Model of the Algorithm ..... 44
6.2 Communication Predicate for Uniform Voting ..... 47
6.3 The Uniform Voting Heard-Of Machine ..... 47
6.4 Preliminary Lemmas ..... 48
6.5 Proof of Irrevocability, Agreement and Integrity ..... 50
6.6 Proof of Termination ..... 58
6.7 UniformVoting Solves Consensus ..... 60
7 Verification of the LastVoting Consensus Algorithm ..... 61
7.1 Model of the Algorithm ..... 61
7.2 Communication Predicate for LastVoting ..... 65
7.3 The LastVoting Heard-Of Machine ..... 65
7.4 Preliminary Lemmas ..... 66
7.5 Boundedness and Monotonicity of Timestamps ..... 71
7.6 Obvious Facts About the Algorithm ..... 72
7.7 Proof of Integrity ..... 80
7.8 Proof of Agreement and Irrevocability ..... 84
7.9 Proof of Termination ..... 94
7.10 LastVoting Solves Consensus ..... 96
8 Verification of the $\mathcal{U}_{T, E, \alpha}$ Consensus Algorithm ..... 97
8.1 Model of the Algorithm ..... 97
8.2 Communication Predicate for $\mathcal{U}_{T, E, \alpha}$ ..... 100
8.3 The $\mathcal{U}_{T, E, \alpha}$ Heard-Of Machine ..... 101
8.4 Preliminary Lemmas ..... 101
8.5 Proof of Agreement and Validity ..... 105
8.6 Proof of Termination ..... 116
8.7 $\mathcal{U}_{T, E, \alpha}$ Solves Weak Consensus ..... 122
9 Verification of the $\mathcal{A}_{T, E, \alpha}$ Consensus algorithm ..... 123
9.1 Model of the Algorithm ..... 123
9.2 Communication Predicate for $\mathcal{A}_{T, E, \alpha}$ ..... 125
9.3 The $\mathcal{A}_{T, E, \alpha}$ Heard-Of Machine ..... 125
9.4 Preliminary Lemmas ..... 126
9.5 Proof of Validity ..... 130
9.6 Proof of Agreement ..... 134
9.7 Proof of Termination ..... 137
$9.8 \mathcal{A}_{T, E, \alpha}$ Solves Weak Consensus ..... 142
10 Verification of the $E I G B y z_{f}$ Consensus Algorithm ..... 143
10.1 Tree Data Structure ..... 143
10.2 Model of the Algorithm ..... 145
10.3 Communication Predicate for $E I G B y z_{f}$ ..... 147
10.4 The EIGByzf Heard-Of Machine ..... 148
10.5 Preliminary Lemmas ..... 148
10.6 Lynch's Lemmas and Theorems ..... 153
10.7 Proof of Agreement, Validity, and Termination ..... 162
10.8 EIGByzf Solves Weak Consensus ..... 164
11 Conclusion ..... 164

## 1 Introduction

We are interested in the verification of fault-tolerant distributed algorithms. The archetypical problem in this area is the Consensus problem that requires a set of distributed nodes to achieve agreement on a common value in the presence of faults. Such algorithms are notoriously hard to design and to get right. This is particularly true in the presence of asynchronous communication: the landmark theorem by Fischer, Lynch, and Paterson [9] shows that there is no algorithm solving the Consensus problem for asynchronous systems in the presence of even a single, permanent fault. Existing solutions therefore rely on assumptions of "partial synchrony" [8].
Different computational models, and different concepts for specifying the kinds and numbers of faults such algorithms must tolerate, have been introduced in the literature on distributed computing. This abundance of subtly different notions makes it very difficult to compare different algorithms, and has sometimes even led to misunderstandings and misinterpretations of what an algorithm claims to achieve. The general lack of rigorous, let alone formal, correctness proofs for this class of algorithms makes it even harder to understand the field.
In this contribution, we formalize in Isabelle/HOL the Heard-Of (HO) model, originally introduced by Charron-Bost and Schiper [7]. This model can represent algorithms that operate in communication-closed rounds, which is true of virtually all known fault-tolerant distributed algorithms. Assumptions on failures tolerated by an algorithm are expressed by communication predicates that impose bounds on the set of messages that are not received during executions. Charron-Bost and Schiper show how the known failure hypotheses from the literature can be represented in this format. The Heard-Of model therefore makes an interesting target for formalizing different algorithms, and for proving their correctness, in a uniform way. In particular, different assumptions can be compared, and the suitability of an algorithm for a particular situation can be evaluated.
The HO model has subsequently been extended [3] to encompass algorithms designed to tolerate value (also known as malicious or Byzantine) faults. In the present work, we propose a generic framework in Isabelle/HOL that encompasses the different variants of HO algorithms, including resilience to benign or value faults, as well as coordinated and non-coordinated algorithms.
A fundamental design decision when modeling distributed algorithm is to determine the unit of atomicity. We formally relate in Isabelle two definitions of runs: we first define "coarse-grained" executions, in which entire rounds are executed atomically, and then define "fine-grained" executions that correspond to conventional interleaving representations of asynchronous networks. We formally prove that every fine-grained execution corresponds
to a certain coarse-grained execution, such that every process observes the same sequence of local states in the two executions, up to stuttering. As a corollary, a large class of correctness properties, including Consensus, can be transferred from coarse-grained to fine-grained executions.
We then apply our framework for verifying six different distributed Consensus algorithms w.r.t. their respective communication predicates. The first three algorithms, One-Third Rule, Uniform Voting, and LastVoting, tolerate benign failures. The three remaining algorithms, $\mathcal{U}_{T, E, \alpha}, \mathcal{A}_{T, E, \alpha}$, and EIG$B y z_{f}$, are designed to tolerate value failures, and solve a weaker variant of the Consensus problem.
A preliminary report on the formalization of the LastVoting algorithm in the HO model appeared in [6]. The paper [4] contains a paper-and-pencil proof of the reduction theorem relating coarse-grained and fine-grained executions, and [5] reports on the formal verification of the $\mathcal{U}_{T, E, \alpha}, \mathcal{A}_{T, E, \alpha}$, and EIGByz $z_{f}$ algorithms.

```
theory HOModel
imports Main
begin
```

declare if-split-asm [split] — perform default perform case splitting on conditionals

## 2 Heard-Of Algorithms

### 2.1 The Consensus Problem

We are interested in the verification of fault-tolerant distributed algorithms. The Consensus problem is paradigmatic in this area. Stated informally, it assumes that all processes participating in the algorithm initially propose some value, and that they may at some point decide some value. It is required that every process eventually decides, and that all processes must decide the same value.
More formally, we represent runs of algorithms as $\omega$-sequences of configurations (vectors of process states). Hence, a run is modeled as a function of type nat $\Rightarrow$ 'proc $\Rightarrow$ 'pst where type variables 'proc and 'pst represent types of processes and process states, respectively. The Consensus property is expressed with respect to a collection vals of initially proposed values (one per process) and an observer function $d e c::$ 'pst $\Rightarrow$ val option that retrieves the decision (if any) from a process state. The Consensus problem is stated as the conjunction of the following properties:

Integrity. Processes can only decide initially proposed values.
Agreement. Whenever processes $p$ and $q$ decide, their decision values must be the same. (In particular, process $p$ may never change the value it
decides, which is referred to as Irrevocability.)
Termination. Every process decides eventually.
The above properties are sometimes only required of non-faulty processes, since nothing can be required of a faulty process. The Heard-Of model does not attribute faults to processes, and therefore the above formulation is appropriate in this framework.

## type-synonym

('proc,'pst) run $=$ nat $\Rightarrow$ 'proc $\Rightarrow$ 'pst

## definition

```
consensus :: ('proc \(\Rightarrow\) 'val \() \Rightarrow\left({ }^{\prime}\right.\) pst \(\Rightarrow\) 'val option \() \Rightarrow\left({ }^{\prime}\right.\) proc,'pst) run \(\Rightarrow\) bool
```

where
consensus vals dec rho $\equiv$
$(\forall n p v . \operatorname{dec}($ rho $n p)=$ Some $v \longrightarrow v \in$ range vals)
$\wedge(\forall m n p q v w . \operatorname{dec}($ rho $m p)=$ Some $v \wedge \operatorname{dec}($ rho $n q)=$ Some $w$
$\longrightarrow v=w)$
$\wedge(\forall p . \exists n . \operatorname{dec}($ rho $n p) \neq$ None $)$

A variant of the Consensus problem replaces the Integrity requirement by
Validity. If all processes initially propose the same value $v$ then every process may only decide $v$.
definition weak-consensus where
weak-consensus vals dec rho $\equiv$
$(\forall v .(\forall p$. vals $p=v) \longrightarrow(\forall n p w . \operatorname{dec}($ rho $n p)=$ Some $w \longrightarrow w=v))$
$\wedge(\forall m n p q v w . \operatorname{dec}($ rho $m p)=$ Some $v \wedge \operatorname{dec}($ rho $n q)=$ Some $w$ $\longrightarrow v=w)$
$\wedge(\forall p . \exists n . \operatorname{dec}($ rho $n p) \neq$ None $)$
Clearly, consensus implies weak-consensus.
lemma consensus-then-weak-consensus:
assumes consensus vals dec rho
shows weak-consensus vals dec rho
using assms by (auto simp: consensus-def weak-consensus-def image-def)
Over Boolean values ("binary Consensus"), weak-consensus implies consensus, hence the two problems are equivalent. In fact, this theorem holds more generally whenever at most two different values are proposed initially (i.e., card (range vals) $\leq 2$ ).
lemma binary-weak-consensus-then-consensus:
assumes bc: weak-consensus (vals::'proc $\Rightarrow$ bool) dec rho
shows consensus vals dec rho
proof -
\{ - Show the Integrity property, the other conjuncts are the same.
fix $n p v$

```
    assume dec: dec (rho \(n\) p) \(=\) Some \(v\)
    have \(v \in\) range vals
    proof (cases \(\exists w . \forall p\). vals \(p=w\) )
        case True
        then obtain \(w\) where \(w: \forall p\). vals \(p=w\)..
    with bc have dec (rho n \(p\) ) \(\in\{\) Some \(w\), None \(\}\) by (auto simp: weak-consensus-def)
        with dec \(w\) show ? thesis by (auto simp: image-def)
    next
        case False
        - In this case both possible values occur in vals, and the result is trivial.
        thus ?thesis by (auto simp: image-def)
    qed
    \(\}\) note integrity \(=\) this
    from \(b c\) show ? thesis
    unfolding consensus-def weak-consensus-def by (auto elim!: integrity)
qed
```

The algorithms that we are going to verify solve the Consensus or weak Consensus problem, under different hypotheses about the kinds and number of faults.

### 2.2 A Generic Representation of Heard-Of Algorithms

Charron-Bost and Schiper [7] introduce the Heard-Of (HO) model for representing fault-tolerant distributed algorithms. In this model, algorithms execute in communication-closed rounds: at any round $r$, processes only receive messages that were sent for that round. For every process $p$ and round $r$, the "heard-of set" $H O(p, r)$ denotes the set of processes from which $p$ receives a message in round $r$. Since every process is assumed to send a message to all processes in each round, the complement of $H O(p, r)$ represents the set of faults that may affect $p$ in round $r$ (messages that were not received, e.g. because the sender crashed, because of a network problem etc.).
The HO model expresses hypotheses on the faults tolerated by an algorithm through "communication predicates" that constrain the sets $H O(p, r)$ that may occur during an execution. Charron-Bost and Schiper show that standard fault models can be represented in this form.
The original HO model is sufficient for representing algorithms tolerating benign failures such as process crashes or message loss. A later extension for algorithms tolerating Byzantine (or value) failures [3] adds a second collection of sets $S H O(p, r) \subseteq H O(p, r)$ that contain those processes $q$ from which process $p$ receives the message that $q$ was indeed supposed to send for round $r$ according to the algorithm. In other words, messages from processes in $H O(p, r) \backslash S H O(p, r)$ were corrupted, be it due to errors during message transmission or because of the sender was faulty or lied deliberately. For both benign and Byzantine errors, the HO model registers the fault but
does not try to identify the faulty component (i.e., designate the sending or receiving process, or the communication channel as the "culprit").
Executions of HO algorithms are defined with respect to collections $H O(p, r)$ and $S H O(p, r)$. However, the code of a process does not have access to these sets. In particular, process $p$ has no way of determining if a message it received from another process $q$ corresponds to what $q$ should have sent or if it has been corrupted.
Certain algorithms rely on the assignment of "coordinator" processes for each round. Just as the collections $H O(p, r)$, the definitions assume an external coordinator assignment such that $\operatorname{coord}(p, r)$ denotes the coordinator of process $p$ and round $r$. Again, the correctness of algorithms may depend on hypotheses about coordinator assignments - e.g., it may be assumed that processes agree sufficiently often on who the current coordinator is.
The following definitions provide a generic representation of HO and SHO algorithms in Isabelle/HOL. A (coordinated) HO algorithm is described by the following parameters:

- a finite type 'proc of processes,
- a type 'pst of local process states,
- a type 'msg of messages sent in the course of the algorithm,
- a predicate CinitState such that CinitState p st crd is true precisely of the initial states st of process $p$, assuming that $c r d$ is the initial coordinator of $p$,
- a function sendMsg where sendMsg r p q st yields the message that process $p$ sends to process $q$ at round $r$, given its local state $s t$, and
- a predicate CnextState where CnextState r pt msgs crd st' characterizes the successor states $s t^{\prime}$ of process $p$ at round $r$, given current state st, the vector msgs :: 'proc $\Rightarrow{ }^{\prime} m s g$ option of messages that $p$ received at round $r$ ( msgs $q=$ None indicates that no message has been received from process $q$ ), and process $c r d$ as the coordinator for the following round.

Note that every process can store the coordinator for the current round in its local state, and it is therefore not necessary to make the coordinator a parameter of the message sending function sendMsg.
We represent an algorithm by a record as follows.

```
record ('proc, 'pst, 'msg) CHOAlgorithm \(=\)
    CinitState :: 'proc \(\Rightarrow\) 'pst \(\Rightarrow\) 'proc \(\Rightarrow\) bool
    sendMsg :: nat \(\Rightarrow\) 'proc \(\Rightarrow\) 'proc \(\Rightarrow\) 'pst \(\Rightarrow\) 'msg
    CnextState : : nat \(\Rightarrow\) 'proc \(\Rightarrow\) 'pst \(\Rightarrow\) ('proc \(\Rightarrow\) 'msg option \() \Rightarrow\) 'proc \(\Rightarrow\) 'pst \(\Rightarrow\)
bool
```

For non-coordinated HO algorithms, the coordinator argument of functions CinitState and CnextState is irrelevant, and we define utility functions that omit that argument.

```
definition isNCAlgorithm where
    isNCAlgorithm alg \(\equiv\)
        \(\left(\forall p\right.\) st crd crd \({ }^{\prime}\). CinitState alg p st crd \(=\) CinitState alg p st crd \(\left.{ }^{\prime}\right)\)
    \(\wedge\left(\forall r p\right.\) st msgs crd crd \({ }^{\prime}\) st'. CnextState alg \(r p\) st msgs crd st'
                                \(=\) CnextState alg r p st msgs crd' \({ }^{\prime} t^{\prime}\) )
```

definition initState where
initState alg $p$ st $\equiv$ CinitState alg p st undefined
definition nextState where
nextState alg r $p$ st msgs st' $\equiv$ CnextState alg r p st msgs undefined $s t^{\prime}$

A heard-of assignment associates a set of processes with each process. The following type is used to represent the collections $H O(p, r)$ and $S H O(p, r)$ for fixed round $r$. Similarly, a coordinator assignment associates a process (its coordinator) to each process.

```
type-synonym
    'proc HO = 'proc }=>\mathrm{ ' 'proc set
type-synonym
    'proc coord = 'proc # 'proc
```

An execution of an HO algorithm is defined with respect to HO and SHO assignments that indicate, for every round $r$ and every process $p$, from which sender processes $p$ receives messages (resp., uncorrupted messages) at round $r$.

The following definitions formalize this idea. We define "coarse-grained" executions whose unit of atomicity is the round of execution. At each round, the entire collection of processes performs a transition according to the CnextState function of the algorithm. Consequently, a system state is simply described by a configuration, i.e. a function assigning a process state to every process. This definition of executions may appear surprising for an asynchronous distributed system, but it simplifies system verification, compared to a "fine-grained" execution model that records individual events such as message sending and reception or local transitions. We will justify later why the "coarse-grained" model is sufficient for verifying interesting correctness properties of HO algorithms.
The predicate CSHOinitConfig describes the possible initial configurations for algorithm $A$ (remember that a configuration is a function that assigns local states to every process).
definition CHOinitConfig where
CHOinitConfig A cfg (coord::'proc coord) $\equiv \forall$ p. CinitState A p $(c f g p)($ coord $p)$

Given the current configuration $c f g$ and the HO and SHO sets $H O p$ and $S H O p$ for process $p$ at round $r$, the function $S H O m s g$ Vectors computes the set of possible vectors of messages that process $p$ may receive. For processes $q \notin H O p, p$ receives no message (represented as value None). For processes $q$ $\in S H O p, p$ receives the message that $q$ computed according to the sendMsg function of the algorithm. For the remaining processes $q \in H O p-S H O p$, $p$ may receive some arbitrary value.
definition SHOmsg Vectors where
SHOmsgVectors A r p cfg HOp SHOp 三

$$
\begin{aligned}
& \{\mu .(\forall q \cdot q \in H O p \longleftrightarrow \mu q \neq \text { None }) \\
& \wedge(\forall q \cdot q \in S H O p \cap H O p \longrightarrow \mu q=\text { Some }(\text { sendMsg Arqp(cfg } q)))\}
\end{aligned}
$$

Predicate CSHOnextConfig uses the preceding function and the algorithm's CnextState function to characterize the possible successor configurations in a coarse-grained step, and predicate CSHORun defines (coarse-grained) executions rho of an HO algorithm.

```
definition CSHOnextConfig where
    CSHOnextConfig A r cfg HO SHO coord cfg' \equiv
    \forall. \exists\mu\inSHOmsgVectors A r p cfg (HO p) (SHO p).
            CnextState A r p (cfg p) }\mu(\mathrm{ coord p) (cfg' p)
```

```
definition CSHORun where
    CSHORun A rho HOs SHOs coords \(\equiv\)
        CHOinitConfig A (rho 0) (coords 0)
        \(\wedge(\forall r\). CSHOnextConfig A r (rho r) (HOs r) (SHOs r) (coords (Suc r))
                            (rho (Suc r)))
```

For non-coordinated algorithms. the coord arguments of the above functions are irrelevant. We define similar functions that omit that argument, and relate them to the above utility functions for these algorithms.

```
definition HOinitConfig where
    HOinitConfig A cfg \(\equiv\) CHOinitConfig A cfg ( \(\lambda q\). undefined)
lemma HOinitConfig-eq:
    HOinitConfig A cfg \(=(\forall\) p. initState A \(p(c f g p))\)
    by (auto simp: HOinitConfig-def CHOinitConfig-def initState-def)
definition SHOnextConfig where
    SHOnextConfig A rcfg HO SHO cfg' 三
        CSHOnextConfig A r cfg HO SHO ( \(\lambda\) q. undefined) \(c f g^{\prime}\)
lemma SHOnextConfig-eq:
    SHOnextConfig A r cfg HO SHO cfg' =
    \((\forall p . \exists \mu \in\) SHOmsgVectors A rpcfg (HO p) (SHO p).
                nextState \(A\) r \(\left.p(c f g p) \mu\left(c f g^{\prime} p\right)\right)\)
    by (auto simp: SHOnextConfig-def CSHOnextConfig-def SHOmsgVectors-def nextState-def)
```

```
definition SHORun where
    SHORun A rho HOs SHOs \equiv
    CSHORun A rho HOs SHOs (\lambdar q. undefined)
lemma SHORun-eq:
    SHORun A rho HOs SHOs=
        (HOinitConfig A (rho 0)
    ^(\forallr. SHOnextConfig A r (rho r) (HOs r) (SHOs r) (rho (Suc r))))
by (auto simp: SHORun-def CSHORun-def HOinitConfig-def SHOnextConfig-def)
```

Algorithms designed to tolerate benign failures are not subject to message corruption, and therefore the SHO sets are irrelevant (more formally, each SHO set equals the corresponding HO set). We define corresponding special cases of the definitions of successor configurations and of runs, and prove that these are equivalent to simpler definitions that will be more useful in proofs. In particular, the vector of messages received by a process in a benign execution is uniquely determined from the current configuration and the HO sets.

```
definition HOrcvdMsgs where
    HOrcvdMsgs A r p HO cfg 三
    \(\lambda q\). if \(q \in H O\) then Some (sendMsg A r q \(p(c f g q)\) ) else None
lemma SHOmsgVectors-HO:
    SHOmsgVectors A r p cfg HO HO \(=\{\) HOrcvdMsgs A r p HO cfg \(\}\)
    unfolding SHOmsgVectors-def HOrcvdMsgs-def by auto
```

With coordinators

```
definition CHOnextConfig where
    CHOnextConfig A r cfg HO coord cfg' \equiv
    CSHOnextConfig A r cfg HO HO coord cfg'
lemma CHOnextConfig-eq:
    CHOnextConfig A r cfg HO coord cfg' =
    ( }\forall\mathrm{ p. CnextState A r p (cfg p) (HOrcvdMsgs A r p (HO p)cfg)
            (coord p) (cfg'p))
    by (auto simp: CHOnextConfig-def CSHOnextConfig-def SHOmsgVectors-HO)
definition CHORun where
    CHORun A rho HOs coords \equivCSHORun A rho HOs HOs coords
lemma CHORun-eq:
    CHORun A rho HOs coords =
        (CHOinitConfig A (rho 0) (coords 0)
        ^(\forallr. CHOnextConfig A r (rho r) (HOs r) (coords (Suc r)) (rho (Suc r))))
    by (auto simp: CHORun-def CSHORun-def CHOinitConfig-def CHOnextCon-
fig-def)
```

Without coordinators

```
definition HOnextConfig where
    HOnextConfig A rcfg HO cfg' \equivSHOnextConfig A r cfg HO HO cfg'
lemma HOnextConfig-eq:
    HOnextConfig A r cfg HO cfg'=
    (\forallp.nextState A r p (cfg p) (HOrcvdMsgs A r p (HO p) cfg) (cfg' p))
    by (auto simp: HOnextConfig-def SHOnextConfig-eq SHOmsgVectors-HO)
definition HORun where
    HORun A rho HOs \equivSHORun A rho HOs HOs
lemma HORun-eq:
    HORun A rho HOs=
    ( HOinitConfig A (rho 0)
    ^(\forall r. HOnextConfig A r (rho r)(HOs r) (rho (Suc r))))
    by (auto simp: HORun-def SHORun-eq HOnextConfig-def)
```

The following derived proof rules are immediate consequences of the definition of CHORun; they simplify automatic reasoning.
lemma CHORun-0:
assumes CHORun A rho HOs coords
and $\bigwedge c f g$. CHOinitConfig A cfg (coords 0$) \Longrightarrow P c f g$ shows $P$ (rho 0)
using assms unfolding CHORun-eq by blast
lemma CHORun-Suc:
assumes CHORun A rho HOs coords
and $\wedge r$. CHOnextConfig Ar (rho r) (HOs r) (coords (Suc r)) (rho (Suc r)) $\Longrightarrow P r$
shows $P n$
using assms unfolding CHORun-eq by blast
lemma CHORun-induct:
assumes run: CHORun A rho HOs coords
and init: CHOinitConfig $A$ (rho 0 ) (coords 0$) \Longrightarrow P 0$
and step: $\wedge r$. 【P r; CHOnextConfig A r (rho r) (HOs r) (coords (Suc r)) (rho (Suc r)) 】 $\Longrightarrow P($ Suc $r)$
shows $P$ n
using run unfolding CHORun-eq by (induct n, auto elim: init step)
Because algorithms will not operate for arbitrary HO , SHO , and coordinator assignments, these are constrained by a communication predicate. For convenience, we split this predicate into a per Round part that is expected to hold at every round and a global part that must hold of the sequence of (S)HO assignments and may thus express liveness assumptions.

In the parlance of [7], a HO machine is an HO algorithm augmented with a communication predicate. We therefore define (C)(S)HO machines as the corresponding extensions of the record defining an HO algorithm.

```
record ('proc, 'pst, 'msg) HOMachine \(=(\) 'proc, 'pst, 'msg) CHOAlgorithm +
    HOcommPerRd::'proc \(\mathrm{HO} \Rightarrow\) bool
    HOcommGlobal::(nat \(\Rightarrow\) 'proc HO\() \Rightarrow\) bool
record ('proc, 'pst, 'msg) CHOMachine \(=\left({ }^{\prime}\right.\) proc, 'pst, 'msg) CHOAlgorithm +
    CHOcommPerRd::nat \(\Rightarrow\) 'proc \(\mathrm{HO} \Rightarrow\) 'proc coord \(\Rightarrow\) bool
    CHOcommGlobal:: (nat \(\Rightarrow\) 'proc HO\() \Rightarrow(\) nat \(\Rightarrow\) 'proc coord \() \Rightarrow\) bool
record ('proc, 'pst, 'msg) SHOMachine \(=(\) 'proc, 'pst, 'msg) CHOAlgorithm +
    SHOcommPerRd::('proc HO) \(\Rightarrow\) ('proc HO\() \Rightarrow\) bool
    SHOcommGlobal::(nat \(\Rightarrow\) 'proc HO\() \Rightarrow(\) nat \(\Rightarrow\) 'proc HO\() \Rightarrow\) bool
record ('proc, 'pst, 'msg) CSHOMachine \(=(\) 'proc, 'pst, 'msg) CHOAlgorithm +
    CSHOcommPerRd::('proc HO) \(\Rightarrow\) ('proc HO\() \Rightarrow\) 'proc coord \(\Rightarrow\) bool
    CSHOcommGlobal:: (nat \(\Rightarrow\) 'proc HO\() \Rightarrow(\) nat \(\Rightarrow\) 'proc HO\()\)
                        \(\Rightarrow\) (nat \(\Rightarrow\) 'proc coord \() \Rightarrow\) bool
end - theory HOModel
theory Reduction
imports HOModel Stuttering-Equivalence.StutterEquivalence
begin
```


## 3 Reduction Theorem

We have defined the semantics of HO algorithms such that rounds are executed atomically, by all processes. This definition is surprising for a model of asynchronous distributed algorithms since it models a synchronous execution of rounds. However, it simplifies representing and reasoning about the algorithms. For example, the communication network does not have to be modeled explicitly, since the possible sets of messages received by processes can be computed from the global configuration and the collections of HO and SHO sets.
We will now define a more conventional "fine-grained" semantics where communication is modeled explicitly and rounds of processes can be arbitrarily interleaved (subject to the constraints of the communication predicates). We will then establish a reduction theorem that shows that for every finegrained run there exists an equivalent round-based ("coarse-grained") run in the sense that the two runs exhibit the same sequences of local states of all processes, modulo stuttering. We prove the reduction theorem for the most general class of coordinated SHO algorithms. It is easy to see that the theorem equally holds for the special cases of uncoordinated or HO algorithms, and since we have in fact defined these classes of algorithms from the more general ones, we can directly apply the general theorem.
As a corollary, interesting properties remain valid in the fine-grained semantics if they hold in the coarse-grained semantics. It is therefore enough to
verify such properties in the coarse-grained semantics, which is much easier to reason about. The essential restriction is that properties may not depend on states of different processes occurring simultaneously. (For example, the coarse-grained semantics ensures by definition that all processes execute the same round at any instant, which is obviously not true of the fine-grained semantics.) We claim that all "reasonable" properties of faulttolerant distributed algorithms are preserved by our reduction. For example, the Consensus (and Weak Consensus) problems fall into this class.
The proofs follow Chaouch-Saad et al. [4], where the reduction theorem was proved for uncoordinated HO algorithms.

### 3.1 Fine-Grained Semantics

In the fine-grained semantics, a run of an HO algorithm is represented as an $\omega$-sequence of system configurations. Each configuration is represented as a record carrying the following information:

- for every process $p$, the current round that process $p$ is executing,
- the local state of every process,
- for every process $p$, the set of processes to which $p$ has already sent a message for the current round,
- for all processes $p$ and $q$, the message (if any) that $p$ has received from $q$ for the round that $p$ is currently executing, and
- the set of messages in transit, represented as triples of the form $(p, r, q, m)$ meaning that process $p$ sent message $m$ to process $q$ for round $r$, but $q$ has not yet received that message.

As explained earlier, the coordinators of processes are not recorded in the configuration, but algorithms may record them as part of the process states.

```
record ('pst, 'proc, 'msg) config =
    round :: 'proc }=>\mathrm{ nat
    state :: 'proc = 'pst
    sent :: 'proc = 'proc set
    rcvd :: 'proc }=>\mathrm{ 'proc }=>\mathrm{ 'msg option
    network :: ('proc * nat * 'proc * 'msg) set
```

type-synonym ('pst ,'proc, 'msg) fgrun $=$ nat $\Rightarrow\left({ }^{\prime} p s t\right.$, 'proc, 'msg) config

In an initial configuration for an algorithm, the local state of every process satisfies the algorithm's initial-state predicate, and all other components have obvious default values.
definition fg-init-config where

```
fg-init-config A (config::('pst,'proc, 'msg) config) (coord::'proc coord) \(\equiv\)
    round config \(=(\lambda p .0)\)
    \(\wedge(\forall p\). CinitState A \(p(\) state config \(p)(\operatorname{coord} p))\)
    \(\wedge\) sent config \(=(\lambda p .\{ \})\)
    \(\wedge\) rcvd config \(=(\lambda p q\). None \()\)
    \(\wedge\) network config \(=\{ \}\)
```

In the fine-grained semantics, we have three types of transitions due to

- some process sending a message,
- some process receiving a message, and
- some process executing a local transition.

The following definition models process $p$ sending a message to process $q$. The transition is enabled if $p$ has not yet sent any message to $q$ for the current round. The message to be sent is computed according to the algorithm's sendMsg function. The effect of the transition is to add $q$ to the sent component of the configuration and the message quadruple to the network component.

```
definition \(f g\)-send-msg where
    fg-send-msg A p q config config' \(\equiv\)
        \(q \notin(\) sent config \(p)\)
\(\wedge\) config' \(=\) config (
        sent \(:=(\) sent config \()(p:=(\) sent config \(p) \cup\{q\})\),
        network \(:=\) network config \(\cup\)
            \(\{(p\), round config \(p, q\),
                        sendMsg A (round config p) \(p q(\) state config \(p))\}\) D
```

The following definition models the reception of a message by process $p$ from process $q$. The action is enabled if $q$ is in the heard-of set $H O$ of process $p$ for the current round, and if the network contains some message from $q$ to $p$ for the round that $p$ is currently executing. W.l.o.g., we model message corruption at reception: if $q$ is not in $p$ 's SHO set (parameter SHO), then an arbitrary value $m^{\prime}$ is received instead of $m$.

```
definition \(f g\)-rcv-msg where
    fg-rcv-msg p q HO SHO config config' \(\equiv\)
        \(\exists m m^{\prime} .(q,(\) round config \(p), p, m) \in\) network config
        \(\wedge q \in H O\)
        \(\wedge\) config \(=\) config 0
            rcvd \(:=(\) rcvd config \()(p:=(\) rcvd config \(p)(q:=\)
                            if \(q \in\) SHO then Some \(m\) else Some \(\left.m^{\prime}\right)\) ),
        network \(:=\) network config \(-\{(q,(\) round config \(p), p, m)\} D\)
```

Finally, we consider local state transition of process $p$. A local transition is enabled only after $p$ has sent all messages for its current round and has received all messages that it is supposed to receive according to its current

HO set (parameter $H O$ ). The local state is updated according to the algorithm's CnextState relation, which may depend on the coordinator crd of the following round. The round of process $p$ is incremented, and the sent and rcvd components for process $p$ are reset to initial values for the new round.

```
definition fg-local where
    fg-local A p HO crd config config' \(\equiv\)
        sent config \(p=U N I V\)
    \(\wedge \operatorname{dom}(\) rcvd config \(p)=H O\)
    \(\wedge(\exists s\). CnextState \(A(\) round config \(p) p(\) state config \(p)(\) rcvd config \(p)\) crd \(s\)
        \(\wedge\) config' \(=\) config \(\mid\)
                round \(:=(\) round config \()(p:=\) Suc (round config \(p)\) ),
                state \(:=(\) state config \()(p:=s)\),
                sent \(:=(\) sent config \()(p:=\{ \})\),
                rcvd \(:=(\) rcvd config \()(p:=\lambda q\). None \() D)\)
```

The next-state relation for process $p$ is just the disjunction of the above three types of transitions.
definition $f g$-next-config where
fg-next-config A p HO SHO crd config config' $\equiv$
( $\exists q$. fg-send-msg A p q config config')
$\vee(\exists q . f g$-rcv-msg p q HO SHO config config')
$\vee$ fg-local A p HO crd config config'
Fine-grained runs are infinite sequences of configurations that start in an initial configuration and where each step corresponds to some process sending a message, receiving a message or performing a local step. We also require that every process eventually executes every round - note that this condition is implicit in the definition of coarse-grained runs.

## definition $f g$-run where

```
fg-run A rho HOs SHOs coords \(\equiv\)
    fg-init-config \(A\) (rho 0 ) (coords 0 )
\(\wedge(\forall i . \exists p . f g\)-next-config \(A p\)
    (HOs (round (rho i) p) p)
    (SHOs (round (rho i) p) p)
    (coords (round (rho (Suc i)) p) p)
    (rho i) (rho (Suc i)))
    \(\wedge(\forall p r . \exists n\). round (rho \(n) p=r)\)
```

The following function computes at which "time point" (index in the finegrained computation) process $p$ starts executing round $r$. This function plays an important role in the correspondence between the two semantics, and in the subsequent proofs.

[^1]
### 3.2 Properties of the Fine-Grained Semantics

In preparation for the proof of the reduction theorem, we establish a number of consequences of the above definitions.

Process states change only when round numbers change during a fine-grained run.
lemma fg-state-change:
assumes rho: fg-run A rho HOs SHOs coords and $r d$ : round (rho (Suc n)) $p=$ round (rho $n$ ) $p$
shows state (rho (Suc n)) $p=$ state (rho $n$ ) $p$
proof -
from rho have $\exists p^{\prime}$. fg-next-config $A p^{\prime}\left(H O s\left(\right.\right.$ round (rho n) $\left.\left.p^{\prime}\right) p^{\prime}\right)$
(SHOs (round (rho n) $p^{\prime}$ ) $p^{\prime}$ )
(coords (round (rho (Suc n)) $p^{\prime}$ ) $p^{\prime}$ )
(rho n) (rho (Suc n))
by (auto simp: fg-run-def)
with $r d$ show ?thesis
by (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)
qed
Round numbers never decrease.
lemma fg-round-numbers-increase:
assumes rho: fg-run A rho HOs SHOs coords and $n: n \leq m$
shows round (rho $n$ ) $p \leq$ round (rho m) $p$
proof -
from $n$ obtain $k$ where $k: m=n+k$ by (auto simp:le-iff-add)
\{
fix $i$
have round (rho n) $p \leq$ round (rho ( $n+i$ ) $p$ (is ?P $i$ )
proof (induct $i$ )
show ?P 0 by simp
next
fix $j$
assume $i h$ : ?P $j$
from rho have $\exists p^{\prime}$. fg-next-config $A p^{\prime}\left(H O s\left(r o u n d ~(r h o(n+j)) p^{\prime}\right) p^{\prime}\right)$
(SHOs (round (rho $\left.(n+j)) p^{\prime}\right) p^{\prime}$ )
(coords (round (rho (Suc $\left.(n+j))) p^{\prime}\right) p^{\prime}$ )
(rho $(n+j))($ rho $(S u c(n+j)))$
by (auto simp: fg-run-def)
hence round (rho $(n+j)$ ) $p \leq$ round (rho $(n+\operatorname{Suc} j)$ ) $p$
by (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)
with ih show ?P (Suc $j$ ) by auto
qed
\}
with $k$ show ?thesis by simp
qed
Combining the two preceding lemmas, it follows that the local states of
process $p$ at two configurations are the same if these configurations have the same round number.

```
lemma fg-same-round-same-state:
    assumes rho: fg-run A rho HOs SHOs coords
        and \(r d\) : round (rho \(m\) ) \(p=\) round (rho \(n\) ) \(p\)
    shows state (rho m) \(p=\) state (rho n) \(p\)
proof -
    \{
        fix \(k i\)
        have round (rho \((k+i)) p=\) round (rho \(k) p\)
                    \(\Longrightarrow\) state (rho \((k+i)) p=\) state (rho \(k\) ) \(p\)
        (is ? \(R i \Longrightarrow\) ?S \(i\) )
    proof (induct i)
        show ?S 0 by simp
    next
            fix \(j\)
            assume \(i h: ? R j \Longrightarrow\) ? \(S j\)
                and \(r\) : round (rho \((k+\) Suc \(j)) p=\) round \((\) rho \(k) p\)
        from rho have 1: round (rho \(k\) ) \(p \leq\) round (rho \((k+j)) p\)
                by (auto elim: fg-round-numbers-increase)
            from rho have 2: round (rho \((k+j)) p \leq r o u n d ~(r h o ~(k+S u c j)) p\)
                by (auto elim: fg-round-numbers-increase)
            from \(12 r\) have 3: round (rho \((k+j)\) ) \(p=\) round (rho \(k\) ) \(p\) by auto
            with \(r\) have round (rho \((S u c(k+j))) p=\) round (rho \((k+j)) p\) by simp
            with rho have state (rho \((S u c(k+j))) p=\) state \((r h o(k+j)) p\)
                by (auto elim: fg-state-change)
            with 3 ih show ?S (Suc j) by simp
        qed
    \}
    note \(a u x=\) this
    show ?thesis
    proof (cases \(n \leq m\) )
        case True
        then obtain \(k\) where \(m=n+k\) by (auto simp: le-iff-add)
        with \(r d\) show ?thesis by (auto simp: aux)
    next
        case False
        hence \(m \leq n\) by \(\operatorname{simp}\)
        then obtain \(k\) where \(n=m+k\) by (auto simp: le-iff-add)
        with \(r d\) show ?thesis by (auto simp: aux)
    qed
qed
```

Since every process executes every round, function fg-startRound is welldefined. We also list a few facts about fg-startRound that will be used to show that it is a "stuttering sampling function", a notion introduced in the theories about stuttering equivalence.
lemma fg-start-round:

```
    assumes fg-run A rho HOs SHOs coords
    shows round (rho (fg-start-round rho p r)) p=r
using assms by (auto simp: fg-run-def fg-start-round-def intro: LeastI-ex)
lemma fg-start-round-smallest:
    assumes round (rho k) p=r
    shows fg-start-round rho p r\leq (k::nat)
using assms unfolding fg-start-round-def by (rule Least-le)
lemma fg-start-round-later:
    assumes rho: fg-run A rho HOs SHOs coords
        and r: round (rho n) p=r and r':r< r'
    shows n< fg-start-round rho p r'(is - < ?start)
proof (rule ccontr)
    assume \neg ?thesis
    hence start:?start }\leqn\mathrm{ by simp
    from rho this have round (rho ?start) p\leq round (rho n) p
        by (rule fg-round-numbers-increase)
    with r have r}\mp@subsup{r}{}{\prime}\leqr by (simp add: fg-start-round[OF rho]
    with r' show False by simp
qed
lemma fg-start-round-0:
    assumes rho: fg-run A rho HOs SHOs coords
    shows fg-start-round rho p 0=0
proof -
    from rho have round (rho 0) p=0 by (auto simp: fg-run-def fg-init-config-def)
    hence fg-start-round rho p 0 \leq 0 by (rule fg-start-round-smallest)
    thus ?thesis by simp
qed
lemma fg-start-round-strict-mono:
    assumes rho: fg-run A rho HOs SHOs coords
    shows strict-mono (fg-start-round rho p)
proof
    fix r r '
    assume r:(r::nat)< < '
    from rho have round (rho (fg-start-round rho pr)) p=r by (rule fg-start-round)
    from rho this r show fg-start-round rho p r < fg-start-round rho p r'
        by (rule fg-start-round-later)
qed
```

Process $p$ is at round $r$ at all configurations between the start of round $r$ and the start of round $r+1$. By lemma fg-same-round-same-state, this implies that the local state of process $p$ is the same at all these configurations.
lemma fg-round-between-start-rounds:
assumes rho: fg-run A rho HOs SHOs coords
and 1: fg-start-round rho pr$\leq n$
and 2: $n<f g$-start-round rho $p$ (Suc r)

```
shows round (rho n) \(p=r(\) is \(? r d=r)\)
proof (rule antisym)
    from 1 have round (rho (fg-start-round rho \(p r\) )) \(p \leq\) ? \(r d\)
        by (rule fg-round-numbers-increase[OF rho])
    thus \(r \leq\) ?rd by (simp add: fg-start-round \([\) OF rho])
next
    show ? \(r d \leq r\)
    proof (rule ccontr)
        assume \(\neg\) ?thesis
        hence Suc \(r \leq\) ? \(r d\) by simp
        hence \(f g\)-start-round rho \(p\) (Suc r) \(\leq\) fg-start-round rho \(p\) ?rd
            by (rule rho[THEN fg-start-round-strict-mono, THEN strict-mono-mono,
                    THEN monoD])
    also have \(\ldots \leq n\) by (auto intro: fg-start-round-smallest)
    also note 2
    finally show False by simp
    qed
qed
```

For any process $p$ and round $r$ there is some instant $n$ where $p$ executes a local transition from round $r$. In fact, $n+1$ marks the start of round $r+1$.

```
lemma fg-local-transition-from-round:
assumes rho: fg-run A rho HOs SHOs coords
obtains n where round (rho n) }p=
            and fg-start-round rho p (Suc r)=Suc n
            and fg-local A p (HOs r p) (coords (Suc r) p) (rho n) (rho (Suc n))
proof -
    have fg-start-round rho p (Suc r)}\not=0\mathrm{ (is ?start }\not=0
    proof
        assume contr: ?start =0
        from rho have round (rho ?start) p=Suc r by (rule fg-start-round)
        with contr rho show False by (auto simp: fg-run-def fg-init-config-def)
    qed
    then obtain n where n: ?start = Suc n by (auto simp: gr0-conv-Suc)
    with fg-start-round[OF rho, of p Suc r]
    have 0:round (rho (Suc n)) p=Suc r by simp
    have 1: round (rho n) p=r
    proof (rule fg-round-between-start-rounds[OF rho])
        have fg-start-round rho p r<fg-start-round rho p (Suc r)
            by (rule fg-start-round-strict-mono[OF rho, THEN strict-monoD]) simp
        with n show fg-start-round rho pr\leqn by simp
    next
        from n show n < ?start by simp
    qed
    from rho obtain p' where
        fg-next-config A p'(HOs (round (rho n) p') p')
                                    (SHOs (round (rho n) p') p')
                                    (coords (round (rho (Suc n)) p') p')
                            (rho n) (rho (Suc n))
```

```
    (is fg-next-config - ? ?HO ?SHO ?crd ?cfg ?cfg')
    by (force simp: fg-run-def)
    hence fg-local A p (HOsrp)(coords (Suc r) p) (rho n) (rho (Suc n))
    proof (auto simp: fg-next-config-def)
        fix q
        assume fg-send-msg A p' q ?cfg ?cfg'
            - impossible because round changes
        with 01 show ?thesis by (auto simp: fg-send-msg-def)
    next
        fix q
        assume fg-rcv-msg p' q ?HO ?SHO ?cfg ?cfg'
            - impossible because round changes
    with 01 show ?thesis by (auto simp: fg-rcv-msg-def)
next
    assume fg-local A p' ?HO ?crd ?cfg ?cfg'
    with 01 show ?thesis by (cases p}\mp@subsup{p}{}{\prime}=p)(\mathrm{ auto simp: fg-local-def)
qed
with 1n that show ?thesis by auto
qed
```

We now prove two invariants asserted in [4]. The first one states that any message $m$ in transit from process $p$ to process $q$ for round $r$ corresponds to the message computed by $p$ for $q$, given $p$ 's state at its $r$ th local transition.

```
lemma fg-invariant1:
    assumes rho: fg-run A rho HOs SHOs coords
        and m: (p,r,q,m)\in network (rho n) (is ?msg n)
    shows m= sendMsg A r p q (state (rho (fg-start-round rho pr)) p)
using m proof (induct n)
    - the base case is trivial because the network is empty
    assume ?msg 0 with rho show ?thesis
    by (auto simp: fg-run-def fg-init-config-def)
next
    fix n
    assume m': ?msg (Suc n) and ih: ?msg n \Longrightarrow ?thesis
    from rho obtain p' where
    fg-next-config A p'(HOs(round (rho n) p') p')
                            (SHOs (round (rho n) p') p
                            (coords (round (rho (Suc n)) p') p')
                            (rho n) (rho (Suc n))
    (is fg-next-config - - ?HO ?SHO ?crd ?cfg ?cfg')
    by (force simp: fg-run-def)
thus ?thesis
proof (auto simp: fg-next-config-def)
```

Only $f g$-send-msg transitions for process $p$ are interesting, since all other transitions cannot add a message for $p$, hence we can apply the induction hypothesis.
fix $q^{\prime}$
assume send: fg-send-msg A $p^{\prime} q^{\prime} ? c f g ? c f g^{\prime}$
show ?thesis

```
    proof (cases ?msg n)
        case True
        with ih show ?thesis .
    next
        case False
        with send m' have 1: p' = p round ?cfg p=r
                    and 2:m= sendMsg A rpq(state ?cfg p)
        by (auto simp: fg-send-msg-def)
    from rho 1 have state ?cfg p = state (rho (fg-start-round rho p r)) p
        by (auto simp: fg-start-round fg-same-round-same-state)
    with }12\mathrm{ show ?thesis by simp
    qed
next
    fix q}\mp@subsup{q}{}{\prime
    assume fg-rcv-msg p' q' ?HO ?SHO ?cfg ?cfg'
    with m' have ?msg n by (auto simp: fg-rcv-msg-def)
    with ih show ?thesis.
next
    assume fg-local A p' ?HO ?crd ?cfg ?cfg'
    with m' have ?msg n by (auto simp: fg-local-def)
    with ih show ?thesis.
qed
qed
```

The second invariant states that if process $q$ received message $m$ from process $p$, then (a) $p$ is in $q$ 's HO set for that round $m$, and (b) if $p$ is moreover in $q$ 's SHO set, then $m$ is the message that $p$ computed at the start of that round.
lemma fg-invariant2a:
assumes rho: fg-run A rho HOs SHOs coords
and $m$ : rcvd (rho $n$ ) $q p=$ Some $m$ (is ?rcvd $n$ )
shows $p \in H O s$ (round (rho $n$ ) q) $q$
(is $p \in H O s(? r d n) q$ is ?P $n$ )
using $m$ proof (induct $n$ )

- The base case is trivial because $q$ has not received any message initially
assume ? rcvd 0 with rho show ?P 0
by (auto simp: fg-run-def fg-init-config-def)
next
fix $n$
assume rcvd: ?rcvd (Suc $n$ ) and $i h:$ ?rcvd $n \Longrightarrow$ ?P $n$
- For the inductive step we distinguish the possible transitions
from rho obtain $p^{\prime}$ where
fg-next-config A $p^{\prime}$ (HOs (round (rho n) $p^{\prime}$ ) $p^{\prime}$ )
(SHOs (round (rho n) $p^{\prime}$ ) $p^{\prime}$ )
(coords (round (rho (Suc n)) $p^{\prime}$ ) $p^{\prime}$ )
(rho n) (rho (Suc n))
(is $f g$-next-config - - ?HO ?SHO ?crd ?cfg ?cfg')
by (force simp: fg-run-def)
thus ? P (Suc n)

```
proof (auto simp: fg-next-config-def)
```

Except for $f g$-rcv-msg steps of process $q$, the proof is immediately reduced to the induction hypothesis.
fix $q^{\prime}$
assume rcvmsg: fg-rcv-msg $p^{\prime} q^{\prime}$ ?HO ?SHO ?cfg ?cfg'
hence $r d$ : ?rd (Suc $n$ ) $=$ ? ? $\mathrm{rd} n$ by (auto simp: fg-rcv-msg-def)
show ?P (Suc n)
proof (cases ? rcvd n)
case True
with ih rd show ?thesis by simp
next
case False
with rcvd rcumsg rd show ?thesis by (auto simp: fg-rcv-msg-def)
qed
next
fix $q^{\prime}$
assume $f g$-send-msg $A p^{\prime} q^{\prime} ? c f g$ ? $c f g^{\prime}$
with rcvd have ?rcvd $n$ and ?rd (Suc $n$ ) $=$ ? rd $n$
by (auto simp: fg-send-msg-def)
with ih show ?P (Suc n) by simp
next
assume fg-local A p' ?HO ?crd ?cfg ?cfg'
with rcvd have ?rcvd $n$ and ?rd (Suc n) $=$ ? $r d n$
— in fact, $p^{\prime}=q$ is impossible because the rcvd field of $p^{\prime}$ is cleared by (auto simp: fg-local-def)
with $i h$ show ?P (Suc $n$ ) by simp
qed
qed
lemma fg-invariant2b:
assumes rho: fg-run A rho HOs SHOs coords
and $m$ : rcvd (rho $n$ ) $q p=$ Some $m$ (is ?rcvd $n$ )
and sho: $p \in$ SHOs (round (rho n) q) $q$ (is $p \in S H O s(? r d n) q$ )
shows $m=\operatorname{sendMsg} A(? r d n) p q$
(state (rho (fg-start-round rho $p(? r d n))) p$ )
(is ? $P n$ )
using $m$ sho proof (induct $n$ )

- The base case is trivial because $q$ has not received any message initially
assume ?rcvd 0 with rho show ?P 0
by (auto simp: fg-run-def fg-init-config-def)
next
fix $n$
assume rcvd: ?rcvd (Suc n) and $p: p \in S H O s(? r d$ (Suc n)) $q$ and $i h: ? r c v d ~ n \Longrightarrow p \in S H O s(? r d n) q \Longrightarrow$ ?P $n$
- For the inductive step we again distinguish the possible transitions
from rho obtain $p^{\prime}$ where
fg-next-config A $p^{\prime}\left(H O s\left(\right.\right.$ round (rho n) $\left.p^{\prime}\right) p^{\prime}$ )
(SHOs (round (rho n) $p^{\prime}$ ) $p^{\prime}$ )

```
    (coords (round (rho (Suc n)) p') p')
    (rho n) (rho (Suc n))
    (is fg-next-config - ? ?HO ?SHO ?crd ?cfg ?cfg')
    by (force simp: fg-run-def)
thus ?P (Suc n)
proof (auto simp: fg-next-config-def)
```

Except for $f g$ - $r c v$ - $m s g$ steps of process $q$, the proof is immediately reduced to the induction hypothesis.

```
        fix q
```

        assume rcvmsg: fg-rcv-msg \(p^{\prime} q^{\prime}\) ?HO ?SHO ?cfg ?cfg'
        hence \(r d\) : ?rd (Suc \(n\) ) \(=\) ? ? \(d\) d \(n\) by (auto simp: fg-rcv-msg-def)
        show ?P (Suc n)
        proof (cases ?rcvd n)
            case True
            with ih \(p\) rd show ?thesis by simp
        next
            case False
            from rcumsg obtain \(m^{\prime} m^{\prime \prime}\) where
                ( \(q^{\prime}\), round ?cfg \(p^{\prime}, p^{\prime}, m^{\prime}\) ) \(\in\) network ?cfg
                rcvd ? \(\mathrm{cfg}^{\prime}=(\) rcvd ? \(c f g)\left(p^{\prime}:=\left(\right.\right.\) rcvd \(\left.? c f g p^{\prime}\right)\left(q^{\prime}:=\right.\)
                                    if \(q^{\prime} \in\) ?SHO then Some \(m^{\prime}\) else Some \(\left.m^{\prime \prime}\right)\) )
                by (auto simp: fg-rcv-msg-def split del: if-split-asm)
        with False rcvd \(p\) rd have \((p\), ?rd \(n, q, m) \in\) network ?cfg by auto
        with rho rd show ?thesis by (auto simp: fg-invariant1)
        qed
    next
        fix \(q^{\prime}\)
        assume \(f g\)-send-msg A \(p^{\prime} q^{\prime} ? c f g\) ? \(c f g^{\prime}\)
        with rcvd have ?rcvd \(n\) and ?rd (Suc \(n\) ) \(=\) ? \(r d n\)
        by (auto simp: fg-send-msg-def)
        with \(p\) ih show ?P (Suc n) by simp
        next
        assume fg-local A p' ?HO ?crd ?cfg ?cfg'
        with rcvd have ?rcvd \(n\) and ?rd (Suc \(n\) ) \(=\) ? rd \(n\)
        - in fact, \(p^{\prime}=q\) is impossible because the rcvd field of \(p^{\prime}\) is cleared
        by (auto simp: fg-local-def)
    with \(p\) ih show ?P (Suc \(n\) ) by simp
    qed
qed

### 3.3 From Fine-Grained to Coarse-Grained Runs

The reduction theorem asserts that for any fine-grained run rho there is a coarse-grained run such that every process sees the same sequence of local states in the two runs, modulo stuttering. In other words, no process can locally distinguish the two runs.
Given fine-grained run rho, the corresponding coarse-grained run sigma is
defined as the sequence of state vectors at the beginning of every round. Notice in particular that the local states sigma $r p$ and sigma $r q$ of two different processes $p$ and $q$ appear at different instants in the original run rho. Nevertheless, we prove that sigma is a coarse-grained run of the algorithm for the same $\mathrm{HO}, \mathrm{SHO}$, and coordinator assignments. By definition (and the fact that local states remain equal between $f g$-start-round instants), the sequences of process states in rho and sigma are easily seen to be stuttering equivalent, and this will be formally stated below.

```
definition coarse-run where
    coarse-run rho r \(p \equiv\) state (rho (fg-start-round rho pr)) p
theorem reduction:
    assumes rho: fg-run A rho HOs SHOs coords
    shows CSHORun A (coarse-run rho) HOs SHOs coords
        (is CSHORun - ?cr - - -)
proof (auto simp: CSHORun-def)
    from rho show CHOinitConfig A (?.cr 0) (coords 0)
        by (auto simp: fg-run-def fg-init-config-def CHOinitConfig-def
                        coarse-run-def fg-start-round- 0 [OF rho])
next
    fix \(r\)
    show CSHOnextConfig A r (?cr r) (HOs r) (SHOs r) (coords (Suc r))
                            (?cr (Suc r))
    proof (auto simp add: CSHOnextConfig-def)
        fix \(p\)
        from rho[THEN fg-local-transition-from-round] obtain \(n\)
            where \(n\) : round (rho \(n\) ) \(p=r\)
                and start: fg-start-round rho \(p(\) Suc r) \(=\) Suc \(n\) (is ?start \(=-\) )
                and loc: fg-local A \(p\) (HOs rp) (coords (Suc r) p) (rho n) (rho (Suc n))
                        (is fg-local - - ?HO ?crd ?cfg ?cfg')
            by blast
        have cfg: ?cr r \(p=\) state ?cfg \(p\)
            unfolding coarse-run-def proof (rule fg-same-round-same-state[OF rho])
            from \(n\) show round (rho (fg-start-round rho \(p r\) )) \(p=\) round ? \(c f g ~ p\)
                by (simp add: fg-start-round [OF rho])
            qed
            from start have \(c f g^{\prime}:\) ? cr (Suc r) \(p=\) state ?cfg' \(p\)
                by (simp add: coarse-run-def)
            have rcvd: rcvd ?cfg \(p \in\) SHOmsgVectors A r \(p\) (?cr r) ?HO (SHOs r p)
            proof (auto simp: SHOmsgVectors-def)
                fix \(q\)
                assume \(q \in\) ? \(H O\)
                with \(n\) loc show \(\exists m\). rcvd ? cfg \(p q=\) Some \(m\) by (auto simp: fg-local-def)
            next
                fix \(q m\)
                assume rcvd?cfg p \(q=\) Some \(m\)
                with rho \(n\) show \(q \in\) ?HO by (auto simp: fg-invariant2a)
    next
```

```
    fix q
    assume sho: q\inSHOs r p and ho: q\in?HO
    from ho n loc obtain m}\mathrm{ where rcvd?cfg p q=Some m
    by (auto simp: fg-local-def)
    with rho n sho show rcvd ?cfg p q = Some (sendMsg A r q p (?cr r q))
    by (auto simp: fg-invariant2b coarse-run-def)
    qed
    with n loc cfg cfg'
    show \exists\mu\inSHOmsgVectors A r p (?cr r) ?HO (SHOs r p).
        CnextState A r p (?cr r p) \mu ?crd (?cr (Suc r) p)
        by (auto simp: fg-local-def)
    qed
qed
```


### 3.4 Locally Similar Runs and Local Properties

We say that two sequences of configurations (vectors of process states) are locally similar if for every process the sequences of its process states are stuttering equivalent. Observe that different stuttering reduction may be applied for every process, hence the original sequences of configurations need not be stuttering equivalent and can indeed differ wildly in the combinations of local states that occur.

A property of a sequence of configurations is called local if it is insensitive to local similarity.

```
definition locally-similar where
    locally-similar ( \(\sigma::\) nat \(\Rightarrow\) 'proc \(\Rightarrow\) 'pst) \(\tau \equiv\)
    \(\forall p:: ' p r o c .(\lambda n . \sigma n p) \approx(\lambda n . \tau n p)\)
definition local-property where
    local-property \(P \equiv\)
    \(\forall \sigma \tau\). locally-similar \(\sigma \tau \longrightarrow P \sigma \longrightarrow P \tau\)
```

Local similarity is an equivalence relation.
lemma locally-similar-refl: locally-similar $\sigma \sigma$ by (simp add: locally-similar-def stutter-equiv-refl)
lemma locally-similar-sym: locally-similar $\sigma \tau \Longrightarrow$ locally-similar $\tau \sigma$ by (simp add: locally-similar-def stutter-equiv-sym)
lemma locally-similar-trans [trans]:
locally-similar $\varrho \sigma \Longrightarrow$ locally-similar $\sigma \tau \Longrightarrow$ locally-similar $\varrho \tau$
by (force simp add: locally-similar-def elim: stutter-equiv-trans)
lemma local-property-eq:
local-property $P=(\forall \sigma \tau$. locally-similar $\sigma \tau \longrightarrow P \sigma=P \tau)$
by (auto simp: local-property-def dest: locally-similar-sym)
Consider any fine-grained run rho. The projection of rho to vectors of pro-
cess states is locally similar to the coarse-grained run computed from rho.

```
lemma coarse-run-locally-similar:
    assumes rho: fg-run A rho HOs SHOs coords
    shows locally-similar (state o rho) (coarse-run rho)
proof (auto simp: locally-similar-def)
    fix p
    show (\lambdan. state (rho n) p) \approx (\lambdan. coarse-run rho n p) (is ?fgr }\approx\mathrm{ ?cgr)
    proof (rule stutter-equivI)
        show stutter-sampler (fg-start-round rho p) ?fgr
        proof (auto simp: stutter-sampler-def)
            from rho show fg-start-round rho p 0 = 0
                by (rule fg-start-round-0)
        next
            show strict-mono (fg-start-round rho p)
                by (rule fg-start-round-strict-mono[OF rho])
    next
            fix r n
            assume fg-start-round rho pr<n and n<fg-start-round rho p (Suc r)
            with rho have round (rho n) p=round (rho (fg-start-round rho p r)) p
                by (simp add: fg-start-round fg-round-between-start-rounds)
            with rho show state (rho n) p= state (rho (fg-start-round rho p r)) p
                by (rule fg-same-round-same-state)
        qed
    next
        show stutter-sampler id ?cgr
            by (rule id-stutter-sampler)
    next
        show ?fgr \circ fg-start-round rho p=?cgr \circ id
            by (auto simp: coarse-run-def)
    qed
qed
```

Therefore, in order to verify a local property $P$ for a fine-grained run over given $\mathrm{HO}, \mathrm{SHO}$, and coord collections, it is enough to show that $P$ holds for all coarse-grained runs for these same collections. Indeed, one may restrict attention to coarse-grained runs whose initial states agree with that of the given fine-grained run.
theorem local-property-reduction:
assumes rho: fg-run A rho HOs SHOs coords
and $P$ : local-property $P$
and coarse-correct:
$\bigwedge$ crho. 【CSHORun A crho HOs SHOs coords; crho $0=$ state (rho 0)】 $\Longrightarrow P$ crho
shows $P($ state $\circ$ rho $)$
proof -
have coarse-run rho $0=$ state (rho 0 )
by (rule ext, simp add: coarse-run-def fg-start-round- 0 [OF rho])
from rho[THEN reduction] this
have $P$ (coarse-run rho) by (rule coarse-correct)
with coarse-run-locally-similar[OF rho] $P$
show ?thesis by (auto simp: local-property-eq)
qed

### 3.5 Consensus as a Local Property

Consensus and Weak Consensus are local properties and can therefore be verified just over coarse-grained runs, according to theorem local-property-reduction.

```
lemma integrity-is-local:
    assumes sim: locally-similar \(\sigma \tau\)
        and val: \(\bigwedge n\). dec \((\sigma n p)=\) Some \(v \Longrightarrow v \in\) range vals
        and dec: dec ( \(\tau \quad n\) p) \(=\) Some \(v\)
    shows \(v \in\) range vals
proof -
    from \(\operatorname{sim}\) have \((\lambda r . \sigma r p) \approx(\lambda r . \tau r p)\) by (simp add: locally-similar-def)
    then obtain \(m\) where \(\sigma m p=\tau n p\) by (rule stutter-equiv-element-left)
    from sym[OF this] dec show ?thesis by (auto elim: val)
qed
lemma validity-is-local:
    assumes sim: locally-similar \(\sigma \tau\)
        and val: \(\bigwedge n\). \(\operatorname{dec}(\sigma n p)=\) Some \(w \Longrightarrow w=v\)
        and dec: \(\operatorname{dec}(\tau n p)=\) Some \(w\)
    shows \(w=v\)
proof -
    from \(\operatorname{sim}\) have \((\lambda r . \sigma r p) \approx(\lambda r . \tau r p)\) by (simp add: locally-similar-def)
    then obtain \(m\) where \(\sigma m p=\tau n p\) by (rule stutter-equiv-element-left)
    from sym[OF this] dec show ?thesis by (auto elim: val)
qed
lemma agreement-is-local:
    assumes sim: locally-similar \(\sigma \tau\)
    and agr: \(\bigwedge m n . \llbracket \operatorname{dec}(\sigma m p)=\operatorname{Some} v ; \operatorname{dec}(\sigma n q)=\) Some \(w \rrbracket \Longrightarrow v=w\)
    and \(v: \operatorname{dec}(\tau m p)=\) Some \(v\) and \(w: \operatorname{dec}(\tau n q)=\) Some \(w\)
    shows \(v=w\)
proof -
    from \(\operatorname{sim}\) have \((\lambda r . \sigma r p) \approx(\lambda r . \tau r p)\) by (simp add: locally-similar-def)
    then obtain \(m^{\prime}\) where \(m^{\prime}: \sigma m^{\prime} p=\tau m p\) by (rule stutter-equiv-element-left)
    from \(\operatorname{sim}\) have \((\lambda r . \sigma r q) \approx(\lambda r . \tau r q)\) by (simp add: locally-similar-def)
    then obtain \(n^{\prime}\) where \(n^{\prime}: \sigma n^{\prime} q=\tau n q\) by (rule stutter-equiv-element-left)
    from \(\operatorname{sym}[O F m] \operatorname{sym}[O F n] v w\) show \(v=w\) by (auto elim: agr)
qed
lemma termination-is-local:
    assumes sim: locally-similar \(\sigma \tau\)
        and trm: dec \((\sigma m p)=\) Some \(v\)
    shows \(\exists n\). dec \((\tau n p)=\) Some \(v\)
proof -
```

```
    from sim have ( }\lambdar.\sigmarp)\approx(\lambdar.\tau r p) by (simp add:locally-similar-def
    then obtain n where \sigmamp=\tau n p by (rule stutter-equiv-element-right)
    with trm show ?thesis by auto
qed
theorem consensus-is-local: local-property (consensus vals dec)
proof (auto simp: local-property-def consensus-def)
    fix }\sigma\taunp
    assume locally-similar \sigma \tau
    and }\forallnpv.dec (\sigmanp)=Some v\longrightarrowv\in range val
    and dec (\tau n p)=Some v
    thus v\in range vals by (blast intro: integrity-is-local)
next
    fix }\sigma\taumnpqv
    assume locally-similar \sigma\tau
    and \forallmn pqv w. dec (\sigmamp)=Some v^dec (\sigma n q) = Some w\longrightarrowv=w
    and dec (\tau m p) = Some v and dec (\tau n q) = Some w
    thus v=w by (blast intro:agreement-is-local)
next
    fix }\sigma\tau
    assume locally-similar \sigma \tau
    and }\forallp.\existsmv.dec (\sigma m p)=Some 
    thus \existsnw.dec (\tau n p) = Some w by (blast dest: termination-is-local)
qed
theorem weak-consensus-is-local: local-property (weak-consensus vals dec)
proof (auto simp: local-property-def weak-consensus-def)
    fix }\sigma\taunpv
    assume locally-similar \sigma \tau
    and }\forallnpw.\operatorname{dec}(\sigmanp)=Some w\longrightarroww=
    and dec (\tau n p) = Some w
    thus w=v by (blast intro: validity-is-local)
next
    fix \sigma\tau m n pqvw
    assume locally-similar \sigma \tau
    and }\forallmnpqvw.\operatorname{dec}(\sigmamp)=Some v\wedge dec (\sigmanq)=Some w\longrightarrowv=
    and dec (\tau m p) = Some v and w: dec (\tau n q) = Some w
    thus v=w by (blast intro: agreement-is-local)
next
    fix }\sigma\tau
    assume locally-similar \sigma \tau
    and }\forallp.\existsmv.dec (\sigma m p)=Some 
    thus \existsn w. dec (\tau n p) = Some w by (blast dest: termination-is-local)
qed
end
theory Majorities
imports Main
```

begin

## 4 Utility Lemmas About Majorities

Consensus algorithms usually ensure that a majority of processes proposes the same value before taking a decision, and we provide a few utility lemmas for reasoning about majorities.

Any two subsets $S$ and $T$ of a finite set $E$ such that the sum of their cardinalities is larger than the size of $E$ have a non-empty intersection.

```
lemma abs-majorities-intersect:
    assumes crd: card E < card S + card T
            and s:S\subseteqE and t:T\subseteqE and e: finite E
        shows }S\capT\not={
proof (clarify)
    assume contra: S\capT={}
    from st e have finite S and finite T by (auto simp: finite-subset)
    with crd contra have card E<card (S\cupT) by (auto simp add: card-Un-Int)
    moreover
    from ste have card (S\cupT)\leqcard E by (simp add:card-mono)
    ultimately
    show False by simp
qed
lemma abs-majoritiesE:
    assumes crd: card E<card S + card T
        and s:S\subseteqE and t:T\subseteqE and e: finite E
    obtains p where p\inS and p\inT
proof -
    from assms have S\capT\not={} by (rule abs-majorities-intersect)
    then obtain p where p\inS\capT by blast
    with that show ?thesis by auto
qed
Special case: both sets S and T are majorities.
lemma abs-majoritiesE':
    assumes Smaj: card S > (card E) div 2 and Tmaj:card T > (card E) div 2
        and s:S\subseteqE and t:T\subseteqE and e: finite E
    obtains p where p\inS and p\inT
proof (rule abs-majoritiesE[OF - s t e])
    from Smaj Tmaj show card E<card S + card T by auto
qed
```

We restate the above theorems for the case where the base type is finite (taking $E$ as the universal set).
lemma majorities-intersect:
assumes crd: card (UNIV::('a::finite) set) < card (S::'a set) $+\operatorname{card} T$

```
    shows S\capT\not={}
    by (rule abs-majorities-intersect[OF crd]) auto
lemma majoritiesE:
    assumes crd: card (UNIV::('a::finite) set) < card (S::'a set) + card (T::'a set)
    obtains p where p\inS and p\inT
using crd majorities-intersect by blast
lemma majoritiesE':
    assumes S: card (S::('a::finite) set) > (card (UNIV::'a set)) div 2
    and T: card (T::'a set) > (card (UNIV::'a set)) div 2
    obtains p where p\inS and p\inT
by (rule abs-majoritiesE'[OF ST]) auto
end
theory OneThirdRuleDefs
imports ../HOModel
begin
```


## 5 Verification of the One-Third Rule Consensus Algorithm

We now apply the framework introduced so far to the verification of concrete algorithms, starting with algorithm One-Third Rule, which is one of the simplest algorithms presented in [7]. Nevertheless, the algorithm has some interesting characteristics: it ensures safety (i.e., the Integrity and Agreement) properties in the presence of arbitrary benign faults, and if everything works perfectly, it terminates in just two rounds. One-Third Rule is an uncoordinated algorithm tolerating benign faults, hence SHO or coordinator sets do not play a role in its definition.

### 5.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic HO model.
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)

## abbreviation

$$
N \equiv \operatorname{card}(U N I V:: \text { Proc set })
$$

The state of each process consists of two fields: $x$ holds the current value proposed by the process and decide the value (if any, hence the option type) it has decided.
record 'val pstate $=$

```
x :: 'val
decide :: 'val option
```

The initial value of field $x$ is unconstrained, but no decision has been taken initially.
definition $O T R$-initState where
OTR-initState $p$ st $\equiv$ decide st $=$ None
Given a vector msgs of values (possibly null) received from each process, HOV msgs $v$ denotes the set of processes from which value $v$ was received.

```
definition \(H O V::(\) Proc \(\Rightarrow\) 'val option \() \Rightarrow\) 'val \(\Rightarrow\) Proc set where
    HOV msgs \(v \equiv\{q \cdot m s g s q=\) Some \(v\}\)
```

MFR msgs $v$ ("most frequently received") holds for vector msgs if no value has been received more frequently than $v$.
Some such value always exists, since there is only a finite set of processes and thus a finite set of possible cardinalities of the sets HOV msgs $v$.

```
definition \(M F R::(\) Proc \(\Rightarrow\) 'val option \() \Rightarrow\) 'val \(\Rightarrow\) bool where
    MFR msgs \(v \equiv \forall w\). card (HOV msgs \(w) \leq \operatorname{card}\) (HOV msgs v)
lemma MFR-exists: \(\exists v . M F R\) msgs \(v\)
proof -
    let ? cards \(=\{\) card (HOV msgs v) |v.True \(\}\)
    let ?mfr = Max ? cards
    have \(\forall v\). card (HOV msgs \(v\) ) \(\leq N\) by (auto intro: card-mono)
    hence ?cards \(\subseteq\{0 . . N\}\) by auto
    hence fin: finite ? cards by (metis atLeast0AtMost finite-atMost finite-subset)
    hence ?mfr \(\in\) ?cards by (rule Max-in) auto
    then obtain \(v\) where \(v: ? m f r=\operatorname{card}(H O V m s g s v)\) by auto
    have MFR msgs \(v\)
    proof (auto simp: MFR-def)
        fix \(w\)
        from fin have card (HOV msgs w) \(\leq\) ? mfr by (rule Max-ge) auto
        thus card (HOV msgs w) \(\leq\) card (HOV msgs v) by (unfold \(v\) )
    qed
    thus ?thesis ..
qed
```

Also, if a process has heard from at least one other process, the most frequently received values are among the received messages.

```
lemma \(M F R\)-in-msgs:
    assumes \(H O\) :HOs \(m p \neq\{ \}\)
        and \(v\) : MFR (HOrcvdMsgs OTR-M mp(HOs m p) (rho m)) v
                (is MFR ?msgs \(v\) )
    shows \(\exists q \in H O s m p . v=\) the \((\) ?msgs \(q)\)
proof -
    from \(H O\) obtain \(q\) where \(q: q \in H O s m p\)
```

```
    by auto
    with v have HOV ?msgs (the (?msgs q)) = {}
    by (auto simp: HOV-def HOrcvdMsgs-def)
    hence HOp: 0 < card (HOV ?msgs (the (?msgs q)))
    by auto
    also from v have \ldots\leqcard (HOV ?msgs v)
    by (simp add: MFR-def)
finally have HOV ?msgs v}\not={
    by auto
thus ?thesis
    by (auto simp: HOV-def HOrcvdMsgs-def)
qed
```

Two Thirds msgs $v$ holds if value $v$ has been received from more than $2 / 3$ of all processes.

```
definition TwoThirds where
    TwoThirds msgs v \equiv(2*N) div 3 < card (HOV msgs v)
```

The next-state relation of algorithm One-Third Rule for every process is defined as follows: if the process has received values from more than $2 / 3$ of all processes, the $x$ field is set to the smallest among the most frequently received values, and the process decides value $v$ if it received $v$ from more than $2 / 3$ of all processes. If $p$ hasn't heard from more than $2 / 3$ of all processes, the state remains unchanged. (Note that Some is the constructor of the option datatype, whereas $\epsilon$ is Hilbert's choice operator.) We require the type of values to be linearly ordered so that the minimum is guaranteed to be well-defined.

```
definition OTR-nextState where
    OTR-nextState r p (st::('val::linorder) pstate) msgs st' \equiv
    if (2*N) div 3 < card {q. msgs q}\not=\mathrm{ None}
    then st' = \ x = Min {v.MFR msgs v},
            decide =( if (\existsv. TwoThirds msgs v)
                then Some (\epsilonv. TwoThirds msgs v)
                else decide st) ()
    else st' = st
```

The message sending function is very simple: at every round, every process sends its current proposal (field $x$ of its local state) to all processes.

## definition $O T R$-sendMsg where

OTR-sendMsgrpqst $\equiv x$ st

### 5.2 Communication Predicate for One-Third Rule

We now define the communication predicate for the One-Third Rule algorithm to be correct. It requires that, infinitely often, there is a round where all processes receive messages from the same set $\Pi$ of processes where $\Pi$
contains more than two thirds of all processes. The "per-round" part of the communication predicate is trivial.

```
definition \(O T R\)-commPerRd where
    OTR-commPerRd HOrs \(\equiv\) True
definition OTR-commGlobal where
    OTR-commGlobal HOs \(\equiv\)
        \(\forall r . \exists r 0 \Pi . r 0 \geq r \wedge(\forall p . H O s r 0 p=\Pi) \wedge \operatorname{card} \Pi>(2 * N) \operatorname{div} 3\)
```


### 5.3 The One-Third Rule Heard-Of Machine

We now define the HO machine for the One-Third Rule algorithm by assembling the algorithm definition and its communication-predicate. Because this is an uncoordinated algorithm, the crd arguments of the initial- and next-state predicates are unused.

```
definition OTR-HOMachine where
    OTR-HOMachine =
        \ CinitState = ( }\lambda\mathrm{ p st crd. OTR-initState p st),
        sendMsg=OTR-sendMsg,
        CnextState = ( }\lambda\mathrm{ r p st msgs crd st'. OTR-nextState r p st msgs st'})
        HOcommPerRd = OTR-commPerRd,
        HOcommGlobal = OTR-commGlobal D
```

abbreviation $O T R-M \equiv O T R-H O M a c h i n e::\left(P r o c,{ }^{\prime}\right.$ val::linorder pstate, 'val) HOMachine
end
theory OneThirdRuleProof
imports OneThirdRuleDefs ../Reduction ../Majorities
begin
We prove that One-Third Rule solves the Consensus problem under the communication predicate defined above. The proof is split into proofs of the Integrity, Agreement, and Termination properties.

### 5.4 Proof of Integrity

Showing integrity of the algorithm is a simple, if slightly tedious exercise in invariant reasoning. The following inductive invariant asserts that the values of the $x$ and decide fields of the process states are limited to the $x$ values present in the initial states since the algorithm does not introduce any new values.

```
definition VInv where
    VInv rho \(n \equiv\)
    let xinit \(=(\) range \((x \circ(\) rho 0\()))\)
    in range \((x \circ(\) rho \(n)) \subseteq\) xinit
```

```
^range (decide ○ (rho n))\subseteq{None }}\cup(Some 'xinit)
```

```
lemma vinv-invariant:
    assumes run:HORun OTR-M rho HOs
    shows VInv rho \(n\)
proof (induct \(n\) )
    from run show VInv rho 0
        by (simp add: HORun-eq HOinitConfig-eq OTR-HOMachine-def initState-def
                                    OTR-initState-def VInv-def image-def)
next
    fix \(m\)
    assume ih: VInv rho \(m\)
    let ?xinit \(=\) range \((x \circ(\) rho 0\())\)
    have range \((x \circ(\) rho \((\) Suc \(m))) \subseteq\) ? xinit
    proof (clarsimp cong del: image-cong-simp)
        fix \(p\)
        from run
        have nxt: OTR-nextState m \(p\) (rho \(m p\) )
                            (HOrcvdMsgs OTR-M mp(HOs m p) (rho m))
                    (rho (Suc m) p)
            (is OTR-nextState - - ?st ?msgs ?st')
    by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
    show \(x\) ?st' \(\in\) ?xinit
    proof \((\) cases \((2 * N)\) div \(3<\operatorname{card}(H O s m p))\)
        case True
        hence \(H O\) : HOs m \(p \neq\{ \}\) by auto
        let \(?\) MFRs \(=\{v . M F R\) ?msgs \(v\}\)
        have Min ?MFRs \(\in\) ?MFRs
        proof (rule Min-in)
            from \(H O\) have ? \(M F R s \subseteq(\text { the } \circ \text { ? } m s g s)^{\text {' }}(H O s m p)\)
            by (auto simp: image-def intro: MFR-in-msgs)
        thus finite ?MFRs by (auto elim: finite-subset)
    next
                from \(M F R\)-exists show ?MFRs \(\neq\{ \}\) by auto
    qed
    with \(H O\) have \(\exists q \in H O s m p\). Min ?MFRs \(=\) the \((\) ? msgs \(q)\)
        by (intro MFR-in-msgs) auto
    hence \(\exists q \in H O s m p\). Min ? \({ }^{\text {MFR }}=x(\) rho \(m q)\)
                by (auto simp: HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)
    moreover
        from True nxt have \(x\) ?st \(=\) Min ?MFRs
            by (simp add: OTR-nextState-def HOrcvdMsgs-def)
    ultimately
        show ?thesis using ih by (auto simp: VInv-def image-def)
    next
            case False
        with nxt ih show ?thesis
            by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def Let-def)
    qed
```

```
    qed
    moreover
    have }\forallp\mathrm{ . decide ((rho (Suc m)) p) {{None } U(Some`?xinit)
    proof
        fix p
        from run
        have nxt: OTR-nextState m p (rho m p)
                                    (HOrcvdMsgs OTR-M m p (HOs m p) (rho m))
                                    (rho (Suc m) p)
            (is OTR-nextState - - ?st ?msgs ?st')
    by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
    show decide ?st' }\in{None}\cup(Some'?xinit
    proof (cases (2*N) div 3<card {q. ?msgs q \not=None})
        assume HO: (2*N) div 3< card {q. ?msgs q\not=None}
        show ?thesis
        proof (cases \existsv. TwoThirds ?msgs v)
            case True
            let ?dec = \epsilonv. TwoThirds?msgs v
            from True have TwoThirds ?msgs ?dec by (rule someI-ex)
            hence HOV ?msgs ?dec }\not={}\mathrm{ by (auto simp add: TwoThirds-def)
            then obtain q}\mathrm{ where x (rho m q) = ?dec
                by (auto simp: HOV-def HOrcvdMsgs-def OTR-HOMachine-def
                    OTR-sendMsg-def)
            from sym[OF this] nxt ih show ?thesis
                by (auto simp: OTR-nextState-def VInv-def image-def)
        next
            case False
            with HO nxt ih show ?thesis
                by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def image-def)
        qed
    next
        case False
        with nxt ih show ?thesis
            by (auto simp:OTR-nextState-def VInv-def image-def)
        qed
    qed
    hence range (decide ○ (rho (Suc m)))\subseteq{None} \cup(Some' ?xinit) by auto
    ultimately
    show VInv rho (Suc m) by (auto simp: VInv-def image-def)
qed
Integrity is an immediate consequence.
```

```
theorem OTR-integrity:
```

theorem OTR-integrity:
assumes run:HORun OTR-M rho HOs and dec: decide (rho n p) $=$ Some $v$
assumes run:HORun OTR-M rho HOs and dec: decide (rho n p) $=$ Some $v$
shows $\exists q . v=x($ rho $0 q)$
shows $\exists q . v=x($ rho $0 q)$
proof -
proof -
let ?xinit $=$ range $(x \circ($ rho 0$))$
let ?xinit $=$ range $(x \circ($ rho 0$))$
from run have VInv rho $n$ by (rule vinv-invariant)
from run have VInv rho $n$ by (rule vinv-invariant)
hence range $($ decide $\circ($ rho $n)) \subseteq\{$ None $\} \cup($ Some '?xinit $)$

```
    hence range \((\) decide \(\circ(\) rho \(n)) \subseteq\{\) None \(\} \cup(\) Some '?xinit \()\)
```

```
    by (auto simp:VInv-def Let-def)
    hence decide ((rho n) p)\in{None} \cup (Some'?xinit)
    by (auto simp: image-def)
    with dec show ?thesis by auto
qed
```


### 5.5 Proof of Agreement

The following lemma $A 1$ asserts that if process $p$ decides in a round on a value $v$ then more than $2 / 3$ of all processes have $v$ as their $x$ value in their local state.

We show a few simple lemmas in preparation.
lemma nextState-change:
assumes HORun OTR-M rho HOs
and $\neg((2 * N)$ div 3

$$
<\operatorname{card}\{q .(\text { HOrcvdMsgs OTR-M n } p(\text { HOs } n p)(\text { rho } n)) q \neq \text { None }\})
$$

shows rho (Suc n) $p=$ rho $n p$
using assms
by (auto simp: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def OTR-nextState-def)
lemma nextState-decide:
assumes run:HORun OTR-M rho HOs
and chg: decide (rho (Suc n) p) $=$ decide (rho n p)
shows TwoThirds (HOrcvdMsgs OTR-Mnp(HOs n p) (rho n))
(the (decide (rho (Suc n) p)))
proof -
from run
have OTR-nextState $n$ p (rho $n$ p)
(HOrcvdMsgs OTR-M n p (HOs n p) (rho n)) (rho (Suc n) p)
by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
with chg show ?thesis by (auto simp: OTR-nextState-def elim: someI)
qed
lemma A1:
assumes run:HORun OTR-M rho HOs
and dec: decide (rho (Suc n) p) = Some $v$
and chg: decide (rho (Suc n) $p$ ) $\neq$ decide (rho $n p$ ) (is decide ?st' $\neq$ decide ?st)
shows $(2 * N)$ div $3<\operatorname{card}\{q . x($ rho $n q)=v\}$
proof -
from run chg
have TwoThirds (HOrcvdMsgs OTR-M n $p$ (HOs $n$ p) (rho n))
(the (decide ?st'))
(is TwoThirds ?msgs -)
by (rule nextState-decide)
with dec have TwoThirds ?msgs $v$ by simp
hence $(2 * N)$ div $3<$ card $\{q$. ?msgs $q=$ Some $v\}$
by ( simp add: TwoThirds-def HOV-def)

```
    moreover
    have {q. ?msgs q=Some v }\subseteq{q.x (rho n q) =v }
    by (auto simp: OTR-HOMachine-def OTR-sendMsg-def HOrcvdMsgs-def)
    hence card {q. ?msgs q=Some v}\leqcard {q.x (rho n q)=v}
    by (simp add: card-mono)
ultimately
show ?thesis by simp
qed
```

The following lemma A2 contains the crucial correctness argument: if more than $2 / 3$ of all processes send $v$ and process $p$ hears from more than $2 / 3$ of all processes then the $x$ field of $p$ will be updated to $v$.

```
lemma \(A 2\) :
    assumes run: HORun OTR-M rho HOs
    and \(H O:(2 * N)\) div 3
    \(<\operatorname{card}\{q\). HOrcvdMsgs OTR-M n \(p\) (HOs n \(p\) ) (rho \(n) q \neq\) None \(\}\)
    and maj: \((2 * N)\) div \(3<\operatorname{card}\{q \cdot x(\) rho \(n q)=v\}\)
    shows \(x\) (rho (Suc n) p) \(=v\)
proof -
    from run
    have nxt: OTR-nextState \(n\) p (rho \(n\) p)
                    (HOrcvdMsgs OTR-M n p (HOs n p) (rho n))
                        (rho (Suc n) p)
        (is OTR-nextState - ? ?st ?msgs ?st')
    by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
    let ?HOVothers \(=\bigcup\{H O V\) ?msgs \(w \mid w \cdot w \neq v\}\)
    - processes from which \(p\) received values different from \(v\)
have \(w\) : card ?HOVothers \(\leq N\) div 3
proof -
    have card ?HOVothers \(\leq\) card \((U N I V-\{q \cdot x(\) rho \(n q)=v\})\)
    by (auto simp: HOV-def HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def
                intro: card-mono)
    also have \(\ldots=N-\operatorname{card}\{q \cdot x(\) rho \(n q)=v\}\)
        by (auto simp: card-Diff-subset)
    also from maj have \(\ldots \leq N\) div 3 by auto
    finally show?thesis.
qed
have hov: HOV ?msgs \(v=\{q\). ?msgs \(q \neq\) None \(\}-\) ?HOVothers
    by (auto simp: HOV-def) blast
have othHO: ?HOVothers \(\subseteq\{q\).?msgs \(q \neq\) None \(\}\)
    by (auto simp: HOV-def)
```

Show that $v$ has been received from more than $N / 3$ processes.

```
from HO have N div 3<card {q. ?msgs q}\not=\mathrm{ None } - (N div 3)
```

    by auto
    ```
also from \(w H O\) have \(\ldots \leq \operatorname{card}\{q\). ?msgs \(q \neq\) None \(\}-\) card ?HOVothers
    by auto
also from hov othHO have \(\ldots=\operatorname{card}\) (HOV ?msgs v)
    by (auto simp: card-Diff-subset)
finally have HOV: \(N\) div \(3<\operatorname{card}\) (HOV?msgs v).
```

All other values are received from at most $N / 3$ processes.

```
have }\forallw.w\not=v\longrightarrow\mathrm{ card (HOV ?msgs w) < card ?HOVothers
    by (force intro: card-mono)
with w have cardw: }\forallw.w\not=v\longrightarrow\operatorname{card (HOV ?msgs w)}\leqN\mathrm{ div 3 by auto
```

In particular, $v$ is the single most frequently received value.
with $H O V$ have $M F R$ ? msgs $v$ by (auto simp: MFR-def)

```
moreover
have }\forallw.w\not=v\longrightarrow\neg(MFR\mathrm{ ?msgs w)
proof (auto simp:MFR-def not-le)
    fix w
    assume w\not=v
    with cardw HOV have card (HOV ?msgs w) < card (HOV ?msgs v) by auto
    thus \existsv.card (HOV ?msgs w) < card (HOV ?msgs v) ..
qed
ultimately
have mfrv: {w.MFR ?msgs w}={v} by auto
have card {q.?msgs q=Some v }}\leq\operatorname{card {q.?msgs q}\not=\mathrm{ None }
    by (auto intro: card-mono)
with HO mfrv nxt show ?thesis by (auto simp: OTR-nextState-def)
qed
```

Therefore, once more than two thirds of the processes hold $v$ in their $x$ field, this will remain true forever.

```
lemma A3:
    assumes run:HORun OTR-M rho HOs
    and n:(2*N) div 3<card {q.x (rho n q)=v} (is ?twothird n)
    shows ?twothird ( }n+k
proof (induct k)
    from n show ?twothird ( }n+0)\mathrm{ by simp
next
    fix m
    assume m: ?twothird ( }n+m\mathrm{ )
    have }\forallq.x(rho (n+m)q)=v\longrightarrowx(rho (n+Suc m)q)=
    proof (rule+)
    fix q
    assume q: x ((rho (n+m)) q) =v
    let ?msgs = HOrcvdMsgs OTR-M (n+m) q(HOs (n+m) q) (rho (n+m))
    show x (rho (n+Suc m)q)=v
    proof (cases (2*N) div 3<card {q.?msgs q}\not=\mathrm{ None })
```

```
        case True
        from m have (2*N) div 3<card {q.x (rho (n+m)q)=v} by simp
        with True run show ?thesis by (auto elim: A2)
    next
        case False
        with run q show ?thesis by (auto dest: nextState-change)
    qed
    qed
    hence card {q.x (rho (n+m)q)=v}\leq card {q. x (rho (n+Sucm)q)=v}
    by (auto intro: card-mono)
    with m show ?twothird ( }n+\mathrm{ Suc m) by simp
qed
```

It now follows that once a process has decided on some value $v$, more than two thirds of all processes continue to hold $v$ in their $x$ field.

```
lemma A4:
    assumes run: HORun OTR-M rho HOs
    and dec: decide (rho n p)=Some v (is ?dec n)
    shows }\forallk.(2*N) div 3<\operatorname{card { q. x (rho (n+k)q)=v}
        (is }\forallk\mathrm{ . ?twothird ( }n+k)\mathrm{ )
using dec proof (induct n)
    - The base case is trivial since no process has decided
    assume ?dec 0 with run show }\forallk\mathrm{ . ?twothird ( }0+k
    by (simp add: HORun-eq HOinitConfig-eq OTR-HOMachine-def
                initState-def OTR-initState-def)
next
- For the inductive step, we assume that process p has decided on v.
fix m
assume ih: ?dec m\Longrightarrow\forallk. ?twothird ( }m+k\mathrm{ ) and m: ?dec (Suc m)
show }\forallk\mathrm{ . ?twothird ((Suc m) + k)
proof
    fix }
    have ?twothird (m + Suc k)
```

There are two cases to consider: if $p$ had already decided on $v$ before, the assertion follows from the induction hypothesis. Otherwise, the assertion follows from lemmas $A 1$ and $A 3$.

```
proof (cases ?dec m)
    case True with ih show ?thesis by blast
    next
            case False
            with run m have ?twothird m by (auto elim: A1)
            with run show ?thesis by (blast dest: A3)
    qed
    thus ?twothird ((Suc m) +k) by simp
    qed
qed
```

The Agreement property follows easily from lemma $A_{4}$ : if processes $p$ and
$q$ decide values $v$ and $w$, respectively, then more than two thirds of the processes must propose $v$ and more than two thirds must propose $w$. Because these two majorities must have an intersection, we must have $v=w$.
We first prove an "asymmetric" version of the agreement property before deriving the general agreement theorem.

```
lemma A5:
    assumes run:HORun OTR-M rho HOs
    and p: decide (rho n p)=Some v
    and p}\mp@subsup{p}{}{\prime}\mathrm{ : decide (rho (n+k) p})=\mathrm{ Some w
    shows v=w
proof -
    from run p
    have (2*N) div 3 < card {q. x (rho (n+k)q)=v} (is - <card ?V)
        by (blast dest: A4)
    moreover
    from run p'
    have (2*N) div 3 < card {q. x (rho ((n+k)+0) q) =w} (is - < card ?W)
        by (blast dest: A4)
    ultimately
    have N< card?V + card ?W by auto
    then obtain proc where proc \in?V \cap?W by (auto dest: majorities-intersect)
    thus ?thesis by auto
qed
```

theorem OTR-agreement:
assumes run:HORun OTR-M rho HOs
and $p$ : decide (rho $n$ p) = Some $v$
and $p^{\prime}$ : decide (rho m $p^{\prime}$ ) =Some $w$
shows $v=w$
proof (cases $n \leq m$ )
case True
then obtain $k$ where $m=n+k$ by (auto simp add: le-iff-add)
with run $p p^{\prime}$ show ?thesis by (auto elim: A5)
next
case False
hence $m \leq n$ by auto
then obtain $k$ where $n=m+k$ by (auto simp add: le-iff-add)
with run $p p^{\prime}$ have $w=v$ by (auto elim: A5)
thus ?thesis..
qed

### 5.6 Proof of Termination

We now show that every process must eventually decide.
The idea of the proof is to observe that the communication predicate guarantees the existence of two uniform rounds where every process hears from the same two-thirds majority of processes. The first such round serves to
ensure that all $x$ fields hold the same value, the second round copies that value into all decision fields.
Lemma A2 is instrumental in this proof.

```
theorem OTR-termination:
    assumes run: HORun OTR-M rho HOs
        and commG: HOcommGlobal OTR-M HOs
    shows \(\exists r v\). decide (rho r \(p\) ) \(=\) Some \(v\)
proof -
    from comm \(G\) obtain \(r 0 \Pi\) where
        \(p i: \forall q\). HOs r0 \(q=\Pi\) and pic: card \(\Pi>(2 * N)\) div 3
        by (auto simp: OTR-HOMachine-def OTR-commGlobal-def)
    let ?msgs q \(r=\) HOrcvdMsgs OTR-Mr \(q\) (HOs \(r q\) ) (rho r)
    from run \(p i\) have \(\forall p q\). ?msgs q r0 \(=\) ? msgs p r0
    by (auto simp: HORun-eq OTR-HOMachine-def HOrcvdMsgs-def OTR-sendMsg-def)
    then obtain \(\mu\) where \(\forall q\). ?msgs q r0 \(=\mu\) by auto
    moreover
    from pi pic have \(\forall p .(2 * N)\) div \(3<\) card \(\{q\). ? msgs p r0 \(q \neq\) None \(\}\)
    by (auto simp: HORun-eq HOnextConfig-eq HOrcvdMsgs-def)
    with run have \(\forall q . x(\) rho \((\) Suc r0) \(q)=\operatorname{Min}\{v . \operatorname{MFR}(? m s g s q r 0) v\}\)
    by (auto simp: HORun-eq HOnextConfig-eq OTR-HOMachine-def
                nextState-def OTR-nextState-def)
    ultimately
    have \(\forall q . x(\) rho \((S u c r 0) q)=\operatorname{Min}\{v . M F R \mu v\}\) by auto
    then obtain \(v\) where \(v: \forall q . x(\) rho (Suc r0) \(q)=v\) by auto
    have \(P: \forall k . \forall q . x(r h o(S u c r 0+k) q)=v\)
    proof
    fix \(k\)
    show \(\forall q \cdot x(r h o(S u c r 0+k) q)=v\)
    proof (induct \(k\) )
        from \(v\) show \(\forall q . x(\) rho \((S u c r 0+0) q)=v\) by \(\operatorname{simp}\)
    next
        fix \(k\)
        assume \(i h: \forall q\). \(x(\) rho \((S u c r 0+k) q)=v\)
        show \(\forall q . x(\) rho \((\) Suc r0 + Suc \(k) q)=v\)
        proof
            fix \(q\)
            show \(x\) (rho (Suc r0 + Suc \(k\) ) \(q\) ) \(=v\)
            proof (cases \((2 * N)\) div \(3<\) card \(\{p\). ?msgs \(q(\) Suc r0 \(+k) p \neq\) None \(\})\)
                    case True
                    have \(N>0\) by (rule finite-UNIV-card-ge-0) simp
                with \(i h\)
                    have \((2 * N)\) div \(3<\operatorname{card}\{p . x(\) rho \((\) Suc r0 \(+k) p)=v\}\) by auto
                with True run show ?thesis by (auto elim: A2)
            next
                case False
                with run ih show ?thesis by (auto dest: nextState-change)
```

```
            qed
        qed
        qed
    qed
    from commG obtain r0' \Pi'
        where r0':r0'\geqSuc r0
        and pi': \forallq. HOs r0' }q=\mp@subsup{\Pi}{}{\prime
        and pic': card }\mp@subsup{\Pi}{}{\prime}>(2*N) div 
    by (force simp: OTR-HOMachine-def OTR-commGlobal-def)
    from r\mp@subsup{O}{}{\prime}P\mathrm{ have v}\mp@subsup{v}{}{\prime}:\forallq.x (rho r0'q)=v by (auto simp:le-iff-add)
    from run
    have OTR-nextState r0' p (rho r0' p) (?msgs p r0') (rho (Suc r0') p)
    by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
    moreover
    from pi' pic' have (2*N) div 3<card {q. (?msgs p r0') q\not=None}
        by (auto simp: HOrcvdMsgs-def OTR-sendMsg-def)
    moreover
    from pi' pic' v' have TwoThirds (?msgs p r0') v
    by (simp add: TwoThirds-def HOrcvdMsgs-def OTR-HOMachine-def
                            OTR-sendMsg-def HOV-def)
    ultimately
    have decide (rho (Suc r0') p)= Some (\epsilonv. TwoThirds (?msgs p r0')v)
    by (auto simp: OTR-nextState-def)
    thus ?thesis by blast
qed
```


### 5.7 One-Third Rule Solves Consensus

Summing up, all (coarse-grained) runs of One-Third Rule for HO collections that satisfy the communication predicate satisfy the Consensus property.
theorem OTR-consensus:
assumes run: HORun OTR-M rho HOs and commG: HOcommGlobal OTR-M HOs
shows consensus ( $x \circ($ rho 0$)$ ) decide rho
using OTR-integrity[OF run] OTR-agreement[OF run] OTR-termination[OF run commG]
by (auto simp: consensus-def image-def)
By the reduction theorem, the correctness of the algorithm also follows for fine-grained runs of the algorithm. It would be much more tedious to establish this theorem directly.
theorem OTR-consensus-fg:
assumes run: fg-run OTR-M rho HOs HOs ( $\lambda$ r q. undefined)
and commG: HOcommGlobal OTR-M HOs
shows consensus ( $\lambda p . x$ (state (rho 0) p)) decide (state $\circ$ rho)
(is consensus ?inits - -)

```
proof (rule local-property-reduction[OF run consensus-is-local])
    fix crun
    assume crun: CSHORun OTR-M crun HOs HOs ( \(\lambda r\) q. undefined)
        and init: crun \(0=\) state (rho 0)
    from crun have HORun OTR-M crun HOs by (unfold HORun-def SHORun-def)
    from this commG have consensus ( \(x \circ\) (crun 0 )) decide crun by (rule OTR-consensus)
    with init show consensus? inits decide crun by (simp add: o-def)
qed
```

end
theory UvDefs
imports ../HOModel
begin

## 6 Verification of the UniformVoting Consensus Algorithm

Algorithm UniformVoting is presented in [7]. It can be considered as a deterministic version of Ben-Or's well-known probabilistic Consensus algorithm [2]. We formalize in Isabelle the correctness proof given in [7], using the framework of theory HOModel.

### 6.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic HO model.
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
abbreviation
$N \equiv$ card (UNIV ::Proc set) — number of processes
The algorithm proceeds in phases of 2 rounds each (we call steps the individual rounds that constitute a phase). The following utility functions compute the phase and step of a round, given the round number.
abbreviation $n$ Steps $\equiv 2$
definition phase where phase (r::nat) $\equiv r$ div nSteps
definition step where step $(r:: n a t) \equiv r$ mod nSteps
The following record models the local state of a process.

```
record 'val pstate \(=\)
    \(x\) :: 'val - current value held by process
```

```
vote :: 'val option - value the process voted for, if any
decide :: 'val option - value the process has decided on, if any
```

Possible messages sent during the execution of the algorithm, and characteristic predicates to distinguish types of messages.

```
datatype 'val msg =
    Val 'val
| ValVote 'val 'val option
| Null - dummy message in case nothing needs to be sent
```

definition isValVote where isValVote $m \equiv \exists z v . m=$ ValVote $z v$
definition isVal where isVal $m \equiv \exists v . m=\operatorname{Val} v$

Selector functions to retrieve components of messages. These functions have a meaningful result only when the message is of appropriate kind.

```
fun getvote where
    getvote (ValVote z v)=v
```


## fun getval where

```
    getval (ValVote \(z v)=z\)
```

$\mid \operatorname{getval}($ Val z) $=z$

The $x$ field of the initial state is unconstrained, all other fields are initialized appropriately.

```
definition UV-initState where
    UV-initState p st \equiv(vote st = None) ^(decide st = None)
```

We separately define the transition predicates and the send functions for each step and later combine them to define the overall next-state relation.
definition $m s g R c v d$ where - processes from which some message was received msgRcvd (msgs:: Proc $\rightharpoonup^{\prime}$ 'val msg) $=\{q$. msgs $q \neq$ None $\}$

## definition smallestValRcvd where

```
smallestValRcvd (msgs::Proc \rightharpoonup ('val::linorder) msg) \equiv
    Min {v.\existsq.msgs q = Some (Val v)}
```

In step 0 , each process sends its current $x$ value.
It updates its $x$ field to the smallest value it has received. If the process has received the same value $v$ from all processes from which it has heard, it updates its vote field to $v$.

## definition send0 where <br> $$
\text { send0 r p q st } \equiv \operatorname{Val}(x s t)
$$

## definition next0 where

 $(\exists v .(\forall q \in$ msgRcvd msgs. msgs $q=$ Some $($ Val $v))$$$
\begin{aligned}
& \left.\wedge s t^{\prime}=\text { st } 0 \text { vote }:=\text { Some } v, x:=\text { smallestValRcvd msgs } D\right) \\
& \vee \neg(\exists v . \forall q \in \text { msgRcvd msgs. msgs } q=\text { Some }(\text { Val } v)) \\
& \wedge s t^{\prime}=\text { st }(x:=\text { smallestValRcvd msgs })
\end{aligned}
$$

In step 1, each process sends its current $x$ and vote values.

```
definition send1 where
    send1 r p q st \(\equiv\) ValVote ( \(x\) st) (vote st)
definition valVoteRcvd where
    - processes from which values and votes were received
    valVoteRcvd (msgs :: Proc \(\rightharpoonup\) 'val msg) \(\equiv\)
    \(\{q . \exists z v\). msgs \(q=\) Some (ValVote \(z v)\}\)
definition smallestValNo VoteRcvd where
    smallestValNoVoteRcvd (msgs::Proc \(-(' v a l:: l i n o r d e r) ~ m s g) ~ \equiv\)
    Min \(\{v . \exists q\). msgs \(q=\) Some (ValVote \(v\) None) \(\}\)
definition someVoteRcvd where
    - set of processes from which some vote was received
    someVoteRcvd (msgs :: Proc - 'val msg) \(\equiv\)
    \(\{q \cdot q \in \operatorname{msgRcvd} m s g s \wedge\) isValVote \((\) the \((\) msgs \(q)) \wedge\) getvote \((\) the \((\) msgs \(q)) \neq\)
None \}
definition identicalVoteRcvd where
    identicalVoteRcvd (msgs :: Proc - 'val msg) \(v \equiv\)
    \(\forall q \in \operatorname{msgRcvd} \operatorname{msgs}\). isValVote \((\) the \((\) msgs \(q)) \wedge\) getvote \((\) the \((m s g s q))=\) Some
\(v\)
definition \(x\)-update where
\(x\)-update st msgs st \({ }^{\prime} \equiv\)
    \(\left(\exists q \in\right.\) someVoteRcvd msgs \(. x s t^{\prime}=\) the \((\) getvote \((\) the \((\) msgs \(\left.q)))\right)\)
\(\vee\) someVoteRcvd msgs \(=\{ \} \wedge x t^{\prime}=\) smallestValNoVoteRcvd msgs
definition dec-update where
    dec-update st msgs st' \(\equiv\)
        ( \(\exists\) v. identicalVoteRcvd msgs \(v \wedge\) decide \(s t^{\prime}=\) Some \(v\) )
    \(\vee \neg(\exists v\). identicalVoteRcvd msgs \(v) \wedge\) decide st' \(=\) decide st
definition next1 where
    next1 r p st msgs st \({ }^{\prime} \equiv\)
        \(x\)-update st msgs st'
        \(\wedge\) dec-update st msgs st \({ }^{\prime}\)
        \(\wedge\) vote st \({ }^{\prime}=\) None
```

The overall send function and next-state relation are simply obtained as the composition of the individual relations defined above.

```
definition UV-sendMsg where
    UV-sendMsg (r::nat) \equiv if step r = 0 then send0 r else send1 r
```


## definition $U V$-nextState where

UV-nextState $r \equiv$ if step $r=0$ then next0 $r$ else next1 $r$

### 6.2 Communication Predicate for UniformVoting

We now define the communication predicate for the UniformVoting algorithm to be correct.
The round-by-round predicate requires that for any two processes there is always one process heard by both of them. In other words, no "split rounds" occur during the execution of the algorithm [7]. Note that in particular, heard-of sets are never empty.

```
definition UV-commPerRd where
    UV-commPerRd HOrs }\equiv\forallpq.\existspq. pq\inHOrs p\capHOrs q
```

The global predicate requires the existence of a (space-) uniform round during which the heard-of sets of all processes are equal. (Observe that [7] requires infinitely many uniform rounds, but the correctness proof uses just one such round.)

```
definition UV-commGlobal where
    UV-commGlobal HOs \equiv\existsr.\forallpq. HOs r p = HOs r q
```


### 6.3 The Uniform Voting Heard-Of Machine

We now define the HO machine for Uniform Voting by assembling the algorithm definition and its communication predicate. Notice that the coordinator arguments for the initialization and transition functions are unused since Uniform Voting is not a coordinated algorithm.

```
definition UV-HOMachine where
    UV-HOMachine = 0
        CinitState = ( }\lambdap\mathrm{ st crd. UV-initState p st),
        sendMsg=UV-sendMsg,
        CnextState = (\lambdar p st msgs crd st'.UV-nextState r p st msgs st'})\mathrm{ ,
        HOcommPerRd = UV-commPerRd,
        HOcommGlobal =UV-commGlobal
    D
abbreviation
    UV-M \equiv(UV-HOMachine::(Proc,'val::linorder pstate,'val msg) HOMachine)
end
theory UvProof
imports UvDefs ../Reduction
begin
```


### 6.4 Preliminary Lemmas

At any round, given two processes $p$ and $q$, there is always some process which is heard by both of them, and from which $p$ and $q$ have received the same message.

```
lemma some-common-msg:
    assumes HOcommPerRd UV-M (HOs r)
    shows \exists pq. pq\inmsgRcvd (HOrcvdMsgs UV-M r p (HOs r p) (rho r))
                \wedge pq\inmsgRcvd (HOrcvdMsgs UV-M rq(HOs r q) (rho r))
    ^(HOrcvdMsgs UV-M r p (HOs r p) (rho r)) pq
        = (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) pq
```

    using assms
    by (auto simp: UV-HOMachine-def UV-commPerRd-def HOrcvdMsgs-def
        \(U V\)-sendMsg-def send0-def send1-def msgRcvd-def)
    When executing step 0 , the minimum received value is always well defined.

```
lemma minval-step 0 :
    assumes com: HOcommPerRd UV-M (HOs r) and s0: step \(r=0\)
    shows smallestValRcvd (HOrcvdMsgs UV-M r q (HOs r q) (rho r))
        \(\in\{v . \exists p .(\) HOrcvdMsgs UV-Mrq(HOs r q) (rho r)) \(p=\) Some (Val v) \(\}\)
        (is smallestValRcvd?msgs \(\in\) ?vals)
unfolding smallestValRcvd-def proof (rule Min-in)
    have ?vals \(\subseteq\) getval' ( \((\) the ○?msgs) ' \((\) HOs r q) )
        by (auto simp: HOrcvdMsgs-def image-def)
    thus finite? vals by (auto simp: finite-subset)
next
    from some-common-msg[of HOs, OF com]
    obtain \(p\) where \(p \in\) msgRcvd ?msgs by blast
    with s0 show ?vals \(\neq\{ \}\)
        by (auto simp: msgRcvd-def HOrcvdMsgs-def UV-HOMachine-def
                                    UV-sendMsg-def send0-def)
qed
```

When executing step 1 and no vote has been received, the minimum among values received in messages carrying no vote is well defined.
lemma minval-step1:
assumes com: HOcommPerRd UV-M (HOs r) and s1: step $r \neq 0$
and nov: someVoteRcvd (HOrcvdMsgs UV-M rq(HOs r q) (rho r)) $=\{ \}$
shows smallestValNoVoteRcvd (HOrcvdMsgs UV-Mrq(HOs rq) (rho r))
$\in\{v . \exists p .(H O r c v d M s g s U V-M r q(H O s r q)(r h o r)) p$
$=$ Some (ValVote v None) $\}$
(is smallestValNoVoteRcvd?msgs $\in$ ?vals)
unfolding smallestValNoVoteRcvd-def proof (rule Min-in)
have ?vals $\subseteq$ getval' ( $($ the ○?msgs) ' $(H O s$ r q) )
by (auto simp: HOrcvdMsgs-def image-def)
thus finite ?vals by (auto simp: finite-subset)
next
from some-common-msg[of HOs, OF com]
obtain $p$ where $p \in m s g R c v d$ ?msgs by blast
with s1 nov show ?vals $\neq\{ \}$
by (auto simp: msgRcvd-def HOrcvdMsgs-def someVoteRcvd-def isValVote-def $U V$-HOMachine-def $U V$-sendMsg-def send1-def)
qed
The vote field is reset every time a new phase begins.

```
lemma reset-vote:
    assumes run: HORun UV-M rho HOs and s0: step r'}=
    shows vote (rho r' p) = None
proof (cases r')
    assume r' = 0
    with run show ?thesis
    by (auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq
                        initState-def UV-initState-def)
next
    fix r
    assume sucr: r' = Suc r
    from run
    have nxt: nextState UV-M r p (rho r p)
                                    (HOrcvdMsgs UV-M r p (HOs r p) (rho r))
                                    (rho (Suc r) p)
        by (auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def)
    from s0 sucr have step r = 1 by (auto simp: step-def mod-Suc)
    with nxt sucr show ?thesis
    by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def next1-def)
qed
```

Processes only vote for the value they hold in their $x$ field.

```
lemma \(x\)-vote-eq:
    assumes run: HORun \(U V-M\) rho HOs
        and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and vote: vote (rho r \(p\) ) \(=\) Some \(v\)
    shows \(v=x\) (rho r \(p\) )
proof (cases r)
    case 0
    with run vote show ?thesis - no vote in initial state
    by (auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq
                        initState-def UV-initState-def)
next
    fix \(r^{\prime}\)
    assume \(r: r=\) Suc \(r^{\prime}\)
    let ?msgs \(=\) HOrcvdMsgs \(U V-M r^{\prime} p\left(H O s r^{\prime} p\right)\left(r h o r r^{\prime}\right)\)
    from run have nextState \(U V-M r^{\prime} p\left(\right.\) rho \(r^{\prime} p\) ) ?msgs (rho (Suc r') \(p\) )
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    with vote \(r\)
    have nxt0: next0 \(r^{\prime} p\left(r h o r^{\prime} p\right.\) ) ?msgs (rho \(r p\) ) and s0: step \(r^{\prime}=0\)
    by (auto simp: nextState-def UV-HOMachine-def UV-nextState-def next1-def)
    from run s0 have vote (rho \(\left.r^{\prime} p\right)=\) None by (rule reset-vote)
```

with vote $n x t 0$
have $i d v: \forall q \in m s g R c v d$ ?msgs. ?msgs $q=$ Some (Val v)
and $x$ : $x$ (rho r $p$ ) $=$ smallestValRcvd ?msgs
by (auto simp: next0-def)
moreover
from com obtain $q$ where $q \in m s g R c v d ? m s g s$
by (force dest: some-common-msg)
with $i d v$ have $\{x . \exists q q$. ?msgs $q q=$ Some $($ Val $x)\}=\{v\}$
by (auto simp: msgRcvd-def)
hence smallestValRcvd?msgs $=v$
by (auto simp: smallestValRcvd-def)
ultimately
show?thesis by simp
qed

### 6.5 Proof of Irrevocability, Agreement and Integrity

A decision can only be taken in the second round of a phase.

```
lemma decide-step:
    assumes run: HORun UV-M rho HOs
        and decide: decide (rho (Suc r) \(p\) ) \(\neq\) decide (rho r \(p\) )
    shows step \(r=1\)
proof -
    let ?msgs \(=\) HOrcvdMsgs \(U V-M\) r \(p(H O s r p)(r h o r)\)
    from run have nextState \(U V-M\) r \(p\) (rho r p) ?msgs (rho (Suc r) p)
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    with decide show ?thesis
        by (auto simp: nextState-def UV-HOMachine-def UV-nextState-def
                next0-def step-def)
```

qed

No process ever decides None.
lemma decide-nonnull:
assumes run: HORun $U V-M$ rho HOs
and decide: decide (rho (Suc r) p) $\neq$ decide (rho r p)
shows decide (rho (Suc r) p) $\neq$ None
proof -
let ?msgs $=$ HOrcvdMsgs $U V-M r p(H O s r p)(r h o r)$
from assms have s1: step $r=1$ by (rule decide-step)
with run have next1 r p (rho r p) ?msgs (rho (Suc r) p)
by (auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq
nextState-def UV-nextState-def)
with decide show ?thesis
by (auto simp: next1-def dec-update-def)
qed
If some process $p$ votes for $v$ at some round $r$, then any message that $p$ received in $r$ was holding $v$ as a value.

```
lemma msgs-unanimity:
    assumes run: HORun UV-M rho HOs
        and vote: vote (rho (Suc r) p)=Some v
        and q:q\in msgRcvd (HOrcvdMsgs UV-M r p (HOs r p) (rho r))
            (is - \in msgRcvd ?msgs)
    shows getval (the (?msgs q)) =v
proof -
    have s0: step r=0
    proof (rule ccontr)
        assume step r\not=0
        hence step (Suc r)=0 by (simp add: step-def mod-Suc)
        with run vote show False by (auto simp: reset-vote)
    qed
    with run have novote: vote (rho r p)=None by (auto simp: reset-vote)
    from run have nextState UV-M r p (rho r p) ?msgs (rho (Suc r) p)
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    with s0 have nxt: next0 r p (rho r p) ?msgs (rho (Suc r) p)
        by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
    with novote vote q show ?thesis by (auto simp: next0-def)
qed
```

Any two processes can only vote for the same value.

```
lemma vote-agreement:
    assumes run: HORun UV-M rho HOs
        and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and \(p\) : vote (rho r p) = Some \(v\)
        and \(q\) : vote (rhor \(q\) ) = Some \(w\)
    shows \(v=w\)
proof (cases r)
    case 0
    with run \(p\) show ?thesis - no votes in initial state
        by (auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq
                                initState-def UV-initState-def)
next
    fix \(r^{\prime}\)
    assume \(r: r=\) Suc \(r^{\prime}\)
    let ?msgs \(p=\) HOrcvdMsgs UV-M \(r^{\prime} p\left(H O s r^{\prime} p\right)\left(r h o r r^{\prime}\right)\)
    from com obtain \(p q\)
        where ?msgs \(p p q=\) ? msgs \(q p q\)
            and \(s m p: p q \in m s g R c v d\) (?msgs \(p\) ) and \(s m q: p q \in \operatorname{msgRcvd}(? m s g s q\) )
            by (force dest: some-common-msg)
    moreover
    from run \(p\) smp \(r\) have getval (the (?msgs \(p p q)\) ) \(=v\)
        by (simp add: msgs-unanimity)
    moreover
    from run \(q\) smq \(r\) have getval (the (?msgs \(q\) pq)) \(=w\)
        by (simp add: msgs-unanimity)
    ultimately
    show ?thesis by simp
```


## qed

If a process decides value $v$ then all processes must have $v$ in their $x$ fields.

```
lemma decide-equals-x:
    assumes run: HORun UV-M rho HOs
        and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and decide: decide (rho (Suc r) p) \(\neq\) decide (rho r p)
        and decval: decide (rho (Suc r) p) \(=\) Some \(v\)
    shows \(x(\) rho \((\) Suc \(r) q)=v\)
proof -
    let ?msgs \(p^{\prime}=\) HOrcvdMsgs UV-M r \(p^{\prime}\left(\right.\) HOs r \(\left.p^{\prime}\right)\left(\begin{array}{l}\text { rho } r)\end{array}\right.\)
    from run decide have s1: step \(r=1\) by (rule decide-step)
    from run have nextState \(U V-M r p(r h o r p)(? m s g s p)(r h o(S u c r) p)\)
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    with \(s 1\) have nxtp: next1 r p (rho r p) (?msgs p) (rho (Suc r) p)
        by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
    from run have nextState \(U V-M r q\) (rhorq) (?msgs q) (rho (Suc r) q)
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    with \(s 1\) have nxtq: next1 \(r q(\) rho \(r q)(? m s g s q)(r h o(S u c ~ r) ~ q) ~\)
        by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
    from com obtain \(p q\) where
        \(p q: p q \in \operatorname{msgRcvd}(\) ?msgs \(p) p q \in \operatorname{msgRcvd}(? m s g s q)\)
            (?msgs \(p\) ) \(p q=(\) ? \(m\) sgs \(q) p q\)
        by (force dest: some-common-msg)
    with decide decval nxtp
    have vote: isValVote (the (?msgs p pq))
                getvote (the (?msgs p pq)) = Some \(v\)
    by (auto simp: next1-def dec-update-def identicalVoteRcvd-def)
    with \(n x t q p q\) obtain \(q^{\prime}\) where
        \(q^{\prime}: q^{\prime} \in\) someVoteRcvd (?msgs \(q\) )
            \(x(\) rho \((\) Suc \(r) q)=\) the \(\left(\right.\) getvote \(\left(\right.\) the \(\left(\right.\) ?msgs \(\left.\left.\left.q q^{\prime}\right)\right)\right)\)
        by (auto simp: next1-def \(x\)-update-def someVoteRcvd-def)
    with \(s 1\) pq vote show ?thesis
    by (auto simp: HOrcvdMsgs-def UV-HOMachine-def UV-sendMsg-def send1-def
                someVoteRcvd-def msgRcvd-def vote-agreement[OF run com])
qed
```

If at some point all processes hold value $v$ in their $x$ fields, then this will still be the case at the next step.

```
lemma same-x-stable:
    assumes run: HORun \(U V-M\) rho \(H O s\)
        and comm: \(\forall r\). HOcommPerRd \(U V-M(H O s r)\)
        and \(x: \forall p . x(\) rho \(r p)=v\)
    shows \(x(\) rho \((\) Suc r) \(q)=v\)
proof -
    let ?msgs \(=\) HOrcvdMsgs UV-Mrq(HOs rq)(rhor)
    from comm obtain \(p\) where \(p: p \in m s g R c v d\) ?msgs
        by (force dest: some-common-msg)
```

```
from run have nextState UV-M r q (rho r q) ?msgs (rho (Suc r) q)
    by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
hence next0 r q (rho r q) ?msgs (rho (Suc r) q) ^ step r = 0
    \checkmark ~ n e x t 1 ~ r ~ q ~ ( r h o ~ r ~ q ) ~ ? m s g s ~ ( r h o ~ ( S u c ~ r ) ~ q ) ~ \wedge ~ s t e p ~ r ~ f = 0
    (is ?nxt0 \vee ?nxt1)
    by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
thus ?thesis
proof
    assume nxt0: ?nxt0
    hence x (rho (Suc r) q) = smallestValRcvd ?msgs
        by (auto simp: next0-def)
    moreover
    from nxt0 x have }\forallp\inmsgRcvd ?msgs. ?msgs p=Some (Val v
        by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                msgRcvd-def send0-def)
    from this p have {x.\existsp. ?msgs p = Some (Val x)}={v}
    by (auto simp: msgRcvd-def)
    hence smallestValRcvd ?msgs =v
        by (auto simp: smallestValRcvd-def)
    ultimately
    show ?thesis by simp
next
    assume nxt1: ?nxt1
    show ?thesis
    proof (cases someVoteRcvd ?msgs = {})
    case True
    with nxt1 have x (rho (Suc r) q) = smallestValNoVoteRcvd ?msgs
        by (auto simp: next1-def x-update-def)
    moreover
    from nxt1 x True
    have }\forallp\inmsgRcvd ?msgs. ?msgs p=Some (ValVote v None)
        by (auto simp:UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                                msgRcvd-def send1-def someVoteRcvd-def isValVote-def)
    from this p have {x.\existsp.?msgs p=Some (ValVote x None)}={v}
        by (auto simp: msgRcvd-def)
    hence smallestValNoVoteRcvd?msgs =v
            by (auto simp: smallestValNoVoteRcvd-def)
    ultimately show ?thesis by simp
next
    case False
    with nxt1 obtain p}\mp@subsup{p}{}{\prime}\mp@subsup{v}{}{\prime}\mathrm{ where
        p
            getvote (the (?msgs p')) = Some v'x (rho (Suc r) q) = v'
        by (auto simp: someVoteRcvd-def next1-def x-update-def)
        with nxt1 have x (rho (Suc r) q) =x (rho r p')
            by (auto simp:UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                        msgRcvd-def send1-def isValVote-def
                        x-vote-eq[OF run comm])
    with }x\mathrm{ show ?thesis by auto
```

```
        qed
    qed
qed
```

Combining the last two lemmas, it follows that as soon as some process decides value $v$, all processes hold $v$ in their $x$ fields.

```
lemma safety-argument:
    assumes run: HORun \(U V-M\) rho HOs
        and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and decide: decide (rho (Suc r) p) \(\neq\) decide (rho r p)
        and decval: decide (rho (Suc r) p) \(=\) Some \(v\)
    shows \(x(\) rho \((\) Suc \(r+k) q)=v\)
proof (induct \(k\) arbitrary: \(q\) )
    fix \(q\)
    from decide-equals-x[OF assms] show \(x(\) rho \((S u c r+0) q)=v\) by simp
next
    fix \(k q\)
    assume \(\wedge q . x(\) rho \((S u c r+k) q)=v\)
    with run com show \(x\) (rho (Suc \(r+\) Suc k) q) \(=v\)
        by (auto dest: same-x-stable)
qed
```

Any process that holds a non-null decision value has made a decision sometime in the past.

```
lemma decided-then-past-decision:
    assumes run: HORun UV-M rho HOs
            and dec:decide (rho n p) = Some v
    shows }\existsm<n\mathrm{ . decide (rho (Suc m) p) # decide (rho m p)
            ^decide (rho (Suc m) p)=Some v
proof -
    let ?dec k= decide (rho k p)
    have ( }\forall\textrm{m}<n\mathrm{ . ?dec (Suc m) }\not=\mathrm{ ? dec m }\longrightarrow\mathrm{ ?dec (Suc m) }=\mathrm{ Some v)
                \longrightarrow \text { ?dec n} = \text { Some v}
        (is ?P n is ?A n\longrightarrow-)
    proof (induct n)
        from run show ?P 0
            by (auto simp: HORun-eq UV-HOMachine-def HOinitConfig-eq
                                    initState-def UV-initState-def)
    next
            fix n
            assume ih: ?P n thus ?P (Suc n) by force
    qed
    with dec show ?thesis by auto
qed
```

We can now prove the safety properties of the algorithm, and start with proving Integrity.
lemma $x$-values-initial:

```
    assumes run:HORun UV-M rho HOs
    and com:\forallr.HOcommPerRd UV-M (HOs r)
    shows }\exists\textrm{q}.x(\mathrm{ rho r p) =x (rho 0 q)
proof (induct r arbitrary: p)
    fix p
    show \existsq. x (rho 0 p)=x (rho 0 q) by auto
next
    fix r p
    assume ih: \bigwedgep'. \existsq. x (rhor p ')=x (rho 0 q)
    let ?msgs = HOrcvdMsgs UV-M r p (HOs r p)(rho r)
    from run have nextState UV-M r p (rho r p) ?msgs (rho (Suc r) p)
        by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
    hence next0 r p (rho r p) ?msgs (rho (Suc r) p) ^ step r =0
                \vee ~ n e x t 1 ~ r ~ p ~ ( r h o ~ r ~ p ) ~ ? m s g s ~ ( r h o ~ ( S u c ~ r ) ~ p ) ~ \wedge ~ s t e p ~ r ~ f = 0
    (is ?nxt0 \vee ?nxt1)
    by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
thus \existsq. x (rho (Suc r) p)=x(rho 0 q)
proof
    assume nxt0: ?nxt0
    hence x (rho (Suc r) p)= smallestValRcvd ?msgs
        by (auto simp: next0-def)
    also with com nxt0 have \ldots.\in{v.\existsq. ?msgs q=Some (Val v)}
        by (intro minval-step0) auto
    also with nxt0 have ... ={x (rho r q)|q.q\inmsgRcvd ?msgs }
        by (auto simp:UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                        msgRcvd-def send0-def)
    finally obtain q}\mathrm{ where x (rho (Suc r) p)=x (rhorq) by auto
    with ih show ?thesis by auto
next
    assume nxt1: ?nxt1
    show ?thesis
    proof (cases someVoteRcvd ?msgs = {})
        case True
        with nxt1 have x (rho (Suc r) p)= smallestValNoVoteRcvd ?msgs
            by (auto simp: next1-def x-update-def)
        also with com nxt1 True
    have ...\in{v.\existsq. ?msgs q=Some (ValVote v None)}
            by (intro minval-step1) auto
    also with nxt1 True
    have ...={x(rho r q)|q.q\inmsgRcvd ?msgs }
        by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                someVoteRcvd-def isValVote-def msgRcvd-def send1-def)
    finally obtain q}\mathrm{ where x (rho (Suc r) p)=x (rhor q) by auto
    with ih show ?thesis by auto
    next
    case False
    with nxt1 obtain q where
        q\in someVoteRcvd ?msgs
        x (rho (Suc r) p) = the (getvote (the (?msgs q)))
```

```
            by (auto simp: next1-def x-update-def)
        with nxt1 have vote (rho r q) = Some (x (rho (Suc r) p))
            by (auto simp:UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                someVoteRcvd-def is ValVote-def msgRcvd-def send1-def)
        with run com have x (rho (Suc r) p)=x(rho r q)
            by (rule x-vote-eq)
        with ih show ?thesis by auto
    qed
    qed
qed
theorem uv-integrity:
    assumes run: HORun UV-M rho HOs
        and com: }\forallr.HOcommPerRd UV-M(HOs r
        and dec: decide (rho r p) = Some v
    shows \existsq.v=x (rho 0 q)
proof -
    from run dec obtain k where
        decide (rho (Suc k) p)\not= decide (rho k p)
        decide (rho (Suc k) p)=Some v
        by (auto dest: decided-then-past-decision)
    with run com have x (rho (Suc k) p)=v
        by (rule decide-equals-x)
    with run com show ?thesis
        by (auto dest: x-values-initial)
qed
We now turn to Agreement.
lemma two-decisions-agree:
assumes run: HORun UV-M rho HOs
and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
and decidep: decide (rho (Suc r) p) \(=\) decide (rho r \(p\) )
and decvalp: decide (rho (Suc r) p) = Some \(v\)
and decideq: decide (rho \((\operatorname{Suc}(r+k)) q) \neq\) decide \((r h o(r+k) q)\)
and decvalq: decide (rho \((\operatorname{Suc}(r+k)) q\) ) \(=\) Some \(w\)
shows \(v=w\)
proof -
from run com decidep decvalp have \(x(\) rho \((\) Suc \(r+k) q)=v\)
by (rule safety-argument)
moreover
from run com decideq decvalq have \(x(\) rho \((S u c(r+k)) q)=w\)
by (rule decide-equals- \(x\) )
ultimately
show?thesis by simp
qed
theorem uv-agreement:
assumes run: HORun \(U V-M\) rho HOs
and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
```

and $p$ : decide (rho $m p$ ) $=$ Some $v$ and $q$ : decide (rho $n q$ ) $=$ Some $w$
shows $v=w$
proof -
from run $p$ obtain $k$ where
$k$ : decide (rho (Suc k) p) $\neq$ decide (rho $k p$ ) decide (rho (Suc k) p) = Some v
by (auto dest: decided-then-past-decision)
from run $q$ obtain $l$ where
$l$ : decide (rho $($ Suc $l) q) \neq$ decide (rho l q) decide $($ rho $($ Suc l) $)$ ) $=$ Some $w$
by (auto dest: decided-then-past-decision)
show ?thesis
proof (cases $k \leq l$ )
case True
then obtain $m$ where $m: l=k+m$ by (auto simp: le-iff-add)
from run com $k l m$ show ?thesis by (blast dest: two-decisions-agree)
next
case False
hence $l \leq k$ by simp
then obtain $m$ where $m: k=l+m$ by (auto simp: le-iff-add)
from run com $k l m$ show ?thesis by (blast dest: two-decisions-agree)
qed
qed
Irrevocability is a consequence of Agreement and the fact that no process can decide None.

```
theorem uv-irrevocability:
    assumes run: HORun \(U V-M\) rho HOs
        and com: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and \(p\) : decide (rho \(m p\) ) = Some \(v\)
    shows decide (rho \((m+n) p)=\) Some \(v\)
proof (induct \(n\) )
    from \(p\) show decide (rho \((m+0) p)=\) Some \(v\) by simp
next
    fix \(n\)
    assume ih: decide (rho \((m+n) p)=\) Some \(v\)
    show decide (rho \((m+\) Suc \(n) p\) ) \(=\) Some \(v\)
    proof (rule classical)
        assume \(\neg\) ?thesis
        with run ih obtain \(w\) where \(w\) : decide (rho \((m+\) Suc n) \(p\) ) \(=\) Some \(w\)
            by (auto dest!: decide-nonnull)
        with \(p\) have \(w=v\) by (auto simp: uv-agreement[OF run com])
        with \(w\) show ?thesis by simp
    qed
qed
```


### 6.6 Proof of Termination

Two processes having the same Heard-Of set at some round will hold the same value in their $x$ variable at the next round.
lemma hoeq-xeq:
assumes run: HORun UV-M rho HOs
and com: $\forall r$. HOcommPerRd UV-M (HOs r)
and hoeq: $H O$ s $r p=H O s r q$
shows $x($ rho $($ Suc r) $)=x($ rho $($ Suc r) $) ~ q) ~$
proof -
let ?msgs $p=$ HOrcvadMsgs UV-Mrp(HOs rp) (rho r)
from hoeq have msgeq: ?msgs $p=$ ? msgs $q$
by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def send0-def send1-def)
show ?thesis
proof (cases step $r=0$ )
case True
with run

by (force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def UV-nextState-def)
hence ?nxt0 $p$ ?nxt0 $q$ by auto
with msgeq show ?thesis by (auto simp: next0-def)
next
assume stp: step $r \neq 0$
with run
have $\forall p$. next1 rp(rhorp)(?msgs p)(rho (Suc r)p) (is $\forall p$. ?nxt1 p)
by (force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def UV-nextState-def)
hence $x$-update (rho r p) (?msgs p) (rho (Suc r) p) $x$-update (rho r q) (?msgs q) (rho (Suc r) q)
by (auto simp: next1-def)
with $m s g e q$ have
$x^{\prime}: x$-update (rho r $p$ ) (?msgs p) (rho (Suc r) p)
$x$-update (rhorq) (?msgs p)(rho (Suc r) q)
by auto
show ?thesis
proof (cases someVoteRcvd (?msgs $p$ ) $=\{ \}$ )
case True
with $x^{\prime}$ show ?thesis
by (auto simp: $x$-update-def)
next
case False
with $x^{\prime} s t p$ obtain $q p q q$ where
vote (rho r qp) $=$ Some $(x$ (rho (Suc r) p)) and vote $($ rho $r q q)=$ Some $(x($ rho $($ Suc r) q) $)$
by (force simp: UV-HOMachine-def HOrcudMsgs-def UV-sendMsg-def $x$-update-def someVoteRcvd-def is ValVote-def

```
                    msgRcvd-def send1-def)
        with run com show ?thesis by (rule vote-agreement)
    qed
    qed
qed
```

We now prove that Uniform Voting terminates.

```
theorem uv-termination:
    assumes run: HORun \(U V\)-M rho HOs
        and commR: \(\forall r\). HOcommPerRd UV-M (HOs r)
        and commG: HOcommGlobal UV-M HOs
    shows \(\exists r v\). decide (rho r \(p\) ) \(=\) Some \(v\)
proof -
```

First obtain a round where all $x$ values agree.
from commG obtain r0 where $r 0: \forall q$. HOs r0 $q=$ HOs r0 p
by (force simp: UV-HOMachine-def UV-commGlobal-def)
let $? v=x($ rho (Suc r0) $p$ )
from run commR r0 have $x s: \forall q \cdot x(r h o(S u c r 0) q)=? v$
by (auto dest: hoeq-xeq)
Now obtain a round where all votes agree.

```
define \(r^{\prime}\) where \(r^{\prime}=(\) if step \((\) Suc \(r 0)=0\) then Suc r0 else Suc (Suc r0))
have stp \({ }^{\prime}\) : step \(r^{\prime}=0\)
    by (simp add: \(r^{\prime}\)-def step-def mod-Suc)
have \(x^{\prime}: \forall q . x\left(\right.\) rho \(\left.r^{\prime} q\right)=? v\)
proof (auto simp: \(r^{\prime}\)-def)
    fix \(q\)
    from \(x s\) show \(x(\) rho (Suc r0) \(q)=? v\)..
next
    fix \(q\)
    from run commR xs show \(x(\) rho \((\) Suc (Suc ro)) \(q)=? v\)
        by (rule same-x-stable)
qed
have vote': \(\forall q\). vote (rho (Suc \(\left.r^{\prime}\right) q\) ) = Some ?v
proof
    fix \(q\)
    let ? msgs \(=\) HOrcvdMsgs \(U V-M r^{\prime} q\left(H O s r^{\prime} q\right)\left(r h o r r^{\prime}\right)\)
    from run stp' have next0 \(r^{\prime} q\) (rho \(r^{\prime} q\) ) ?msgs (rho (Suc \(\left.r^{\prime}\right) ~ q\) )
        by (force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq
                                    nextState-def UV-nextState-def)
    moreover
    from stp \({ }^{\prime} x^{\prime}\) have \(\forall q^{\prime} \in\) msgRcvd ?msgs. ?msgs \(q^{\prime}=\) Some (Val ?v)
        by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
                        send0-def msgRcvd-def)
    moreover
    from commR have msgRcvd ? msgs \(\neq\{ \}\)
    by (force dest: some-common-msg)
    ultimately
```

```
    show vote (rho (Suc r') q) = Some ?v
    by (auto simp: next0-def)
qed
```

At the subsequent round, process $p$ will decide.

```
let ? r ' ' = Suc r'
let ?msg\mp@subsup{s}{}{\prime}= HOrcvdMsgs UV-M ?r'r p (HOs ?r'r p)(rho ?r')
from stp' have stp": step ?r '" = 1
    by (simp add: step-def mod-Suc)
with run have next1 ?r'r p (rho ?r'\prime p) ?msgs'(rho (Suc ?r') p)
    by (auto simp:UV-HOMachine-def HORun-eq HOnextConfig-eq
                        nextState-def UV-nextState-def)
moreover
from stp" vote' have identicalVoteRcvd ?msgs' ?v
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
        send1-def identicalVoteRcvd-def isValVote-def msgRcvd-def)
moreover
from commR have msgRcvd ?msgs' }\not={
    by (force dest: some-common-msg)
ultimately
have decide (rho (Suc ?r') p)= Some ?v
    by (force simp: next1-def dec-update-def identicalVoteRcvd-def
                msgRcvd-def isValVote-def)
    thus ?thesis by blast
qed
```


### 6.7 Uniform Voting Solves Consensus

Summing up, all (coarse-grained) runs of UniformVoting for HO collections that satisfy the communication predicate satisfy the Consensus property.

```
theorem uv-consensus:
    assumes run: HORun UV-M rho HOs
        and commR: \forallr. HOcommPerRd UV-M (HOs r)
        and commG: HOcommGlobal UV-M HOs
    shows consensus ( }x\circ(\mathrm{ rho 0)) decide rho
    using assms unfolding consensus-def image-def
    by (auto elim: uv-integrity uv-agreement uv-termination)
```

By the reduction theorem, the correctness of the algorithm carries over to the fine-grained model of runs.
theorem $u v$-consensus- $f g$ :
assumes run: fg-run $U V-M$ rho HOs HOs ( $\lambda r$ q. undefined)
and commR: $\forall r$. HOcommPerRd UV-M (HOs r)
and commG: HOcommGlobal UV-M HOs
shows consensus ( $\lambda p . x$ (state (rho 0) p)) decide (state $\circ$ rho)
(is consensus ?inits - -)

```
proof (rule local-property-reduction[OF run consensus-is-local])
    fix crun
    assume crun: CSHORun UV-M crun HOs HOs (\lambdar q. undefined)
        and init: crun 0 = state (rho 0)
    from crun have HORun UV-M crun HOs
        by (unfold HORun-def SHORun-def)
    from this commR commG have consensus (x\circ(crun 0)) decide crun
        by (rule uv-consensus)
    with init show consensus ?inits decide crun
        by (simp add: o-def)
qed
end
theory LastVotingDefs
imports ../HOModel
begin
```


## 7 Verification of the LastVoting Consensus Algorithm

The LastVoting algorithm can be considered as a representation of Lamport's Paxos consensus algorithm [11] in the Heard-Of model. It is a coordinated algorithm designed to tolerate benign failures. Following [7], we formalize its proof of correctness in Isabelle, using the framework of theory HOModel.

### 7.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic CHO model.
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
abbreviation
$N \equiv \operatorname{card}(U N I V::$ Proc set) - number of processes
The algorithm proceeds in phases of 4 rounds each (we call steps the individual rounds that constitute a phase). The following utility functions compute the phase and step of a round, given the round number.
definition phase where phase ( $r::$ nat $) \equiv r$ div 4
definition step where step $(r:: n a t) \equiv r \bmod 4$
lemma phase-zero [simp]: phase $0=0$
by (simp add: phase-def)
lemma step-zero [simp]: step $0=0$
by (simp add: step-def)
lemma phase-step: (phase $r * 4)+$ step $r=r$
by (auto simp add: phase-def step-def)
The following record models the local state of a process.

```
record 'val pstate =
    x :: 'val - current value held by process
    vote :: 'val option - value the process voted for, if any
    commt :: bool - did the process commit to the vote?
    ready :: bool - for coordinators: did the round finish successfully?
    timestamp :: nat - time stamp of current value
    decide :: 'val option - value the process has decided on, if any
    coord }\Phi\mathrm{ :: Proc - coordinator for current phase
```

Possible messages sent during the execution of the algorithm.

```
datatype 'val msg =
    ValStamp 'val nat
| Vote 'val
| Ack
| Null - dummy message in case nothing needs to be sent
```

Characteristic predicates on messages.

```
definition isValStamp where isValStamp m\equiv\existsvts. m= ValStamp vts
```

definition isVote where isVote $m \equiv \exists v . m=$ Vote $v$
definition isAck where isAck $m \equiv m=A c k$

Selector functions to retrieve components of messages. These functions have a meaningful result only when the message is of an appropriate kind.

```
fun val where
    val (ValStamp v ts) =v
| val (Vote v)=v
fun stamp where
    stamp (ValStamp v ts) = ts
```

The $x$ field of the initial state is unconstrained, all other fields are initialized appropriately.

```
definition LV-initState where
    LV-initState p st crd \equiv
        vote st = None
    \neg(commt st)
    \neg(ready st)
```

```
\ timestamp st = 0
decide st = None
\wedge \operatorname { c o o r d } \Phi \text { st = crd}
```

We separately define the transition predicates and the send functions for each step and later combine them to define the overall next-state relation.

```
definition valStampsRcvd where
valStampsRcvd (msgs :: Proc - 'val msg) \(\equiv\)
\(\{q . \exists v\) ts. msgs \(q=\) Some (ValStamp \(v t s)\}\)
```

definition highestStampRcvd where
highestStampRcvd msgs $\equiv$
$\operatorname{Max}\left\{t s . \exists q v .\left(m s g s::\right.\right.$ Proc $\rightharpoonup^{\prime}$ 'val msg) $q=$ Some (ValStamp $\left.\left.v t s\right)\right\}$

In step 0 , each process sends its current $x$ and timestamp values to its coordinator.
A process that considers itself to be a coordinator updates its vote field if it has received messages from a majority of processes. It then sets its commt field to true.

```
definition send0 where
```

```
send0 r p q st \(\equiv\)
```

send0 r p q st $\equiv$
if $q=\operatorname{coord} \Phi$ st then ValStamp ( $x$ st) (timestamp st) else Null

```
    if \(q=\operatorname{coord} \Phi\) st then ValStamp ( \(x\) st) (timestamp st) else Null
```

definition next0 where
next0 $r$ p st msgs crd st' $\equiv$
if $p=\operatorname{coord} \Phi$ st $\wedge$ card (valStampsRcvd msgs) $>N$ div 2
then $(\exists p$ v.msgs $p=$ Some (ValStamp $v$ (highestStampRcvd msgs))
$\wedge s t^{\prime}=$ st ( vote $:=$ Some $v$, commt $:=$ True ) )
else $s t^{\prime}=s t$

In step 1, coordinators that have committed send their vote to all processes. Processes update their $x$ and timestamp fields if they have received a vote from their coordinator.

```
definition send1 where
    send1 r p q st \equiv
    if p=\operatorname{coord}\Phi\mathrm{ st }\wedge commt st then Vote (the (vote st)) else Null
definition next1 where
    next1 r p st msgs crd st' \equiv
    if msgs (coord }\Phi\mathrm{ st) }==\mathrm{ None ^ isVote (the (msgs (coord }\Phi\mathrm{ st)))
    then st' = st \ x := val (the (msgs (coord \Phi st))), timestamp := Suc(phaser)|)
    else st'}=s
```

In step 2, processes that have current timestamps send an acknowledgement to their coordinator.
A coordinator sets its ready field to true if it receives a majority of acknowledgements.

```
definition send2 where
    send2 \(r p q\) st \(\equiv\)
    if timestamp st \(=\operatorname{Suc}(\) phase \(r) \wedge q=\operatorname{coord} \Phi\) st then Ack else Null
- processes from which an acknowledgement was received
definition acksRcvd where
acksRcvd (msgs :: Proc - 'val msg) \(\equiv\)
    \(\{q . \operatorname{msgs} q \neq\) None \(\wedge\) isAck (the (msgs \(q))\}\)
definition next2 where
    next2 r \(p\) st msgs crd \(s t^{\prime} \equiv\)
    if \(p=\operatorname{coord} \Phi\) st \(\wedge\) card (acksRcvd msgs) \(>N\) div 2
    then \(s t^{\prime}=\) st ( ready \(:=\) True )
    else \(s t^{\prime}=s t\)
```

In step 3, coordinators that are ready send their vote to all processes.
Processes that received a vote from their coordinator decide on that value. Coordinators reset their ready and commt fields to false. All processes reset the coordinators as indicated by the parameter of the operator.

```
definition send3 where
    send3 r p q st \(\equiv\)
    if \(p=\operatorname{coord} \Phi\) st \(\wedge\) ready st then Vote (the (vote st)) else Null
definition next3 where
next3 r p st msgs crd st \({ }^{\prime} \equiv\)
    (if msgs \((\operatorname{coord} \Phi\) st) \(\neq\) None \(\wedge\) isVote \((\) the \((\) msgs \((\operatorname{coord} \Phi\) st)))
        then decide st \({ }^{\prime}=\) Some (val (the (msgs \((\operatorname{coord} \Phi\) st))))
        else decide st' \(=\) decide \(s t)\)
    \(\wedge\) (if \(p=\operatorname{coord} \Phi s t\)
        then \(\neg\left(\right.\) ready st') \(\wedge \neg\left(\right.\) commt st \(\left.t^{\prime}\right)\)
        else ready st \({ }^{\prime}=\) ready st \(\wedge\) commt st \({ }^{\prime}=\) commt st)
    \(\wedge x s t^{\prime}=x s t\)
    \(\wedge\) vote st \({ }^{\prime}=\) vote st
    \(\wedge\) timestamp st \({ }^{\prime}=\) timestamp st
    \(\wedge \operatorname{coord} \Phi\) st \(t^{\prime}=c r d\)
```

The overall send function and next-state relation are simply obtained as the composition of the individual relations defined above.

```
definition LV-sendMsg :: nat \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) 'val msg where
    \(L V\)-sendMsg ( \(r::\) nat \() \equiv\)
    if step \(r=0\) then send0 \(r\)
    else if step \(r=1\) then send1 \(r\)
    else if step \(r=2\) then send2 \(r\)
    else send3 \(r\)
```


## definition

```
LV-nextState :: nat \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) (Proc \(\rightharpoonup^{\prime}\) 'val msg)
```

$$
\Rightarrow \text { Proc } \Rightarrow \text { 'val pstate } \Rightarrow \text { bool }
$$

where
LV-nextState $r \equiv$
if step $r=0$ then next0 $r$
else if step $r=1$ then next $1 r$
else if step $r=2$ then next2 $r$
else next3 $r$

### 7.2 Communication Predicate for LastVoting

We now define the communication predicate that will be assumed for the correctness proof of the LastVoting algorithm. The "per-round" part is trivial: integrity and agreement are always ensured.
For the "global" part, Charron-Bost and Schiper propose a predicate that requires the existence of infinitely many phases $p h$ such that:

- all processes agree on the same coordinator $c$,
- $c$ hears from a strict majority of processes in steps 0 and 2 of phase $p h$, and
- every process hears from $c$ in steps 1 and 3 (this is slightly weaker than the predicate that appears in [7], but obviously sufficient).

Instead of requiring infinitely many such phases, we only assume the existence of one such phase (Charron-Bost and Schiper note that this is enough.)

## definition

```
\(L V\)-commPerRd where
\(L V\)-commPerRd r (HO::Proc HO) (coord::Proc coord) \(\equiv\) True
```


## definition

```
LV-commGlobal where
LV-commGlobal HOs coords \(\equiv\)
        \(\exists\) ph::nat. \(\exists c::\) Proc.
            \((\forall p . \operatorname{coords}(4 * p h) p=c)\)
            \(\wedge\) card \((H O s(4 * p h) c)>N\) div 2
            \(\wedge \operatorname{card}(\) HOs \((4 * p h+2) c)>N\) div 2
            \(\wedge(\forall p . c \in H O s(4 * p h+1) p \cap H O s(4 * p h+3) p)\)
```


### 7.3 The LastVoting Heard-Of Machine

We now define the coordinated HO machine for the LastVoting algorithm by assembling the algorithm definition and its communication-predicate.

```
definition \(L V\)-CHOMachine where
    LV-CHOMachine \(\equiv\)
    ( CinitState \(=L V\)-initState,
        sendMsg \(=L V\)-sendMsg,
```

CnextState $=L V$-nextState, CHOcommPerRd $=L V$-commPerRd, CHOcommGlobal $=L V$-commGlobal )

## abbreviation

```
LV-M \equiv(LV-CHOMachine::(Proc,'val pstate,'val msg) CHOMachine)
```


## end

theory LastVotingProof
imports LastVotingDefs ../ Majorities ../ Reduction
begin

### 7.4 Preliminary Lemmas

We begin by proving some simple lemmas about the utility functions used in the model of LastVoting. We also specialize the induction rules of the generic CHO model for this particular algorithm.

```
lemma timeStampsRcvdFinite:
    finite {ts.\existsqv.(msgs::Proc \rightharpoonup'val msg) q = Some (ValStamp v ts)}
    (is finite ?ts)
proof -
    have ?ts = stamp'the ' msgs '(valStampsRcvd msgs)
    by (force simp add: valStampsRcvd-def image-def)
    thus ?thesis by auto
qed
lemma highestStampRcvd-exists:
    assumes nempty: valStampsRcvd msgs }\not={
    obtains pv where msgs p=Some (ValStamp v (highestStampRcvd msgs))
proof -
    let ?ts ={ts.\existsqv.msgs q=Some(ValStamp v ts)}
    from nempty have ?ts }\not={}\mathrm{ by (auto simp add: valStampsRcvd-def)
    with timeStampsRcvdFinite
    have highestStampRcvd msgs \in?ts
        unfolding highestStampRcvd-def by (rule Max-in)
    then obtain pv where msgs p=Some (ValStamp v (highestStampRcvd msgs))
        by (auto simp add: highestStampRcvd-def)
    with that show thesis .
qed
lemma highestStampRcvd-max:
    assumes msgs p = Some (ValStamp v ts)
    shows ts \leqhighestStampRcvd msgs
    using assms unfolding highestStampRcvd-def
    by (blast intro: Max-ge timeStampsRcvdFinite)
lemma phase-Suc:
    phase (Suc r)=(if step r=3 then Suc (phase r)
    else phase r)
```

unfolding step－def phase－def by presburger
Many proofs are by induction on runs of the LastVoting algorithm，and we derive a specific induction rule to support these proofs．

```
lemma \(L V\)-induct:
    assumes run: CHORun LV-M rho HOs coords
    and init: \(\forall p\). CinitState \(L V-M p(\) rho \(0 p)(\) coords \(0 p) \Longrightarrow P 0\)
    and step \(0: \bigwedge r\).
                        \(\llbracket\) step \(r=0 ; P r ;\) phase \((\) Suc \(r)=\) phase \(r ;\) step \((\) Suc \(r)=1\);
                        \(\forall p\). next0 r p (rho r p)
                            (HOrcvdMsgs LV-Mrp(HOs rp)(rho r))
                            (coords (Suc r) p)
                            (rho (Suc r) p) 】
        \(\Longrightarrow P(\) Suc \(r)\)
    and step \(1: \bigwedge r\).
        【 step \(r=1 ;\) Pr;phase \((\) Suc \(r)=\) phase \(r\); step \((\) Suc \(r)=\) 2;
        \(\forall p\). next1 \(r\) p (rho r \(p\) )
                            (HOrcvdMsgs LV-M r p (HOs r p) (rho r))
                            (coords (Suc r) p)
                            (rho (Suc r) p) 】
        \(\Longrightarrow P(\) Suc \(r)\)
    and step2: \(\bigwedge\) r.
            【 step \(r=2 ; P\) r; phase \((\) Suc \(r)=\) phase \(r\); step \((\) Suc \(r)=3\);
            \(\forall p\). next2 \(r p\) (rho r \(p\) )
                                    (HOrcvdMsgs LV-Mrp(HOs rp) (rho r))
                                    (coords (Suc r) p)
                            (rho (Suc r) p) 】
        \(\Longrightarrow P(\) Suc \(r)\)
    and step3: \(\bigwedge r\).
        \(\llbracket\) step \(r=3 ;\) Pr \(r\) phase \((\) Suc \(r)=\) Suc \((\) phase \(r) ;\) step \((\) Suc \(r)=0\);
        \(\forall p\). next3 \(r\) p (rho r \(p\) )
            (HOrcvdMsgs LV-M r p (HOs r p) (rho r))
            (coords (Suc r) p)
                            (rho (Suc r) p) 】
        \(\Longrightarrow P(\) Suc \(r)\)
    shows \(P\) n
proof (rule CHORun-induct[OF run])
    assume CHOinitConfig LV-M (rho 0) (coords 0)
    thus P 0 by (auto simp add: CHOinitConfig-def init)
next
    fix \(r\)
    assume \(i h: P r\)
        and nxt: CHOnextConfig LV-M r (rho r) (HOs r)
                            (coords (Suc r)) (rho (Suc r))
    have step \(r \in\{0,1,2,3\}\) by (auto simp add: step-def)
    thus \(P\) (Suc r)
    proof auto
    assume stp: step \(r=0\)
    hence step (Suc r) = 1
```

```
        by (auto simp add: step-def mod-Suc)
    with ih nxt stp show ?thesis
        by (intro step0)
            (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                                    LV-nextState-def LV-sendMsg-def phase-Suc)
next
    assume stp: step r = Suc 0
    hence step (Suc r) = 2
        by (auto simp add: step-def mod-Suc)
    with ih nxt stp show ?thesis
        by (intro step1)
            (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                                    LV-nextState-def LV-sendMsg-def phase-Suc)
next
    assume stp: step r = 2
    hence step (Suc r)=3
        by (auto simp add: step-def mod-Suc)
    with ih nxt stp show ?thesis
        by (intro step2)
            (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                                    LV-nextState-def LV-sendMsg-def phase-Suc)
next
    assume stp: step r=3
    hence step (Suc r) = 0
        by (auto simp add: step-def mod-Suc)
    with ih nxt stp show ?thesis
        by (intro step3)
            (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                                    LV-nextState-def LV-sendMsg-def phase-Suc)
qed
qed
```

The following rule similarly establishes a property of two successive config－ urations of a run by case distinction on the step that was executed．
lemma $L V$－Suc：
assumes run：CHORun LV－M rho HOs coords
and step $0: \llbracket$ step $r=0$ ；step $($ Suc $r)=1$ ；phase $($ Suc $r)=$ phase $r$ ；
$\forall p$ ．next0 r $p$（rho r $p$ ）
（HOrcvdMsgs LV－M rp（HOs rp）（rho r））
（coords（Suc r）p）（rho（Suc r）p）】
$\Longrightarrow P r$
and step 1：【 step $r=1$ ；step $($ Suc $r)=2$ ；phase $($ Suc $r)=$ phase $r$ ； $\forall p$ ．next1 r $p$（rho r $p$ ）
（HOrcvdMsgs LV－M rp（HOs rp）（rho r））
（coords（Suc r）p）（rho（Suc r）p）】

$$
\Longrightarrow P r
$$

and step2：$\llbracket$ step $r=2$ ；step $($ Suc $r)=3$ ；phase（Suc $r$ ）$=$ phase $r$ ； $\forall p$ ．next2 r $p$（rho r $p$ ）
（HOrcvdMsgs LV－M r p（HOs r p）（rho r））

```
                        (coords (Suc r) p)(rho (Suc r) p)\rrbracket
        \Longrightarrow P r
    and step3: \llbracket step r = 3; step (Suc r)=0; phase (Suc r)=Suc (phase r);
            \forall\mp@code{next3 r p (rho r p)}
                (HOrcvdMsgs LV-M r p (HOs r p) (rho r))
                (coords (Suc r) p)(rho (Suc r) p)\rrbracket
        Pr
    shows Pr
proof -
    from run
    have nxt:CHOnextConfig LV-M r (rho r) (HOs r)
                            (coords (Suc r)) (rho (Suc r))
    by (auto simp add: CHORun-eq)
    have step }r\in{0,1,2,3} by (auto simp add: step-def
    thus Pr
    proof (auto)
    assume stp: step r=0
    hence step (Suc r) = 1
            by (auto simp add: step-def mod-Suc)
    with nxt stp show ?thesis
            by (intro step0)
                (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                            LV-nextState-def LV-sendMsg-def phase-Suc)
    next
    assume stp: step r = Suc 0
    hence step (Suc r) = 2
            by (auto simp add: step-def mod-Suc)
    with nxt stp show ?thesis
            by (intro step1)
                    (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                                    LV-nextState-def LV-sendMsg-def phase-Suc)
    next
        assume stp: step r=2
        hence step (Suc r) = 3
            by (auto simp add: step-def mod-Suc)
        with nxt stp show ?thesis
            by (intro step2)
                (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                            LV-nextState-def LV-sendMsg-def phase-Suc)
    next
        assume stp: step r=3
        hence step (Suc r) = 0
            by (auto simp add: step-def mod-Suc)
            with nxt stp show ?thesis
            by (intro step3)
                (auto simp:LV-CHOMachine-def CHOnextConfig-eq
                    LV-nextState-def LV-sendMsg-def phase-Suc)
    qed
qed
```

Sometimes the assertion to prove talks about a specific process and follows from the next－state relation of that particular process．We prove corre－ sponding variants of the induction and case－distinction rules．When these variants are applicable，they help automating the Isabelle proof．

```
lemma \(L V\)-induct':
    assumes run: CHORun LV-M rho HOs coords
    and init: CinitState \(L V-M p(r h o 0 p)(\) coords \(0 p) \Longrightarrow P p 0\)
    and step \(0: \bigwedge r\). 【 step \(r=0 ; P\) pr;phase \((\) Suc \(r)=\) phase \(r\); step \((\) Suc \(r)=1\);
        next0 r \(p\) (rho r \(p\) )
                            (HOrcvdMsgs LV-M rp(HOs rp) (rho r))
                            (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p\) (Suc r)
    and step \(1: \bigwedge r\). 【 step \(r=1 ; P\) pr;phase \((\) Suc \(r)=\) phase \(r\); step \((\) Suc \(r)=2\);
        next1 r \(p\) (rho r \(p\) )
                            (HOrcvdMsgs LV-M r p (HOs rp) (rho r))
                            (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p(\) Suc \(r)\)
    and step2: \(\bigwedge\) r. 【 step \(r=2\); P pr;phase \((\) Suc \(r)=\) phase \(r\); step \((\) Suc \(r)=3\);
        next2 \(r\) p (rho r \(p\) )
                            (HOrcvdMsgs LV-M rp (HOs rp) (rho r))
                            (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p(\) Suc \(r)\)
    and step3: \(\bigwedge r\). 【 step \(r=3 ;\) P pr;phase (Suc r)=Suc (phase r); step (Suc r)
\(=0\);
        next3 r \(p\) (rho r \(p\) )
                            (HOrcvdMsgs LV-M rp (HOs rp) (rho r))
                            (coords (Suc r) p) (rho (Suc r) p) 】
        \(\Longrightarrow P p(\) Suc \(r)\)
    shows \(P\) p \(n\)
    by (rule LV-induct[OF run])
    (auto intro: init step0 step1 step2 step3)
lemma \(L V\)-Suc':
    assumes run: CHORun LV-M rho HOs coords
    and step \(0: \llbracket\) step \(r=0\); step \((\) Suc \(r)=1\); phase \((\) Suc \(r)=\) phase \(r\);
        next0 r \(p\) (rho r \(p\) )
        (HOrcvdMsgs LV-M r p (HOs r p) (rho r))
        (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p r\)
    and step 1: 【 step \(r=1\); step \((\) Suc \(r)=2\); phase (Suc \(r)=\) phase \(r\);
        next1 r \(p\) (rho r \(p\) )
        (HOrcvdMsgs LV-Mrp (HOs r p) (rho r))
        (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p_{r}\)
and step2: 【 step \(r=2\); step \((\) Suc \(r)=3\); phase (Suc \(r\) ) \(=\) phase \(r\);
        next2 \(r\) p (rho r \(p\) )
        (HOrcvdMsgs LV-Mrp(HOs rp) (rho r))
        (coords (Suc r) p) (rho (Suc r) p) 】
        \(\Longrightarrow P p r\)
```

```
and step3: 【 step \(r=3\); step \((\) Suc \(r)=0 ;\) phase (Suc \(r)=\) Suc (phase \(r\) );
    next3 r \(p\) (rho r \(p\) )
        (HOrcvdMsgs LV-Mrp(HOs rp) (rho r))
        (coords (Suc r) p) (rho (Suc r) p) 】
    \(\Longrightarrow P p r\)
```

shows P pr
by (rule $L V$-Suc[OF run])
(auto intro: step0 step1 step2 step3)

### 7.5 Boundedness and Monotonicity of Timestamps

The timestamp of any process is bounded by the current phase.

```
lemma LV-timestamp-bounded:
    assumes run: CHORun LV-M rho HOs coords
    shows timestamp (rho n p)\leq(if step n<2 then phase n else Suc (phase n))
        (is ?P p n)
    by (rule LV-induct' [OF run, where P=?P])
    (auto simp:LV-CHOMachine-def LV-initState-def
        next0-def next1-def next2-def next3-def)
```

Moreover, timestamps can only grow over time.

```
lemma LV-timestamp-increasing:
    assumes run: CHORun LV-M rho HOs coords
    shows timestamp (rho n \(p\) ) \(\leq\) timestamp (rho (Suc n) p)
    (is ? \(P p n\) is ? \(t s \leq-\) )
proof (rule \(L V\)-Suc' \([\) OF run, where \(P=? P]\) )
```

The case of next1 is the only interesting one because the timestamp may change: here we use the previously established fact that the timestamp is bounded by the phase number.

```
    assume stp: step \(n=1\)
    and nxt: next1 \(n p\) (rho \(n\) p)
                                    (HOrcvdMsgs LV-M \(n\) p (HOs \(n\) p) (rho \(n\) ) )
            (coords (Suc n) p) (rho (Suc n) p)
from stp have ?ts \(\leq\) phase \(n\)
    using \(L V\)-timestamp-bounded \([\) OF run, where \(n=n\), where \(p=p]\) by auto
    with nxt show ?thesis by (auto simp add: next1-def)
qed (auto simp add: next0-def next2-def next3-def)
lemma LV-timestamp-monotonic:
    assumes run: CHORun LV-M rho HOs coords and le: \(m \leq n\)
    shows timestamp (rho mp) timestamp (rho n p)
    (is?ts \(m \leq-\) )
proof -
    from le obtain \(k\) where \(k: n=m+k\)
        by (auto simp add: le-iff-add)
    have ?ts \(m \leq\) ?ts \((m+k)\) (is ?P \(k\) )
    proof (induct \(k\) )
```

```
        case 0 show ?P 0 by simp
    next
        fix }
        assume ih:?P k
        from run have ?ts ( }m+k\mathrm{ ) }\leq\mathrm{ ?ts ( }m+\mathrm{ Suc k)
            by (auto simp add: LV-timestamp-increasing)
    with ih show ?P (Suc k) by simp
    qed
    with k show ?thesis by simp
qed
```

The following definition collects the set of processes whose timestamp is beyond a given bound at a system state.
definition procsBeyondTS where

```
procsBeyondTS ts cfg \equiv{ p.ts \leq timestamp (cfg p) }
```

Since timestamps grow monotonically, so does the set of processes that are beyond a certain bound.
lemma procsBeyondTS-monotonic:
assumes run: CHORun LV-M rho HOs coords
and $p: p \in$ procsBeyondTS ts (rho $m$ ) and le: $m \leq n$
shows $p \in$ procsBeyondTS ts (rho $n$ )
proof -
from $p$ have $t s \leq$ timestamp (rho $m p$ ) (is - $\leq$ ?ts $m$ )
by (simp add: procsBeyondTS-def)
moreover
from run le have ?ts $m \leq$ ?ts $n$ by (rule LV-timestamp-monotonic)
ultimately show ?thesis
by (simp add: procsBeyondTS-def)
qed

### 7.6 Obvious Facts About the Algorithm

The following lemmas state some very obvious facts that follow "immediately" from the definition of the algorithm. We could prove them in one fell swoop by defining a big invariant, but it appears more readable to prove them separately.

Coordinators change only at step 3 .
lemma notStep3EqualCoord:
assumes run: CHORun LV-M rho HOs coords and stp:step $r \neq 3$
shows $\operatorname{coord} \Phi($ rho $($ Suc r) $)=\operatorname{coord} \Phi($ rho r $p$ ) (is ?P pr)
by (rule $L V$-Suc' $[O F$ run, where $P=? P]$ )
(auto simp: stp next0-def next1-def next2-def)
lemma coordinators:
assumes run: CHORun LV-M rho HOs coords
shows $\operatorname{coord} \Phi($ rho r $p)=\operatorname{coords}(4 *($ phase r $)) p$

```
proof -
    let ? \(r 0=(4 *(\) phase \(r)-1)\)
    let ? \(r 1=(4 *(\) phase \(r))\)
    have \(\operatorname{coord} \Phi(\) rho ? r1 \(p)=\) coords ? r \(1 p\)
    proof (cases phase \(r>0\) )
    case False
    hence phase \(r=0\) by auto
    with run show ?thesis
            by (auto simp: LV-CHOMachine-def CHORun-eq CHOinitConfig-def
                                    LV-initState-def)
    next
        case True
        hence step (Suc ?r0) \(=0\) by (auto simp: step-def)
        hence step ?r0 \(=3\) by (auto simp: mod-Suc step-def)
        moreover
        from run
        have LV-nextState ?r0 \(p\) (rho ?r0 p)
                            (HOrcvdMsgs LV-M ?r0 p (HOs ?r0 p) (rho ?r0))
                            (coords (Suc ?r0) p) (rho (Suc ?r0) p)
            by (auto simp: LV-CHOMachine-def CHORun-eq CHOnextConfig-eq)
    ultimately
    have nxt: next3 ?r0 \(p\) (rho ?r0 p)
                            (HOrcvdMsgs LV-M ?r0 p (HOs ?r0 p) (rho ?r0))
                            (coords (Suc ?r0) p) (rho (Suc ?r0) p)
            by (auto simp: LV-nextState-def)
    hence coord \(\Phi\) (rho (Suc ?r0) p) \(=\) coords (Suc ?r0) \(p\)
            by (auto simp: next3-def)
    with True show ?thesis by auto
qed
moreover
from run
have coord \(\Phi(\) rho \((\) Suc (Suc (Suc ?r1) )) \(p)=\operatorname{coord} \Phi(\) rho ?r1 \(p)\)
        \(\wedge \operatorname{coord} \Phi(r h o(S u c(S u c ~ ? r 1)) p)=\operatorname{coord} \Phi(\) rho ?r1 p)
        \(\wedge \operatorname{coord} \Phi(\) rho \((\) Suc ?r1) \(p)=\operatorname{coord} \Phi(\) rho ? \(r 1\) p)
    by (auto simp: notStep3EqualCoord step-def phase-def mod-Suc)
    moreover
have \(r \in\{\) ?r1, Suc ?r1, Suc (Suc ?r1), Suc (Suc (Suc ?r1)) \}
    by (auto simp: step-def phase-def mod-Suc)
ultimately
show ?thesis by auto
qed
Votes only change at step 0.
lemma notStep0EqualVote [rule-format]:
assumes run: CHORun LV-M rho HOs coords
shows step \(r \neq 0 \longrightarrow\) vote (rho (Suc r) \(p\) ) \(=\) vote (rho r \(p\) ) (is ?P pr)
by (rule \(L V\)-Suc' \([\) OF run, where \(P=? P]\) )
    (auto simp: next0-def next1-def next2-def next3-def)
```

Commit status only changes at steps 0 and 3 .

```
lemma notStep03EqualCommit [rule-format]:
    assumes run: CHORun LV-M rho HOs coords
    shows step \(r \neq 0 \wedge\) step \(r \neq 3 \longrightarrow\) commt (rho (Suc r) \(p\) ) \(=\) commt (rhorp)
        (is ? P \(p r\) )
    by (rule \(L V-S u c\) ' \([\) OF run, where \(P=? P]\) )
    (auto simp: next0-def next1-def next2-def next3-def)
```

Timestamps only change at step 1 .
lemma notStep 1EqualTimestamp [rule-format]:
assumes run: CHORun $L V-M$ rho $H O s$ coords
shows step $r \neq 1 \longrightarrow$ timestamp (rho (Suc r) $p$ ) $=$ timestamp (rho r $p$ ) (is ? $P$ pr)
by (rule $L V-S u c$ ' $[$ OF run, where $P=? P]$ )
(auto simp: next0-def next1-def next2-def next3-def)
The $x$ field only changes at step 1 .
lemma notStep1EqualX [rule-format]:
assumes run: CHORun LV-M rho HOs coords
shows step $r \neq 1 \longrightarrow x($ rho $($ Suc $r) p)=x($ rho r $p)($ is ?P $p r)$
by (rule $L V-S u c^{\prime}[$ OF run, where $P=? P]$ )
(auto simp: next0-def next1-def next2-def next3-def)
A process $p$ has its commt flag set only if the following conditions hold:

- the step number is at least 1 ,
- $p$ considers itself to be the coordinator,
- $p$ has a non-null vote,
- a majority of processes consider $p$ as their coordinator.
lemma commitE:
assumes run: CHORun LV-M rho HOs coords and cmt: commt (rho r p)
and conds: $\llbracket 1 \leq$ step $r ; \operatorname{coord} \Phi($ rho $r p)=p ;$ vote $($ rho $r p) \neq$ None;
card $\{q \cdot \operatorname{coord} \Phi($ rho $r q)=p\}>N$ div 2

$$
\rrbracket \Longrightarrow A
$$

shows $A$
proof -
have commt (rho r $p$ ) $\longrightarrow$
$1 \leq$ step r
$\wedge \operatorname{coord} \Phi($ rho $r p)=p$
$\wedge$ vote (rho r $p$ ) $\neq$ None
$\wedge \operatorname{card}\{q \cdot \operatorname{coord} \Phi($ rho $r q)=p\}>N$ div 2
(is ? P $p r$ is - $\longrightarrow$ ? $R r$ )
proof (rule $L V$-induct ${ }^{\prime}[O F$ run, where $P=? P]$ )

- the only interesting step is step 0
fix $n$
assume nxt: next0 n $p($ rho $n$ p) (HOrcvdMsgs LV-M $n$ p (HOs n p) (rho n))

```
                                    (coords (Suc n) p) (rho (Suc n) p)
        and ph:phase (Suc n) = phase n
        and stp: step n = 0 and stp': step (Suc n)=1
        and ih: ?P p n
    show ?P p (Suc n)
    proof
        assume cm': commt (rho (Suc n) p)
        from stp ih have cm: ᄀcommt (rho n p) by simp
        with nxt cm'
        have coord \Phi (rho n p)=p
        \wedge vote (rho (Suc n) p) \not= None
        ^card (valStampsRcvd (HOrcvdMsgs LV-M n p (HOs n p) (rho n)))
                > N div 2
        by (auto simp add: next0-def)
    moreover
    from stp
    have valStampsRcvd (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
                \subseteq \{ q \cdot \operatorname { c o o r d } \Phi ( \text { rho n q) = p\}}
        by (auto simp: valStampsRcvd-def LV-CHOMachine-def
                            HOrcvdMsgs-def LV-sendMsg-def send0-def)
    hence card (valStampsRcvd (HOrcvdMsgs LV-M n p (HOs n p) (rho n)))
                \leqcard {q. coord \Phi (rho n q) = p}
        by (auto intro: card-mono)
    moreover
    note stp stp' run
    ultimately
    show ?R (Suc n) by (auto simp: notStep3EqualCoord)
    qed
    - the remaining cases are all solved by expanding the definitions
    qed (auto simp: LV-CHOMachine-def LV-initState-def next1-def next2-def
        next3-def notStep3EqualCoord[OF run])
    with cmt show ?thesis by (intro conds, auto)
qed
```

A process has a current timestamp only if:

- it is at step 2 or beyond,
- its coordinator has committed,
- its $x$ value is the vote of its coordinator.
lemma currentTimestampE:
assumes run: CHORun LV-M rho HOs coords
and ts: timestamp (rho r p) $=$ Suc (phase r)
and conds: $\mathbb{2} \leq$ step $r$;
commt (rho r $(\operatorname{coord} \Phi($ rho $r p)))$;
$x($ rho $r p)=$ the $($ vote $($ rho $r(\operatorname{coord} \Phi($ rho r $p))))$
$\rrbracket \Longrightarrow A$
shows $A$

```
proof -
    let ?ts n = timestamp (rho n p)
    let ?crd n = coord \Phi (rho n p)
    have ?ts r = Suc (phase r)}
                    2 s step r
                commt (rho r (?crd r))
            \wedge (rho r p)=the (vote (rho r (?crd r)))
        (is ?Q pr is -\longrightarrow ?R r)
    proof (rule LV-induct'[OF run, where P=?Q])
        - The assertion is trivially true initially because the timestamp is 0.
    assume CinitState LV-M p (rho 0 p) (coords 0 p) thus ?Q p 0
            by (auto simp: LV-CHOMachine-def LV-initState-def)
    next
```

The assertion is trivially preserved by step 0 because the timestamp in the poststate cannot be current (cf. lemma LV-timestamp-bounded).
fix $n$
assume stp': step (Suc n) $=1$
with run LV-timestamp-bounded[where $n=$ Suc $n$ ]
have ?ts (Suc n) $\leq$ phase (Suc $n$ ) by auto
thus ?Q $p$ (Suc $n$ ) by simp
next
Step 1 establishes the assertion by definition of the transition relation.

```
fix \(n\)
assume stp: step \(n=1\) and stp \(^{\prime}:\) step \((S u c ~ n)=2\)
    and ph: phase (Suc \(n\) ) \(=\) phase \(n\)
    and nxt: next1 \(n\) p (rho \(n\) p) (HOrcvdMsgs LV-M n \(p\) (HOs \(n p\) ) (rhon))
            (coords (Suc n) p) (rho (Suc n) p)
show ?Q \(p\) (Suc n)
proof
    assume ts: ?ts (Suc n) = Suc (phase (Suc n))
    from run stp LV-timestamp-bounded [where \(n=n\) ]
    have ?ts \(n \leq\) phase \(n\) by auto
    moreover
    from run stp
    have vote (rho (Suc n) (?crd (Suc n))) = vote (rho \(n(? c r d n))\)
        by (auto simp: notStep3EqualCoord notStep0EqualVote)
    moreover
    from run stp
    have commt (rho (Suc n) (?crd (Suc n))) \(=\) commt (rho \(n(? \operatorname{crd} n))\)
        by (auto simp: notStep3EqualCoord notStep03EqualCommit)
    moreover
    note \(t s n x t\) stp stp' \(p h\)
    ultimately
    show ?R (Suc n)
        by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def
                next1-def send1-def is Vote-def)
qed
```


## next

For step 2, the assertion follows from the induction hypothesis, observing that none of the relevant state components change.
fix $n$
assume stp: step $n=2$ and stp': step $($ Suc $n)=3$
and ph: phase (Suc $n$ ) $=$ phase $n$
and $i h$ :? $Q$ p $n$
and nxt: next2 $n p($ rho $n$ p) (HOrcvdMsgs LV-M $n p$ (HOs $n p$ ) (rho n))
(coords (Suc n) p) (rho (Suc n) p)
show ? $Q p$ (Suc n)
proof
assume ts: ?ts (Suc n) = Suc (phase (Suc n))
from run stp
have vt: vote $($ rho $($ Suc $n)(? c r d ~(S u c ~ n)))=\operatorname{vote}($ rho $n(? c r d n))$
by (auto simp add: notStep3EqualCoord notStep0EqualVote)
from run stp
have cmt: commt (rho (Suc n) (?crd (Suc n))) $=$ commt (rho $n(? c r d n))$ by (auto simp add: notStep3EqualCoord notStep03EqualCommit)
with vt ts ph stp stp' ih nxt
show ?R (Suc n) by (auto simp add: next2-def)
qed
next
The assertion is trivially preserved by step 3 because the timestamp in the poststate cannot be current (cf. lemma LV-timestamp-bounded).
fix $n$
assume stp ${ }^{\prime}$ : step (Suc $\left.n\right)=0$
with run LV-timestamp-bounded $[$ where $n=S u c n]$
have ?ts (Suc n) $\leq$ phase (Suc $n$ ) by auto
thus ?Q $p(S u c n)$ by $\operatorname{simp}$
qed
with $t s$ show ?thesis by (intro conds) auto
qed
If a process $p$ has its ready bit set then:

- it is at step 3 ,
- it considers itself to be the coordinator of that phase and
- a majority of processes considers $p$ to be the coordinator and has a current timestamp.
lemma readyE:
assumes run: CHORun LV-M rho HOs coords and rdy: ready (rho r p)
and conds: 【 step $r=3 ; \operatorname{coord} \Phi($ rho $r p)=p$;
card $\{q \cdot \operatorname{coord} \Phi($ rho $r q)=p$

```
                    ^ timestamp (rho r q) = Suc (phase r) } > N div 2
                        \Longrightarrow P
    shows P
proof -
    let ?qs }n={q\cdot\operatorname{coord}\Phi(rho n q)=
            \imestamp (rho n q) = Suc (phase n) }
    have ready (rho r p)\longrightarrow
                step r = 3
            ^coord\Phi (rho r p) = p
            card (?qs r)>N div 2
    (is ?Q pr is - \longrightarrow?R pr)
proof (rule LV-induct' }[OF run, where P=?Q]
            - the interesting case is step 2
    fix n
    assume stp: step n =2 and stp': step (Suc n) = 3
            and ih:?Q p n and ph: phase (Suc n) = phase n
            and nxt: next2 n p (rho n p) (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
                                    (coords (Suc n) p) (rho (Suc n) p)
    show ?Q p (Suc n)
    proof
        assume rdy: ready (rho (Suc n) p)
        from stp ih have nrdy: \neg ready (rho n p) by simp
        with rdy nxt have coord \Phi (rho n p)=p
        by (auto simp: next2-def)
        with run stp have coord: coord }\Phi\mathrm{ (rho (Suc n) p)=p
            by (simp add: notStep3EqualCoord)
    let ?acks = acksRcvd (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
    from nrdy rdy nxt have aRcvd: card ?acks > N div 2
        by (auto simp: next2-def)
    have ?acks \subseteq?qs (Suc n)
    proof (clarify)
        fix q
        assume q:q\in?acks
        with stp
        have n: coord \Phi (rho n q) = p^ timestamp (rho n q) = Suc (phase n)
            by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def
                                acksRcvd-def send2-def isAck-def)
        with run stp ph
        show coord \Phi (rho (Suc n) q) = p
                \timestamp (rho (Suc n) q) = Suc (phase (Suc n))
            by (simp add: notStep3EqualCoord notStep1EqualTimestamp)
    qed
    hence card ?acks \leq card (?qs (Suc n))
        by (intro card-mono) auto
        with stp' coord aRcvd show ?R p (Suc n)
        by auto
    qed
    - the remaining steps are all solved trivially
qed (auto simp:LV-CHOMachine-def LV-initState-def
```

> next0-def next1-def next3-def)
with rdy show ?thesis by (blast intro: conds)
qed
A process decides only if the following conditions hold:

- it is at step 3 ,
- its coordinator votes for the value the process decides on,
- the coordinator has its ready and commt bits set.

```
lemma decisionE:
    assumes run: CHORun LV-M rho HOs coords
    and dec: decide (rho (Suc r) p) \(\neq\) decide (rho r \(p\) )
    and conds: 【
        step \(r=3\);
        decide \((\) rho \((\) Suc r) \(p)=\) Some \((\) the \((\operatorname{vote}(\) rho \(r(\operatorname{coord} \Phi(\) rho r \(p)))))\);
        ready \((\) rho \(r(\operatorname{coord} \Phi(\) rho \(r p))) ; \operatorname{commt}(\) rho \(r(\operatorname{coord} \Phi(\) rho r \(p)))\)
        \(\rrbracket \Longrightarrow P\)
    shows \(P\)
proof -
    let ?cfg \(=\) rho \(r\)
    let ? \({ }^{\prime} \mathrm{fg}^{\prime}=r h o(S u c r)\)
    let ? crd \(p=\operatorname{coord} \Phi(? c f g p)\)
    let ? \(\mathrm{dec}^{\prime}=\) decide \((\) ?cfg' \(p)\)
```

Except for the assertion about the commt field, the assertion can be proved directly from the next-state relation.

```
have 1: step \(r=3\)
    \(\wedge\) ? dec \({ }^{\prime}=\) Some \((\) the \((\) vote \((? c f g(? c r d p))))\)
    \(\wedge\) ready (?cfg (?crd p))
    (is ? \(Q p r\) )
    proof (rule \(L V\)-Suc' \([\) OF run, where \(P=\) ? \(Q]\) )
    - for step 3, we prove the thesis by expanding the relevant definitions
    assume next3 r p (?cfg p) (HOrcvdMsgs LV-M r p (HOs r p) ?cfg)
                                (coords (Suc r) p) (?cfg' p)
        and step \(r=3\)
    with dec show ?thesis
        by (auto simp: next3-def send3-def isVote-def LV-CHOMachine-def
                                    HOrcudMsgs-def LV-sendMsg-def)
next
    - the other steps don't change the decision
    assume next0 r \(p\) (?cfg p) (HOrcvdMsgs LV-M rp(HOs rp) ?cfg)
                (coords (Suc r) p) (?cfg' p)
    with dec show ?thesis by (auto simp: next0-def)
next
    assume next1 r \(p\) (?cfg p) (HOrcvdMsgs LV-Mrp(HOs r p) ?cfg)
                (coords (Suc r) p) (?cfg' p)
    with dec show ?thesis by (auto simp: next1-def)
```

```
next
    assume next2 r \(p\) (?cfg p) (HOrcvdMsgs LV-M r p (HOs r p) ?cfg)
                    (coords (Suc r) p) (?cfg' p)
    with dec show ?thesis by (auto simp: next2-def)
qed
hence ready (?cfg (?crd p)) by blast
```

Because the coordinator is ready, there is a majority of processes that consider it to be the coordinator and that have a current timestamp.

```
with run
have card {q. ?crd q = ?crd p ^ timestamp (?cfg q) = Suc (phase r)}
    > N div 2 by (rule readyE)
- Hence there is at least one such process ...
hence card {q. ?crd q=?crd p\wedge timestamp (?cfg q)=Suc (phase r)}\not=0
    by arith
then obtain q where ?crd q = ?crd p and timestamp (?cfg q) = Suc (phase r)
    by auto
- ... and by a previous lemma the coordinator must have committed.
with run have commt (?cfg (?crd p))
    by (auto elim: currentTimestampE)
    with 1 show ?thesis by (blast intro: conds)
qed
```


### 7.7 Proof of Integrity

Integrity is proved using a standard invariance argument that asserts that only values present in the initial state appear in the relevant fields.

```
lemma lv-integrityInvariant:
    assumes run: CHORun LV-M rho HOs coords
    and inv: 【range \((x \circ(\) rho \(n)) \subseteq\) range \((x \circ(\) rho 0\())\);
            range \((\) vote \(\circ(\) rho \(n)) \subseteq\{\) None \(\} \cup\) Some 'range \((x \circ(\) rho 0\())\);
            range \((\) decide \(\circ(\) rho \(n)) \subseteq\{\) None \(\} \cup\) Some'range \((x \circ(\) rho 0\())\)
        \(\rrbracket \Longrightarrow A\)
    shows \(A\)
proof -
    let \(? x 0=\) range \((x \circ\) rho 0\()\)
    let ? \(x 0\) opt \(=\{\) None \(\} \cup\) Some '? \(x 0\)
    have range ( \(x\) ○ rho \(n\) ) \(\subseteq\) ? \(x 0\)
        \(\wedge\) range (vote \(\circ\) rho \(n\) ) \(\subseteq\) ? x0opt
        \(\wedge\) range (decide \(\circ\) rho \(n\) ) \(\subseteq\) ?x0opt
    (is ?Inv \(n\) is ? \(X n \wedge\) ? Vote \(n \wedge\) ?Decide \(n\) )
proof (induct \(n\) )
    from run show ?Inv 0
            by (auto simp: CHORun-eq CHOinitConfig-def LV-CHOMachine-def
                LV-initState-def)
    next
    fix \(n\)
    assume ih: ?Inv \(n\) thus ?Inv (Suc \(n\) )
    proof (clarify)
```

assume $x$ : ? $X n$ and $v t$ : ?Vote $n$ and dec: ?Decide $n$
Proof of first conjunct

```
have \mp@subsup{x}{}{\prime}:?X (Suc n)
proof (clarsimp)
    fix p
    from run
    show x (rho (Suc n) p)\in range ( }\lambdaq.x(rho 0 q)) (is ?P p n)
    proof (rule LV-Suc'[where P=?P])
        - only step1 is of interest
        assume stp: step n=1
        and nxt: next1 n p (rho n p)
                            (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
                            (coords (Suc n) p) (rho (Suc n) p)
    show ?thesis
    proof (cases rho (Suc n) p=rho n p)
        case True
        with }x\mathrm{ show ?thesis by auto
    next
        case False
        with stp nxt have cmt: commt (rho n (coord \Phi (rho n p)))
            and xp:x (rho (Suc n) p) = the (vote (rho n (coord\Phi (rho n p))))
        by (auto simp: next1-def LV-CHOMachine-def HOrcvdMsgs-def
                            LV-sendMsg-def send1-def isVote-def)
        from run cmt have vote (rho n (coord }\Phi(\mathrm{ rho n p))) }=\mathrm{ None
            by (rule commitE)
        moreover
        from vt have vote (rho n (coord \Phi (rho n p))) \in ?x0opt
            by (auto simp add: image-def)
        moreover
        note }x
        ultimately
        show ?thesis by (force simp add: image-def)
    qed
    - the other steps don't change }
    next
        assume step n=0
        with run have x (rho (Suc n) p)=x (rho n p)
        by (simp add: notStep1EqualX)
        with }x\mathrm{ show ?thesis by auto
    next
        assume step n=2
        with run have x (rho (Suc n) p)=x(rho n p)
        by (simp add: notStep1EqualX)
    with }x\mathrm{ show ?thesis by auto
    next
        assume step n = 3
        with run have x (rho (Suc n) p)=x(rho n p)
        by (simp add: notStep1EqualX)
```

```
            with }x\mathrm{ show ?thesis by auto
    qed
qed
```

Proof of second conjunct

```
have \(v t^{\prime}\) : ?Vote (Suc n)
proof (clarsimp simp: image-def)
    fix \(p v\)
    assume \(v\) : vote (rho (Suc n) p) \(=\) Some \(v\)
    from run
    have vote (rho (Suc n) p) = Some \(v \longrightarrow v \in\) ? \(x 0\) (is ?P p n)
    proof (rule LV-Suc' \([\) where \(P=\) ? \(P]\) )
    - here only step 0 is of interest
    assume stp: step \(n=0\)
        and nxt: next0 \(n p\) (rho \(n p\) )
                            (HOrcvdMsgs LV-M \(n \mathrm{p}\) (HOs \(n \mathrm{p}\) ) (rho \(n\) ) )
                            (coords (Suc n) p) (rho (Suc n) p)
    show ?thesis
    proof (cases rho (Suc n) \(p=\) rho \(n\) p)
        case True
        from \(v t\) have vote (rho \(n p\) ) \(\in\) ?x0opt
            by (auto simp: image-def)
        with True show ?thesis by auto
    next
        case False
        from nxt stp False \(v\) obtain \(q\) where \(v=x(r h o n q)\)
            by (auto simp: next0-def send0-def LV-CHOMachine-def
                        HOrcvdMsgs-def LV-sendMsg-def)
        with \(x\) show ?thesis by (auto simp: image-def)
    qed
    - the other cases don't change the vote
    next
        assume step \(n=1\)
        with run have vote (rho (Suc n) p) = vote (rho n p)
            by (simp add: notStep0EqualVote)
        moreover
        from \(v t\) have vote (rho \(n p\) ) \(\in\) ?x0opt
            by (auto simp: image-def)
        ultimately
        show ?thesis by auto
    next
        assume step \(n=2\)
        with run have vote (rho (Suc n) p) = vote (rho n p)
        by (simp add: notStep0EqualVote)
    moreover
    from \(v t\) have vote (rho \(n p\) ) \(\in\) ? \(x 0 o p t\)
        by (auto simp: image-def)
    ultimately
    show ?thesis by auto
```

```
            next
            assume step n=3
            with run have vote (rho (Suc n) p)= vote (rho n p)
                by (simp add: notStep0EqualVote)
            moreover
            from vt have vote (rho n p) \in?x0opt
                by (auto simp: image-def)
            ultimately
            show ?thesis by auto
            qed
            with v show \existsq.v=x (rho 0 q) by auto
        qed
Proof of third conjunct
        have dec': ?Decide (Suc n)
        proof (clarsimp simp: image-def)
            fix }p
            assume v: decide (rho (Suc n) p) = Some v
            show \existsq.v=x (rho 0 q)
            proof (cases decide (rho (Suc n) p)= decide (rho n p))
            case True
            with dec True v show ?thesis by (auto simp: image-def)
            next
                    case False
                    let ?crd = coord }\Phi(\mathrm{ rho n p)
            from False run
            have d': decide (rho (Suc n) p) = Some (the (vote (rho n ?crd)))
                and cmt: commt (rho n ?crd)
                by (auto elim: decisionE)
                    from vt have vtc: vote (rho n ?crd) \in?x0opt
                    by (auto simp: image-def)
            from run cmt have vote (rho n ?crd)}\not=\mathrm{ None
                by (rule commitE)
            with d' v vtc show ?thesis by auto
            qed
        qed
        from x'vt' dec' show ?thesis by simp
        qed
    qed
    with inv show ?thesis by simp
qed
Integrity now follows immediately.
theorem lv-integrity:
    assumes run: CHORun LV-M rho HOs coords
        and dec: decide (rho n p)=Some v
    shows \existsq.v=x(rho 0 q)
proof -
    from run have decide (rho n p) \in{None } USome '(range (x\circ(rho 0)))
```

> by (rule lv-integrityInvariant) (auto simp: image-def) with dec show ?thesis by (auto simp: image-def) qed

### 7.8 Proof of Agreement and Irrevocability

The following lemmas closely follow a hand proof provided by Bernadette Charron-Bost.
If a process decides, then a majority of processes have a current timestamp.
lemma decisionThenMajorityBeyondTS:
assumes run: CHORun LV-M rho HOs coords
and dec: decide (rho (Suc r) p) $\neq$ decide (rho r $p$ )
shows card (procsBeyondTS (Suc (phase r)) (rho r)) $>N$ div 2
using run dec proof (rule decisionE)
Lemma decisionE tells us that we are at step 3 and that the coordinator is ready.

```
let ? crd \(=\operatorname{coord} \Phi(\) rho \(r p)\)
let ? \(q s=\{q \cdot \operatorname{coord} \Phi(\) rho \(r q)=\) ?crd
    \(\wedge\) timestamp (rho r q) = Suc (phase r) \(\}\)
assume stp: step \(r=3\) and rdy: ready (rho \(r\) ? crd)
```

Now, lemma readyE implies that a majority of processes have a recent timestamp.

```
    from run rdy have card ?qs > N div 2 by (rule readyE)
    moreover
    from stp LV-timestamp-bounded[OF run, where n=r]
    have }\forallq. timestamp (rho r q) \leqSuc (phase r) by aut
    hence ?qs \subseteqprocsBeyondTS (Suc (phase r)) (rho r)
    by (auto simp: procsBeyondTS-def)
    hence card ?qs \leq card (procsBeyondTS (Suc (phase r))(rho r))
    by (intro card-mono) auto
    ultimately show ?thesis by simp
qed
```

No two different processes have their commit flag set at any state.

```
lemma committedProcsEqual:
    assumes run: CHORun LV-M rho HOs coords
    and cmt: commt (rho r p) and \(\mathrm{cmt}^{\prime}\) : commt (rho r \(p^{\prime}\) )
    shows \(p=p^{\prime}\)
proof -
    from run cmt have card \(\{q \cdot \operatorname{coord} \Phi(\) rho \(r q)=p\}>N\) div 2
        by (blast elim: commitE)
    moreover
    from run \(c m t^{\prime}\) have card \(\left\{q \cdot \operatorname{coord} \Phi(\right.\) rho \(\left.r q)=p^{\prime}\right\}>N\) div 2
        by (blast elim: commitE)
    ultimately
    obtain \(q\) where \(\operatorname{coord} \Phi(\) rho \(r q)=p\) and \(p^{\prime}=\operatorname{coord} \Phi(\) rho r \(q)\)
    by (auto elim: majorities \(E^{\prime}\) )
    thus ?thesis by simp
```


## qed

No two different processes have their ready flag set at any state.

```
lemma readyProcsEqual:
    assumes run: CHORun LV-M rho HOs coords
    and rdy: ready (rho r p) and rdy': ready (rho r \(p^{\prime}\) )
    shows \(p=p^{\prime}\)
proof -
    let ? \(C p=\{q \cdot \operatorname{coord} \Phi(\) rho \(r q)=p \wedge\) timestamp \((\) rho \(r q)=\) Suc \((p h a s e r)\}\)
    from run rdy have card \((\) ? \(C p)>N\) div 2
        by (blast elim: readyE)
    moreover
    from run \(r d y^{\prime}\) have card \(\left(? C p^{\prime}\right)>N\) div 2
        by (blast elim: readyE)
    ultimately
    obtain \(q\) where \(\operatorname{coord} \Phi(\) rho \(r q)=p\) and \(p^{\prime}=\operatorname{coord} \Phi(\) rho \(r q)\)
    by (auto elim: majorities \(E^{\prime}\) )
    thus ?thesis by simp
qed
```

The following lemma asserts that whenever a process $p$ commits at a state where a majority of processes have a timestamp beyond $t s$, then $p$ votes for a value held by some process whose timestamp is beyond $t s$.

```
lemma commitThenVoteRecent:
    assumes run: CHORun LV-M rho HOs coords
    and maj: card (procsBeyondTS ts (rho r)) > N div 2
    and cmt: commt (rho rp)
    shows \(\exists q \in\) procsBeyondTS ts (rho r). vote (rho r \(p\) ) \(=\) Some ( \(x\) (rhor \(q\) ) )
        (is? \(Q r\) )
proof -
    let ?bynd \(n=\) procsBeyondTS ts (rho \(n\) )
    have card (?bynd \(r\) ) >N div \(2 \wedge\) commt (rho r \(p\) ) \(\longrightarrow\) ?Q \(r\) (is ?P pr)
    proof (rule LV-induct[OF run])
```

next0 establishes the property
fix $n$
assume stp: step $n=0$
and $n x t: \forall q$. next $0 n q($ rho $n q)$

```
                                    (HOrcudMsgs LV-M \(n q(H O s n q)(\) rho \(n))\)
                    (coords (Suc n) q)
                    (rho (Suc n) q)
```

                    (is \(\forall q\). ?nxt q)
    from \(n x t\) have \(n x p\) : ?nxt \(p\)..
    show ?P \(p\) (Suc n)
    proof (clarify)
        assume mj: card (?bynd (Suc n)) > N div 2
            and ct: commt (rho (Suc n) p)
        show ?Q (Suc n)
        proof -
    ```
let ?msgs = HOrcvdMsgs LV-M n p (HOs n p) (rho n)
from stp run have ᄀ commt (rho n p) by (auto elim: commitE)
with nxp ct obtain qv where
    v: ?msgs q = Some (ValStamp v (highestStampRcvd ?msgs)) and
    vote: vote (rho (Suc n) p)=Some v and
    rcvd: card (valStampsRcvd ?msgs) > N div 2
    by (auto simp: next0-def)
from mj rcvd obtain q' where
    q1': q' }\in\mathrm{ ?bynd (Suc n) and q2': q' 
    by (rule majoritiesE')
have timestamp (rho n q}\mp@subsup{q}{}{\prime})\leq\mathrm{ timestamp (rho n q)
proof -
    from q2' obtain v' ts'
        where ts':?msgs q' = Some (ValStamp v'ts')
        by (auto simp: valStampsRcvd-def)
    hence ts'\leq highestStampRcvd ?msgs
    by (rule highestStampRcvd-max)
    moreover
    from ts'stp have timestamp (rho n q}\mp@subsup{q}{}{\prime})=t\mp@subsup{s}{}{\prime
        by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def
                            LV-sendMsg-def send0-def)
    moreover
    from v stp have timestamp (rho n q) = highestStampRcvd ?msgs
    by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def
                LV-sendMsg-def send0-def)
    ultimately
    show ?thesis by simp
qed
moreover
from run stp
have timestamp (rho (Suc n) q') = timestamp (rho n q
    by (simp add: notStep1EqualTimestamp)
moreover
from run stp
have timestamp (rho (Suc n) q) = timestamp (rho n q)
    by (simp add: notStep1EqualTimestamp)
moreover
note q1'
ultimately
have q}\in\mathrm{ ?bynd (Suc n)
    by (simp add: procsBeyondTS-def)
moreover
from v vote stp
have vote (rho (Suc n) p) = Some (x (rho n q))
    by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def
        LV-sendMsg-def send0-def)
moreover
from run stp have x (rho (Suc n) q) = x (rho n q)
    by (simp add: notStep1EqualX)
```

```
        ultimately
        show ?thesis by force
        qed
    qed
```

next

We now prove that next1 preserves the property. Observe that next1 may establish a majority of processes with current timestamps, so we cannot just refer to the induction hypothesis. However, if that happens, there is at least one process with a fresh timestamp that copies the vote of the (only) committed coordinator, thus establishing the property.
fix $n$
assume stp: step $n=1$
and $n x t: \forall q$. next1 $n q($ rho $n q)$
(HOrcvdMsgs LV-M $n q($ HOs $n q)($ rho $n))$
(coords (Suc n) q)
(rho (Suc n) q)
(is $\forall q$. ?nxt $q$ )
and $i h$ : ? P $p n$
from $n x t$ have $n x p$ : ?nxt $p$..
show ?P $p$ (Suc n)
proof (clarify)
assume $m j^{\prime}$ : card (?bynd (Suc n)) > N div 2
and $c^{\prime}$ ': commt (rho (Suc n) p)
from run stp $c t^{\prime}$ have ct: commt (rho n p)
by (simp add: notStep03EqualCommit)
from run stp have vote': vote (rho (Suc n) p) = vote (rho n p)
by (simp add: notStep0EqualVote)
show ?Q (Suc n)
proof (cases $\exists q \in$ ?bynd (Suc n). rho (Suc n) $q \neq$ rho $n q$ ) case True
in this case the property holds because $q$ updates its $x$ field to the vote

```
then obtain \(q\) where
    \(q 1: q \in\) ?bynd (Suc n) and \(q 2\) : rho (Suc n) \(q \neq\) rho \(n ~ q .\).
from \(n x t\) have ?nxt \(q\)..
with q2 stp
have \(x^{\prime}: x(\) rho \((\) Suc \(n) q)=\) the \((\) vote \((\) rho \(n(\operatorname{coord} \Phi(\) rho \(n q))))\)
    and coord: commt (rho \(n(\operatorname{coord} \Phi(\) rho \(n q))\) )
    by (auto simp: next1-def send1-def LV-CHOMachine-def HOrcvdMsgs-def
                                    LV-sendMsg-def is Vote-def)
    from run ct have vote: vote (rho \(n\) p) \(\neq\) None
    by (rule commitE)
    from run coord \(c t\) have \(\operatorname{coord} \Phi(\) rho \(n q)=p\)
    by (rule committedProcsEqual)
    with \(q 1 x^{\prime}\) vote vote' show ?thesis by auto
next
    case False
```

if no relevant process moves then procsBeyondTS doesn't change and we invoke the induction hypothesis

```
    hence bynd: ?bynd (Suc \(n\) ) = ?bynd \(n\)
    proof (auto simp: procsBeyondTS-def)
            fix \(r\)
            assume ts: ts timestamp (rho \(n\) r)
            from run have timestamp (rho \(n r\) ) \(\leq\) timestamp (rho (Suc n) r)
                by (simp add: LV-timestamp-monotonic)
            with \(t s\) show \(t s \leq\) timestamp (rho (Suc n) r) by simp
        qed
        with \(m j^{\prime}\) have \(m j\) : card (?bynd \(n\) ) \(>N\) div 2 by \(\operatorname{simp}\)
        with ct ih obtain \(q\) where
            \(q \in\) ?bynd \(n\) and vote (rho \(n p\) ) \(=\) Some \((x(\) rho \(n q))\)
            by blast
        with vote' bynd False show ?thesis by auto
        qed
```

    qed
    next
    step2 preserves the property, via the induction hypothesis.
fix $n$
assume stp: step $n=2$
and $n x t$ : $\forall q$. next2 $n q($ rho $n q)$
(HOrcvdMsgs LV-M $n q($ HOs $n q)($ rho $n)$ )
(coords (Suc n) q)
(rho (Suc n) q)
(is $\forall q$. ?nxt $q$ )
and $i h$ : ? $P$ p $n$
from nxt have nxp: ?nxt $p$..
show ?P $p$ (Suc n)
proof (clarify)
assume $m j^{\prime}:$ card (?bynd (Suc n)) $>N$ div 2
and ct $^{\prime}$ : commt (rho (Suc n) p)
from run stp $c t^{\prime}$ have ct: commt (rho n $p$ )
by (simp add: notStep03EqualCommit)
from run stp have vote': vote (rho (Suc n) p) = vote (rho n $p$ )
by (simp add: notStep0EqualVote)
from run stp have $\forall$. timestamp (rho (Suc n) q) = timestamp (rho n q)
by (simp add: notStep1EqualTimestamp)
hence bynd': ?bynd (Suc n) = ?bynd $n$
by (auto simp add: procsBeyondTS-def)
from run stp have $\forall q$. $x($ rho $($ Suc $n) q)=x($ rho $n q)$
by (simp add: notStep1EqualX)
with bynd' vote' ct mj' ih show ?Q (Suc n)
by auto
qed
the initial state and the step3 transition are trivial because the commt flag cannot be set.
qed (auto elim: commitE[OF run])
with maj cmt show? ?thesis by simp
qed
The following lemma gives the crucial argument for agreement: after some process $p$ has decided, all processes whose timestamp is beyond the timestamp at the point of decision contain the decision value in their $x$ field.

```
lemma XOfTimestampBeyondDecision:
    assumes run: CHORun LV-M rho HOs coords
    and dec: decide (rho (Suc r) p) \(\neq\) decide (rho r \(p\) )
    shows \(\forall q \in\) procsBeyondTS (Suc (phase \(r\) )) (rho \((r+k)\) ).
        \(x(r h o(r+k) q)=\) the \((\) decide (rho \((\) Suc \(r) p))\)
    (is \(\forall q \in ? b y n d k .=? v\) is ? \(P\) p \(k\) )
proof (induct \(k\) )
    - base step
    show ? P \(p 0\)
    proof (clarify)
    fix \(q\)
    assume \(q: q \in\) ?bynd 0
```

use preceding lemmas about the decision value and the $x$ field of processes with fresh timestamps

```
    from run dec
    have stp: step \(r=3\)
    and \(v\) : decide \((\) rho \((\) Suc \(r) p)=\) Some \((\) the \((\operatorname{vote}(\operatorname{rho} r(\operatorname{coord} \Phi(\) rho r \(p)))))\)
    and cmt: commt (rho r (coord \(\Phi\) (rho r p)))
    by (auto elim: decisionE)
    from stp \(L V\)-timestamp-bounded [OF run, where \(n=r\) ]
    have timestamp (rho \(r q\) ) \(\leq\) Suc (phase \(r\) ) by simp
    with \(q\) have timestamp (rho r \(q\) ) \(=\) Suc (phase \(r\) )
    by (simp add: procsBeyondTS-def)
    with run
    have \(x: x(\) rho \(r q)=\) the \((\) vote \((\) rho \(r(\operatorname{coord} \Phi(\) rho r \(q))))\)
        and \(\mathrm{cmt}^{\prime}\) : commt (rho r (coord \(\Phi(\) rho \(\left.r q)\right)\) )
        by (auto elim: currentTimestampE)
    from run cmt \(c m t^{\prime}\) have \(\operatorname{coord} \Phi(\) rho r \(p)=\operatorname{coord} \Phi(\) rho \(r q)\)
        by (rule committedProcsEqual)
    with \(x v\) show \(x(r h o(r+0) q)=? v\) by \(\operatorname{simp}\)
qed
next
- induction step
fix \(k\)
assume \(i h: ? P p k\)
show ?P \(p\) (Suc \(k\) )
proof (clarify)
    fix \(q\)
    assume \(q: q \in\) ?bynd (Suc k)
    - distinguish the kind of transition - only step 1 is interesting
    have \(x(r h o(S u c(r+k)) q)=? v\) (is ? \(X q(r+k))\)
```

```
proof (rule LV-Suc'[OF run, where P=?X])
    assume stp: step (r +k)=1
    and nxt: next1 (r+k)q(rho (r+k)q)
                                    (HOrcvdMsgs LV-M (r+k) q(HOs (r+k) q) (rho (r+k)))
                                    (coords (Suc (r+k)) q)
                                    (rho (Suc (r+k)) q)
    show ?thesis
    proof (cases rho (Suc (r+k)) q = rho (r+k)q)
        case True
        with q ih show ?thesis by (auto simp: procsBeyondTS-def)
    next
        case False
        from run dec have card (?bynd 0) > N div 2
        by (simp add: decisionThenMajorityBeyondTS)
    moreover
    have ?bynd 0}\subseteq\mathrm{ ?bynd }
        by (auto elim: procsBeyondTS-monotonic[OF run])
    hence card (?bynd 0) \leq card (?bynd k)
        by (auto intro: card-mono)
    ultimately
    have maj: card (?bynd k) > N div 2 by simp
    let ?crd = coord \Phi (rho (r+k) q)
    from False stp nxt have
        cmt: commt (rho (r+k)?crd) and
        x: x (rho (Suc (r+k)) q) = the (vote (rho (r+k) ?crd))
        by (auto simp: next1-def LV-CHOMachine-def HOrcvdMsgs-def
                            LV-sendMsg-def send1-def isVote-def)
    from run maj cmt stp obtain q'
        where q1': q' }\mp@subsup{q}{}{\prime}\in\mathrm{ ?bynd }
            and q2': vote (rho (r+k)?crd) = Some (x (rho (r+k) q'))
            by (blast dest: commitThenVoteRecent)
    with x ih show ?thesis by auto
    qed
next
    - all other steps hold by induction hypothesis
    assume step (r+k)=0
    with run have x: x (rho (Suc (r+k)) q) =x (rho (r+k)q)
        and ts: timestamp (rho (Suc (r+k)) q) = timestamp (rho (r+k)q)
        by (auto simp: notStep1EqualX notStep1EqualTimestamp)
    from ts q have q\in ?bynd k
        by (auto simp: procsBeyondTS-def)
    with x ih show ?thesis by auto
next
    assume step (r+k)=2
    with run have x: x (rho (Suc (r+k)) q) =x (rho (r+k)q)
    and ts: timestamp (rho (Suc (r+k)) q) = timestamp (rho (r+k)q)
    by (auto simp: notStep1EqualX notStep1EqualTimestamp)
    from ts q have q\in ?bynd k
    by (auto simp: procsBeyondTS-def)
```

```
            with \(x\) ih show ?thesis by auto
    next
            assume step \((r+k)=3\)
            with run have \(x\) : \(x(\) rho \((S u c(r+k)) q)=x(r h o(r+k) q)\)
                and ts: timestamp (rho \((S u c(r+k)) q)=\) timestamp \((r h o(r+k) q)\)
                by (auto simp: notStep1EqualX notStep1EqualTimestamp)
            from ts \(q\) have \(q \in\) ? bynd \(k\)
                by (auto simp: procsBeyondTS-def)
            with \(x\) ih show ?thesis by auto
    qed
    thus \(x(r h o(r+S u c k) q)=? v\) by simp
qed
qed
```

We are now in position to prove Agreement: if some process decides at step $r$ and another (or possibly the same) process decides at step $r+k$ then they decide the same value.

```
lemma laterProcessDecidesSameValue:
    assumes run: CHORun LV-M rho HOs coords
    and \(p\) : decide (rho (Suc r) p) \(\neq\) decide (rho r \(p\) )
    and \(q\) : decide (rho \((S u c(r+k)) q) \neq\) decide \((r h o(r+k) q)\)
    shows decide (rho \((\) Suc \((r+k)) q\) ) \(=\) decide (rho (Suc r) p)
proof -
    let ?bynd \(k=\) procsBeyondTS (Suc (phase \(r)\) ) (rho \((r+k))\)
    let ? \(q\) crd \(=\operatorname{coord} \Phi(r h o(r+k) q)\)
    from run \(p\) have notNone: decide (rho (Suc r) p) \(\neq\) None
    by (auto elim: decisionE)
    - process \(q\) decides on the vote of its coordinator
    from run \(q\)
    have dec: decide (rho \((\operatorname{Suc}(r+k)) q)=\) Some \((\) the \((\operatorname{vote}(r h o(r+k)\) ?qcrd \()))\)
    and cmt: commt (rho ( \(r+k\) ) ?qcrd)
    by (auto elim: decisionE)
    - that vote is the \(x\) field of some process \(q^{\prime}\) with a recent timestamp
    from run \(p\) have card (?bynd 0) \(>N\) div 2
        by (simp add: decisionThenMajorityBeyondTS)
    moreover
    from run have ?bynd \(0 \subseteq\) ?bynd \(k\)
    by (auto elim: procsBeyondTS-monotonic)
    hence card (?bynd 0) \(\leq\) card (?bynd \(k\) )
    by (auto intro: card-mono)
    ultimately
    have maj: card (?bynd \(k\) ) \(>N\) div 2 by simp
    from run maj cmt obtain \(q^{\prime}\)
            where \(q^{\prime} 1: q^{\prime} \in\) ? bynd \(k\)
            and \(q^{\prime} 2:\) vote \((r h o(r+k) ? q c r d)=\) Some \(\left(x\left(r h o(r+k) q^{\prime}\right)\right)\)
    by (auto dest: commitThenVoteRecent)
    - the \(x\) field of process \(q^{\prime}\) is the value \(p\) decided on
    from run \(p q^{\prime} 1\)
    have \(x\left(r h o(r+k) q^{\prime}\right)=\) the \((\) decide \((\) rho \((\) Suc \(r) p))\)
```

```
    by (auto dest: XOfTimestampBeyondDecision)
    - which proves the assertion
    with dec q'2 notNone show ?thesis by auto
qed
```

A process that holds some decision $v$ has decided $v$ sometime in the past.

```
lemma decisionNonNullThenDecided:
```

    assumes run: CHORun LV-M rho HOs coords
        and dec: decide (rho \(n\) p) \(=\) Some \(v\)
    shows \(\exists m<n\). decide (rho (Suc m) \(p\) ) \(\neq\) decide (rho \(m p\) )
                \(\wedge\) decide (rho \((\) Suc \(m) p\) ) \(=\) Some \(v\)
    proof -
let ?dec $k=$ decide (rho $k p$ )
have $(\forall m<n$. ?dec $($ Suc $m) \neq$ ?dec $m \longrightarrow$ ?dec $($ Suc $m) \neq$ Some $v)$
$\longrightarrow$ ?dec $n \neq$ Some $v$
(is? $P n$ is ? $A n \longrightarrow-$ )
proof (induct $n$ )
from run show ?P 0
by (auto simp: CHORun-eq LV-CHOMachine-def
CHOinitConfig-def LV-initState-def)
next
fix $n$
assume $i h: ? P n$
show ?P (Suc n)
proof (clarify)
assume p: ?A (Suc n) and $v$ : ?dec (Suc $n$ ) $=$ Some $v$
from $p$ have? A $n$ by simp
with ih have ?dec $n \neq$ Some $v$ by simp
moreover
from $p$
have ?dec $($ Suc $n) \neq$ ? dec $n \longrightarrow$ ? dec $($ Suc $n) \neq$ Some $v$ by simp
ultimately
have ?dec (Suc $n$ ) $\neq$ Some $v$ by auto
with $v$ show False by simp
qed
qed
with dec show ?thesis by auto
qed

Irrevocability and Agreement are straightforward consequences of the two preceding lemmas.

```
theorem lv-irrevocability:
    assumes run: CHORun LV-M rho HOs coords
            and \(p\) : decide (rho \(m p\) ) \(=\) Some \(v\)
    shows decide (rho \((m+k) p\) ) \(=\) Some \(v\)
proof -
    from run \(p\) obtain \(n\) where
            \(n 1: n<m\) and
            n2: decide (rho (Suc n) \(p\) ) \(\neq\) decide (rho \(n p\) ) and
```

```
    n3: decide (rho (Suc n) p) = Some v
    by (auto dest: decisionNonNullThenDecided)
    have }\foralli\mathrm{ . decide (rho (Suc (n+i)) p)=Some v (is }\foralli\mathrm{ . ?dec i)
    proof
    fix }
    show ?dec i
    proof (induct i)
        from n3 show ?dec 0 by simp
    next
        fix }
        assume ih:?dec j
        show ?dec (Suc j)
        proof (rule ccontr)
            assume ctr: \neg (?dec (Suc j))
            with ih
            have decide (rho (Suc (n+Suc j)) p)\not= decide (rho (n+Suc j) p)
                by simp
            with run n2
            have decide (rho (Suc (n+Suc j)) p)= decide (rho (Suc n) p)
            by (rule laterProcessDecidesSameValue)
            with ctr n3 show False by simp
        qed
    qed
qed
moreover
from n1 obtain j where m+k=Suc(n+j)
    by (auto dest: less-imp-Suc-add)
    ultimately
    show ?thesis by auto
qed
theorem lv-agreement:
    assumes run: CHORun LV-M rho HOs coords
        and p: decide (rho m p)=Some v
        and q: decide (rho n q) = Some w
    shows v=w
proof -
    from run p obtain k
        where k1: decide (rho (Suc k) p)\not= decide (rho k p)
            and k2: decide (rho (Suc k) p)=Some v
    by (auto dest: decisionNonNullThenDecided)
    from run q obtain l
        where l1: decide (rho (Suc l) q) = decide (rho l q)
            and l2: decide (rho (Suc l) q) = Some w
    by (auto dest: decisionNonNullThenDecided)
    show ?thesis
    proof (cases k\leql)
    case True
    then obtain m}\mathrm{ where m:l=k+m by (auto simp:le-iff-add)
```

```
    from run k1 l1 m
    have decide (rho (Suc l)q) = decide (rho (Suc k) p)
        by (auto elim: laterProcessDecidesSameValue)
    with k2l2 show ?thesis by simp
next
    case False
    hence l\leqk by simp
    then obtain m where m:k=l+m by (auto simp:le-iff-add)
    from run l1 k1 m
    have decide (rho (Suc k) p)= decide (rho (Suc l) q)
        by (auto elim: laterProcessDecidesSameValue)
    with l2 k2 show ?thesis by simp
    qed
qed
```


### 7.9 Proof of Termination

The proof of termination relies on the communication predicate, which stipulates the existence of some phase during which there is a single coordinator that (a) receives a majority of messages and (b) is heard by everybody. Therefore, all processes successfully execute the protocol, deciding at step 3 of that phase.

```
theorem lv-termination:
    assumes run: CHORun LV-M rho HOs coords
        and commG:CHOcommGlobal LV-M HOs coords
    shows }\existsr.\forallp\mathrm{ . decide (rho r p)}\not=\mathrm{ None
proof -
```

The communication predicate implies the existence of a "successful" phase $p h$, coordinated by some process $c$ for all processes.

```
from comm \(G\) obtain \(p h c\)
    where \(c: \forall p\). coords \((4 * p h) p=c\)
    and maj0: card \((H O s(4 * p h) c)>N \operatorname{div} 2\)
    and maj2: card \((H O s(4 * p h+2) c)>N \operatorname{div} 2\)
    and rcv1: \(\forall p . c \in H O s(4 * p h+1) p\)
    and rcv3: \(\forall p . c \in H O s(4 * p h+3) p\)
    by (auto simp: LV-CHOMachine-def LV-commGlobal-def)
let \(? r 0=4 * p h\)
let \(? r 1=\) Suc \(? r 0\)
let \(? r 2=\) Suc \(? r 1\)
let \(?\) r \(3=\) Suc \(?\) ? 2
let \(?_{r} r_{4}=S u c\) ? \(r_{3}\)
```

Process $c$ is the coordinator of all steps of phase $p h$.

```
from run c have c}\mp@subsup{c}{}{\prime}:\forallp.\operatorname{coord}\Phi(rho ?r p)=
    by (auto simp add: phase-def coordinators)
with run have c1: }\forall\textrm{p}.\operatorname{coord}\Phi(rho ?r1 p)=
    by (auto simp add: step-def mod-Suc notStep3EqualCoord)
```

```
with run have \(c 2: \forall p\). \(\operatorname{coord} \Phi(r h o\) ? \(r 2 p)=c\)
    by (auto simp add: step-def mod-Suc notStep3EqualCoord)
with run have \(c 3: \forall p\). \(\operatorname{coord} \Phi(r h o\) ? \(r 3\) p \()=c\)
    by (auto simp add: step-def mod-Suc notStep3EqualCoord)
```

The coordinator receives ValStamp messages from a majority of processes at step 0 of phase $p h$ and therefore commits during the transition at the end of step 0 .

```
have 1: commt (rho ?r1 c) (is ?P \(c(4 * p h)\) )
proof (rule \(L V\)-Suc' \([\) OF run, where \(P=? P]\), auto simp: step-def)
    assume next0 ?r c (rho ?r c) (HOrcvdMsgs LV-M ?r c (HOs ?r c) (rho ?r))
            (coords (Suc ?r) c) (rho (Suc ?r) c)
    with \(c^{\prime}\) maj0 show commt (rho (Suc ?r) c)
        by (auto simp: step-def next0-def send0-def valStampsRcvd-def
                                LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def)
qed
```

All processes receive the vote of $c$ at step 1 and therefore update their time stamps during the transition at the end of step 1.

```
have 2: \(\forall\) p. timestamp (rho ?r2 \(p\) ) \(=\) Suc \(p h\)
proof
    fix \(p\)
    let ?msgs = HOrcvdMsgs LV-M ?r1 p (HOs ?r1 p) (rho ?r1)
    let ?crd \(=\operatorname{coord} \Phi(r h o\) ?r1 p)
    from run 1 c1 rcv1
    have cnd: ?msgs ?crd \(\neq\) None \(\wedge\) isVote (the (?msgs ?crd))
        by (auto elim: commitE
                        simp: step-def LV-CHOMachine-def HOrcvdMsgs-def
                            LV-sendMsg-def send1-def is Vote-def)
    show timestamp (rho ?r2 p) \(=\) Suc ph (is ?P p (Suc (4*ph)))
    proof (rule LV-Suc' \([\) OF run, where \(P=\) ? P], auto simp: step-def mod-Suc)
        assume next1 ?r1 p (rho ?r1 p) ?msgs (coords (Suc ?r1) p) (rho ?r2 p)
        with cnd show ?thesis by (auto simp: next1-def phase-def)
    qed
qed
```

The coordinator receives acknowledgements from a majority of processes at step 2 and sets its ready flag during the transition at the end of step 2.

```
have 3: ready (rho ?r3 c) (is ?P c (Suc (Suc (4*ph))))
proof (rule LV-Suc' \([\) OF run, where \(P=\) ?P], auto simp: step-def mod-Suc)
    assume next2 ?r2 c (rho ?r2 c)
                            (HOrcvdMsgs LV-M ?r2 c (HOs ?r2 c) (rho ?r2))
                            (coords (Suc ?r2) c) (rho ?r3 c)
    with 2 c2 maj2 show ?thesis
        by (auto simp: mod-Suc step-def LV-CHOMachine-def HOrcvdMsgs-def
                    LV-sendMsg-def next2-def send2-def acksRcvd-def
                        isAck-def phase-def)
qed
```

All processes receive the vote of the coordinator during step 3 and decide during the transition at the end of that step.

```
have 4: \(\forall p\). decide (rho ?r4 \(p\) ) \(\neq\) None
proof
    fix \(p\)
    let ?msgs \(=\) HOrcvdMsgs LV-M ?r3 \(p(H O s\) ?r3 \(p)(r h o\) ? r3)
    let ? crd \(=\operatorname{coord} \Phi(\) rho ? \(r 3\) p \()\)
    from run 3 c3 rcv3
    have cnd: ?msgs ?crd \(\neq\) None \(\wedge\) isVote (the (?msgs ?crd))
        by (auto elim: readyE
                    simp: step-def mod-Suc LV-CHOMachine-def HOrcvdMsgs-def
                                    LV-sendMsg-def send3-def is Vote-def numeral-3-eq-3)
    show decide (rho ?r4 \(p\) ) \(\neq\) None (is ?P p \((S u c(S u c(S u c(4 * p h))))\) )
    proof (rule LV-Suc' \([O F\) run, where \(P=\) ? P], auto simp: step-def mod-Suc)
        assume next3 ? r3 \(p\) (rho ?r3 p) ?msgs (coords (Suc ?r3) p) (rho ?r4 p)
        with cnd show \(\exists v\). decide (rho ? \(r_{4} p\) ) \(=\) Some \(v\)
            by (auto simp: next3-def)
    qed
qed
```

This immediately proves the assertion.
from 4 show ?thesis ..
qed

### 7.10 LastVoting Solves Consensus

Summing up, all (coarse-grained) runs of LastVoting for HO collections that satisfy the communication predicate satisfy the Consensus property.
theorem lv-consensus:
assumes run: CHORun LV-M rho HOs coords and commG: CHOcommGlobal LV-M HOs coords
shows consensus ( $x \circ(r h o 0)$ ) decide rho
proof -

- the above statement of termination is stronger than what we need
from lv-termination[OF assms]
obtain $r$ where $\forall p$. decide (rho $r p) \neq$ None ..
hence $\forall p$. $\exists r$. decide (rho $r p$ ) $\neq$ None by blast
with lv-integrity[OF run] lv-agreement[OF run]
show ?thesis by (auto simp: consensus-def image-def)
qed
By the reduction theorem, the correctness of the algorithm carries over to the fine-grained model of runs.

```
theorem lv-consensus-fg:
    assumes run: fg-run LV-M rho HOs HOs coords
        and commG:CHOcommGlobal LV-M HOs coords
    shows consensus ( }\lambdap.x(\mathrm{ state (rho 0) p)) decide (state ○ rho)
    (is consensus ?inits - -)
proof (rule local-property-reduction[OF run consensus-is-local])
    fix crun
```

```
    assume crun: CSHORun LV-M crun HOs HOs coords
        and init: crun 0 = state (rho 0)
    from crun have CHORun LV-M crun HOs coords
        by (unfold CHORun-def SHORun-def)
    from this commG have consensus ( }x\circ(\mathrm{ crun 0)) decide crun
        by (rule lv-consensus)
    with init show consensus ?inits decide crun
    by (simp add: o-def)
qed
end
theory UteDefs
imports ../HOModel
begin
```


## 8 Verification of the $\mathcal{U}_{T, E, \alpha}$ Consensus Algorithm

Algorithm $\mathcal{U}_{T, E, \alpha}$ is presented in [3]. It is an uncoordinated algorithm that tolerates value (a.k.a. Byzantine) faults, and can be understood as a variant of Uniform Voting. The parameters $T, E$, and $\alpha$ appear as thresholds of the algorithm and in the communication predicates. Their values can be chosen within certain bounds in order to adapt the algorithm to the characteristics of different systems.
We formalize in Isabelle the correctness proof of the algorithm that appears in [3], using the framework of theory HOModel.

### 8.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic HO model.

```
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
abbreviation
    N \equivcard (UNIV::Proc set) - number of processes
```

The algorithm proceeds in phases of 2 rounds each (we call steps the individual rounds that constitute a phase). The following utility functions compute the phase and step of a round, given the round number.

```
abbreviation
    nSteps \equiv2
definition phase where phase (r::nat) \equivr div nSteps
definition step where step (r::nat) \equivr mod nSteps
lemma phase-zero [simp]: phase 0 = 0
```

```
by (simp add: phase-def)
lemma step-zero [simp]: step \(0=0\)
by (simp add: step-def)
lemma phase-step: (phase \(r *\) nSteps \()+\) step \(r=r\)
    by (auto simp add: phase-def step-def)
```

The following record models the local state of a process.

```
record 'val pstate =
    x :: 'val - current value held by process
    vote :: 'val option - value the process voted for, if any
    decide :: 'val option - value the process has decided on, if any
```

Possible messages sent during the execution of the algorithm.

```
datatype 'val msg=
    Val 'val
| Vote 'val option
```

The $x$ field of the initial state is unconstrained, all other fields are initialized appropriately.

```
definition Ute-initState where
    Ute-initState p st \equiv
    (vote st = None) ^(decide st = None)
```

The following locale introduces the parameters used for the $\mathcal{U}_{T, E, \alpha}$ algorithm and their constraints [3].

```
locale ute-parameters \(=\)
    fixes \(\alpha:: n a t\) and \(T::\) nat and \(E:: n a t\)
    assumes majE: \(2 * E \geq N+2 * \alpha\)
        and \(\operatorname{majT}: 2 * T \geq N+2 * \alpha\)
        and EltN: \(E<N\)
        and \(T l t N: T<N\)
begin
```

Simple consequences of the above parameter constraints.
lemma alpha-lt-N: $\alpha<N$
using EltN majE by auto
lemma alpha-lt-T: $\alpha<T$
using majT alpha-lt-N by auto
lemma alpha-lt-E: $\alpha<E$
using majE alpha-lt- $N$ by auto
We separately define the transition predicates and the send functions for each step and later combine them to define the overall next-state relation.

In step 0 , each process sends its current $x$. If it receives the value $v$ more than $T$ times, it votes for $v$, otherwise it doesn't vote.

## definition

```
    send0 :: nat \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) 'val msg
where
    send0 rpqst \(\equiv \operatorname{Val}(x s t)\)
```


## definition

$$
\begin{gathered}
\text { next0 }:: \text { nat } \Rightarrow \text { Proc } \Rightarrow \text { 'val pstate } \Rightarrow \text { (Proc } \Rightarrow \text { 'val msg option }) \\
\Rightarrow \text { 'val pstate } \Rightarrow \text { bool }
\end{gathered}
$$

where

```
next0 r p st msgs st \({ }^{\prime} \equiv\)
    \(\left(\exists\right.\) v. card \(\{q\). msgs \(q=\) Some \((\) Val \(v)\}>T \wedge\) st \(t^{\prime}=\) st \(\cap\) vote \(:=\) Some \(\left.v \downarrow\right)\)
    \(\vee \neg(\exists v\). card \(\{q\). msgs \(q=\) Some \((\) Val \(v)\}>T) \wedge\) st \(t^{\prime}=\) st ( vote \(:=\) None )
```

In step 1, each process sends its current vote.
If it receives more than $\alpha$ votes for a given value $v$, it sets its $x$ field to $v$, else it sets $x$ to a default value.
If the process receives more than $E$ votes for $v$, it decides $v$, otherwise it leaves its decision unchanged.

## definition

```
    send1 \(::\) nat \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) 'val msg
where
    send1 rpqst \(\equiv\) Vote (vote st)
```


## definition

$$
\begin{gathered}
\text { next1 }:: \text { nat } \Rightarrow \text { Proc } \Rightarrow \text { 'val pstate } \Rightarrow(\text { Proc } \Rightarrow \text { 'val msg option }) \\
\quad \Rightarrow \text { 'val pstate } \Rightarrow \text { bool }
\end{gathered}
$$

## where

```
next1 r p st msgs st' }
```

    \(\left(\left(\exists v\right.\right.\). card \(\{q\). msgs \(q=\) Some \((\operatorname{Vote}(\) Some \(v))\}>\alpha \wedge x\) st \(\left.t^{\prime}=v\right)\)
        \(\vee \neg(\exists v\). card \(\{q\). msgs \(q=\) Some \((\) Vote \((\) Some \(v))\}>\alpha)\)
            \(\wedge x s t^{\prime}=\) undefined \()\)
    $\wedge\left(\left(\exists v\right.\right.$. card $\{q$. msgs $q=$ Some $($ Vote $($ Some $v))\}>E \wedge$ decide st ${ }^{\prime}=$ Some $\left.v\right)$
$\vee \neg(\exists v$. card $\{q$. msgs $q=$ Some $($ Vote $($ Some $v))\}>E)$
$\wedge$ decide st ${ }^{\prime}=$ decide st $)$
$\wedge$ vote $s t^{\prime}=$ None

The overall send function and next-state relation are simply obtained as the composition of the individual relations defined above.

```
definition
    Ute-sendMsg :: nat \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) 'val msg
where
    Ute-sendMsg (r::nat) \(\equiv\) if step \(r=0\) then send0 \(r\) else send1 \(r\)
definition
    Ute-nextState :: nat \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) (Proc \(\Rightarrow{ }^{\prime}\) 'val msg option \()\)
```

$$
\Rightarrow \text { 'val pstate } \Rightarrow \text { bool }
$$

where
Ute-nextState $r \equiv$ if step $r=0$ then next0 $r$ else next1 $r$

### 8.2 Communication Predicate for $\mathcal{U}_{T, E, \alpha}$

Following [3], we now define the communication predicate for the $\mathcal{U}_{T, E, \alpha}$ algorithm to be correct.
The round-by-round predicate stipulates the following conditions:

- no process may receive more than $\alpha$ corrupted messages, and
- every process should receive more than $\max (T, N+2 * \alpha-E-1)$ correct messages.
[3] also requires that every process should receive more than $\alpha$ correct messages, but this is implied, since $T>\alpha$ (cf. lemma alpha-lt-T).

```
definition Ute-commPerRd where
    Ute-commPerRd HOrs SHOrs \equiv
    \forall.card (HOrs p - SHOrs p) \leq\alpha
        card (SHOrs p \capHOrs p)>N+2*\alpha-E-1
        ^card (SHOrs p\capHOrs p)>T
```

The global communication predicate requires there exists some phase $\Phi$ such that:

- all HO and SHO sets of all processes are equal in the second step of phase $\Phi$, i.e. all processes receive messages from the same set of processes, and none of these messages is corrupted,
- every process receives more than $T$ correct messages in the first step of phase $\Phi+1$, and
- every process receives more than $E$ correct messages in the second step of phase $\Phi+1$.

The predicate in the article [3] requires infinitely many such phases, but one is clearly enough.

```
definition Ute-commGlobal where
    Ute-commGlobal HOs \(\mathrm{SHOs} \equiv\)
    \(\exists\). . let \(r=\) Suc ( \(n\) Steps \(* \Phi)\)
        in \((\exists \pi . \forall p . \pi=H O s r p \wedge \pi=\) SHOs \(r p)\)
            \(\wedge(\forall p\). card \((\) SHOs \((\) Suc r) \(p \cap H O s(\) Suc r) \(p)>T)\)
            \(\wedge(\forall p\).card \((\) SHOs \((\) Suc \((\) Suc r) \() p \cap \operatorname{HOs}(\) Suc \((\) Suc r) \() p)>E))\)
```


### 8.3 The $\mathcal{U}_{T, E, \alpha}$ Heard-Of Machine

We now define the coordinated HO machine for the $\mathcal{U}_{T, E, \alpha}$ algorithm by assembling the algorithm definition and its communication-predicate.

```
definition Ute-SHOMachine where
    Ute-SHOMachine = 0
        CinitState = ( }\lambda\mathrm{ p st crd. Ute-initState p st),
        sendMsg = Ute-sendMsg,
        CnextState = (\lambda r p st msgs crd st'. Ute-nextState r p st msgs st'),
        SHOcommPerRd = Ute-commPerRd,
        SHOcommGlobal = Ute-commGlobal
    D
abbreviation
    Ute-M \equiv(Ute-SHOMachine::(Proc,'val pstate,'val msg) SHOMachine)
end - locale ute-parameters
end
theory UteProof
imports UteDefs ../Majorities ../Reduction
begin
context ute-parameters
begin
```


### 8.4 Preliminary Lemmas

Processes can make a vote only at first round of each phase.

```
lemma vote-step:
    assumes nxt: nextState Ute-M r p (rho r p) \(\mu(\) rho (Suc r) \(p\) )
    and vote (rho (Suc r) \(p\) ) \(\neq\) None
    shows step \(r=0\)
proof (rule ccontr)
    assume step \(r \neq 0\)
    with assms have vote (rho (Suc r) p) = None
        by (auto simp:Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def)
    with assms show False by auto
qed
```

Processes can make a new decision only at second round of each phase.
lemma decide-step:
assumes run: SHORun Ute-M rho HOs SHOs
and d1: decide (rho r $p$ ) $\neq$ Some $v$
and d2: decide (rho (Suc r) p) = Some $v$
shows step $r \neq 0$
proof
assume sr:step $r=0$

```
    from run obtain }\mu\mathrm{ where Ute-nextState r p (rho r p) }\mu(\mathrm{ rho (Suc r) p)
        unfolding Ute-SHOMachine-def nextState-def SHORun-eq SHOnextConfig-eq
        by force
    with sr have next0 r p (rho r p) }\mu(rho(Suc r) p
    unfolding Ute-nextState-def by auto
    hence decide (rho r p)= decide (rho (Suc r) p)
    by (auto simp:next0-def)
    with d1 d2 show False by auto
qed
lemma unique-majority-E:
    assumes majv:card {qq::Proc. F qq=Some m}>E
    and majw: card {qq::Proc. F qq=Some m'}>E
    shows m= m'
proof -
    from majv majw majE
    have card {qq::Proc. F qq=Some m}>N div 2
        and card {qq::Proc.F qq=Some m'}>N div 2
    by auto
    then obtain qq
    where }qq\in{qq::Proc. F qq= Some m
            and qq\in{qq::Proc. F qq=Some m'}
    by (rule majoritiesE')
    thus?thesis by auto
qed
lemma unique-majority-E-\alpha:
    assumes majv: card {qq::Proc. F qq=m} > E-\alpha
    and majw: card {qq::Proc. F qq= m'}>E-\alpha
    shows m= m'
proof -
    from majE alpha-lt-N majv majw
    have card {qq::Proc. F qq=m}>N div 2
        and card {qq::Proc. F qq=m'}>N div 2
    by auto
    then obtain qq
    where qq\in{qq::Proc. F qq=m}
            and qq\in{qq::Proc. F qq=m
    by (rule majoritiesE')
    thus?thesis by auto
qed
lemma unique-majority-T:
    assumes majv: card {qq::Proc. F qq=Some m} > T
    and majw: card {qq::Proc. F qq=Some m'} > T
    shows m= m'
proof -
    from majT majv majw
    have card {qq::Proc. F qq=Some m}>N div 2
```

and card $\left\{q q::\right.$ Proc. $F q q=$ Some $\left.m^{\prime}\right\}>N$ div 2
by auto
then obtain $q q$
where $q q \in\{q q::$ Proc. $F q q=$ Some $m\}$
and $q q \in\left\{q q:\right.$ Proc. $F q q=$ Some $\left.m^{\prime}\right\}$
by (rule majoritiesE')
thus ?thesis by auto
qed
No two processes may vote for different values in the same round.

```
lemma common-vote:
    assumes usafe: SHOcommPerRd Ute-M HO SHO
    and nxtp: nextState Ute-M r p (rho r p) \(\mu \mathrm{p}\) (rho (Suc r) p)
    and mup: \(\mu \mathrm{p} \in\) SHOmsgVectors Ute-M r p (rho r) (HO p) (SHO p)
    and nxtq: nextState Ute-M r q (rho r q) \(\mu q\) (rho (Suc r) q)
    and muq: \(\mu q \in\) SHOmsgVectors Ute-M r q (rho r) \((\mathrm{HO} q)(S H O q)\)
    and \(v p\) : vote (rho (Suc r) p) \(=\) Some \(v p\)
    and \(v q\) : vote (rho (Suc r) q) \(=\) Some \(v q\)
    shows \(v p=v q\)
using assms proof -
    have gtn: card \(\{q q\). sendMsg Ute-M r qq \(p(\) rho r \(q q)=\) Val \(v p\}\)
                \(+\operatorname{card}\{q q\). sendMsg Ute-Mrqq q(rho r qq) \(=\) Val vq\} \(>N\)
    proof -
    have card \(\{q q\). sendMsg Ute-M r qq \(p(\) rho \(r q q)=\) Val \(v p\}>T-\alpha\)
        \(\wedge\) card \(\{q q\). sendMsg Ute-Mrqq \(q(\) rho \(r q q)=\) Val \(v q\}>T-\alpha\)
        (is card ?vsentp \(>-\wedge\) card ? vsentq \(>-\) )
    proof -
        from nxtp vp have stp:step \(r=0\) by (auto simp: vote-step)
        from mup
        have \(\{q q \cdot \mu p q q=\) Some (Val vp) \(\}-(H O p-S H O p)\)
            \(\subseteq\{q q\). sendMsg Ute-M r qq \(p(\) rho \(r q q)=\) Val \(v p\}\)
                        (is ?vrcvdp - ?ahop \(\subseteq\) ?vsentp)
            by (auto simp: SHOmsgVectors-def)
        hence card (?vrcudp - ?ahop) \(\leq\) card ?vsentp
            and card (?vrcvdp - ?ahop) \(\geq\) card ?vrcvdp - card ?ahop
            by (auto simp: card-mono diff-card-le-card-Diff)
        hence card ?vsentp \(\geq\) card ?vrcvdp - card ?ahop by auto
        moreover
        from nxtp stp have next0 r \(p\) (rho r \(p\) ) \(\mu p\) (rho (Suc r) p)
            by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
            with \(v p\) have card ?vrcvdp \(>T\)
            unfolding next0-def by auto
    moreover
    from \(m u q\)
    have \(\{q q . \mu q q q=\) Some \((\) Val \(v q)\}-(H O q-S H O q)\)
                \(\subseteq\{q q\). sendMsg Ute-M r qq \(q(\) rho \(r q q)=\) Val \(v q\}\)
                (is ? \(v r c v d q-\) ?ahoq \(\subseteq\) ?vsentq)
            by (auto simp: SHOmsgVectors-def)
    hence card (?vrcvdq - ?ahoq) \(\leq\) card ?vsentq
```

```
        and card (?vrcvdq - ?ahoq) \geq card ?vrcvdq - card ?ahoq
        by (auto simp: card-mono diff-card-le-card-Diff)
    hence card ?vsentq \geq card ?vrcvdq - card ?ahoq by auto
    moreover
    from nxtq stp have next0 r q (rho r q) \muq (rho (Suc r) q)
        by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
    with vq have card {qq. \muq qq = Some (Val vq)} > T
        by (unfold next0-def, auto)
    moreover
    from usafe have card ?ahop }\leq\alpha\mathrm{ and card ?ahoq }\leq
        by (auto simp:Ute-SHOMachine-def Ute-commPerRd-def)
    ultimately
    show ?thesis using alpha-lt-T by auto
    qed
    thus ?thesis using majT by auto
qed
show ?thesis
proof (rule ccontr)
    assume vpq:vp }\not=v
    have \forallqq. sendMsg Ute-M r qq p (rho r qq)
                = sendMsg Ute-M r qq q (rho r qq)
        by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def
                step-def send0-def send1-def)
    with vpq
    have {qq. sendMsg Ute-M r qq p (rho r qq) = Val vp}
            \cap {qq. sendMsg Ute-M r qq q (rhor rq) = Val vq} = {}
        by auto
    with gtn
    have card ({qq. sendMsg Ute-M rqq p (rho r qq) = Val vp}
                                    \cup{q. sendMsg Ute-Mrqq q(rho r qq) = Val vq})>N
    by (auto simp: card-Un-Int)
    moreover
    have card ({qq. sendMsg Ute-M r qq p (rho r qq) = Val vp}
                            \cup \{ q q . ~ s e n d M s g ~ U t e - M ~ r ~ q q ~ q ( r h o ~ r ~ q q ) = V a l ~ v q \} ) ~ \leq N
    by (auto simp: card-mono)
    ultimately
    show False by auto
qed
qed
```

No decision may be taken by a process unless it received enough messages holding the same value.
lemma decide-with-threshold-E:
assumes run: SHORun Ute-M rho HOs SHOs
and usafe: SHOcommPerRd Ute-M (HOs r) (SHOs r)
and d1: decide (rho r $p$ ) $\neq$ Some $v$
and d2: decide (rho (Suc r) $p$ ) = Some $v$
shows card $\{q$. sendMsg Ute-Mrqp(rhorq) $=$ Vote (Some $v)\}$

```
    > E-\alpha
proof -
    from run obtain }\mu
        where nxt:nextState Ute-M r p (rho r p) \mup (rho (Suc r) p)
            and }\forallqq.qq\inHOs r p\longleftrightarrow\mupqq\not=Non
            and }\forallqq.qq\inSHOs r p\capHOs r p
                    \longrightarrowpqq=Some (sendMsg Ute-Mr qq p (rho r qq))
        unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq SHOmsgVec-
tors-def
        by blast
    hence {qq. \mup qq = Some (Vote (Some v))} - (HOs r p - SHOs r p)
                \subseteq \{ q q . ~ s e n d M s g ~ U t e - M ~ r ~ q q ~ p ~ ( r h o ~ r ~ q q ) ~ = ~ V o t e ~ ( S o m e ~ v ) \}
            (is ?vrcvdp - ?ahop \subseteq?vsentp) by auto
    hence card (?vrcvdp - ?ahop) \leq card ?vsentp
        and card (?vrcvdp - ?ahop) \geq card ?vrcvdp - card ?ahop
        by (auto simp: card-mono diff-card-le-card-Diff)
    hence card ?vsentp \geq card ?vrcvdp - card ?ahop by auto
    moreover
    from usafe have card (HOs r p-SHOs r p) \leq\alpha
        by (auto simp:Ute-SHOMachine-def Ute-commPerRd-def)
    moreover
    from run d1 d2 have step r f=0 by (rule decide-step)
    with nxt have next1 r p (rho r p) \mup (rho (Suc r) p)
    by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
    with run d1 d2 have card {qq. \mup qq=Some (Vote (Some v))}>E
        unfolding next1-def by auto
    ultimately
    show ?thesis using alpha-lt-E by auto
qed
```


### 8.5 Proof of Agreement and Validity

If more than $E-\alpha$ messages holding $v$ are sent to some process $p$ at round $r$, then every process $p p$ correctly receives more than $\alpha$ such messages.

```
lemma common-x-argument-1:
    assumes usafe:SHOcommPerRd Ute-M (HOs (Suc r)) (SHOs (Suc r))
    and threshold: card {q. sendMsg Ute-M (Suc r) q p (rho (Suc r) q)
                                    = Vote (Some v)}>E-\alpha
    (is card (?msgs p v)> -)
    shows card (?msgs pp v\cap(SHOs (Suc r) pp\capHOs (Suc r) pp))>\alpha
proof -
    have card (?msgs pp v) + card (SHOs (Suc r) pp\capHOs (Suc r) pp)>N+\alpha
    proof -
    have }\forallq. sendMsg Ute-M (Suc r) q p (rho (Suc r) q)
                        = sendMsg Ute-M (Suc r) q pp (rho (Suc r) q)
        by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def
                        step-def send0-def send1-def)
    moreover
    from usafe
```

```
    have card (SHOs (Suc r) pp \capHOs (Suc r) pp) >N + 2*\alpha - E - 1
            by (auto simp: Ute-SHOMachine-def step-def Ute-commPerRd-def)
    ultimately
    show ?thesis using threshold by auto
    qed
    moreover
    have card (?msgs pp v) + card (SHOs (Suc r) pp \cap HOs (Suc r) pp)
            = card (?msgs pp v\cup(SHOs (Suc r) pp\capHOs (Suc r) pp))
                        + card (?msgs pp v\cap(SHOs(Suc r) pp\capHOs(Suc r) pp))
    by (auto intro: card-Un-Int)
    moreover
    have card (?msgs pp v\cup(SHOs(Suc r) pp\capHOs(Suc r) pp))\leqN
    by (auto simp: card-mono)
    ultimately
    show ?thesis by auto
qed
```

If more than $E-\alpha$ messages holding $v$ are sent to $p$ at some round $r$, then any process $p p$ will set its $x$ to value $v$ in $r$.

```
lemma common-x-argument-2:
    assumes run: SHORun Ute-M rho HOs SHOs
    and usafe: \(\forall r\). SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and nxtpp: nextState Ute-M (Suc r) pp (rho (Suc r) pp)
        \(\mu p p\) (rho (Suc (Suc r)) pp)
    and mupp: \(\mu p p \in\) SHOmsgVectors Ute-M (Suc r) pp (rho (Suc r))
                                    (HOs (Suc r) pp) (SHOs (Suc r) pp)
    and threshold: card \(\{q\). sendMsg Ute-M (Suc r) q p (rho (Suc r) q)
                \(=\operatorname{Vote}(\) Some \(v)\}>E-\alpha\)
            (is card (?sent p \(v\) ) >-)
    shows \(x(\) rho \((\) Suc \((\) Suc r) \() p p)=v\)
proof -
    have stp:step (Suc r) \(\neq 0\)
    proof
    assume sr: step (Suc r) \(=0\)
    hence \(\forall q\). sendMsg Ute-M (Suc r) \(q\) p (rho (Suc r) q)
                        \(=\operatorname{Val}(x(r h o(S u c r) q))\)
        by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send0-def)
    moreover
    from threshold obtain \(q q\) where
            sendMsg Ute-M (Suc r) qq \(p\) (rho (Suc r) qq) = Vote (Some \(v\) )
            by force
    ultimately
    show False by simp
    qed
    have va: card \(\{q q . \mu p p q q=\) Some \((\) Vote \((\) Some \(v))\}>\alpha\)
        (is card (?msgs \(v\) ) >
    proof -
    from mupp
```

```
    have SHOs (Suc r) pp \capHOs (Suc r) pp
        \subseteq \{ q q . ~ \mu p p ~ q q = S o m e ~ ( s e n d M s g ~ U t e - M ~ ( S u c ~ r ) ~ q q ~ p p ~ ( r h o ~ ( S u c ~ r ) ~ q q ) ) \}
        unfolding SHOmsgVectors-def by auto
    moreover
    hence (?msgs v) \supseteq(?sent pp v) \cap(SHOs (Suc r) pp\capHOs (Suc r) pp)
        by auto
    hence card (?msgs v)
        \geq \operatorname { c a r d ~ ( ( ? s e n t ~ p p ~ v ) ~ \cap ( S H O s ~ ( S u c ~ r ) ~ p p ~ \cap H O s ( S u c ~ r ) ~ p p ) ) }
    by (auto intro: card-mono)
    moreover
    from usafe threshold
    have alph:card ((?sent pp v) \cap(SHOs (Suc r) pp \capHOs (Suc r) pp)) > 人
    by (blast dest: common-x-argument-1)
    ultimately
    show ?thesis by auto
qed
moreover
from nxtpp stp
have next1 (Suc r) pp (rho (Suc r) pp) \mupp (rho (Suc (Suc r)) pp)
    by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
ultimately
obtain w where wa:card (?msgs w)>\alpha and xw:x (rho (Suc (Suc r)) pp)=w
unfolding next1-def by auto
```

```
have \(v=w\)
```

have $v=w$
proof -
proof -
note usafe
note usafe
moreover
moreover
obtain $q v$ where $q v \in \operatorname{SHOs}($ Suc $r) p p$ and $\mu p p q v=$ Some $($ Vote $($ Some $v))$
obtain $q v$ where $q v \in \operatorname{SHOs}($ Suc $r) p p$ and $\mu p p q v=$ Some $($ Vote $($ Some $v))$
proof -
proof -
have $\neg($ ?msgs $v \subseteq H O s($ Suc r) $p p-S H O s($ Suc r) pp)
have $\neg($ ?msgs $v \subseteq H O s($ Suc r) $p p-S H O s($ Suc r) pp)
proof
proof
assume ?msgs $v \subseteq H O s$ (Suc r) pp - SHOs (Suc r) pp
assume ?msgs $v \subseteq H O s$ (Suc r) pp - SHOs (Suc r) pp
hence card (?msgs v) $\operatorname{card}(($ HOs (Suc r) pp) $-($ SHOs (Suc r) pp))
hence card (?msgs v) $\operatorname{card}(($ HOs (Suc r) pp) $-($ SHOs (Suc r) pp))
by (auto simp: card-mono)
by (auto simp: card-mono)
moreover
moreover
from usafe
from usafe
have card (HOs (Suc r) pp-SHOs (Suc r) pp) $\leq \alpha$
have card (HOs (Suc r) pp-SHOs (Suc r) pp) $\leq \alpha$
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
moreover
moreover
note $v a$
note $v a$
ultimately
ultimately
show False by auto
show False by auto
qed
qed
then obtain $q v$
then obtain $q v$
where $q v \notin H O s(S u c r) p p-S H O s(S u c r) p p$
where $q v \notin H O s(S u c r) p p-S H O s(S u c r) p p$
and $q s v: \mu p p q v=$ Some (Vote (Some $v)$ )
and $q s v: \mu p p q v=$ Some (Vote (Some $v)$ )
by auto
by auto
with mupp have $q v \in S H O s$ (Suc r) pp

```
    with mupp have \(q v \in S H O s\) (Suc r) pp
```

unfolding SHOmsgVectors-def by auto with qsv that show ?thesis by auto

## qed

with stp mupp have vote (rho (Suc r) qv) = Some $v$ by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send1-def)

## moreover

obtain $q w$ where
$q w \in S H O s($ Suc $r) p p$ and $\mu p p q w=$ Some (Vote (Some $w)$ )
proof -
have $\neg($ ?msgs $w \subseteq H O s(S u c r) p p-S H O s(S u c r) p p)$
proof
assume ?msgs $w \subseteq H O s$ (Suc r) pp-SHOs (Suc r) pp
hence card (?msgs w) $\operatorname{card}(($ HOs (Suc r) pp) $-($ SHOs (Suc r) pp))
by (auto simp: card-mono)
moreover
from usafe
have card (HOs (Suc r) pp - SHOs (Suc r) pp) $\leq \alpha$
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
moreover
note wa
ultimately
show False by auto
qed
then obtain $q w$
where $q w \notin H O s(S u c r) p p-S H O s(S u c r) p p$ and qsw: $\mu p p q w=$ Some (Vote (Some w))
by auto
with mupp have $q w \in S H O s$ (Suc r) pp
unfolding SHOmsgVectors-def by auto
with qsw that show ?thesis by auto
qed
with stp mupp have vote (rho (Suc r) qw) = Some $w$
by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def
Ute-sendMsg-def send1-def)

## moreover

from run obtain $\mu q v \mu q w$
where nextState Ute-M r qv ((rho r) qv) $\mu q v(r h o(S u c ~ r) ~ q v) ~$
and $\mu q v \in S H O m s g V e c t o r s$ Ute-M r qv (rho r) (HOs r qv) (SHOs r qv)
and nextState Ute-M r qw ((rho r) qw) $\mu q w(r h o$ (Suc r) qw)
and $\mu q w \in$ SHOmsgVectors Ute-M r qw (rho r) (HOs r qw) (SHOs r qw)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq) blast ultimately
show ?thesis using usafe by (auto dest: common-vote)
qed
with $x w$ show $x(r h o(S u c(S u c r)) p p)=v$ by auto
qed
Inductive argument for the agreement and validity theorems.

```
lemma safety-inductive-argument:
    assumes run: SHORun Ute-M rho HOs SHOs
    and comm: }\forallr.SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and ih: E - \alpha<card {q. sendMsg Ute-M r'q p (rho r'q) = Vote (Some v)}
    and stp1: step r' = Suc 0
    shows E - \alpha<
        card {q. sendMsg Ute-M (Suc (Suc r')) q p (rho (Suc (Suc r')) q)
                        = Vote (Some v)}
proof -
    from stp1 have r'> 0 by (auto simp: step-def)
    with stp1 obtain r where rr':}\mp@subsup{r}{}{\prime}=\mathrm{ Suc r and stpr:step (Suc r) = Suc 0
        by (auto dest: gr0-implies-Suc)
    have }\forallpp.x(rho (Suc (Suc r)) pp)=
    proof
        fix pp
        from run obtain }\mup
            where }\mupp\inSHOmsgVectors Ute-M r'pp (rho r') (HOs r'pp) (SHOs r'pp
            and nextState Ute-M r'pp (rho r'pp) \mupp (rho (Suc r') pp)
            by (auto simp:Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
        with run comm ih rr' show x (rho (Suc (Suc r)) pp) =v
            by (auto dest: common-x-argument-2)
    qed
    with run stpr
    have \forallpp p. sendMsg Ute-M (Suc (Suc r)) pp p (rho (Suc (Suc r)) pp) = Val v
        by (auto simp:Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
                        Ute-sendMsg-def send0-def mod-Suc step-def)
    with rr '
    have }\wedgep\mu\mp@subsup{p}{}{\prime}.\mu\mp@subsup{p}{}{\prime}\inSHOmsgVectors Ute-M (Suc r') p (rho (Suc r'))
                            (HOs (Suc r') p)(SHOs (Suc r') p)
            CSHOs(Suc r') p\capHOs(Suc r') p
                        \subseteq \{ q . \mu p ^ { \prime } q = \text { Some (Val v)\}}
    by (auto simp:SHOmsgVectors-def)
hence }\p\mu\mp@subsup{p}{}{\prime}.\mu\mp@subsup{p}{}{\prime}\in\mathrm{ SHOmsgVectors Ute-M (Suc r') p (rho (Suc r'))
                                    (HOs (Suc r') p)(SHOs (Suc r') p)
                card (SHOs (Suc r') p\capHOs(Suc r') p)
                \leqcard {q. \mup' q = Some (Val v)}
    by (auto simp: card-mono)
moreover
from comm have }\bigwedgep.T<\operatorname{card}(SHOs(Suc r') p\capHOs(Suc r') p
    by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
ultimately
have vT:\p }\mu\mp@subsup{p}{}{\prime}.\mu\mp@subsup{p}{}{\prime}\inSHOmsgVectors Ute-M (Suc r') p (rho (Suc r'))
                            (HOs (Suc r') p)(SHOs (Suc r') p)
                \LongrightarrowT<card {q. \mup' q = Some (Valv)}
    by (auto dest: less-le-trans)
show ?thesis
proof -
```

```
    have \(\forall p p\). vote \(\left(\left(\right.\right.\) rho \(\left(\right.\) Suc \(\left(\right.\) Suc \(\left.\left.\left.r^{\prime}\right)\right)\right)\) pp) \(=\) Some \(v\)
    proof
        fix \(p p\)
        from run obtain \(\mu p p\)
            where nxtpp: nextState Ute-M (Suc r\(\left.r^{\prime}\right) p p\left(r h o\left(S u c r^{\prime}\right) p p\right) \mu p p\)
                    (rho (Suc (Suc r \({ }^{\prime}\) )) pp)
            and mupp: \(\mu \mathrm{pp} \in\) SHOmsgVectors Ute-M (Suc r') pp (rho (Suc r\(\left.\left.r^{\prime}\right)\right)\)
                    (HOs (Suc \(\left.r^{\prime}\right) p p\) ) (SHOs (Suc \(\left.\left.r^{\prime}\right) p p\right)\)
        by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
    with \(v T\) have \(v T^{\prime}:\) card \(\{q . \mu p p q=\) Some (Val v) \(\}>T\)
        by auto
    moreover
    from stpr \(r r^{\prime}\) have step (Suc \(\left.r^{\prime}\right)=0\)
        by (auto simp: mod-Suc step-def)
    with nxtpp
    have next0 (Suc \(\left.r^{\prime}\right) p p\left(r h o\left(S u c r^{\prime}\right) p p\right) \mu p p\left(r h o\left(S u c\left(S u c r^{\prime}\right)\right) p p\right)\)
        by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
    ultimately
    obtain \(w\)
        where \(w T\) :card \(\{q . \mu p p q=\) Some \((\) Val w) \(\}>T\)
            and votew:vote (rho (Suc (Suc \(\left.r^{\prime}\right)\) ) pp) \(=\) Some \(w\)
        by (auto simp: next0-def)
    from \(v T^{\prime} w T\) have \(v=w\)
        by (auto dest: unique-majority-T)
    with votew show vote (rho (Suc (Suc \(\left.\left.\left.r^{\prime}\right)\right) p p\right)=\) Some \(v\) by simp
    qed
    with run stpr \(r r^{\prime}\)
    have \(\forall p . N=\operatorname{card}\{q\). sendMsg Ute-M (Suc (Suc (Suc r))) q \(p\)
                            \(((\) rho \((\) Suc \((\) Suc \((\) Suc r \()))) q)=\operatorname{Vote}(\) Some \(v)\}\)
    by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
    Ute-sendMsg-def send1-def step-def mod-Suc)
    with \(r r^{\prime}\) majE Elt \(N\) show ?thesis by auto
qed
qed
```

A process that holds some decision $v$ has decided $v$ sometime in the past.

```
lemma decisionNonNullThenDecided:
    assumes run:SHORun Ute-M rho HOs SHOs and dec: decide (rho n p) = Some
\(v\)
    shows \(\exists m<n\). decide \((\) rho \((\) Suc \(m) p) \neq\) decide \((\) rho \(m p)\)
            \(\wedge\) decide (rho (Suc m) p) \(=\) Some \(v\)
proof -
    let ?dec \(k=\) decide \(((\) rho \(k) p)\)
    have \((\forall m<n\). ?dec \((\) Suc \(m) \neq\) ?dec \(m \longrightarrow\) ?dec \((\) Suc \(m) \neq\) Some \(v)\)
                \(\longrightarrow\) ?dec \(n \neq\) Some \(v\)
            (is?P \(n\) is ?A \(n \longrightarrow-\) )
proof (induct \(n\) )
    from run show ?P 0
            by (auto simp: Ute-SHOMachine-def SHORun-eq HOinitConfig-eq
```

```
initState-def Ute-initState-def)
```

next
fix $n$
assume ih: ?P $n$ thus ?P (Suc $n$ ) by force
qed
with dec show ?thesis by auto

## qed

If process $p 1$ has decided value $v 1$ and process $p 2$ later decides, then $p 2$ must decide $v 1$.

```
lemma laterProcessDecidesSameValue:
    assumes run:SHORun Ute-M rho HOs SHOs
    and comm: \(\forall\) r. SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and dv1:decide (rho (Suc r) p1) = Some v1
    and dn2:decide (rho \((r+k) p 2) \neq\) Some v2
    and dv2:decide (rho \((S u c(r+k))\) p2) \(=\) Some v2
    shows \(v 2=v 1\)
proof -
    from run dv1 obtain r1
    where r1r:r1 < Suc r
        and dn1:decide (rho r1 p1) \(\neq\) Some v1
        and \(d v 1\) ':decide (rho (Suc r1) p1) = Some v1
    by (auto dest: decisionNonNullThenDecided)
    from r1r obtain \(s\) where \(r r 1: S u c r=\) Suc \((r 1+s)\)
    by (auto dest: less-imp-Suc-add)
    then obtain \(k^{\prime}\) where \(k k^{\prime}: r+k=r 1+k^{\prime}\)
    by auto
    with dn2 dv2
    have dn2': decide (rho \(\left.\left(r 1+k^{\prime}\right) p 2\right) \neq\) Some v2
    and dv2': decide (rho \(\left(S u c ~^{\prime}\left(r 1+k^{\prime}\right)\right)\) p2) \(=\) Some v2
    by auto
    from run \(d n 1 d v 1^{\prime} d n 2^{\prime} d v 2^{\prime}\)
    have rs0:step r1 = Suc 0 and rks0:step \(\left(r 1+k^{\prime}\right)=\) Suc 0
    by (auto simp: mod-Suc step-def dest: decide-step)
    have step \(\left(r 1+k^{\prime}\right)=\) step (step \(\left.r 1+k^{\prime}\right)\)
    unfolding step-def by (simp add: mod-add-left-eq)
    with rs0 rks0 have step \(k^{\prime}=0\) by (auto simp: step-def mod-Suc)
    then obtain \(k^{\prime \prime}\) where \(k^{\prime}=k^{\prime \prime} *\) nSteps by (auto simp: step-def)
    with \(d n 2^{\prime} d v 2^{\prime}\)
    have dn2 \({ }^{\prime \prime}\) :decide (rho ( \(\left.\left.r 1+k^{\prime \prime} * n S t e p s\right) ~ p 2\right) \neq\) Some v2
    and dv2 \(^{\prime \prime}\) :decide (rho (Suc (r1 \(+k^{\prime \prime} * n\) Steps \()\) ) p2) \(=\) Some v2
    by auto
    from rs0 have stp:step \(\left(r 1+k^{\prime \prime} * n\right.\) Steps \()=\) Suc 0
    unfolding step-def by auto
```

```
    have inv:card \{q. sendMsg Ute-M (r1 \(\left.+k^{\prime \prime} * n S t e p s\right) ~ q ~ p 1 ~\left(r h o\left(r 1+k^{\prime \prime} * n S t e p s\right)\right.\)
q)
    \(=\operatorname{Vote}(\) Some v1 \()\}>E-\alpha\)
    proof (induct \(k^{\prime \prime}\) )
    from stp have step \((r 1+0 * n S t e p s)=\) Suc 0
        by (auto simp: step-def)
    from run comm dn1 dv1'
    show card \(\{\). sendMsg Ute-M \((r 1+0 * n\) Steps \() q p 1(r h o(r 1+0 * n S t e p s) q)\)
                        \(=\operatorname{Vote}\) (Some v1) \(\}>E-\alpha\)
        by (intro decide-with-threshold-E) auto
next
    fix \(k^{\prime \prime}\)
    assume ih: \(E-\alpha<\)
        card \{q. sendMsg Ute-M (r1 \(\left.+k^{\prime \prime} * n S t e p s\right) ~ q p 1\left(r h o\left(r 1+k^{\prime \prime} * n S t e p s\right) q\right)\)
                        \(=\) Vote (Some v1) \}
    from rs0 have stps: step \(\left(r 1+k^{\prime \prime} * n S t e p s\right)=\) Suc 0
        by (auto simp: step-def)
    with run comm ih
    have \(E-\alpha<\)
        card \{q. sendMsg Ute-M (Suc (Suc ( \(\left.\left.\left.r 1+k^{\prime \prime} * n S t e p s\right)\right)\right)\) q p1
                                    (rho (Suc (Suc (r1 \(\left.\left.+k^{\prime \prime} * n S t e p s\right)\right)\) ) q)
                \(=\operatorname{Vote}(\) Some v1) \(\}\)
        by (rule safety-inductive-argument)
    thus \(E-\alpha<\)
        card \(\left\{q\right.\). sendMsg Ute-M \(\left(r 1+\right.\) Suc \(\left.k^{\prime \prime} * n S t e p s\right) q p 1\)
                                    (rho (r1 + Suc \(k^{\prime \prime} *\) nSteps) \(q\) )
                                    \(=\) Vote (Some v1) \}
        by auto
qed
moreover
from run
have \(\forall q\). sendMsg Ute-M \(\left(r 1+k^{\prime \prime} * n S t e p s\right) ~ q ~ p 1\left(r h o\left(r 1+k^{\prime \prime} * n S t e p s\right) q\right)\)
            \(=\) sendMsg Ute-M (r1 \(\left.+k^{\prime \prime} * n S t e p s\right) ~ q ~ p 2\left(r h o\left(r 1+k^{\prime \prime} * n S t e p s\right) q\right)\)
    by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def
                step-def send0-def send1-def)
moreover
from run comm dn2" dv2"
have \(E-\alpha<\)
        card \(\left\{q\right.\). sendMsg Ute-M (r1 \(\left.+k^{\prime \prime} * n S t e p s\right) ~ q ~ p 2\left(r h o\left(r 1+k^{\prime \prime} * n S t e p s\right) q\right)\)
                        \(=\operatorname{Vote}\) (Some v2) \(\}\)
    by (auto dest: decide-with-threshold-E)
ultimately
show \(v 2=v 1\) by (auto dest: unique-majority-E- \(\alpha\) )
qed
```

The Agreement property is an immediate consequence of the two preceding lemmas.
theorem ute-agreement:
assumes run: SHORun Ute-M rho HOs SHOs

```
    and comm: \forallr.SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and p: decide (rho m p) = Some v
    and q: decide (rho n q) = Some w
    shows v=w
proof -
    from run p obtain k
        where k1: decide (rho (Suc k) p) \not= decide (rho k p)
            and k2: decide (rho (Suc k) p) = Some v
    by (auto dest:decisionNonNullThenDecided)
    from run q obtain l
    where l1: decide (rho (Suc l) q) = decide (rho l q)
            and l2: decide (rho (Suc l) q) = Some w
    by (auto dest: decisionNonNullThenDecided)
    show ?thesis
    proof (cases k\leql)
    case True
    then obtain m where m:l=k+m by (auto simp add:le-iff-add)
    from run comm k2 l1 l2 m have w=v
            by (auto elim!: laterProcessDecidesSameValue)
        thus ?thesis by simp
    next
        case False
        hence l\leqk by simp
        then obtain m}\mathrm{ where m:k=l+m by (auto simp add:le-iff-add)
        from run comm l2 k1 k2 m show ?thesis
            by (auto elim!: laterProcessDecidesSameValue)
    qed
qed
Main lemma for the proof of the Validity property.
lemma validity-argument:
    assumes run: SHORun Ute-M rho HOs SHOs
    and comm: \forallr.SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and init: }\forallp.x ((rho 0) p)=
    and dw: decide (rho r p) = Some w
    and stp: step r' = Suc 0
    shows card {q. sendMsg Ute-M r'q p (rho r'q)=Vote (Some v)}>E-\alpha
proof -
    define k where k= r' div nSteps
    with stp have stp: r' = Suc 0 + k* nSteps
        using div-mult-mod-eq [of r' nSteps]
        by (simp add: step-def)
    moreover
    have E-\alpha<
        card {q. sendMsg Ute-M (Suc 0 + k*nSteps) q p ((rho (Suc 0 + k*nSteps))
q)
            = Vote (Some v)}
    proof (induct k)
    have }\forallpp.vote ((rho (Suc 0)) pp)=Some 
```

```
    proof
    fix pp
    from run obtain \mupp
        where nxtpp:nextState Ute-M 0 pp (rho 0 pp) \mupp (rho (Suc 0) pp)
            and mupp:\mupp \inSHOmsgVectors Ute-M 0 pp (rho 0) (HOs 0 pp) (SHOs
0 pp)
            by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
    have majv:card {q. upp q=Some (Val v)}>T
    proof -
            from run init have }\forallq. sendMsg Ute-M 0 q pp (rho 0 q) = Val v
            by (auto simp:Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
                                    Ute-sendMsg-def send0-def step-def)
        moreover
        from comm have shoT:card (SHOs 0 pp \capHOs 0 pp)> T
            by (auto simp:Ute-SHOMachine-def Ute-commPerRd-def)
        moreover
        from mupp
        have SHOs 0 pp \capHOs 0 pp
                        \subseteq \{ q . \mu p p ~ q = S o m e ~ ( s e n d M s g ~ U t e - M ~ 0 ~ q ~ p p ~ ( r h o ~ 0 ~ q ) ) \}
            by (auto simp: SHOmsgVectors-def)
        hence card (SHOs 0 pp \cap HOs 0 pp)
                        \leqcard {q. \mupp q=Some (sendMsg Ute-M 0 q pp (rho 0 q))}
            by (auto simp: card-mono)
        ultimately
        show ?thesis by (auto simp: less-le-trans)
    qed
    moreover
    from nxtpp have next0 0 pp ((rho 0) pp) \mupp (rho (Suc 0) pp)
    by (auto simp:Ute-SHOMachine-def nextState-def Ute-nextState-def step-def)
    ultimately
    obtain w where majw:card {q. \mupp q= Some (Val w)}>T
                        and votew:vote (rho (Suc 0) pp) = Some w
            by (auto simp: next0-def)
        from majv majw have v=w by (auto dest: unique-majority-T)
        with votew show vote ((rho (Suc 0)) pp)=Some v by simp
    qed
    with run
    have card {q. sendMsg Ute-M (Suc 0) q p (rho (Suc 0) q) = Vote (Some v)}
= N
        by (auto simp:Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
                    Ute-nextState-def step-def Ute-sendMsg-def send1-def)
    thus E-\alpha<
        card {q. sendMsg Ute-M (Suc 0 + 0*nSteps) qp(rho(Suc 0 + 0*nSteps)
q)
                        = Vote (Some v)}
        using majE EltN by auto
next
    fix }
```

```
    assume ih:E - \alpha<
        card {q. sendMsg Ute-M(Suc 0 +k*nSteps) q p(rho(Suc 0 +k*nSteps)
q)
                = Vote (Some v)}
    have step (Suc 0 + k* nSteps) = Suc 0
    by (auto simp: mod-Suc step-def)
    from run comm ih this
    have E-\alpha<
        card {q. sendMsg Ute-M (Suc (Suc (Suc 0 + k*nSteps))) q p
                                    (rho (Suc (Suc (Suc 0 + k* nSteps))) q)
            = Vote (Some v)}
    by (rule safety-inductive-argument)
    thus E-\alpha<
        card {q. sendMsg Ute-M (Suc 0 + Suc k*nSteps) q p
                    (rho (Suc 0 + Suc k*nSteps) q)
            = Vote (Some v)} by simp
    qed
    ultimately
    show ?thesis by simp
qed
The following theorem shows the Validity property of algorithm \(\mathcal{U}_{T, E, \alpha}\).
```

```
theorem ute-validity:
```

theorem ute-validity:
assumes run: SHORun Ute-M rho HOs SHOs
assumes run: SHORun Ute-M rho HOs SHOs
and comm: $\forall r$. SHOcommPerRd Ute-M (HOs r) (SHOs r)
and comm: $\forall r$. SHOcommPerRd Ute-M (HOs r) (SHOs r)
and init: $\forall p . x($ rho $0 p)=v$
and init: $\forall p . x($ rho $0 p)=v$
and dw: decide (rho r $p$ ) = Some $w$
and dw: decide (rho r $p$ ) = Some $w$
shows $v=w$
shows $v=w$
proof -
proof -
from run dw obtain r1
from run dw obtain r1
where dnr1:decide ((rho r1) p) $=$ Some $w$
where dnr1:decide ((rho r1) p) $=$ Some $w$
and dwr1:decide ((rho (Suc r1)) p) = Some $w$
and dwr1:decide ((rho (Suc r1)) p) = Some $w$
by (force dest: decisionNonNullThenDecided)
by (force dest: decisionNonNullThenDecided)
with run have step $r 1 \neq 0$ by (rule decide-step)
with run have step $r 1 \neq 0$ by (rule decide-step)
hence step r1 = Suc 0 by (simp add: step-def mod-Suc)
hence step r1 = Suc 0 by (simp add: step-def mod-Suc)
with assms
with assms
have $E-\alpha<$
have $E-\alpha<$
card $\{q$. sendMsg Ute-M r1 q $p($ rho r1 $q)=\operatorname{Vote}($ Some $v)\}$
card $\{q$. sendMsg Ute-M r1 q $p($ rho r1 $q)=\operatorname{Vote}($ Some $v)\}$
by (rule validity-argument)
by (rule validity-argument)
moreover
moreover
from run comm dnr1 dwr1
from run comm dnr1 dwr1
have card $\{q$. sendMsg Ute-Mr1 q p (rho r1 q) $=\operatorname{Vote}($ Some $w)\}>E-\alpha$
have card $\{q$. sendMsg Ute-Mr1 q p (rho r1 q) $=\operatorname{Vote}($ Some $w)\}>E-\alpha$
by (auto dest: decide-with-threshold-E)
by (auto dest: decide-with-threshold-E)
ultimately
ultimately
show $v=w$ by (auto dest: unique-majority- $E-\alpha$ )
show $v=w$ by (auto dest: unique-majority- $E-\alpha$ )
qed

```
qed
```


### 8.6 Proof of Termination

At the second round of a phase that satisfies the conditions expressed in the global communication predicate, processes update their $x$ variable with the value $v$ they receive in more than $\alpha$ messages.

```
lemma set-x-from-vote:
    assumes run: SHORun Ute-M rho HOs SHOs
    and comm: SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and stp: step \((\) Suc r) \(=\) Suc 0
    and \(\pi: \forall p\). HOs (Suc r) \(p=\) SHOs (Suc r) \(p\)
    and nxt: nextState Ute-M (Suc r) \(p(\) rho (Suc r) p) \(\mu(\) rho (Suc (Suc r)) \(p\) )
    and mu: \(\mu \in\) SHOmsgVectors Ute-M (Suc r) p (rho (Suc r))
                            (HOs (Suc r) p) (SHOs (Suc r) p)
    and vp: \(\alpha<\operatorname{card}\{q q . \mu q q=\) Some \((\) Vote \((\) Some \(v))\}\)
    shows \(x((\) rho \((\) Suc \((\) Suc r) \())) p)=v\)
proof -
    from nxt stp \(v p\) obtain \(w p\)
    where xwp: \(\alpha<\operatorname{card}\{q q . \mu q q=\) Some \((\) Vote \((\) Some wp \())\}\)
        and \(x p: x(r h o(S u c(S u c r)) p)=w p\)
    by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def)
    have \(w p=v\)
    proof -
    from \(x w p\) obtain \(p p\) where \(s m w: \mu p p=\) Some (Vote (Some wp))
        by force
    have vote (rho (Suc r) pp) = Some wp
    proof -
        from \(s m w m u \pi\)
        have \(\mu p p=\) Some (sendMsg Ute-M (Suc r) pp p(rho (Suc r) pp))
            unfolding SHOmsgVectors-def by force
        with stp have \(\mu p p=\) Some (Vote (vote (rho (Suc r) pp)))
            by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def)
        with \(s m w\) show ?thesis by auto
    qed
    moreover
    from \(v p\) obtain \(q q\) where \(s m v: \mu q=\) Some (Vote (Some \(v\) ))
        by force
    have vote (rho (Suc r) qq) \(=\) Some \(v\)
    proof -
        from \(s m v m u \pi\)
        have \(\mu q q=\) Some (sendMsg Ute-M (Suc r) qq \(p\) (rho (Suc r) qq))
            unfolding SHOmsg Vectors-def by force
        with stp have \(\mu q q=\) Some (Vote (vote (rho (Suc r) qq)))
                by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def)
        with smv show ?thesis by auto
    qed
    moreover
    from run obtain \(\mu p p \mu q q\)
        where nextState Ute-M r pp (rho r pp) \(\mu p p\) (rho (Suc r) pp)
```

```
            and \mupp\inSHOmsgVectors Ute-M r pp (rho r) (HOs r pp) (SHOs r pp)
            and nextState Ute-M r qq ((rho r) qq) \muqq(rho (Suc r) qq)
            and \muqq\inSHOmsgVectors Ute-M r qq (rho r)(HOs r qq) (SHOs r qq)
            unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq by blast
        ultimately
        show ?thesis using comm by (auto dest: common-vote)
    qed
    with xp show ?thesis by simp
qed
```

Assume that HO and SHO sets are uniform at the second step of some phase. Then at the subsequent round there exists some value $v$ such that any received message which is not corrupted holds $v$.

```
lemma termination-argument-1:
    assumes run: SHORun Ute-M rho HOs SHOs
    and comm: SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and stp: step (Suc r) = Suc 0
    and \(\pi: \forall p . \pi 0=H O s(\) Suc r) \(p \wedge \pi 0=\) SHOs (Suc r) \(p\)
    obtains \(v\) where
        \(\wedge p \mu p^{\prime} q\).
            \(\llbracket q \in \operatorname{SHOs}(\) Suc \((\) Suc r)) \(p \cap H O s(\) Suc (Suc r)) \(p\);
                        \(\mu p^{\prime} \in\) SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r)))
                            (HOs (Suc (Suc r)) p) (SHOs (Suc (Suc r)) p)
        \(\rrbracket \Longrightarrow \mu p^{\prime} q=(\) Some \((\) Val \(v))\)
proof -
    from \(\pi\) have hosho: \(\forall\) p.SHOs (Suc r) \(p=\) SHOs (Suc r) \(p \cap H O s\) (Suc r) \(p\)
        by \(\operatorname{simp}\)
```

    have \(\bigwedge p q . x(r h o(S u c(\) Suc r) \() p)=x(r h o(S u c(S u c ~ r)) q)\)
    proof -
        fix \(p q\)
        from run obtain \(\mu p\)
            where nxt: nextState Ute-M (Suc r) \(p\) (rho (Suc r) p)
                                    \(\mu p(r h o(S u c(S u c ~ r)) p)\)
            and mu: \(\mu p \in\) SHOmsgVectors Ute-M (Suc r) \(p\) (rho (Suc r))
                                    (HOs (Suc r) p) (SHOs (Suc r) p)
            by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
            from run obtain \(\mu q\)
            where nxtq: nextState Ute-M (Suc r) q (rho (Suc r) q)
                        \(\mu q(\) rho \((\) Suc (Suc r)) q)
            and muq: \(\mu q \in\) SHOmsgVectors Ute-M (Suc r) \(q\) (rho (Suc r))
                                    (HOs (Suc r) q) (SHOs (Suc r) q)
            by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
    have \(\forall q q . \mu p q q=\mu q q q\)
    proof
            fix \(q q\)
            show \(\mu p q q=\mu q q q\)
    ```
proof (cases \mup qq= None)
    case False
    with mu \pi have 1:qq\inSHOs (Suc r) p and 2:qq\inSHOs (Suc r) q
        unfolding SHOmsgVectors-def by auto
    from mu \pi 1
    have }\mupqq=Some (sendMsg Ute-M (Suc r) qq p (rho (Suc r)qq))
        unfolding SHOmsgVectors-def by auto
    moreover
    from muq \pi 2
    have }\muqqq=Some (sendMsg Ute-M (Suc r) qq q (rho (Suc r) qq))
        unfolding SHOmsgVectors-def by auto
    ultimately
    show ?thesis
        by (auto simp:Ute-SHOMachine-def Ute-sendMsg-def step-def
                        send0-def send1-def)
    next
    case True
    with mu have qq\not\inHOs (Suc r) punfolding SHOmsgVectors-def by auto
    with \pi muq have }\muqqq=None unfolding SHOmsgVectors-def by aut
    with True show ?thesis by simp
    qed
qed
hence vsets:\v. {qq. \mup qq=Some (Vote (Some v))}
                        ={qq. \muq qq=Some (Vote (Some v))}
by auto
show x (rho (Suc (Suc r)) p)=x(rho (Suc (Suc r)) q)
proof (cases \existsv.\alpha<card {qq. \mup qq=Some (Vote (Some v))}, clarify)
    fix v
    assume vp: \alpha<card {qq. \mup qq=Some (Vote (Some v))}
    with run comm stp \pi nxt mu have x (rho (Suc (Suc r)) p)=v
    by (auto dest: set-x-from-vote)
moreover
from vsets vp
have }\alpha<\mathrm{ card {qq. }\muqqq=Some (Vote (Some v))} by auto
with run comm stp \pi nxtq muq have x (rho (Suc (Suc r)) q) =v
    by (auto dest: set-x-from-vote)
    ultimately
    show x (rho (Suc (Suc r)) p)=x(rho (Suc (Suc r)) q)
    by auto
next
    assume nov: \neg(\existsv. \alpha< card {qq. \mup qq=Some (Vote (Some v))})
    with nxt stp have x (rho (Suc (Suc r)) p)= undefined
        by (auto simp:Ute-SHOMachine-def nextState-def
                    Ute-nextState-def next1-def)
moreover
from vsets nov
have }\neg(\existsv.\alpha<\operatorname{card {qq. \muq qq=Some (Vote (Some v))}) by auto
```

```
        with nxtq stp have x (rho (Suc (Suc r)) q) = undefined
        by (auto simp:Ute-SHOMachine-def nextState-def
                            Ute-nextState-def next1-def)
        ultimately
        show ?thesis by simp
    qed
qed
then obtain v}\mathrm{ where }\Lambdaq.x(rho(Suc(Sucr))q)=v by blas
moreover
from stp have step (Suc (Suc r)) = 0
    by (auto simp: step-def mod-Suc)
hence }\p\mu\mp@subsup{p}{}{\prime}q\mathrm{ .
    \llbracketq\inSHOs(Suc (Suc r)) p\capHOs(Suc (Suc r)) p;
        \mup}\mp@subsup{p}{}{\prime}\in\mathrm{ SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r)))
                                    (HOs (Suc (Suc r)) p) (SHOs (Suc (Suc r)) p)
    \rrbracket\Longrightarrow \mup'q=Some (Val (x (rho (Suc (Suc r)) q)))
    by (auto simp:Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send0-def)
ultimately
have }\Lambdap\mu\mp@subsup{p}{}{\prime}q
    \llbracketq\inSHOs(Suc (Suc r)) p \cap HOs (Suc (Suc r)) p;
        \mup}\mp@subsup{p}{}{\prime}\in\mathrm{ SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r)))
                                (HOs (Suc (Suc r)) p) (SHOs (Suc (Suc r)) p)
    \rrbracket\Longrightarrow\mu\mp@subsup{p}{}{\prime}q=(Some(\mathrm{ Val v))}
    by auto
    with that show thesis by blast
qed
```

If a process $p$ votes $v$ at some round $r$, then all messages received by $p$ in $r$ that are not corrupted hold $v$.
lemma termination-argument-2:
assumes mup: $\mu p \in$ SHOmsgVectors Ute-M (Suc r) $p$ (rho (Suc r)) (HOs (Suc r) p) (SHOs (Suc r) p)
and nxtq: nextState Ute-M r q (rho r q) $\mu q$ (rho (Suc r) q)
and vq: vote (rho (Suc r) q) = Some $v$
and qsho: $q \in$ SHOs (Suc r) $p \cap H O s($ Suc r) $p$
shows $\mu p q=$ Some (Vote (Some v))
proof -
from $n x t q v q$ have step $r=0$ by (auto simp: vote-step)
with mup qsho have $\mu p q=$ Some (Vote (vote (rho (Suc r) q) ))
by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def step-def send1-def mod-Suc)
with $v q$ show $\mu p q=\operatorname{Some}(\operatorname{Vote}($ Some $v)$ ) by auto
qed
We now prove the Termination property.
theorem ute-termination:
assumes run: SHORun Ute-M rho HOs SHOs
and commR: $\forall r$. SHOcommPerRd Ute-M (HOs r) (SHOs r)
and commG: SHOcommGlobal Ute-M HOs SHOs

```
    shows \existsrv.decide (rho r p)=Some v
proof -
    from commG
    obtain $ \pi r0
    where rr:r0 = Suc (nSteps * \Phi)
            and \pi: \forallp.\pi=HOs r0 p\wedge\pi=SHOs r0 p
            and t:\forallp.card (SHOs (Suc r0) p\capHOs(Suc r0) p)> T
            and e:\forallp.card (SHOs (Suc (Suc r0)) p \capHOs (Suc (Suc r0)) p)>E
    by (auto simp: Ute-SHOMachine-def Ute-commGlobal-def Let-def)
    from rr have stp:step r0 = Suc 0 by (auto simp: step-def)
    obtain w where votew:\forallp.(vote (rho (Suc (Suc r0)) p)) = Some w
    proof -
    have abc:\forallp.\exists w. vote (rho (Suc (Suc r0)) p)=Some w
    proof
        fix p
        from run stp obtain }\mu
            where nxt:nextState Ute-M (Suc r0) p (rho (Suc r0) p) \mup
                    (rho (Suc (Suc r0)) p)
            and mup:\mup G SHOmsgVectors Ute-M (Suc r0) p (rho (Suc r0))
                            (HOs (Suc rO) p)(SHOs (Suc r0) p)
                by (auto simp:Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
        have }\existsv.T<\operatorname{card {qq. \mup qq=Some (Val v)}
        proof -
            from t have card (SHOs (Suc r0) p\capHOs(Suc rO) p)>T ..
            moreover
            from run commR stp \pi rr
            obtain v where
                \p \mu\mp@subsup{p}{}{\prime}q.
                    \llbracketq\inSHOs(Suc r0) p\capHOs(Suc r0) p;
                        \mup}\mp@subsup{p}{}{\prime}\in\mathrm{ SHOmsgVectors Ute-M (Suc r0) p (rho (Suc r0))
                            (HOs (Suc r0) p) (SHOs (Suc r0) p)
                    \Longrightarrow }\mu\mp@subsup{p}{}{\prime}q=Some(Val v
                    using termination-argument-1 by blast
            with mup obtain v}\mathrm{ where
                        \qq.qq\inSHOs(Suc r0) p\capHOs(Suc r0) p\Longrightarrow < < qq = Some(Val v)
                by auto
                hence SHOs(Suc r0) p\capHOs(Suc r0) p\subseteq{qq. pp qq=Some (Val v)}
                    by auto
            hence card (SHOs (Suc r0) p\capHOs(Suc r0) p)
                    \leq card {qq. \mup qq = Some (Val v)}
                    by (auto intro: card-mono)
            ultimately
            have T<card {qq. \mup qq=Some (Val v)} by auto
            thus?thesis by auto
        qed
        with stp nxt show \exists}w\mathrm{ . vote ((rho (Suc (Suc r0))) p) = Some w
```

by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def step-def mod-Suc next0-def)
qed
then obtain $q q$ where $q q w$ :vote (rho (Suc (Suc r0)) qq) $=$ Some $w$ by blast
have $\forall p p$. vote (rho (Suc (Suc r0)) pp) = Some $w$
proof
fix $p p$
from $a b c$ obtain $w p$ where $p w p:$ vote $(($ rho $($ Suc (Suc r0))) $p p)=$ Some $w p$ by blast
from run obtain $\mu p p \mu q q$
where nxtp: nextState Ute-M (Suc r0) pp (rho (Suc r0) pp)
$\mu p p(r h o(S u c(S u c r 0)) p p)$
and mup: $\mu \mathrm{pp} \in$ SHOmsgVectors Ute-M (Suc rO) pp (rho (Suc rO))
(HOs (Suc r0) pp) (SHOs (Suc r0) pp)
and nxtq: nextState Ute-M (Suc r0) qq (rho (Suc r0) qq)
$\mu q q($ rho (Suc (Suc ro)) qq)
and muq: $\mu q q \in$ SHOmsgVectors Ute-M (Suc r0) qq (rho (Suc r0))
(HOs (Suc r0) qq) (SHOs (Suc r0) qq)
unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq by blast
from commR this pwp qqw have $w p=w$
by (auto dest: common-vote)
with $p w p$ show vote $(($ rho (Suc (Suc r0))) pp) $=$ Some $w$
by auto
qed
with that show ?thesis by auto
qed
from run obtain $\mu p^{\prime}$
where nxtp: nextState Ute-M (Suc (Suc r0)) $p$ (rho (Suc (Suc ro)) p)

$$
\mu p^{\prime}(\text { rho }(\text { Suc }(\text { Suc }(\text { Suc r0) )) p) }
$$

and mup': $\mu p^{\prime} \in$ SHOmsgVectors Ute-M (Suc (Suc r0)) p (rho (Suc (Suc $r 0)$ )
(HOs (Suc (Suc r0)) p) (SHOs (Suc (Suc ro)) p)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
have $\bigwedge q q . q q \in \operatorname{SHOs}($ Suc (Suc r0)) $p \cap H O s(S u c$ (Suc r0)) $p$

$$
\Longrightarrow \mu p^{\prime} q q=\text { Some }(\text { Vote }(\text { Some } w))
$$

proof -
fix $q q$
assume qqsho:qq $\in \operatorname{SHOs}(S u c(S u c r 0)) p \cap H O s(S u c$ (Suc ro)) $p$
from run obtain $\mu q q$ where
nxtqq:nextState Ute-M (Suc r0) qq (rho (Suc r0) qq)
$\mu q q$ (rho (Suc (Suc r0)) qq)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
from commR mup' nxtqq votew qqsho show $\mu p^{\prime} q q=$ Some (Vote (Some w)) by (auto dest: termination-argument-2)
qed
hence SHOs (Suc (Suc r0)) p $\cap$ HOs (Suc (Suc r0)) p $\subseteq\left\{q q \cdot \mu p^{\prime} q q=\right.$ Some $($ Vote $($ Some $\left.w))\right\}$

```
    by auto
    hence wsho: card (SHOs (Suc (Suc r0)) p \(\cap\) HOs (Suc (Suc ro)) p)
            \(\leq \operatorname{card}\left\{q q . \mu p^{\prime} q q=\operatorname{Some}(\operatorname{Vote}(\right.\) Some \(\left.w))\right\}\)
    by (auto simp: card-mono)
    from stp have step (Suc (Suc r0)) = Suc 0
    unfolding step-def by auto
    with nxtp have next1 (Suc (Suc ro)) p (rho (Suc (Suc ro)) p) \(\mu p^{\prime}\)
                            (rho (Suc (Suc (Suc r0))) p)
    by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
moreover
from \(e\) have \(E<\operatorname{card}(S H O s(S u c\) (Suc r0)) \(p \cap H O s\) (Suc (Suc ro)) p)
    by auto
with wsho have majv:card \(\left\{q q . \mu p^{\prime} q q=\operatorname{Some}(\operatorname{Vote}(\right.\) Some \(\left.w))\right\}>E\)
    by auto
ultimately
show ?thesis by (auto simp: next1-def)
qed
```


## $8.7 \mathcal{U}_{T, E, \alpha}$ Solves Weak Consensus

Summing up, all (coarse-grained) runs of $\mathcal{U}_{T, E, \alpha}$ for HO and SHO collections that satisfy the communication predicate satisfy the Weak Consensus property.

```
theorem ute-weak-consensus:
    assumes run: SHORun Ute-M rho HOs SHOs
    and commR: \forallr.SHOcommPerRd Ute-M (HOs r) (SHOs r)
    and commG:SHOcommGlobal Ute-M HOs SHOs
    shows weak-consensus ( }x\circ\mathrm{ (rho 0)) decide rho
    unfolding weak-consensus-def
    using ute-validity[OF run commR]
    ute-agreement[OF run commR]
    ute-termination[OF run commR commG]
    by auto
```

By the reduction theorem, the correctness of the algorithm carries over to the fine-grained model of runs.

```
theorem ute-weak-consensus-fg:
    assumes run: fg-run Ute-M rho HOs SHOs ( \(\lambda r\) q. undefined)
        and commR: \(\forall r\). SHOcommPerRd Ute-M (HOs r) (SHOs r)
        and commG: SHOcommGlobal Ute-M HOs SHOs
    shows weak-consensus ( \(\lambda p . x\) (state (rho 0) p)) decide (state ○ rho)
    (is weak-consensus ?inits - -)
proof (rule local-property-reduction[OF run weak-consensus-is-local])
    fix crun
    assume crun: CSHORun Ute-M crun HOs SHOs ( \(\lambda r\) q. undefined)
        and init: crun \(0=\) state (rho 0)
    from crun have SHORun Ute-M crun HOs SHOs by (unfold SHORun-def)
```

```
    from this commR commG
    have weak-consensus (x\circ(crun 0)) decide crun
    by (rule ute-weak-consensus)
    with init show weak-consensus?:nits decide crun
    by (simp add: o-def)
qed
end - context ute-parameters
end
theory AteDefs
imports ../HOModel
begin
```


## 9 Verification of the $\mathcal{A}_{T, E, \alpha}$ Consensus algorithm

Algorithm $\mathcal{A}_{T, E, \alpha}$ is presented in [3]. Like $\mathcal{U}_{T, E, \alpha}$, it is an uncoordinated algorithm that tolerates value faults, and it is parameterized by values $T$, $E$, and $\alpha$ that serve a similar function as in $\mathcal{U}_{T, E, \alpha}$, allowing the algorithm to be adapted to the characteristics of different systems. $\mathcal{A}_{T, E, \alpha}$ can be understood as a variant of OneThirdRule tolerating Byzantine faults.
We formalize in Isabelle the correctness proof of the algorithm that appears in [3], using the framework of theory HOModel.

### 9.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic HO model.
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)

## abbreviation

$N \equiv$ card (UNIV ::Proc set) — number of processes
The following record models the local state of a process.

```
record 'val pstate =
    x :: 'val - current value held by process
    decide :: 'val option - value the process has decided on, if any
```

The $x$ field of the initial state is unconstrained, but no decision has yet been taken.
definition Ate-initState where
Ate-initState $p$ st $\equiv($ decide st $=$ None $)$

The following locale introduces the parameters used for the $\mathcal{A}_{T, E, \alpha}$ algorithm and their constraints［3］．

```
locale ate-parameters =
    fixes }\alpha::nat and T::nat and E::na
    assumes TNaE:T\geq2*(N+2*\alpha-E)
        and TltN:T<N
        and EltN:E<N
```


## begin

The following are consequences of the assumptions on the parameters．

```
lemma majE: 2 * (E-\alpha)\geqN
using TNaE TltN by auto
lemma Egta: E> 人
using majE EltN by auto
lemma Tge2a: T\geq2 * 人
using TNaE EltN by auto
```

At every round，each process sends its current $x$ ．If it received more than $T$ messages，it selects the smallest value and store it in $x$ ．As in algorithm OneThirdRule，we therefore require values to be linearly ordered．
If more than $E$ messages holding the same value are received，the process decides that value．

```
definition mostOftenRcvd where
    mostOftenRcvd (msgs::Proc }=>\mathrm{ 'val option) 三
    {v.\forallw.card {qq. msgs qq = Some w}}\leq\mathrm{ card {qq. msgs qq=Some v}}
```


## definition

```
    Ate-sendMsg \(::\) nat \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) 'val pstate \(\Rightarrow\) 'val
```

where
Ate-sendMsgrpqst $\equiv x$ st

## definition

$$
\begin{gathered}
\text { Ate-nextState }:: \text { nat } \Rightarrow \text { Proc } \Rightarrow(\text { ('val::linorder }) \text { pstate } \Rightarrow(\text { Proc } \Rightarrow \text { 'val option }) \\
\Rightarrow{ }^{\prime} \text { 'val pstate } \Rightarrow \text { bool }
\end{gathered}
$$

where
Ate－nextState r $p$ st msgs st ${ }^{\prime} \equiv$

$$
\text { (if card }\{q . \text { msgs } q \neq \text { None }\}>T
$$

$$
\text { then } x s t^{\prime}=\text { Min }(\text { mostOftenRcvd msgs })
$$

$$
\text { else } x s t^{\prime}=x \text { st) }
$$

$\wedge\left(\quad\left(\exists v\right.\right.$. card $\{q$. msgs $q=$ Some $v\}>E \wedge$ decide st $^{\prime}=$ Some $\left.v\right)$ $\vee \neg(\exists v$ ．card $\{q$ ．msgs $q=$ Some $v\}>E)$
$\wedge$ decide $s t^{\prime}=$ decide $\left.s t\right)$

### 9.2 Communication Predicate for $\mathcal{A}_{T, E, \alpha}$

Following [3], we now define the communication predicate for the $\mathcal{A}_{T, E, \alpha}$ algorithm. The round-by-round predicate requires that no process may receive more than $\alpha$ corrupted messages at any round.

```
definition Ate-commPerRd where
    Ate-commPerRd HOrs SHOrs \equiv
    \forallp.card (HOrs p - SHOrs p) \leq\alpha
```

The global communication predicate stipulates the three following conditions:

- for every process $p$ there are infinitely many rounds where $p$ receives more than $T$ messages,
- for every process $p$ there are infinitely many rounds where $p$ receives more than $E$ uncorrupted messages,
- and there are infinitely many rounds in which more than $E-\alpha$ processes receive uncorrupted messages from the same set of processes, which contains more than $T$ processes.


## definition

```
Ate-commGlobal where
Ate-commGlobal HOs \(\mathrm{SHOs} \equiv\)
    \(\left(\forall r p . \exists r^{\prime}>r . c a r d\left(H O s r^{\prime} p\right)>T\right)\)
\(\wedge\left(\forall r p . \exists r^{\prime}>r\right.\).card \(\left(\right.\) SHOs \(\left.\left.r^{\prime} p \cap H O s r^{\prime} p\right)>E\right)\)
\(\wedge\left(\forall r . \exists r^{\prime}>r . \exists \pi 1 \pi\right.\) 2.
            card \(\pi 1>E-\alpha\)
    \(\wedge\) card \(\pi 2>T\)
    \(\wedge\left(\forall p \in \pi 1\right.\). HOs \(r^{\prime} p=\pi 2 \wedge\) SHOs \(\left.\left.r^{\prime} p \cap H O s r^{\prime} p=\pi \mathcal{Z}\right)\right)\)
```


### 9.3 The $\mathcal{A}_{T, E, \alpha}$ Heard-Of Machine

We now define the non-coordinated SHO machine for the $\mathcal{A}_{T, E, \alpha}$ algorithm by assembling the algorithm definition and its communication-predicate.

```
definition Ate-SHOMachine where
    Ate-SHOMachine \(=0\)
        CinitState \(=(\lambda\) p st crd. Ate-initState \(p(s t::(' v a l:: l i n o r d e r) ~ p s t a t e))\),
        sendMsg \(=\) Ate-sendMsg,
        CnextState \(=\left(\lambda r p\right.\) st msgs crd st' \({ }^{\prime}\). Ate-nextState \(r p\) st msgs st \(\left.{ }^{\prime}\right)\),
        SHOcommPerRd \(=(\) Ate-commPerRd:: Proc \(\mathrm{HO} \Rightarrow\) Proc \(\mathrm{HO} \Rightarrow\) bool \()\),
        SHOcommGlobal \(=\) Ate-commGlobal
    D
```

abbreviation
Ate-M $\equiv$ (Ate-SHOMachine::(Proc, 'val::linorder pstate, 'val) SHOMachine)

```
end - locale ate-parameters
end
theory AteProof
imports AteDefs ../Reduction
begin
context ate-parameters
begin
```


### 9.4 Preliminary Lemmas

If a process newly decides value $v$ at some round, then it received more than $E-\alpha$ messages holding $v$ at this round.

```
lemma decide-sent-msgs-threshold:
    assumes run: SHORun Ate-M rho HOs SHOs
    and comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and nvp: decide (rho r p) \(\neq\) Some \(v\)
    and \(v p\) : decide (rho (Suc r) p) = Some \(v\)
    shows card \(\{q q\). sendMsg Ate-Mr qq \(p(\) rho \(r q q)=v\}>E-\alpha\)
proof -
    from run obtain \(\mu p\)
            where mu: \(\mu p \in\) SHOmsgVectors Ate-M r \(p\) (rho r) (HOs r p) (SHOs r p)
                and nxt: nextState Ate-M r p (rho r p) \(\mu \mathrm{p}\) (rho (Suc r) p)
            by (auto simp: SHORun-eq SHOnextConfig-eq)
    from \(m u\)
    have \(\{q q . \mu p q q=\) Some \(v\}-(\) HOs r \(p-\) SHOs \(r\) r \()\)
                            \(\subseteq\{q q . \operatorname{sendMsg}\) Ate-Mrqq \(p(\) rho \(r q q)=v\}\)
            (is ?vrcvdp - ?ahop \(\subseteq\) ?vsentp)
        by (auto simp: SHOmsgVectors-def)
    hence card (?vrcudp - ?ahop) \(\leq\) card ?vsentp
        and card (?vrcvdp - ?ahop) \(\geq\) card ?vrcvdp - card ?ahop
        by (auto simp: card-mono diff-card-le-card-Diff)
    hence card ?vsentp \(\geq\) card ?vrcvdp - card ?ahop by auto
    moreover
    from nxt nvp vp have card ?vrcvdp > E
        by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
    moreover
    from comm have card (HOs r \(p-S H O s\) r \(p\) ) \(\leq \alpha\)
        by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
    ultimately
    show ?thesis using Egta by auto
qed
```

If more than $E-\alpha$ processes send a value $v$ to some process $q$ at some round, then $q$ will receive at least $N+2 * \alpha-E$ messages holding $v$ at this round.
lemma other-values-received:

```
    assumes comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and nxt: nextState Ate-M r q (rho r q) \muq ((rho (Suc r)) q)
    and muq: }\muq\inSHOmsgVectors Ate-M r q (rho r) (HOs r q) (SHOs r q)
    and vsent: card {qq. sendMsg Ate-Mr qq q (rho r qq) =v}>E - 人
        (is card ?vsent > -)
    shows card ({qq. \muq qq = Some v}\capHOs r q) \leqN+2*\alpha - E
proof -
    from nxt muq
    have ({qq. }\muqqq\not=S\mathrm{ Some v} }\cap\mathrm{ HOs r q) - (HOs r q - SHOs r q)
        \subseteq \{ q q . ~ s e n d M s g ~ A t e - M ~ r ~ q q ~ q ~ ( r h o ~ r ~ q q ) ~ \neq v \}
    (is ?notvrcvd - ?aho \subseteq ?notvsent)
    unfolding SHOmsgVectors-def by auto
    hence card ?notvsent }\geq\mathrm{ card (?noturcvd - ?aho)
    and card (?notvrcvd - ?aho) \geq card ?notvrcvd - card ?aho
    by (auto simp: card-mono diff-card-le-card-Diff)
    moreover
    from comm have card ?aho \leq <
        by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
    moreover
    have 1:card ?notvsent + card ?vsent = card (?notvsent \cup ?vsent)
    by (subst card-Un-Int) auto
    have ?notvsent \cup ?vsent = (UNIV::Proc set) by auto
    hence card (?notvsent \cup?vsent) =N by simp
    with 1 vsent have card ?notvsent }\leqN-(E+1-\alpha) by aut
    ultimately
    show ?thesis using EltN Egta by auto
qed
```

If more than $E-\alpha$ processes send a value $v$ to some process $q$ at some round $r$, and if $q$ receives more than $T$ messages in $r$, then $v$ is the most frequently received value by $q$ in $r$.

```
lemma mostOftenRcvd-v:
    assumes comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and nxt: nextState Ate-M r q (rho r q) \muq ((rho (Suc r)) q)
    and muq: }\muq\inSHOmsgVectors Ate-Mrq(rho r) (HOs r q) (SHOs r q)
    and threshold-T: card {qq. \muq qq = None} > T
    and threshold-E: card {qq. sendMsg Ate-Mrqq q (rho r qq) =v}>E - 人
    shows mostOftenRcvd }\muq={v
proof -
    from muq have hodef:HOs r q = {qq. \muq qq = None}
    unfolding SHOmsgVectors-def by auto
    from comm nxt muq threshold-E
    have card ({qq. \muq qq = Some v}\capHOs r q) \leqN+2*\alpha - E
    (is card ?heardnotv \leq -)
    by (rule other-values-received)
    moreover
    have card ?heardnotv \geqT+1- card {qq. }\muqqq=Some v
    proof -
```

```
    from muq
    have ?heardnotv = (HOs r q) - {qq. \muq qq= Some v}
        and {qq. \muq qq=Some v}\subseteqHOs r q
        unfolding SHOmsgVectors-def by auto
    hence card ?heardnotv = card (HOs r q) - card {qq. \muq qq = Some v}
        by (auto simp: card-Diff-subset)
    with hodef threshold-T show ?thesis by auto
qed
ultimately
have card {qq. \muq qq = Some v} > card ?heardnotv
    using TNaE by auto
moreover
{
    fix w
    assume w: w\not=v
    with hodef have {qq. }\muqqq=Some w}\subseteq\mathrm{ ?heardnotv by auto
    hence card {qq. \muq qq= Some w}\leqcard ?heardnotv by (auto simp: card-mono)
}
ultimately
have {w. card {qq. }\muqqq=\mathrm{ Some w} }\geq\mathrm{ card {qq. }\muqqq=Some v}}={v
    by force
    thus ?thesis unfolding mostOftenRcvd-def by auto
qed
If at some round more than \(E-\alpha\) processes have their \(x\) variable set to \(v\), then this is also true at next round.
```

```
lemma common-x-induct:
```

lemma common-x-induct:
assumes run: SHORun Ate-M rho HOs SHOs
assumes run: SHORun Ate-M rho HOs SHOs
and comm:SHOcommPerRd Ate-M (HOs (r+k)) (SHOs (r+k))
and comm:SHOcommPerRd Ate-M (HOs (r+k)) (SHOs (r+k))
and ih: card {qq. x (rho (r +k)qq) = v}> > - 人
and ih: card {qq. x (rho (r +k)qq) = v}> > - 人
shows card {qq. x (rho (r + Suc k) qq) =v}>E-\alpha
shows card {qq. x (rho (r + Suc k) qq) =v}>E-\alpha
proof -
proof -
from ih
from ih
have thrE:\forallpp.card {qq. sendMsg Ate-M (r+k) qq pp (rho (r+k)qq)=v}
have thrE:\forallpp.card {qq. sendMsg Ate-M (r+k) qq pp (rho (r+k)qq)=v}
> E-\alpha
> E-\alpha
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
{
{
fix qq
fix qq
assume kv:x (rho (r+k)qq)=v
assume kv:x (rho (r+k)qq)=v
from run obtain }\muq
from run obtain }\muq
where nxt: nextState Ate-M (r+k) qq (rho (r+k)qq) \muqq((rho (Suc (r
where nxt: nextState Ate-M (r+k) qq (rho (r+k)qq) \muqq((rho (Suc (r

+ k))) qq)
+ k))) qq)
and muq: \muqq\inSHOmsgVectors Ate-M (r+k)qq (rho (r +k))
and muq: \muqq\inSHOmsgVectors Ate-M (r+k)qq (rho (r +k))
(HOs (r+k)qq)(SHOs (r+k)qq)
(HOs (r+k)qq)(SHOs (r+k)qq)
by (auto simp: SHORun-eq SHOnextConfig-eq)
by (auto simp: SHORun-eq SHOnextConfig-eq)
have x (rho (r+Suc k)qq)=v
have x (rho (r+Suc k)qq)=v
proof (cases card {pp. \muqq pp\not= None} > T)

```
    proof (cases card {pp. \muqq pp\not= None} > T)
```

```
        case True
        with comm nxt muq thrE have mostOftenRcvd }\muqq={v
            by (auto dest: mostOftenRcvd-v)
        with nxt True show x (rho (r + Suc k) qq) =v
            by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
        next
            case False
            with nxt have x (rho (r+Suc k)qq)=x (rho (r+k)qq)
            by (auto simp:Ate-SHOMachine-def nextState-def Ate-nextState-def)
            with kv show }x\mathrm{ (rho (r + Suc k)qq)=v by simp
        qed
    }
    hence {qq.x (rho (r+k)qq)=v}\subseteq{qq. x (rho (r+Suck)qq)=v}
    by auto
    hence card {qq. x (rho (r+k)qq) =v}\leq card {qq. x (rho (r+Suc k)qq)=
v}
    by (auto simp: card-mono)
    with ih show ?thesis by auto
qed
```

Whenever some process newly decides value $v$, then any process that updates its $x$ variable will set it to $v$.

```
lemma common-x:
    assumes run: SHORun Ate-M rho HOs SHOs
    and comm: \forallr.SHOcommPerRd (Ate-M::(Proc,'val::linorder pstate, 'val) SHOMa-
chine)
    (HOsr) (SHOs r)
    and d1: decide (rho r p)\not=Some v
    and d2: decide (rho (Suc r) p)=Some v
    and qupdatex: x (rho (r +Suc k) q) \not=x (rho (r +k)q)
    shows }x\mathrm{ (rho (r + Suc k) q) =v
proof -
    from comm
    have SHOcommPerRd (Ate-M::(Proc,'val::linorder pstate,'val) SHOMachine)
            (HOs (r+k)) (SHOs (r+k)) ..
    moreover
    from run obtain }\mu
    where nxt: nextState Ate-M (r+k) q(rho (r+k)q) \muq(rho (r + Suc k)q)
        and muq: }\muq\in\mathrm{ SHOmsgVectors Ate-M (r+k) q(rho (r+k))
                                    (HOs (r+k)q) (SHOs (r+k)q)
    by (auto simp: SHORun-eq SHOnextConfig-eq)
moreover
from nxt qupdatex
have threshold-T: card {qq. \muq qq # None } > T
    and xsmall: x (rho (r+Suc k) q) = Min (mostOftenRcvd \muq)
    by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
moreover
have E-\alpha<card {qq. x (rho (r+k)qq)=v}
proof (induct k)
```

```
    from run comm d1 d2
    have E-\alpha<card {qq. sendMsg Ate-Mr qq p (rho r qq) =v}
    by (auto dest: decide-sent-msgs-threshold)
    thus E-\alpha<card {qq. x (rho (r+0)qq)=v}
        by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
    next
    fix k
    assume E-\alpha<card {qq. x (rho (r+k)qq)=v}
    with run comm show E-\alpha<card {qq.x (rho (r+Suck)qq)=v}
        by (auto dest: common-x-induct)
    qed
    with run
    have E-\alpha< card {qq. sendMsg Ate-M (r+k) qq q(rho (r+k)qq)=v}
    by (auto simp:Ate-SHOMachine-def Ate-sendMsg-def SHORun-eq SHOnextCon-
fig-eq)
    ultimately
    have mostOftenRcvd }\muq={v}\mathrm{ by (auto dest:mostOftenRcvd-v)
    with xsmall show?thesis by auto
qed
```

A process that holds some decision $v$ has decided $v$ sometime in the past.

```
lemma decisionNonNullThenDecided:
    assumes run: SHORun Ate-M rho HOs SHOs
        and dec: decide (rho n p) = Some v
    obtains m}\mathrm{ where m<n
            and decide (rho m p) \not= Some v
            and decide (rho (Suc m) p)=Some v
proof -
    let ?dec k= decide (rho k p)
    have ( }\forall\textrm{m}<n\mathrm{ . ?dec (Suc m) = ?dec m }\longrightarrow\mathrm{ ? ?dec (Suc m) }=\mathrm{ Some v) }\longrightarrow\mathrm{ ?dec n
# Some v
    (is ?P n is ?A n\longrightarrow-)
    proof (induct n)
            from run show ?P 0
                by (auto simp: Ate-SHOMachine-def SHORun-eq HOinitConfig-eq
                                    initState-def Ate-initState-def)
    next
            fix n
            assume ih:?P n thus ?P (Suc n) by force
    qed
    with dec that show ?thesis by auto
qed
```


### 9.5 Proof of Validity

Validity asserts that if all processes were initialized with the same value, then no other value may ever be decided.
theorem ate-validity:

```
assumes run: SHORun Ate-M rho HOs SHOs
    and comm: \forallr.SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and initv: }\forallq.x(rho 0 q)=
    and dp: decide (rho r p) = Some w
    shows w=v
proof -
    {
        fix r
    have }\forallqq. sendMsg Ate-M r qq p (rho r qq) =v
    proof (induct r)
        from run initv show }\forallqq. sendMsg Ate-M 0 qq p (rho 0 qq) =
        by (auto simp: SHORun-eq SHOnextConfig-eq Ate-SHOMachine-def Ate-sendMsg-def)
    next
        fix r
        assume ih:\forallqq. sendMsg Ate-M r qq p (rho r qq) =v
        have }\forallqq.x (rho (Sucr)qq)=
        proof
            fix qq
            from run obtain }\muq
                where nxt: nextState Ate-M r qq (rho r qq) \muqq (rho (Suc r) qq)
                    and mu: }\muqq\inSHOmsgVectors Ate-M r qq (rho r) (HOs r qq) (SHOs r
qq)
                by (auto simp: SHORun-eq SHOnextConfig-eq)
        from nxt
            have (card {pp. \muqq pp \not= None} > T ^ x (rho (Suc r) qq) = Min
(mostOftenRcvd }\muqq)
            \vee (card {pp. \muqq pp = None} \leqT ^ x (rho (Suc r) qq) =x (rho r qq))
            by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
            thus x (rho (Suc r) qq)=v
            proof safe
                assume x (rho (Suc r) qq) =x (rho r qq)
                with ih show ?thesis
                    by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
            next
                assume threshold-T:T<card {pp. \muqq pp \not=None}
                    and rsmall:x (rho (Suc r) qq) = Min (mostOftenRcvd \muqq)
                    have card {pp.\existsw.w\not=v^\muqq pp=Some w}\leqT div 2
                    proof -
                        from comm have 1:card (HOs r qq-SHOs r qq) \leq 人
                        by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
                    moreover
                    from mu ih
                    have SHOs r qq\capHOsr qq\subseteq{pp. \muqq pp=Some v}
                    and HOs r qq={pp. \muqq pp\not=None}
                    by (auto simp:SHOmsgVectors-def Ate-SHOMachine-def Ate-sendMsg-def)
                    hence {pp. \muqq pp}\not=N\mathrm{ None } - {pp. }\muqq pp=Some v
                        \subseteq \mp@code { H O s ~ r ~ q q - S H O s ~ r ~ q q }
```

```
        by auto
    hence card ({pp. \muqq pp\not=None} - {pp. \muqq pp = Some v})
                \leqcard (HOs r qq-SHOs r qq)
    by (auto simp:card-mono)
    ultimately
    have card ({pp. \muqq pp \not=None} - {pp. \muqq pp=Some v})\leqT div 2
        using Tge2a by auto
    moreover
    have {pp. \muqq pp\not= None} - {pp. \muqq pp = Some v}
        ={pp.\existsw.w\not=v\wedge \muqq pp=Some w} by auto
    ultimately
    show ?thesis by simp
    qed
    moreover
    have {pp. \muqq pp\not=None}
        = {pp.\muqq pp=Some v}\cup{pp.\existsw.w\not=v^\muqq pp=Some w}
    and {pp.\muqq pp=Some v}\cap{pp.\existsw.w\not=v^\muqq pp=Some w}=
{}
    by auto
    hence card {pp. \muqq pp}\not=N=None
        = card {pp. \muqq pp=Some v} + card {pp.\existsw.w\not=v^\muqq pp=
Some w}
    by (auto simp: card-Un-Int)
    moreover
    note threshold-T
    ultimately
    have card {pp. \muqq pp = Some v}> card {pp.\existsw.w\not=v\wedge \muqq pp=
Some w}
    by auto
    moreover
    {
    fix w
    assume w\not=v
```



```
        by auto
    hence card {pp. \muqq pp=Some w}\leqcard {pp.\existsw.w\not=v^\muqq pp=
Some w}
    by (auto simp: card-mono)
    }
ultimately
have zz:\w. w\not=v\Longrightarrow
                        card {pp.\muqq pp=Some w}<card {pp. \muqq pp = Some v}
    by force
    hence }\v.card {pp.\muqq pp=Some v}\leqcard {pp. \muqq pp=Some w
        "w=v
    by force
    with }zz\mathrm{ have mostOftenRcvd }\muqq={v
    by (force simp: mostOftenRcvd-def)
    with xsmall show x (rho (Suc r) qq) =v by auto
```

```
        qed
    qed
    thus \forallqq. sendMsg Ate-M (Suc r) qq p (rho (Suc r) qq)=v
        by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
    qed
}
note P= this
from run dp obtain rp
    where rp: rp<r decide (rho rp p)}\not=\mathrm{ Some w
        decide (rho (Suc rp) p) = Some w
    by (rule decisionNonNullThenDecided)
from run obtain }\mu
    where nxt: nextState Ate-M rp p (rho rp p) \mup (rho (Suc rp) p)
    and mu: }\mu\textrm{p}\inSHOmsgVectors Ate-M rp p (rho rp) (HOs rp p) (SHOs rp p)
    by (auto simp: SHORun-eq SHOnextConfig-eq)
{
    fix w
    assume w: w\not=v
    from comm have card (HOs rp p-SHOs rp p) \leq\alpha
        by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
    moreover
    from mu P
    have SHOs rp p\capHOs rp p\subseteq{pp. \mup pp=Some v}
        and HOs rp p ={pp. \mup pp\not=None}
        by (auto simp:SHOmsgVectors-def)
    hence {pp. \mup pp\not=None} - {pp. \mup pp=Some v}
            \subseteq H O s ~ r p ~ p - S H O s ~ r p ~ p
        by auto
    hence card ({pp. \mup pp\not=None} - {pp. \mup pp=Some v})
            scard (HOs rp p - SHOs rp p)
        by (auto simp: card-mono)
    ultimately
    have card ({pp. \mup pp\not=None} - {pp. \mup pp=Some v})<E
        using Egta by auto
    moreover
    from w have {pp. \mup pp=Some w}
                                    \subseteq \{ \{ p p . \mu p ~ p p \neq N o n e \} - \{ p p . \mu p ~ p p = S o m e v \}
    by auto
    hence card {pp. \mup pp = Some w}
        \leqcard ({pp. \mup pp\not= None} - {pp. \mup pp=Some v})
    by (auto simp: card-mono)
    ultimately
    have card {pp. \mup pp=Some w}<E by simp
}
hence PP: \bigwedgew. card {pp. \mup pp=Some w} \geqE\Longrightarroww=v by force
```

```
    from rp nxt mu have card {q. \mup q=Some w} > E
    by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def
        nextState-def Ate-nextState-def)
    with PP show ?thesis by auto
qed
```


### 9.6 Proof of Agreement

If two processes decide at the some round, they decide the same value.

```
lemma common-decision:
    assumes run: SHORun Ate-M rho HOs SHOs
    and comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and nvp: decide (rho r p) \(\neq\) Some \(v\)
    and \(v p\) : decide (rho (Suc r) p) = Some \(v\)
    and \(n w q\) : decide (rho \(r q\) ) \(\neq\) Some \(w\)
    and \(w q\) : decide (rho (Suc r) q) \(=\) Some \(w\)
    shows \(w=v\)
proof -
    have gtn: card \(\{q q\). sendMsg Ate-Mr qq \(p(\) rho \(r q q)=v\}\)
                + card \(\{q q\). sendMsg Ate-Mr qq q \((\) rho \(r q q)=w\}>N\)
    proof -
        from run comm nvp vp
        have card \(\{q q\). sendMsg Ate-Mr qq \(p(\) rho r \(q q)=v\}>E-\alpha\)
        by (rule decide-sent-msgs-threshold)
    moreover
    from run comm nwq wq
    have card \(\{q q\). sendMsg Ate-Mrqq \(q(\) rho \(r q q)=w\}>E-\alpha\)
        by (rule decide-sent-msgs-threshold)
    ultimately
    show ?thesis using majE by auto
qed
show ?thesis
proof (rule ccontr)
    assume \(v w: w \neq v\)
    have \(\forall q q\). sendMsg Ate-Mrqq \(p(r h o r q q)=\operatorname{sendMsg}\) Ate-Mrqq (rho r \(q q\) )
        by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
    with \(v w\)
    have \(\{q q\). sendMsg Ate-Mr qq \(p(\) rho \(r q q)=v\}\)
            \(\cap\{q q\). sendMsg Ate-Mrqq \(q(\) rho \(r q q)=w\}=\{ \}\)
        by auto
    with gtn
    have card \((\{q q\). sendMsg Ate-M r qq \(p(\) rho \(r q q)=v\}\)
                                    \(\cup\{q q\). sendMsg Ate-Mr \(q q q(\) rho \(r q q)=w\})>N\)
        by (auto simp: card-Un-Int)
    moreover
    have card \((\{q q . \operatorname{sendMsg}\) Ate-M r qq \(p(\) rho \(r q q)=v\}\)
                            \(\cup\{q q\). sendMsg Ate-M r qq \(q(\) rho \(r q q)=w\}) \leq N\)
        by (auto simp: card-mono)
```

```
    ultimately
    show False by auto
    qed
qed
```

If process $p$ decides at step $r$ and process $q$ decides at some later step $r+k$ then $p$ and $q$ decide the same value.
lemma laterProcessDecidesSameValue :
assumes run: SHORun Ate-M rho HOs SHOs
and comm: $\forall r$. SHOcommPerRd Ate-M (HOs r) (SHOs r)
and nd1: decide (rho r p) $=$ Some $v$
and d1: decide (rho (Suc r) p) = Some $v$
and $n d 2$ : decide $(r h o(r+k) q) \neq$ Some $w$
and d2: decide (rho $(\operatorname{Suc}(r+k)) q$ ) $=$ Some $w$
shows $w=v$
proof (rule ccontr)
assume $v d i f w: w \neq v$
have kgt0: $k>0$
proof (rule ccontr)
assume $\neg k>0$
hence $k=0$ by auto
with run comm nd1 d1 nd2 d2 have $w=v$
by (auto dest: common-decision)
with vdifw show False ..
qed
have 1: $\{q q$. sendMsg Ate-M r qq $p($ rho $r q q)=v\}$
$\cap\{q q . \operatorname{sendMsg}$ Ate-M $(r+k) q q q(r h o(r+k) q q)=w\}=\{ \}$
(is ?sentv $\cap$ ? sentw $=\{ \}$ )
proof (rule ccontr)
assume $\neg$ ?thesis
then obtain $q q$
where $x r v: x($ rho $r q q)=v$ and rkw: $x(r h o(r+k) q q)=w$
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
have $\exists k^{\prime}<k . x\left(r h o\left(r+k^{\prime}\right) q q\right) \neq w \wedge x\left(r h o\left(r+S u c k^{\prime}\right) q q\right)=w$
proof (rule ccontr)
assume $f$ : $\neg$ ?thesis
\{
fix $k^{\prime}$
assume $k k^{\prime}: k^{\prime}<k$ hence $x\left(r h o\left(r+k^{\prime}\right) q q\right) \neq w$
proof (induct $k^{\prime}$ )
from xrv vdifw
show $x(r h o(r+0) q q) \neq w$ by $\operatorname{simp}$
next
fix $k^{\prime}$
assume $i h: k^{\prime}<k \Longrightarrow x\left(r h o\left(r+k^{\prime}\right) q q\right) \neq w$
and $k s k^{\prime}:$ Suc $k^{\prime}<k$
from $k s k^{\prime}$ have $k^{\prime}<k$ by simp
with ih $f$ show $x\left(\right.$ rho $\left.\left(r+S u c k^{\prime}\right) q q\right) \neq w$ by auto

```
        qed
    }
    with f have }\forall\mp@subsup{k}{}{\prime}<k.x(rho (r+Suc k')qq) \not=w by aut
    moreover
    from kgt0 have k-1<k and kk:Suc (k-1)=k by auto
    ultimately
    have x (rho (r +Suc (k-1)) qq)}\not=w\mathrm{ by blast
    with rkw kk show False by simp
    qed
    then obtain k'
    where }\mp@subsup{k}{}{\prime}<
        and w: x (rho (r+Suc k')qq) =w
        and qqupdatex: x (rho (r+Suc k') qq)}=x(rho (r+k')qq
    by auto
    from run comm nd1 d1 qqupdatex
    have x (rho (r+Suc k')qq)=v by (rule common-x)
    with w vdifw show False by simp
qed
from run comm nd1 d1 have sentv: card ?sentv > E - \alpha
    by (auto dest: decide-sent-msgs-threshold)
```



```
    by (auto dest: decide-sent-msgs-threshold)
with sentv majE have (card ?sentv) + (card ?sentw) > N
    by simp
with 1 vdifw have 2: card (?sentv \cup ?sentw) > N
    by (auto simp: card-Un-Int)
have card (?sentv \cup ?sentw) \leqN
    by (auto simp: card-mono)
    with 2 show False by simp
qed
```

The Agreement property is now an immediate consequence.

```
theorem ate-agreement:
    assumes run: SHORun Ate-M rho HOs SHOs
    and comm: \(\forall r\). SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and \(p\) : decide (rho m \(p\) ) = Some \(v\)
    and \(q\) : decide (rho \(n q\) ) \(=\) Some \(w\)
    shows \(w=v\)
proof -
    from run \(p\) obtain \(k\) where
        \(k: k<m\) decide (rho \(k p) \neq\) Some \(v\) decide (rho (Suc \(k\) ) \(p\) ) \(=\) Some \(v\)
        by (rule decisionNonNullThenDecided)
    from run \(q\) obtain \(l\) where
        \(l: l<n\) decide \((\) rho \(l q) \neq\) Some \(w\) decide \((\) rho \((\) Suc \(l) q)=\) Some \(w\)
        by (rule decisionNonNullThenDecided)
    show ?thesis
    proof (cases \(k \leq l\) )
    case True
    then obtain \(i\) where \(l=k+i\) by (auto simp add: le-iff-add)
```

```
    with run comm kl show ?thesis
        by (auto dest: laterProcessDecidesSameValue)
    next
    case False
    hence l\leqk by simp
    then obtain i}\mathrm{ where m:k=l+i by (auto simp add:le-iff-add)
    with run comm kl show ?thesis
        by (auto dest: laterProcessDecidesSameValue)
    qed
qed
```


### 9.7 Proof of Termination

We now prove that every process must eventually decide, given the global and round-by-round communication predicates.
theorem ate-termination:
assumes run: SHORun Ate-M rho HOs SHOs
and commR: $\forall r$. (SHOcommPerRd::((Proc, 'val::linorder pstate, 'val) SHOMachine)

$$
\Rightarrow(\text { Proc } \mathrm{HO}) \Rightarrow(\text { Proc } \mathrm{HO}) \Rightarrow \text { bool })
$$

Ate-M (HOs r) (SHOs r)
and commG: SHOcommGlobal Ate-M HOs SHOs
shows $\exists r v$. decide (rho r $p$ ) $=$ Some $v$
proof -
from $\operatorname{comm} G$ obtain $r^{\prime} \pi 1 \pi 2$
where $\pi$ ea: card $\pi 1>E-\alpha$
and $\pi t$ : card $\pi 2>T$
and hosho: $\forall p \in \pi 1$. (HOs $\left.r^{\prime} p=\pi 2 \wedge S H O s r^{\prime} p \cap H O s r^{\prime} p=\pi 2\right)$
by (auto simp: Ate-SHOMachine-def Ate-commGlobal-def)

## obtain $v$ where

P1: $\forall p p$. card $\left\{q q\right.$. sendMsg Ate-M $\left(\right.$ Suc $\left.r^{\prime}\right) q q p p\left(r h o\left(\right.\right.$ Suc $\left.\left.\left.r^{\prime}\right) q q\right)=v\right\}>E$ $-\alpha$
proof -
have $\forall p \in \pi 1 . \forall q \in \pi 1 . x\left(\right.$ rho $\left(\right.$ Suc $\left.\left.r^{\prime}\right) p\right)=x\left(\right.$ rho $\left(\right.$ Suc $\left.\left.r^{\prime}\right) q\right)$
proof (clarify)
fix $p q$
assume $p: p \in \pi 1$ and $q: q \in \pi 1$
from run obtain $\mu p$
where nxtp: nextState Ate-M $r^{\prime} p\left(r h o r^{\prime} p\right) \mu p\left(r h o\left(S u c r^{\prime}\right) p\right)$
and mup: $\mu p \in$ SHOmsgVectors Ate-M $r^{\prime} p$ (rho $r^{\prime}$ ) (HOs r $r^{\prime} p$ ) (SHOs $r^{\prime}$
p)
by (auto simp: SHORun-eq SHOnextConfig-eq)
from run obtain $\mu q$
where nxtq: nextState Ate-M $r^{\prime} q\left(r h o r^{\prime} q\right) \mu q\left(r h o\left(S u c r^{\prime}\right) q\right)$
and muq: $\mu q \in$ SHOmsgVectors Ate-M $r^{\prime} q$ (rho $\left.r^{\prime}\right)\left(\right.$ HOs $\left.r^{\prime} q\right)\left(S H O s r^{\prime}\right.$
q)
by (auto simp: SHORun-eq SHOnextConfig-eq)
from mup muq $p q$
have $\{q q . \mu q q q \neq$ None $\}=H O s r^{\prime} q$
and $2:\left\{q q . \mu q q q=\right.$ Some $\left(\right.$ sendMsg Ate-M $\left.\left.r^{\prime} q q q\left(r h o r^{\prime} q q\right)\right)\right\}$
$\supseteq$ SHOs $r^{\prime} q \cap H O s r^{\prime} q$
and $\{q q . \mu p q q \neq$ None $\}=H O s r^{\prime} p$
and $4:\left\{q q . \mu p q q=\right.$ Some (sendMsg Ate-M $\left.\left.r^{\prime} q q p\left(r h o r^{\prime} q q\right)\right)\right\}$
$\supseteq$ SHOs $r^{\prime} p \cap H O s r^{\prime} p$
by (auto simp: SHOmsgVectors-def)
with $p$ q hosho
have aa: $12=\{q q . \mu q q q \neq$ None $\}$
and $c c: \pi 2=\{q q \cdot \mu p q q \neq$ None $\}$ by auto
from $p$ h hosho 2
have $b b:\left\{q q . \mu q q q=\right.$ Some (sendMsg Ate-M $\left.\left.r^{\prime} q q q\left(r h o r^{\prime} q q\right)\right)\right\} \supseteq \pi 2$
by auto
from $p$ q hosho 4
have $d d:\left\{q q . \mu p q q=\right.$ Some $\left(\right.$ sendMsg Ate-M $r^{\prime} q q p\left(\right.$ rho $\left.\left.\left.r^{\prime} q q\right)\right)\right\} \supseteq \pi \mathcal{Z}$
by auto
have Min $($ mostOftenRcvd $\mu p)=\operatorname{Min}(\operatorname{mostOftenRcvd} \mu q)$
proof -
have $\forall q q$. sendMsg Ate-M $r^{\prime} q q p$ (rho $r^{\prime} q q$ ) $=$ sendMsg Ate-M $r^{\prime} q q q\left(\right.$ rho $\left.r^{\prime} q q\right)$
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
with $a a b b c c d d$ have $\forall q q . \mu p q q \neq N o n e \longrightarrow \mu p q q=\mu q q q$
by force
moreover
from $a a b b c c d d$
have $\{q q . \mu p q q \neq$ None $\}=\{q q . \mu q q q \neq$ None $\}$ by auto
hence $\forall q q . \mu p q q=$ None $\longleftrightarrow \mu q q q=$ None by blast
hence $\forall q q . \mu p q q=$ None $\longrightarrow \mu p q q=\mu q q q$ by auto
ultimately
have $\forall q q . \mu p q q=\mu q q q$ by blast
thus ?thesis by (auto simp: mostOftenRcvd-def)
qed
with $\pi t$ aa nxtq $\pi t$ cc nxtp
show $x\left(\right.$ rho $\left(\right.$ Suc $\left.\left.r^{\prime}\right) p\right)=x\left(\right.$ rho $\left(\right.$ Suc $\left.\left.r^{\prime}\right) q\right)$
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
qed
then obtain $v$ where $P v: \forall p \in \pi 1 . x\left(r h o\left(\right.\right.$ Suc $\left.\left.r^{\prime}\right) p\right)=v$ by blast \{
fix $p p$
from $P v$ have $\forall p \in \pi 1$. sendMsg Ate-M (Suc r') $p$ pp (rho (Suc r') $p$ ) $=v$
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
hence card $\pi 1 \leq$ card $\{q q$. sendMsg Ate-M (Suc r') qq pp (rho (Suc r') qq)
$=v\}$
by (auto intro: card-mono)
with $\pi e a$
have $E-\alpha<$ card $\left\{q q\right.$. sendMsg Ate-M (Suc r $\left.r^{\prime}\right) q q$ pp (rho (Suc $\left.\left.r^{\prime}\right) q q\right)=$
\}
with that show ?thesis by blast
qed
\{
fix $k p p$
have $E-\alpha<$ card $\left\{q q\right.$. sendMsg Ate-M $\left(\right.$ Suc $\left.r^{\prime}+k\right) q q$ pp (rho $\left(S u c r^{\prime}+k\right)$
$q q)=v\}$
(is? ? $k$ )
proof (induct $k$ )
from P1 show ?P 0 by simp
next
fix $k$
assume $i h$ :? $k$
from commR
have (SHOcommPerRd::((Proc, 'val::linorder pstate, 'val) SHOMachine)
$\Rightarrow($ Proc HO$) \Rightarrow($ Proc HO$) \Rightarrow$ bool $)$
Ate-M (HOs $\left(\right.$ Suc $\left.\left.r^{\prime}+k\right)\right)\left(\right.$ SHOs $\left(\right.$ Suc $\left.\left.r^{\prime}+k\right)\right)$..
moreover
from $i h$ have $E-\alpha<\operatorname{card}\left\{q q . x\left(r h o\left(S u c r^{\prime}+k\right) q q\right)=v\right\}$
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
ultimately
have $E-\alpha<$ card $\left\{q q . x\left(\right.\right.$ rho $\left(S u c r^{\prime}+\right.$ Suc $\left.\left.\left.k\right) q q\right)=v\right\}$
by (rule common-x-induct[OF run])
thus ?P (Suc k)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
qed
\}
note $P 2=$ this
\{
fix $k p p$
assume ppupdatex: $x\left(\right.$ rho $\left(\right.$ Suc $r^{\prime}+$ Suc $\left.\left.k\right) p p\right) \neq x\left(r h o\left(S u c r^{\prime}+k\right) p p\right)$
from commR
have (SHOcommPerRd::((Proc, 'val::linorder pstate, 'val) SHOMachine)
$\Rightarrow($ Proc HO$) \Rightarrow($ Proc HO$) \Rightarrow$ bool $)$
Ate-M $\left(\right.$ HOs $\left(\right.$ Suc $\left.\left.r^{\prime}+k\right)\right)\left(\right.$ SHOs $\left(\right.$ Suc $\left.\left.r^{\prime}+k\right)\right) .$.
moreover
from run obtain $\mu p p$
where nxt:nextState Ate-M (Suc $\left.r^{\prime}+k\right) p p\left(r h o\left(S u c r^{\prime}+k\right) p p\right) \mu p p$
(rho (Suc r $r^{\prime}+$ Suc $k$ ) pp)
and $m u: \mu p p \in S H O m s g V e c t o r s$ Ate-M $\left(S u c r^{\prime}+k\right) p p\left(r h o\left(S u c r^{\prime}+k\right)\right)$
(HOs $\left(\right.$ Suc $\left.\left.r^{\prime}+k\right) p p\right)\left(S H O s\left(S u c r^{\prime}+k\right) p p\right)$
by (auto simp: SHORun-eq SHOnextConfig-eq)
moreover
from nxt ppupdatex

```
    have threshold-T: card {qq. \mupp qq = None} > T
        and xsmall: x (rho (Suc r't Suc k) pp) = Min (mostOftenRcvd \mupp)
        by (auto simp:Ate-SHOMachine-def nextState-def Ate-nextState-def)
    moreover
    from P2
    have E-\alpha<card {qq. sendMsg Ate-M (Suc r' + k) qq pp (rho (Suc r'}+k
qq) =v}.
    ultimately
    have mostOftenRcvd }\mupp={v}\mathrm{ by (auto dest!: mostOftenRcvd-v)
    with xsmall
    have }x\mathrm{ (rho (Suc r' +Suc k) pp)=v by simp
}
note P3 = this
have P4:\forallpp.\existsk.x (rho (Suc r't Suc k) pp)=v
proof
    fix pp
    from commG have \existsr '\prime}>\mp@subsup{r}{}{\prime}.card (HOs r'\prime pp)>
    by (auto simp:Ate-SHOMachine-def Ate-commGlobal-def)
    then obtain k where Suc r'}+k>\mp@subsup{r}{}{\prime}\mathrm{ and t:card (HOs (Suc r' +k) pp)>T
    by (auto dest: less-imp-Suc-add)
    moreover
    from run obtain }\mup
        where nxt: nextState Ate-M (Suc r r}+k)pp(rho (Suc r' +k) pp) \mup
                        (rho (Suc r'}+\mathrm{ Suc k) pp)
            and mu: \mupp S SHOmsgVectors Ate-M (Suc r' +k) pp (rho (Suc r' +k))
                        (HOs (Suc r r}+k)pp)(SHOs(Suc r' +k)pp
    by (auto simp: SHORun-eq SHOnextConfig-eq)
    moreover
    have x (rho (Suc r'tSuc k) pp)=v
    proof -
    from commR
    have (SHOcommPerRd::((Proc, 'val::linorder pstate, 'val::linorder) SHOMa-
chine)
\[
\Rightarrow(\text { Proc } H O) \Rightarrow(\text { Proc } H O) \Rightarrow \text { bool })
\]
\[
\text { Ate-M }\left(\text { HOs }\left(\text { Suc } r^{\prime}+k\right)\right)\left(\text { SHOs }\left(\text { Suc } r^{\prime}+k\right)\right) \text {.. }
\]
    moreover
    from mu have HOs (Suc r'}+k)pp={q. \mupp q\not=None
        by (auto simp: SHOmsgVectors-def)
    with nxt t
    have threshold-T: card {q. \mupp q\not=None} > T
        and xsmall: x (rho (Suc r'}+\mathrm{ Suc k) pp) = Min (mostOftenRcvd upp)
        by (auto simp:Ate-SHOMachine-def nextState-def Ate-nextState-def)
    moreover
    from P2
    have E - 人< card {qq. sendMsg Ate-M (Suc r' + k) qq pp (rho (Suc r' +
k) qq) = v} .
    ultimately
    have mostOftenRcvd }\mupp={v
```

```
            using nxt mu by (auto dest!: mostOftenRcvd-v)
            with xsmall show ?thesis by auto
    qed
    thus \existsk.x (rho (Suc r' + Suc k) pp)=v..
qed
have P5a: \forallpp.\existsrr.\forallk.x (rho (rr +k) pp)=v
proof
    fix pp
    from P4 obtain rk where
        xrrv: x (rho (Suc r' + Suc rk) pp)=v(is x (rho ?rr pp)=v)
        by blast
    have }\forallk.x(rho(?rr +k)pp)=
    proof
        fix }
        show x (rho (?rr + k) pp)=v
        proof (induct k)
            from xrrv show x (rho (?rr + 0) pp)=v by simp
        next
            fix }
            assume ih: x (rho (?rr + k) pp)=v
            obtain }\mp@subsup{k}{}{\prime}\mathrm{ where rrk:Suc r' }+\mp@subsup{k}{}{\prime}=?rr +k by aut
            show x (rho (?rr + Suc k) pp) =v
            proof (rule ccontr)
                assume nv: x (rho (?rr + Suck) pp)\not=v
                with rrk ih
                have x (rho (Suc r' +Suc k') pp)}\not=x(rho(Suc r' + k') pp
                by (simp add: ac-simps)
            hence x (rho (Suc r'+Suc k') pp)=v by (rule P3)
            with rrk nv show False by (simp add: ac-simps)
            qed
        qed
    qed
    thus \existsrr.\forallk.x (rho (rr +k) pp)=v by blast
qed
from P5a have \existsF.\forallppk.x (rho (F pp +k) pp)=v by (rule choice)
then obtain R::(Proc => nat)
    where imgR: R'(UNIV::Proc set) }\not={
        and R:\forallppk.x (rho (R pp+k)pp)=v
    by blast
define rr where rr = Max (R'UNIV)
have P5: \forallr'>rr. }\forallpp.x(rho r'pp)=
proof (clarify)
    fix r}\mp@subsup{r}{}{\prime}p
    assume r': r'>rr
    hence }\mp@subsup{r}{}{\prime}>R pp by (auto simp: rr-def
    then obtain i where r'=R pp+i
```

```
        by (auto dest: less-imp-Suc-add)
    with \(R\) show \(x\left(r h o r^{\prime} p p\right)=v\) by auto
    qed
    from commG have \(\exists r^{\prime}>r r\). card (SHOs \(\left.r^{\prime} p \cap H O s r^{\prime} p\right)>E\)
    by (auto simp: Ate-SHOMachine-def Ate-commGlobal-def)
    with \(P 5\) obtain \(r^{\prime}\)
    where \(r^{\prime}>r r\)
        and card (SHOs \(\left.r^{\prime} p \cap H O s r^{\prime} p\right)>E\)
        and \(\forall p p\). sendMsg Ate-M \(r^{\prime} p p p\left(\right.\) rho \(\left.r^{\prime} p p\right)=v\)
    by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
moreover
from run obtain \(\mu p\)
    where nxt: nextState Ate-M \(r^{\prime} p\left(r h o r^{\prime} p\right) \mu p(r h o(S u c ~ r i) p)\)
        and \(m u: \mu p \in S H O m s g\) Vectors Ate-M \(r^{\prime} p\left(r h o r r^{\prime}\right)\left(H O s r^{\prime} p\right)\left(S H O s r^{\prime} p\right)\)
    by (auto simp: SHORun-eq SHOnextConfig-eq)
from \(m u\)
have card (SHOs r'p HOs r \(r^{\prime} p\) )
            \(\leq \operatorname{card}\left\{q . \mu p q=\right.\) Some (sendMsg Ate-M \(\left.\left.r^{\prime} q p\left(r h o r^{\prime} q\right)\right)\right\}\)
    by (auto simp: SHOmsgVectors-def intro: card-mono)
    ultimately
    have threshold-E: card \(\{q . \mu p q=\) Some \(v\}>E\) by auto
    with nxt show ?thesis
    by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
qed
```


## $9.8 \mathcal{A}_{T, E, \alpha}$ Solves Weak Consensus

Summing up, all (coarse-grained) runs of $\mathcal{A}_{T, E, \alpha}$ for HO and SHO collections that satisfy the communication predicate satisfy the Weak Consensus property.

```
theorem ate-weak-consensus:
    assumes run: SHORun Ate-M rho HOs SHOs
    and commR: }\forall\mathrm{ r. SHOcommPerRd Ate-M (HOs r)(SHOs r)
    and commG:SHOcommGlobal Ate-M HOs SHOs
shows weak-consensus (x\circ(rho 0)) decide rho
unfolding weak-consensus-def using assms
by (auto elim: ate-validity ate-agreement ate-termination)
```

By the reduction theorem, the correctness of the algorithm carries over to the fine-grained model of runs.

```
theorem ate-weak-consensus-fg:
    assumes run: fg-run Ate-M rho HOs SHOs (\lambdar q. undefined)
    and commR: \forallr.SHOcommPerRd Ate-M (HOs r) (SHOs r)
    and commG:SHOcommGlobal Ate-M HOs SHOs
    shows weak-consensus ( }\lambdap.x(\mathrm{ state (rho 0) p)) decide (state ○ rho)
    (is weak-consensus ?inits - -)
proof (rule local-property-reduction[OF run weak-consensus-is-local])
```

```
    fix crun
    assume crun: CSHORun Ate-M crun HOs SHOs (\lambdar q. undefined)
        and init: crun 0 = state (rho 0)
    from crun have SHORun Ate-M crun HOs SHOs by (unfold SHORun-def)
    from this commR commG
    have weak-consensus ( }x\circ(\mathrm{ crun 0)) decide crun
        by (rule ate-weak-consensus)
    with init show weak-consensus ?inits decide crun
    by (simp add: o-def)
qed
end - context ate-parameters
end
theory EigbyzDefs
imports ../HOModel
begin
```


## 10 Verification of the EIGByz $_{f}$ Consensus Algo- $^{\text {Con }}$ rithm

Lynch [12] presents $E I G B y z_{f}$, a version of the exponential information gathering algorithm tolerating Byzantine faults, that works in $f$ rounds, and that was originally introduced in [1].
We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable 'proc of the generic HO model.
typedecl Proc - the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
abbreviation
$N \equiv$ card (UNIV ::Proc set) — number of processes
The algorithm is parameterized by $f$, which represents the number of rounds and the height of the tree data structure (see below).
axiomatization $f::$ nat
where $f: f<N$

### 10.1 Tree Data Structure

The algorithm relies on propagating information about the initially proposed values among all the processes. This information is stored in trees whose branches are labeled by lists of (distinct) processes. For example, the interpretation of an entry $[p, q] \mapsto$ Some $v$ is that the current process heard from process $q$ that it had heard from process $p$ that its proposed value is
$v$. The value initially proposed by the process itself is stored at the root of the tree.
We introduce the type of labels, which encapsulate lists of distinct process identifiers and whose length is at most $f+1$.
definition Label $=\{x s::$ Proc list. length $x s \leq$ Suc $f \wedge$ distinct $x s\}$
typedef Label $=$ Label
by (auto simp: Label-def intro: exI[where $x=[]]$ ) - the empty list is a label
There is a finite number of different labels.

```
lemma finite-Label: finite Label
proof -
    have Label \(\subseteq\{x s\). set \(x s \subseteq(U N I V::\) Proc set \() \wedge\) length \(x s \leq\) Suc \(f\}\)
        by (auto simp: Label-def)
    moreover
    have finite \(\{x s\). set \(x s \subseteq(U N I V::\) Proc set \() \wedge\) length \(x s \leq S u c f\}\)
        by (rule finite-lists-length-le) auto
    ultimately
    show ?thesis by (auto elim: finite-subset)
qed
lemma finite-UNIV-Label: finite (UNIV::Label set)
proof -
    from finite-Label have finite (Abs-Label' Label) by simp
    moreover
    \{
        fix \(l:\) :Label
        have \(l \in\) Abs-Label ' Label
            by (rule Abs-Label-cases) auto
    \}
    hence (UNIV::Label set) \(=(\) Abs-Label' Label) by auto
    ultimately show?thesis by simp
qed
lemma finite-Label-set [iff]: finite ( \(S\) :: Label set)
    using finite-UNIV-Label by (auto intro: finite-subset)
```

Utility functions on labels.
definition root-node where
root-node $\equiv$ Abs-Label []
definition length-lbl where
length-lbl $l \equiv$ length (Rep-Label l)
lemma length-lbl [intro]: length-lbl l $\leq$ Suc $f$
unfolding length-lbl-def using Label-def Rep-Label by auto
definition is-leaf where
is-leaf $l \equiv$ length-lbl $l=S u c f$

```
definition last-lbl where
    last-lbl l \equiv last (Rep-Label l)
definition butlast-lbl where
    butlast-lbl l \equivAbs-Label (butlast (Rep-Label l))
definition set-lbl where
    set-lbl l=set (Rep-Label l)
```

The children of a non-leaf label are all possible extensions of that label.

```
definition children where
    children l \(\equiv\)
    if is-leaf \(l\)
    then \(\}\)
    else \(\{\) Abs-Label (Rep-Label l @ \([p]) \mid p . p \notin\) set-lbl l \}
```


### 10.2 Model of the Algorithm

The following record models the local state of a process.

```
record 'val pstate =
    vals :: Label # 'val option
    newvals :: Label => 'val
    decide :: 'val option
```

Initially, no values are assigned to non-root labels, and an arbitrary value is assigned to the root: that value is interpreted as the initial proposal of the process. No decision has yet been taken, and the newvals field is unconstrained.

```
definition EIG-initState where
    EIG-initState p st \(\equiv\)
    \((\forall l .(\) vals st \(l=\) None \()=(l \neq\) root-node \())\)
    \(\wedge\) decide st \(=\) None
```

type-synonym 'val Msg $=$ Label $\Rightarrow$ 'val option

At every round, every process sends its current vals tree to all processes. In fact, only the level of the tree corresponding to the round number is used (cf. definition of extend-vals below).
definition EIG-sendMsg where
EIG-sendMsg r p q st $\equiv$ vals st
During the first $f-1$ rounds, every process extends its tree vals according to the values received in the round. No decision is taken.

```
definition extend-vals where
    extend-vals r p st msgs st'}
    vals st' = ( }\lambdal
```

```
if length-lbl l \(=\) Suc \(r \wedge\) msgs \((\) last-lbl l \() \neq\) None
then (the (msgs (last-lbl l))) (butlast-lbl l)
else if length-lbl \(l=S u c r \wedge\) msgs (last-lbl l) \(=\) None then None
else vals st l)
```


## definition next-main where

```
next-main \(r\) p st msgs \(s t^{\prime} \equiv\) extend-vals \(r p\) st msgs st \({ }^{\prime} \wedge\) decide \(s t^{\prime}=\) None
```

In the final round, in addition to extending the tree as described previously, processes construct the tree newvals, starting at the leaves. The values at the leaves are copied from vals, except that missing values None are replaced by the default value undefined. Moving up, if there exists a majority value among the children, it is assigned to the parent node, otherwise the parent node receives the default value undefined. The decision is set to the value computed for the root of the tree.

```
fun fixupval :: 'val option \(\Rightarrow\) 'val where
    fixupval None \(=\) undefined
| fixupval (Some \(v\) ) \(=v\)
definition has-majority :: 'val \(\Rightarrow\left({ }^{\prime} a \Rightarrow\right.\) 'val \() \Rightarrow\) 'a set \(\Rightarrow\) bool where
    has-majority vg \(S \equiv\) card \(\{e \in S . g e=v\}>(\operatorname{card} S)\) div 2
definition check-newvals :: 'val pstate \(\Rightarrow\) bool where
    check-newvals st \(\equiv\)
        \(\forall l\). is-leaf \(l \wedge\) newvals st \(l=\) fixupval (vals st \(l\) )
        \(\vee \neg(\) is-leaf \(l) \wedge\)
            \(((\exists\). has-majority \(w\) (newvals st) \((\) children \(l) \wedge\) newvals st \(l=w)\)
            \(\vee(\neg(\exists\) w. has-majority \(w\) (newvals st) (children \(l))\)
                    \(\wedge\) newvals st \(l=\) undefined) \()\)
definition next-end where
    next-end \(r p\) st msgs st \({ }^{\prime} \equiv\)
        extend-vals \(r p\) st msgs st'
    \(\wedge\) check-newvals st \({ }^{\prime}\)
    \(\wedge\) decide st' \(=\) Some (newvals st \({ }^{\prime}\) root-node)
```

The overall next-state relation is defined such that every process applies nextMain during rounds $0, \ldots, f-1$, and applies nextEnd during round $f$. After that, the algorithm terminates and nothing changes anymore.

```
definition EIG-nextState where
    EIG-nextState r\equiv
    if r}<f\mathrm{ then next-main r
    else if r = f then next-end r
    else ( }\lambdap\mathrm{ st msgs st'. st' = st)
```


### 10.3 Communication Predicate for $E I G B y z_{f}$

The secure kernel $S K r$ w.r.t. given HO and SHO collections consists of the process from which every process receives the correct message.

```
definition SKr :: Proc HO => Proc HO => Proc set where
    SKr HO SHO \equiv{q.\forallp.q\inHOp\capSHO p}
```

The secure kernel $S K$ of an entire execution (i.e., for sequences of HO and SHO collections) is the intersection of the secure kernels for all rounds. Obviously, only the first $f$ rounds really matter, since the algorithm terminates after that.

```
definition \(S K::(\) nat \(\Rightarrow\) Proc \(H O) \Rightarrow(\) nat \(\Rightarrow\) Proc \(H O) \Rightarrow\) Proc set where
    SK HOs SHOs \(\equiv\{q . \forall r . q \in \operatorname{SKr}(H O s r)(\) SHOs \(r)\}\)
```

The round-by-round predicate requires that the secure kernel at every round contains more than $(N+f)$ div 2 processes.
definition EIG-commPerRd where

$$
\text { EIG-commPerRd HO SHO 三card }(S K r H O S H O)>(N+f) \text { div } 2
$$

The global predicate requires that the secure kernel for the entire execution contains at least $N-f$ processes. Messages from these processes are always correctly received by all processes.

```
definition EIG-commGlobal where
    EIG-commGlobal HOs SHOs \equivcard (SK HOs SHOs)\geqN-f
```

The above communication predicates differ from Lynch's presentation of $E I G B y z_{f}$. In fact, the algorithm was originally designed for synchronous systems with reliable links and at most $f$ faulty processes. In such a system, every process receives the correct message from at least the non-faulty processes at every round, and therefore the global predicate EIG-commGlobal is satisfied. The standard correctness proof assumes that $N>3 f$, and therefore $N-f>(N+f) \div 2$. Since moreover, for any $r$, we obviously have

$$
\left(\bigcap_{p \in \Pi, r^{\prime} \in \mathbb{N}} S H O\left(p, r^{\prime}\right)\right) \subseteq\left(\bigcap_{p \in \Pi} S H O(p, r)\right),
$$

it follows that any execution of EIGByz $f$ where $N>3 f$ also satisfies EIG-commPerRd at any round. The standard correctness hypotheses thus imply our communication predicates.

However, our proof shows that $E I G B y z_{f}$ can indeed tolerate more transient faults than the standard bound can express. For example, consider the case where $N=5$ and $f=2$. Our predicates are satisfied in executions where two processes exhibit transient faults, but never fail simultaneously. Indeed, in such an execution, every process receives four correct messages at every round, hence EIG-commPerRd always holds. Also, EIG-commGlobal is satisfied because there are three processes from which every process receives
the correct messages at all rounds. By our correctness proof, it follows that $E I G B y z_{f}$ then achieves Consensus, unlike what one could expect from the standard correctness predicate. This observation underlines the interest of expressing assumptions about transient faults, as in the HO model.

### 10.4 The EIGByz $z_{f}$ Heard-Of Machine

We now define the non-coordinated SHO machine for EIGByz $f$ by assembling the algorithm definition and its communication-predicate.

```
definition EIG-SHOMachine where
    EIG-SHOMachine = 0
        CinitState = (\lambda p st crd. EIG-initState p st),
        sendMsg = EIG-sendMsg,
        CnextState = ( }\lambdarp\mathrm{ st msgs crd st'. EIG-nextState r p st msgs st'})
        SHOcommPerRd = EIG-commPerRd,
        SHOcommGlobal = EIG-commGlobal
    D
```

abbreviation EIG-M $\equiv$ (EIG-SHOMachine::(Proc, 'val pstate, 'val Msg) SHOMachine)
end
theory EigbyzProof
imports EigbyzDefs ../ Majorities ../Reduction
begin

### 10.5 Preliminary Lemmas

Some technical lemmas about labels and trees.
lemma not-leaf-length:
assumes $l$ : $\neg$ (is-leaf $l$ )
shows length-lbl $l \leq f$
using length-lbl[of l] by (simp add: is-leaf-def)
lemma nil-is-Label: []$\in$ Label
by (auto simp: Label-def)
lemma card-set-lbl: card (set-lbl l) = length-lbl l
unfolding set-lbl-def length-lbl-def
using Rep-Label[of l, unfolded Label-def]
by (auto elim: distinct-card)
lemma Rep-Label-root-node [simp]: Rep-Label root-node $=[]$
using nil-is-Label by (simp add: root-node-def Abs-Label-inverse)
lemma root-node-length $[$ simp $]$ : length-lbl root-node $=0$
by (simp add: length-lbl-def)

```
lemma root-node-not-leaf: }\neg\mathrm{ (is-leaf root-node)
    by (simp add: is-leaf-def)
```

Removing the last element of a non-root label gives a label.

```
lemma butlast-rep-in-label:
    assumes \(l: l \neq\) root-node
    shows butlast (Rep-Label l) \(\in\) Label
proof -
    have Rep-Label \(l \neq[]\)
    proof
        assume Rep-Label \(l=[]\)
        hence Rep-Label l \(=\) Rep-Label root-node by simp
        with \(l\) show False by (simp only: Rep-Label-inject)
    qed
    with Rep-Label[ of l] show ?thesis
        by (auto simp: Label-def elim: distinct-butlast)
qed
```

The label of a child is well-formed.
lemma Rep-Label-append:
assumes $l$ : $\neg$ (is-leaf $l$ )
shows $($ Rep-Label l @ $[p] \in$ Label $)=(p \notin$ set-lbl l)
(is ?lhs $=? r h s$ is $\left.\left(? l^{\prime} \in-\right)=-\right)$
proof
assume lhs: ?lhs thus ?rhs
by (auto simp: Label-def set-lbl-def)
next
assume $p$ : ?rhs
from $l[$ THEN not-leaf-length $]$ have length ? $l^{\prime} \leq$ Suc $f$
by (simp add: length-lbl-def)
moreover
from Rep-Label[of l] have distinct (Rep-Label l)
by (simp add: Label-def)
with $p$ have distinct ? $l^{\prime}$ by (simp add: set-lbl-def)
ultimately
show ?lhs by (simp add: Label-def)
qed

The label of a child is the label of the parent, extended by a process.

```
lemma label-children:
    assumes c:c\in children l
    shows \existsp.p\not\inset-lbl l\wedgeRep-Label c=Rep-Label l @ [p]
proof -
    from c obtain p
        where p:p\not\in set-lbl l and l: \neg(is-leaf l)
            and c:c=Abs-Label (Rep-Label l @ [p])
        by (auto simp: children-def)
    with Rep-Label-append[OF l] show ?thesis
```

```
    by (auto simp:Abs-Label-inverse)
qed
```

The label of any child node is one longer than the label of its parent.

```
lemma children-length:
    assumes l\in children h
    shows length-lbl l = Suc (length-lbl h)
    using label-children[OF assms] by (auto simp: length-lbl-def)
```

The root node is never a child.

```
lemma children-not-root:
    assumes root-node }\in\mathrm{ children l
    shows P
    using label-children[OF assms] Abs-Label-inverse[OF nil-is-Label]
    by (auto simp: root-node-def)
```

The label of a child with the last element removed is the label of the parent.

```
lemma children-butlast-lbl:
    assumes \(c \in\) children \(l\)
    shows butlast-lbl c=l
    using label-children[OF assms]
    by (auto simp: butlast-lbl-def Rep-Label-inverse)
```

The root node is not a child, and it is the only such node.

```
lemma root-iff-no-child: (l = root-node ) = ( }\forall\mp@subsup{l}{}{\prime}.l\not\in\mathrm{ children l'}
proof
    assume l= root-node
    thus \foralll'.l children l' by (auto elim: children-not-root)
next
    assume rhs: \foralll'. l & children l'
    show l= root-node
    proof (rule rev-exhaust[of Rep-Label l])
        assume Rep-Label l= []
        hence Rep-Label l=Rep-Label root-node by simp
        thus ?thesis by (simp only:Rep-Label-inject)
    next
    fix l' q
    assume l':Rep-Label l= l' @ [q]
    let ?l' = Abs-Label l'
    from Rep-Label[of l] l' have l' 
    hence repl': Rep-Label ?l' = l' by (rule Abs-Label-inverse)
    from Rep-Label[of l] l' have l' @ [q]\in Label by (simp add: Label-def)
    with l' have Rep-Label l= Rep-Label (Abs-Label (l' @ [q]))
        by (simp add: Abs-Label-inverse)
    hence l=Abs-Label (l' @ [q]) by (simp add: Rep-Label-inject)
    moreover
    from Rep-Label[of l] l' have length l'}< Suc f q\not\in set l' 
```

```
            by (auto simp: Label-def)
    moreover
    note repl'
    ultimately have l children ?l'
            by (auto simp: children-def is-leaf-def length-lbl-def set-lbl-def)
    with rhs show ?thesis by blast
    qed
qed
```

If some label $l$ is not a leaf, then the set of processes that appear at the end of the labels of its children is the set of all processes that do not appear in $l$.

```
lemma children-last-set:
    assumes l:}\neg\mathrm{ (is-leaf l)
    shows last-lbl'(children l) = UNIV - set-lbl l
proof
    show last-lbl'(children l)\subseteqUNIV - set-lbl l
        by (auto dest: label-children simp: last-lbl-def)
next
    show UNIV - set-lbl l\subseteq last-lbl '(children l)
    proof (auto simp: image-def)
        fix p
        assume p:p\not\in set-lbl l
        with l have c:Abs-Label (Rep-Label l @ [p])\inchildren l
            by (auto simp: children-def)
        with Rep-Label-append[OF l] p
        show }\existsc\in\mathrm{ children l. p = last-lbl c
        by (force simp:last-lbl-def Abs-Label-inverse)
    qed
qed
```

The function returning the last element of a label is injective on the set of children of some given label.

```
lemma last-lbl-inj-on-children:inj-on last-lbl (children l)
proof (auto simp: inj-on-def)
    fix \(c c^{\prime}\)
    assume \(c: c \in\) children \(l\) and \(c^{\prime}: c^{\prime} \in\) children \(l\)
        and eq: last-lbl \(c=l a s t-l b l c^{\prime}\)
    from \(c c^{\prime}\) obtain \(p p^{\prime}\)
        where \(p\) : Rep-Label \(c=\) Rep-Label \(l @[p]\)
            and \(p^{\prime}:\) Rep-Label \(c^{\prime}=\) Rep-Label \(l @[p]\)
        by (auto dest!: label-children)
    from \(p p^{\prime}\) eq have \(p=p^{\prime}\) by (simp add: last-lbl-def)
    with \(p p^{\prime}\) have Rep-Label \(c=\) Rep-Label \(c^{\prime}\) by simp
    thus \(c=c^{\prime}\) by (simp add: Rep-Label-inject)
qed
```

The number of children of any non-leaf label $l$ is the number of processes that do not appear in $l$.
lemma card-children:

```
    assumes }\neg(\mathrm{ is-leaf l)
    shows card (children l)}=N-(\mathrm{ length-lbl l)
proof -
    from assms
    have last-lbl '(children l) = UNIV - set-lbl l
        by (rule children-last-set)
    moreover
    have card (UNIV - set-lbl l) = card (UNIV ::Proc set) - card (set-lbl l)
    by (auto simp: card-Diff-subset-Int)
    moreover
    from last-lbl-inj-on-children
    have card (children l) = card (last-lbl'children l)
    by (rule sym[OF card-image])
    moreover
    note card-set-lbl[of l]
    ultimately
    show ?thesis by auto
qed
```

Suppose a non-root label $l^{\prime}$ of length $r+1$ ending in $q$, and suppose that $q$ is well heard by process $p$ in round $r$. Then the value with which $p$ decorates $l$ is the one that $q$ associates to the parent of $l$.
lemma sho-correct-vals:
assumes run: SHORun EIG-M rho HOs SHOs
and $l^{\prime}: l^{\prime} \in$ children $l$
and shop: last-lbl l' $\in S H O s$ (length-lbl l) $p \cap H O s$ (length-lbl l) $p$
(is ? $q \in S H O s($ ?len $l) p \cap-$ )
shows vals (rho (?len $l^{\prime}$ ) $p$ ) $l^{\prime}=$ vals (rho (?len $l$ ) ?q) $l$
proof -
let $? r=$ ?len $l$
from run obtain $\mu p$
where nxt: nextState EIG-M ?r p (rho ?r p) $\mu \mathrm{p}$ (rho (Suc ?r) p)
and mu: $\mu \mathrm{p} \in$ SHOmsgVectors EIG-M ?r p (rho ?r) (HOs ?r p) (SHOs ?r p)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with shop
have msl: $\mu \mathrm{p}$ ? $q=$ Some (vals (rho ?r ?q))
by (auto simp: EIG-SHOMachine-def EIG-sendMsg-def SHOmsgVectors-def)
from nxt length-lbl[ of l] children-length[OF l]
have extend-vals ?r $p$ (rho ?r p) $\mu$ (rho (Suc ?r) p)
by (auto simp: EIG-SHOMachine-def nextState-def EIG-nextState-def next-main-def next-end-def)
with msl $l^{\prime}$ show ?thesis
by (auto simp: extend-vals-def children-length children-butlast-lbl)
qed
A process fixes the value vals $l$ of a label at state length-lbl $l$, and then never modifies the value.
lemma keep-vals:
assumes run: SHORun EIG-M rho HOs SHOs

```
    shows vals (rho (length-lbl l + n) p) l= vals (rho (length-lbl l) p) l
        (is ?v n=?vl)
proof (induct n)
    show ?v 0 = ?vl by simp
next
    fix n
    assume ih: ?v n=?vl
    let ?r = length-lbl l + n
    from run obtain }\mu
        where nxt: nextState EIG-M ?r p (rho ?r p) \mup (rho (Suc ?r) p)
        by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
    with ih show ?v (Suc n) = ?vl
    by (auto simp: EIG-SHOMachine-def nextState-def EIG-nextState-def
                    next-main-def next-end-def extend-vals-def)
qed
```


### 10.6 Lynch's Lemmas and Theorems

If some process is safely heard by all processes at round $r$, then all processes agree on the value associated to labels of length $r+1$ ending in that process.
lemma lynch-6-15:
assumes run: SHORun EIG-M rho HOs SHOs
and $l^{\prime}: l^{\prime} \in$ children $l$
and skr: last-lbl $l^{\prime} \in S K r(H O s(l e n g t h-l b l ~ l))(S H O s ~(l e n g t h-l b l ~ l)) ~$
shows vals (rho (length-lbl l') p) $l^{\prime}=$ vals (rho (length-lbl l') q) l'
using assms unfolding SKr-def by (auto simp: sho-correct-vals)
Suppose that $l$ is a non-root label whose last element was well heard by all processes at round $r$, and that $l^{\prime}$ is a child of $l$ corresponding to process $q$ that is also well heard by all processes at round $r+1$. Then the values associated with $l$ and $l^{\prime}$ by any process $p$ are identical.

```
lemma lynch-6-16-a:
    assumes run: SHORun EIG-M rho HOs SHOs
        and l:l\in children t
        and skrl:last-lbl l \inSKr (HOs (length-lbl t)) (SHOs (length-lbl t))
    and l}\mp@subsup{l}{}{\prime}:\mp@subsup{l}{}{\prime}\in\mathrm{ children l
    and skrl':last-lbl l' \in SKr (HOs (length-lbl l)) (SHOs (length-lbl l))
    shows vals (rho (length-lbl l') p) l'= vals (rho (length-lbl l) p) l
    using assms by (auto simp: SKr-def sho-correct-vals)
```

For any non-leaf label $l$, more than half of its children end with a process that is well heard by everyone at round length-lbl l.

```
lemma lynch-6-16-c:
    assumes commR: EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))
                (is EIG-commPerRd (HOs ?r) -)
    and \(l: \neg(\) is-leaf \(l)\)
shows card \(\left\{l^{\prime} \in\right.\) children l. last-lbl \(l^{\prime} \in S K r(H O s\) ?r) (SHOs ?r \(\left.)\right\}\)
        \(>\) card (children l) div 2
```

```
    (is card ?lhs > -)
proof -
    let ?skr = SKr (HOs ?r) (SHOs ?r)
    have last-lbl'?lhs = ?skr - set-lbl l
    proof
        from children-last-set[OF l]
        show last-lbl'?lhs \subseteq?skr - set-lbl l
            by (auto simp: children-length)
    next
        {
        fix p
        assume p: p\in?skr p\not\in set-lbl l
        with children-last-set[OF l]
        have p\inlast-lbl`children l by auto
        with p have p\in last-lbl'?lhs
            by (auto simp: image-def children-length)
        }
        thus ?skr - set-lbl l\subseteq last-lbl`?lhs by auto
    qed
    moreover
    from last-lbl-inj-on-children[of l]
    have inj-on last-lbl ?lhs by (auto simp: inj-on-def)
    ultimately
    have card ?lhs = card (?skr - set-lbl l) by (auto dest: card-image)
    also have ... \geq(card ?skr) - (card (set-lbl l))
    by (simp add: diff-card-le-card-Diff)
    finally have card?lhs \geq(card ?skr) - ?r
        using card-set-lbl[of l] by simp
    moreover
    from commR have card ?skr > (N+f) div 2
        by (auto simp: EIG-commPerRd-def)
    with not-leaf-length[OF l] f
    have (card ?skr) - ?r > (N - ?r) div 2 by auto
    with card-children[OF l]
    have (card ?skr) - ?r > card (children l) div 2 by simp
    ultimately show ?thesis by simp
qed
```

If $l$ is a non-leaf label such that all of its children corresponding to well-heard processes at round length-lbl $l$ have a uniform newvals decoration at round $f+1$, then $l$ itself is decorated with that same value.
lemma newvals-skr-uniform:
assumes run: SHORun EIG-M rho HOs SHOs
and commR: EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))
(is EIG-commPerRd (HOs?r) -)
and notleaf: $\neg($ is-leaf $l)$
and unif: $\wedge l^{\prime} . \llbracket l^{\prime} \in$ children $l$;

$$
\text { last-lbl } l^{\prime} \in S K r(H O s(l e n g t h-l b l ~ l))(S H O s(\text { length-lbl l)) }
$$

$\rrbracket \Longrightarrow$ newvals (rho (Suc f) p) $l^{\prime}=v$
shows newvals (rho (Suc f) p) $l=v$
proof -
from unif
have card $\left\{l^{\prime} \in\right.$ children l. last-lbl $\left.l^{\prime} \in S K r(H O s ? r)(S H O s ? r)\right\}$ $\leq$ card $\left\{l^{\prime} \in\right.$ children $l$. newvals (rho (Suc f) p) $\left.l^{\prime}=v\right\}$
by (auto intro: card-mono)
with lynch-6-16-c[of HOs l SHOs, OF commR notleaf]
have maj: has-majority $v$ (newvals (rho (Suc f) p)) (children l)
by (simp add: has-majority-def)
from run have check-newvals (rho (Suc f) p)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq nextState-def EIG-nextState-def next-end-def)
with maj notleaf obtain $w$
where wmaj: has-majority $w$ (newvals (rho (Suc f) p)) (children l) and wupd: newvals (rho (Suc f) p) $l=w$
by (auto simp: check-newvals-def)
from maj wmaj have $w=v$
by (auto simp: has-majority-def elim: abs-majoritiesE')
with wupd show ?thesis by simp
qed
A node whose label $l$ ends with a process which is well heard at round length-lbl $l$ will have its newvals field set (at round $f+1$ ) to the "fixed-up" value given by vals.
lemma lynch-6-16-d:
assumes run: SHORun EIG-M rho HOs SHOs
and commR: $\forall r$. EIG-commPerRd (HOs r) (SHOs r)
and notroot: $l \in$ children $t$

(is - $\in \operatorname{SKr}(H O s(? l e n t))-)$
shows newvals (rho (Suc f) p) $l=$ fixupval (vals (rho (?len l) p) $l$ )
(is ?P l)
using notroot skr proof (induct Suc $f-($ ?len $l$ ) arbitrary: $l$ t)
fix $l t$
assume $0=S u c f-$ ?len $l$
with length-lbl[ of $l]$ have leaf: is-leaf $l$ by (simp add: is-leaf-def)
from run have check-newvals (rho (Suc f) p)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq
nextState-def EIG-nextState-def next-end-def)
with leaf show ?P $l$
by (auto simp: check-newvals-def is-leaf-def)
next
fix $k l t$
assume $i h: \bigwedge l^{\prime} t^{\prime}$.

```
\(\llbracket k=\) Suc \(f-\) length-lbl \(l^{\prime} ; l^{\prime} \in\) children \(t^{\prime} ;\)
last-lbl \(l^{\prime} \in \operatorname{SKr}\left(H O s\left(\right.\right.\) ?len \(\left.\left.t^{\prime}\right)\right)\left(S H O s\left(? l e n ~ t^{\prime}\right)\right) 】\)
\(\Longrightarrow\) ? \(P l^{\prime}\)
```

    and flk: Suc \(k=S u c f-\) ?len \(l\)
    and notroot: \(l \in\) children \(t\)
    and skr: last-lbl l \(\in \operatorname{SKr}(H O s(? l e n t))(S H O s(? l e n t))\)
    let \(? v=\) fixupval \((\) vals \((\) rho \((? l e n ~ l) p) ~ l)\)
    from flk have notlf: \(\neg\) (is-leaf l) by (simp add: is-leaf-def)
    \{
    fix \(l^{\prime}\)
    assume \(l^{\prime}: l^{\prime} \in\) children \(l\)
        and \(s k r^{\prime}: l a s t-l b l l^{\prime} \in S K r(H O s(? l e n ~ l))(S H O s(? l e n ~ l))\)
    from run notroot skr l' skr
    have vals (rho (?len \(l^{\prime}\) ) \(p\) ) \(l^{\prime}=\) vals (rho (?len l) \(p\) ) \(l\)
        by (rule lynch-6-16-a)
    moreover
    from \(f k l^{\prime}\) have \(k=S u c f-\) ?len \(l^{\prime}\) by (simp add: children-length)
    from this \(l^{\prime} s k r^{\prime}\) have ?P \(l^{\prime}\) by (rule ih)
    ultimately
    have newvals (rho (Suc f) p) \(l^{\prime}=\) ?v
        using notroot \(l^{\prime}\) by (simp add: children-length)
    \}
    with run commR notlf show ?P \(l\) by (auto intro: newvals-skr-uniform)
    qed

Following Lynch [12], we introduce some more useful concepts for reasoning about the data structure.

A label is common if all processes agree on the final value it is decorated with.
definition common where
common rho $l \equiv$
$\forall p$. newvals (rho (Suc f) $p$ ) $l=$ newvals $(r h o(S u c f) q) l$
The subtrees of a given label are all its possible extensions.

```
definition subtrees where
    subtrees \(h \equiv\{l . \exists t\). Rep-Label \(l=(\) Rep-Label \(h) @ t\}\)
lemma children-in-subtree:
    assumes \(l \in\) children \(h\)
    shows \(l \in\) subtrees \(h\)
    using label-children[OF assms] by (auto simp: subtrees-def)
lemma subtrees-refl [iff]: \(l \in\) subtrees \(l\)
    by (auto simp: subtrees-def)
```

```
lemma subtrees-root [iff]:l s subtrees root-node
    by (auto simp: subtrees-def)
lemma subtrees-trans:
    assumes l"'}\in\mathrm{ subtrees l' and l}\mp@subsup{l}{}{\prime}\in\mathrm{ subtrees l
    shows ll' \in subtrees l
    using assms by (auto simp: subtrees-def)
lemma subtrees-antisym:
    assumes }l\in\mathrm{ subtrees l' and l'}\mp@subsup{l}{}{\prime}\in\mathrm{ subtrees l
    shows l' = l
    using assms by (auto simp: subtrees-def Rep-Label-inject)
lemma subtrees-tree:
    assumes l':l subtrees l' and l'\prime:l\in subtrees l"
    shows l' }\mp@subsup{l}{}{\prime}\mathrm{ subtrees l}\mp@subsup{l}{}{\prime\prime}\vee\mp@subsup{l}{}{\prime\prime}\in\mathrm{ subtrees }\mp@subsup{l}{}{\prime
using assms proof (auto simp: subtrees-def append-eq-append-conv2)
    fix xs
    assume Rep-Label l'\prime @ xs=Rep-Label l'
    hence Rep-Label l' = Rep-Label l'\prime @ xs by (rule sym)
    thus \existsys.Rep-Label l' = Rep-Label l'\ @ ys ..
qed
lemma subtrees-cases:
    assumes l':}\mp@subsup{l}{}{\prime}\in\mathrm{ subtrees }
        and self: l' =l\LongrightarrowP
        and child: \c.\llbracketc\in children l; l' }\in\mathrm{ subtrees c \ }\Longrightarrow
    shows P
proof -
    from l' obtain t where t:Rep-Label l'=(Rep-Label l) @ t
    by (auto simp: subtrees-def)
    have l'}=l\vee(\existsc\in\mathrm{ children l. l' }\in\mathrm{ subtrees c)
    proof (cases t)
        assume t= []
        with t show ?thesis by (simp add: Rep-Label-inject)
    next
    fix p t'
    assume cons: t=p# t'
    from Rep-Label[of l`] have length (Rep-Label l @ t) \leqSuc f
        by (simp add: Label-def)
    with cons have notleaf: }\neg(\mathrm{ is-leaf l)
            by (auto simp: is-leaf-def length-lbl-def)
    let ?c = Abs-Label (Rep-Label l @ [p])
    from t cons Rep-Label[of l` have p:p\not\in set-lbl l
            by (auto simp: Label-def set-lbl-def)
        with notleaf have c: ?c \in children l
            by (auto simp: children-def)
    moreover
```

```
    from notleaf p have Rep-Label l @ [p]\in Label
        by (simp add: Rep-Label-append)
    hence Rep-Label ?c = (Rep-Label l @ [p])
        by (simp add: Abs-Label-inverse)
    with const have l' }\mp@subsup{l}{}{\prime}\mathrm{ subtrees?c
        by (auto simp: subtrees-def)
    ultimately show ?thesis by blast
    qed
    thus ?thesis by (auto elim!: self child)
qed
lemma subtrees-leaf:
    assumes l: is-leaf l and l':}\mp@subsup{l}{}{\prime}\in\mathrm{ subtrees l
    shows l' = l
using l' proof (rule subtrees-cases)
    fix }
    assume c\in children l - impossible
    with l show ?thesis by (simp add: children-def)
qed
lemma children-subtrees-equal:
    assumes c:c\in children l and c': c'}\in\mathrm{ children l
        and sub: c' }\in\mathrm{ subtrees c
    shows }\mp@subsup{c}{}{\prime}=
proof -
    from assms have Rep-Label c' = Rep-Label c
    by (auto simp: subtrees-def dest!: label-children)
    thus ?thesis by (simp add: Rep-Label-inject)
qed
```

A set $C$ of labels is a subcovering w.r.t. label $l$ if for all leaf subtrees $s$ of $l$ there exists some label $h \in C$ such that $s$ is a subtree of $h$ and $h$ is a subtree of $l$.
definition subcovering where
subcovering $C l \equiv$
$\forall s \in$ subtrees $l$. is-leaf $s \longrightarrow(\exists h \in C . h \in$ subtrees $l \wedge s \in$ subtrees $h)$
A covering is a subcovering w.r.t. the root node.
abbreviation covering where
covering $C \equiv$ subcovering $C$ root-node
The set of labels whose last element is well heard by all processes throughout the execution forms a covering, and all these labels are common.

```
lemma lynch-6-18-a:
    assumes SHORun EIG-M rho HOs SHOs
    and \forallr.EIG-commPerRd (HOs r) (SHOs r)
    and l\in children t
    and last-lbl l GSKr (HOs (length-lbl t)) (SHOs (length-lbl t))
```

```
    shows common rho l
    using assms
    by (auto simp: common-def lynch-6-16-d lynch-6-15
        intro: arg-cong[where f=fixupval])
lemma lynch-6-18-b:
    assumes run: SHORun EIG-M rho HOs SHOs
        and commG: EIG-commGlobal HOs SHOs
        and commR: \forallr. EIG-commPerRd (HOs r) (SHOs r)
    shows covering {l. \existst.l\in children t}\wedge last-lbl l\in(SK HOs SHOs)
proof (clarsimp simp: subcovering-def)
    fix l
    assume is-leaf l
    with card-set-lbl[of l] have card (set-lbl l) = Suc f
    by (simp add: is-leaf-def)
    with commG have N<card (SK HOs SHOs) + card (set-lbl l)
    by (simp add: EIG-commGlobal-def)
    hence }\existsq\in\mathrm{ set-lbl l. q SK HOs SHOs
    by (auto dest: majorities-intersect)
    then obtain l1 q l2 where
    l:Rep-Label l=(l1 @ [q]) @ l2 and q:q\inSK HOs SHOs
    unfolding set-lbl-def by (auto intro: split-list-propE)
    let ?h = Abs-Label (l1 @ [q])
    from Rep-Label[of l] l have l1 @ [q] \in Label by (simp add:Label-def)
    hence reph:Rep-Label ?h = l1 @ [q] by (rule Abs-Label-inverse)
    hence length-lbl ?h \not=0 by (simp add: length-lbl-def)
    hence ?h}\not=\mathrm{ root-node by auto
    then obtain t where t:?h c children t
        by (auto simp: root-iff-no-child)
    moreover
    from reph q have last-lbl ?h \inSK HOs SHOs by (simp add: last-lbl-def)
    moreover
    from reph l have l subtrees ?h by (simp add: subtrees-def)
    ultimately
    show \existsh. (\existst. h\in children t) ^last-lbl h\inSKHOs SHOs ^l\in subtrees h
    by blast
qed
```

If $C$ covers the subtree rooted at label $l$ and if $l \notin C$ then $C$ also covers subtrees rooted at $l$ 's children.

```
lemma lynch-6-19-a:
    assumes cov: subcovering C l
        and l:l\not\inC
        and e:}e\in\mathrm{ children l
    shows subcovering C e
proof (clarsimp simp: subcovering-def)
    fix }
    assume s:s}\in\mathrm{ subtrees e and leaf:is-leaf s
```

```
    from \(s\) children-in-subtree \([O F e]\) have \(s \in\) subtrees \(l\)
    by (rule subtrees-trans)
    with leaf cov obtain \(h\) where \(h: h \in C h \in\) subtrees \(l s \in\) subtrees \(h\)
    by (auto simp: subcovering-def)
    with \(l\) obtain \(e^{\prime}\) where \(e^{\prime}: e^{\prime} \in\) children \(l h \in\) subtrees \(e^{\prime}\)
    by (auto elim: subtrees-cases)
    from \(\langle s \in\) subtrees \(h\rangle\left\langle h \in\right.\) subtrees \(\left.e^{\prime}\right\rangle\) have \(s \in\) subtrees \(e^{\prime}\)
    by (rule subtrees-trans)
    with \(s\) have \(e \in\) subtrees \(e^{\prime} \vee e^{\prime} \in\) subtrees \(e\)
    by (rule subtrees-tree)
    with \(e e^{\prime}\) have \(e^{\prime}=e\)
    by (auto dest: children-subtrees-equal)
    with \(e^{\prime} h\) show \(\exists h \in C . h \in\) subtrees \(e \wedge s \in\) subtrees \(h\) by blast
qed
```

If there is a subcovering $C$ for a label $l$ such that all labels in $C$ are common, then $l$ itself is common as well.

```
lemma lynch-6-19-b:
    assumes run: SHORun EIG-M rho HOs SHOs
    and cov: subcovering Cl
    and com: \(\forall l^{\prime} \in C\). common rho \(l^{\prime}\)
    shows common rho \(l\)
using cov proof (induct Suc \(f\) - length-lbl l arbitrary: \(l\) )
    fix \(l\)
    assume 0: \(0=\) Suc \(f-\) length-lbl \(l\)
        and \(C\) : subcovering \(C l\)
    from 0 length-lbl[ of \(l]\) have is-leaf \(l\)
        by (simp add: is-leaf-def)
    with \(C\) obtain \(h\) where \(h: h \in C h \in\) subtrees \(l l \in\) subtrees \(h\)
    by (auto simp: subcovering-def)
    hence \(l \in C\) by (auto dest: subtrees-antisym)
    with com show common rho \(l\)..
next
    fix \(k l\)
    assume \(k\) : Suc \(k=\) Suc \(f-\) length-lbl \(l\)
        and \(C\) : subcovering \(C l\)
        and \(i h: \bigwedge l^{\prime} . \llbracket k=S u c f-\) length-lbl \(l^{\prime} ;\) subcovering \(C l \rrbracket \Longrightarrow\) common rho \(l^{\prime}\)
    show common rho \(l\)
    proof (cases \(l \in C\) )
        case True
        with com show ?thesis ..
    next
        case False
    with \(C\) have \(\forall e \in\) children l. subcovering \(C e\)
        by (blast intro: lynch-6-19-a)
    moreover
    from \(k\) have \(\forall e \in\) children l. \(k=\) Suc \(f-\) length-lbl \(e\)
        by (auto simp: children-length)
    ultimately
```

```
    have com-ch: }\foralle\in\mathrm{ children l. common rho e
    by (blast intro: ih)
    show ?thesis
    proof (clarsimp simp: common-def)
    fix pq
    from }k\mathrm{ have notleaf: }\neg\mathrm{ (is-leaf l) by (simp add: is-leaf-def)
    let ?r =Suc f
    from com-ch
    have }\foralle\in\mathrm{ children l. newvals (rho ?r p) e= newvals (rho ?r q) e
        by (auto simp: common-def)
    hence }\forallw.{e\in\mathrm{ children l. newvals (rho ?r p) e=w}
                ={e\in children l. newvals (rho ?r q) e=w}
        by auto
    moreover
    from run
    have check-newvals (rho ?r p) check-newvals (rho ?r q)
    by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq nextState-def
                    EIG-nextState-def next-end-def)
    with notleaf have
        ( }\exists\mathrm{ w. has-majority w (newvals (rho ?r p)) (children l)
                newvals (rho ?r p) l=w)
        \vee \neg ( \exists w . h a s - m a j o r i t y ~ w ~ ( n e w v a l s ~ ( r h o ~ ? r ~ p ) ) ~ ( c h i l d r e n ~ l ) ) ~
                \wedge ~ n e w v a l s ~ ( r h o ~ ? r ~ p ) ~ l = ~ u n d e f i n e d ~
        ( }\exists\mathrm{ w. has-majority w (newvals (rho ?r q)) (children l)
            newvals (rho ?r q) l=w)
        \vee \neg ( \exists \text { w. has-majority w (newvals (rho ?r q)) (children l))}
                newvals (rho ?r q) l= undefined
        by (auto simp: check-newvals-def)
    ultimately show newvals (rho ?r p) l= newvals (rho ?r q) l
        by (auto simp: has-majority-def elim: abs-majoritiesE')
    qed
qed
qed
```

The root of the tree is a common node.
lemma lynch-6-20:
assumes run: SHORun EIG-M rho HOs SHOs
and commG: EIG-commGlobal HOs SHOs
and commR: $\forall r$. EIG-commPerRd (HOs r) (SHOs r)
shows common rho root-node
using run lynch-6-18-b[OF assms]
proof (rule lynch-6-19-b, clarify)
fix $l t$
assume $l \in$ children $t$ last-lbl $l \in S K H O s$ SHOs
thus common rho $l$ by (auto simp: SK-def elim: lynch-6-18-a[OF run commR])
qed
A decision is taken only at state $f+1$ and then stays stable.

```
lemma decide:
    assumes run: SHORun EIG-M rho HOs SHOs
    shows decide (rho r p)=
        (if r < Suc f then None
                else Some (newvals (rho (Suc f) p) root-node))
        (is ?P r)
proof (induct r)
    from run show ?P 0
        by (auto simp: EIG-SHOMachine-def SHORun-eq HOinitConfig-eq
                                initState-def EIG-initState-def)
next
    fix r
    assume ih: ?P r
    from run obtain }\mu
        where EIG-nextState r p (rho r p) \mup (rho (Suc r) p)
        by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq
                                nextState-def)
    thus ?P (Suc r)
    proof (auto simp: EIG-nextState-def next-main-def next-end-def)
        assume }\neg(r<f)r\not=
        with ih
        show decide (rho r p) = Some (newvals (rho (Suc f) p) root-node)
            by simp
    qed
qed
```


### 10.7 Proof of Agreement, Validity, and Termination

The Agreement property is an immediate consequence of lemma lynch-6-20.

```
theorem Agreement:
    assumes run: SHORun EIG-M rho HOs SHOs
    and commG: EIG-commGlobal HOs SHOs
    and commR: \forallr. EIG-commPerRd (HOs r) (SHOs r)
    and p: decide (rho m p) = Some v
    and q: decide (rho n q) = Some w
    shows v=w
    using p q lynch-6-20[OF run commG commR]
    by (auto simp: decide[OF run] common-def)
```

We now show the Validity property: if all processes initially propose the same value $v$, then no other value may be decided.

By lemma sho-correct-vals, value $v$ must propagate to all children of the root that are well heard at round 0 , and lemma lynch-6-16-d implies that $v$ is the value assigned to all these children by newvals. Finally, lemma newvals-skr-uniform lets us conclude.
theorem Validity:
assumes run: SHORun EIG-M rho HOs SHOs
and commR: $\forall r$. EIG-commPerRd (HOs r) (SHOs r)

```
        and initv: }\forallq.\mathrm{ the (vals (rho 0 q) root-node) =v
        and dp: decide (rho r p) = Some w
    shows v=w
proof -
    have v:\forallq. vals (rho 0 q) root-node = Some v
    proof
    fix q
    from run have vals (rho 0 q) root-node }\not=\mathrm{ None
            by (auto simp: EIG-SHOMachine-def SHORun-eq HOinitConfig-eq
                initState-def EIG-initState-def)
    then obtain w where w: vals (rho 0 q) root-node = Some w
        by auto
    from initv have the (vals (rho 0 q) root-node) = v ..
    with w}\mathrm{ show vals (rho 0 q) root-node = Some v by simp
qed
let ?len = length-lbl
let ?r = Suc f
{
    fix l}\mp@subsup{l}{}{\prime
    assume l':}\mp@subsup{l}{}{\prime}\in\mathrm{ children root-node
        and skr: last-lbl l' }\mp@subsup{l}{}{\prime}SKr(HOs 0)(SHOs 0)
    with run v have vals (rho (?len l') p) l' =Some v
        by (auto dest: sho-correct-vals simp: SKr-def)
    moreover
    from run commR l' skr
    have newvals (rho ?r p) l'= fixupval (vals (rho (?len l') p) l')
        by (auto intro: lynch-6-16-d)
    ultimately
    have newvals (rho ?r p) l' =v by simp
}
with run commR root-node-not-leaf
have newvals (rho ?r p) root-node =v
    by (auto intro: newvals-skr-uniform)
with dp show ?thesis by (simp add: decide[OF run])
qed
Termination is trivial for \(E I G B y z_{f}\).
theorem Termination:
    assumes SHORun EIG-M rho HOs SHOs
    shows }\existsrv\mathrm{ . decide (rho r p) = Some v
    using assms by (auto simp: decide)
```


### 10.8 EIGByz $z_{f}$ Solves Weak Consensus

Summing up, all (coarse-grained) runs of EIGByzf for HO and SHO collections that satisfy the communication predicate satisfy the Weak Consensus property.

```
theorem eig-weak-consensus:
    assumes run: SHORun EIG-M rho HOs SHOs
    and commR: }\forallr\mathrm{ . EIG-commPerRd (HOs r)(SHOs r)
    and commG: EIG-commGlobal HOs SHOs
shows weak-consensus ( }\lambda\mathrm{ p. the (vals (rho 0 p) root-node)) decide rho
unfolding weak-consensus-def
using Validity[OF run commR]
    Agreement[OF run commG commR]
    Termination[OF run]
    by auto
```

By the reduction theorem, the correctness of the algorithm carries over to the fine-grained model of runs.

```
theorem eig-weak-consensus-fg:
    assumes run: fg-run EIG-M rho HOs SHOs (\lambdar q. undefined)
        and commR: }\forall\mathrm{ r. EIG-commPerRd (HOs r) (SHOs r)
        and commG: EIG-commGlobal HOs SHOs
    shows weak-consensus ( }\lambdap\mathrm{ p. the (vals (state (rho 0) p) root-node))
                decide (state o rho)
        (is weak-consensus ?inits - -)
proof (rule local-property-reduction[OF run weak-consensus-is-local])
    fix crun
    assume crun: CSHORun EIG-M crun HOs SHOs (\lambdar q. undefined)
        and init: crun 0 = state (rho 0)
    from crun have SHORun EIG-M crun HOs SHOs by (unfold SHORun-def)
    from this commR commG
    have weak-consensus ( }\lambdap\mathrm{ . the (vals (crun 0 p) root-node)) decide crun
        by (rule eig-weak-consensus)
    with init show weak-consensus?:inits decide crun
        by (simp add:o-def)
qed
```

end

## 11 Conclusion

In this contribution we have formalized the Heard-Of model in the proof assistant Isabelle/HOL. We have established a formal framework, in which fault-tolerant distributed algorithms can be represented, and that caters for different variants (benign or malicious faults, coordinated and uncoordinated algorithms). We have formally proved a reduction theorem that re-
lates fine-grained (asynchronous) interleaving executions and coarse-grained executions, in which an entire round constitutes the unit of atomicity. As a corollary, many correctness properties, including Consensus, can be transferred from the coarse-grained to the fine-grained representation.
We have applied this framework to give formal proofs in Isabelle/HOL for six different Consensus algorithms known from the literature. Thanks to the reduction theorem, it is enough to verify the algorithms over coarse-grained runs, and this keeps the effort manageable. For example, our LastVoting algorithm is similar to the DiskPaxos algorithm verified in [10], but our proof here is an order of magnitude shorter, although we prove safety and liveness properties, whereas only safety was considered in [10].
We also emphasize that the uniform characterization of fault assumptions via communication predicates in the HO model lets us consider the effects of transient failures, contrary to standard models that consider only permanent failures. For example, our correctness proof for the EIGByz $z_{f}$ algorithm establishes a stronger result than that claimed by the designers of the algorithm. The uniform presentation also paves the way towards comparing assumptions of different algorithms.

The encoding of the HO model as Isabelle/HOL theories is quite straightforward, and we find our Isar proofs quite readable, although they necessarily contain the full details that are often glossed over in textbook presentations. We believe that our framework allows algorithm designers to study different fault-tolerant distributed algorithms, their assumptions, and their proofs, in a clear, rigorous and uniform way.

## References

[1] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. Inf. Comput., 97(2):205-233, 1992.
[2] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In R. L. Probert, N. A. Lynch, and N. Santoro, editors, Proc. 2nd Symp. Principles of Distributed Computing (PODC 1983), pages 27-30, Montreal, Canada, 1983. ACM.
[3] M. Biely, J. Widder, B. Charron-Bost, A. Gaillard, M. Hutle, and A. Schiper. Tolerating corrupted communication. In Proc. 26th Annual ACM Symposium on Principles of Distributed Computing, PODC '07, pages 244-253, New York, NY, USA, 2007. ACM.
[4] M. Chaouch-Saad, B. Charron-Bost, and S. Merz. A reduction theorem for the verification of round-based distributed algorithms. In O. Bournez and I. Potapov, editors, Reachability Problems, volume 5797
of Lecture Notes in Computer Science, pages 93-106, Palaiseau, France, 2009. Springer.
[5] B. Charron-Bost, H. Debrat, and S. Merz. Formal verification of consensus algorithms tolerating malicious faults. In X. Défago, F. Petit, and V. Villain, editors, 13th Intl. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS 2011), volume 6976 of LNCS, pages 120-134, Grenoble, France, 2011. Springer.
[6] B. Charron-Bost and S. Merz. Formal verification of a Consensus algorithm in the Heard-Of model. Intl. J. Software and Informatics, 3(2-3):273-304, 2009.
[7] B. Charron-Bost and A. Schiper. The Heard-Of model: computing in distributed systems with benign faults. Distributed Computing, 22(1):49-71, 2009.
[8] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288-323, Apr. 1988.
[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374-382, Apr. 1985.
[10] M. Jaskelioff and S. Merz. Proving the correctness of DiskPaxos. Archive of Formal Proofs, 2005.
[11] L. Lamport. The part-time parliament. ACM Trans. Comput. Syst., 16(2):133-169, 1998.
[12] N. Lynch. Distributed Algorithms. Morgan Kaufmann Publishers, San Mateo, CA, 1996.


[^0]:    *Bernadette Charron-Bost introduced us to the Heard-Of model and accompanied this work by suggesting algorithms to study, providing or simplifying hand proofs, and giving most valuable feedback on our formalizations. Mouna Chaouch-Saad contributed an initial draft formalization of the reduction theorem.

[^1]:    definition $f g$-start-round where
    fg-start-round rho pr$\equiv \operatorname{LEAST}$ ( $n$ ::nat). round (rho n) $p=r$

