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.

*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.
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 UniformVoting Consensus Algorithm 44
   6.1 Model of the Algorithm ............................ 44
   6.2 Communication Predicate for UniformVoting ........ 47
   6.3 The UniformVoting Heard-Of Machine ............. 47
   6.4 Preliminary Lemmas ................................ 47
   6.5 Proof of Irrevocability, Agreement and Integrity .. 50
   6.6 Proof of Termination .............................. 57
   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
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>Preliminary Lemmas</td>
<td>66</td>
</tr>
<tr>
<td>7.5</td>
<td>Boundedness and Monotonicity of Timestamps</td>
<td>71</td>
</tr>
<tr>
<td>7.6</td>
<td>Obvious Facts About the Algorithm</td>
<td>72</td>
</tr>
<tr>
<td>7.7</td>
<td>Proof of Integrity</td>
<td>80</td>
</tr>
<tr>
<td>7.8</td>
<td>Proof of Agreement and Irrevocability</td>
<td>84</td>
</tr>
<tr>
<td>7.9</td>
<td>Proof of Termination</td>
<td>94</td>
</tr>
<tr>
<td>7.10</td>
<td>LastVoting Solves Consensus</td>
<td>96</td>
</tr>
<tr>
<td>8</td>
<td>Verification of the $UT,E,\alpha$ Consensus Algorithm</td>
<td>97</td>
</tr>
<tr>
<td>8.1</td>
<td>Model of the Algorithm</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Communication Predicate for $UT,E,\alpha$</td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>The $UT,E,\alpha$ Heard-Of Machine</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>Preliminary Lemmas</td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Proof of Agreement and Validity</td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>Proof of Termination</td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>$UT,E,\alpha$ Solves Weak Consensus</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Verification of the $AT,E,\alpha$ Consensus algorithm</td>
<td>123</td>
</tr>
<tr>
<td>9.1</td>
<td>Model of the Algorithm</td>
<td></td>
</tr>
<tr>
<td>9.2</td>
<td>Communication Predicate for $AT,E,\alpha$</td>
<td></td>
</tr>
<tr>
<td>9.3</td>
<td>The $AT,E,\alpha$ Heard-Of Machine</td>
<td></td>
</tr>
<tr>
<td>9.4</td>
<td>Preliminary Lemmas</td>
<td></td>
</tr>
<tr>
<td>9.5</td>
<td>Proof of Validity</td>
<td></td>
</tr>
<tr>
<td>9.6</td>
<td>Proof of Agreement</td>
<td></td>
</tr>
<tr>
<td>9.7</td>
<td>Proof of Termination</td>
<td></td>
</tr>
<tr>
<td>9.8</td>
<td>$AT,E,\alpha$ Solves Weak Consensus</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Verification of the $EIGByz_f$ Consensus Algorithm</td>
<td>143</td>
</tr>
<tr>
<td>10.1</td>
<td>Tree Data Structure</td>
<td></td>
</tr>
<tr>
<td>10.2</td>
<td>Model of the Algorithm</td>
<td></td>
</tr>
<tr>
<td>10.3</td>
<td>Communication Predicate for $EIGByz_f$</td>
<td></td>
</tr>
<tr>
<td>10.4</td>
<td>The $EIGByz_f$ Heard-Of Machine</td>
<td></td>
</tr>
<tr>
<td>10.5</td>
<td>Preliminary Lemmas</td>
<td></td>
</tr>
<tr>
<td>10.6</td>
<td>Lynch’s Lemmas and Theorems</td>
<td></td>
</tr>
<tr>
<td>10.7</td>
<td>Proof of Agreement, Validity, and Termination</td>
<td></td>
</tr>
<tr>
<td>10.8</td>
<td>$EIGByz_f$ Solves Weak Consensus</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Conclusion</td>
<td>164</td>
</tr>
</tbody>
</table>
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, UniformVoting, and LastVoting, tolerate benign failures. The three remaining algorithms, $U_{T,E,\alpha}$, $A_{T,E,\alpha}$, and $EIGByz_f$, are designed to tolerate value failures, and solve a weaker variant of the Consensus problem.


theory HOModel
imports Main
begin

declare split-if-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 $\text{nat} \Rightarrow 'proc \Rightarrow 'pst$ where type variables $'proc$ and $'pst$ represent types of processes and process states, respectively. The Consensus property is expressed with respect to a collection $\text{vals}$ of initially proposed values (one per process) and an observer function $\text{dec}:'pst \Rightarrow \text{val option}$ 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.

```plaintext
type-synonym
  ('proc,'pst) run = nat ⇒ 'proc ⇒ 'pst

definition
  consensus :: ('proc ⇒ 'val) ⇒ ('pst ⇒ 'val option) ⇒ ('proc,'pst) run ⇒ bool
where
  consensus vals dec rho ≡
  (forall n p v. dec (rho n p) = Some v → v ∈ range vals)
  ∧ (forall m n p q w. dec (rho m p) = Some v ∧ dec (rho m q) = Some w
      → v = w)
  ∧ (forall p. ∃ n. dec (rho n p) ≠ 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 \( v \).

```plaintext
definition weak-consensus where
  weak-consensus vals dec rho ≡
  (forall v. (∀ p. vals p = v) → (∀ n p w. dec (rho n p) = Some w → w = v))
  ∧ (forall m n p q w. dec (rho m p) = Some v ∧ dec (rho m q) = Some w
      → v = w)
  ∧ (∀ p. ∃ n. dec (rho n p) ≠ 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., \( \text{card} \ (\text{range} \ vals) \leq 2 \)).

**Lemma** binary-weak-consensus-then-consensus:
  assumes bc: weak-consensus (vals::'proc ⇒ bool) dec rho
  shows consensus vals dec rho
  proof
    { — Show the Integrity property, the other conjuncts are the same.
      fix n p v
```
assume \(\text{dec} : \text{dec} (\rho n p) = \text{Some} \ v\)

have \(v \in \text{range} \ \text{vals}\)

proof (cases \(\exists \ w. \ \forall \ p. \ \text{vals} \ p = w\))

\(\text{case} \ True\)

then obtain \(w \) where \(w : \forall \ p. \ \text{vals} \ p = w \ ..\)

with \(\text{be} \ \text{dec} (\rho n p) \in \{\text{Some} \ w, \ \text{None}\} \) by (auto simp; weak-consensus-def)

\(\text{with} \ \text{dec} \ w \ \text{show} \ \text{thesis} \) by (auto simp; image-def)

next

\(\text{case} \ False\)

— In this case both possible values occur in \(\text{vals}\), and the result is trivial.

thus \(\text{thesis} \) by (auto simp; image-def)

qed

} note integrity = this

from be show \(\text{thesis}\)

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” \(HO(p, r)\) 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 \(HO(p, r)\) 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 \(HO(p, r)\) 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 \(SHO(p, r) \subseteq HO(p, r)\) 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 \(HO(p, r) \setminus SHO(p, r)\) 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 $HO(p, r)$ and $SHO(p, r)$. 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 $HO(p, r)$, the definitions assume an external coordinator assignment such that $coord(p, r)$ 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 $\text{'proc}$ of processes,
- a type $\text{'pst}$ of local process states,
- a type $\text{'msg}$ 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 $crd$ 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 $st$, and
- a predicate $CnextState$ where $CnextState r p st msgs crd st'$ characterizes the successor states $st'$ of process $p$ at round $r$, given current state $st$, the vector $msgs :: \text{'pst} \Rightarrow \text{'msg option}$ of messages that $p$ received at round $r$ ($msgs q = \text{None}$ indicates that no message has been received from process $q$), and process $crd$ 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
```

8
For non-coordinated HO algorithms, the coordinator argument of functions \texttt{CinitState} and \texttt{CnextState} is irrelevant, and we define utility functions that omit that argument.

\textbf{definition} \texttt{isNCAlgorithm} \textbf{where}
\[
\texttt{isNCAlgorithm alg} \equiv \ \forall \ p \ st \ crd \ crd'. \ \texttt{CinitState alg p st crd} = \texttt{CinitState alg p st crd'} \land \forall r p st \ msgs \ crd \ crd' \ st'. \ \texttt{CnextState alg r p st msgs crd st'} = \texttt{CnextState alg r p st msgs crd st'}
\]

\textbf{definition} \texttt{initState} \textbf{where}
\[
\texttt{initState alg p st} \equiv \texttt{CinitState alg p st undefined}
\]

\textbf{definition} \texttt{nextState} \textbf{where}
\[
\texttt{nextState alg r p st msgs st'} \equiv \texttt{CnextState alg r p st msgs undefined st'}
\]

A \textit{heard-of assignment} associates a set of processes with each process. The following type is used to represent the collections \texttt{HO}(p, r) and \texttt{SHO}(p, r) for fixed round \( r \). Similarly, a \textit{coordinator assignment} associates a process (its coordinator) to each process.

\textbf{type-synonym}
\[
\texttt{proc HO} = \texttt{proc} \Rightarrow \texttt{proc set}
\]

\textbf{type-synonym}
\[
\texttt{proc coord} = \texttt{proc} \Rightarrow \texttt{proc}
\]

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 \texttt{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 \texttt{CSHOinitConfig} describes the possible initial configurations for algorithm \( A \) (remember that a configuration is a function that assigns local states to every process).

\textbf{definition} \texttt{CSHOinitConfig} \textbf{where}
\[
\texttt{CSHOinitConfig A} \ \texttt{cfg (coord::'proc coord)} \equiv \forall p. \ \texttt{CinitState A p (cfg p) (coord p)}
\]
Given the current configuration $cfg$ and the HO and SHO sets $HOp$ and $SHOp$ for process $p$ at round $r$, the function $SHOmsgVectors$ computes the set of possible vectors of messages that process $p$ may receive. For processes $q \notin HOp$, $p$ receives no message (represented as value None). For processes $q \in SHOp$, $p$ receives the message that $q$ computed according to the $sendMsg$ function of the algorithm. For the remaining processes $q \in HOp - SHOp$, $p$ may receive some arbitrary value.

**Definition** $SHOmsgVectors$ where

$$SHOmsgVectors A r p cfg HOp SHOp \equiv \{ \mu. (\forall q. q \in HOp \leftrightarrow \mu q \neq \text{None}) \land (\forall q. q \in SHOp \cap HOp \rightarrow \mu q = \text{Some} (sendMsg A r q p (cfg q)))\}$$

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

$$CSHOnextConfig A r cfg HO SHO coord cfg' \equiv \forall p. \exists \mu \in SHOmsgVectors A r p cfg (HO p) (SHO p). CnextState A r p (cfg p) \mu (coord p) (cfg' p)$$

**Definition** $CSHORun$ where

$$CSHORun A rho HOs SHOs coords \equiv CHOinitConfig A (rho 0) (coords 0) \land (\forall r. CSHOnextConfig A r (rho r) (HOs r) (SHOs r) (coords (Suc r)) (rho (Suc r)))$$

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

$$HOinitConfig A cfg \equiv CHOinitConfig A cfg (\lambda q. \text{undefined})$$

**Lemma** $HOinitConfig-eq$:

$$HOinitConfig A cfg = (\forall p. \text{initState} A p (cfg p))$$

by (auto simp: HOinitConfig-def CHOinitConfig-def initState-def)

**Definition** $SHOnextConfig$ where

$$SHOnextConfig A r cfg HO SHO cfg' \equiv CSHOnextConfig A r cfg HO SHO (\lambda q. \text{undefined}) cfg'$$

**Lemma** $SHOnextConfig-eq$:

$$SHOnextConfig A r cfg HO SHO cfg' = (\forall p. \exists \mu \in SHOmsgVectors A r p cfg (HO p) (SHO p). nextState A r p (cfg p) \mu (cfg' p))$$

by (auto simp: SHOnextConfig-def CSHOnextConfig-def SHOmsgVectors-def nextState-def)
**definition** \textit{SHORun} where
\[
\text{SHORun} \ A \ \rho \ \text{HOs} \ \text{SHOs} \equiv
\hspace{1cm}
\text{CSHORun} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ (\lambda r. \text{undefined})
\]

**lemma** \textit{SHORun-eq}:
\[
\text{SHORun} \ A \ \rho \ \text{HOs} \ \text{SHOs} =
\hspace{1cm}
(\text{HOinitConfig} \ A \ (\rho \ 0))
\land (\forall r. \ \text{SHOnextConfig} \ A \ r \ \rho \ r \ (\text{HOs} \ r) \ (\text{SHOs} \ r) \ (\rho \ (\text{Suc} \ r)))
\]
by (auto simp: \text{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** \textit{HOrcvdMsgs} where
\[
\text{HOrcvdMsgs} \ A \ r \ p \ \text{HO} \ \text{cfg} \equiv
\lambda q. \begin{cases}
\text{if } q \in \text{HO} \text{ then } \text{Some} & (\text{sendMsg} \ A \ r \ q \ p \ (\text{cfg} \ q)) \\
\text{else } \text{None}
\end{cases}
\]

**lemma** \textit{SHOmsgVectors-HO}:
\[
\text{SHOmsgVectors} \ A \ r \ p \ \text{cfg} \ \text{HO} \ \text{HO} = \{\text{HOrcvdMsgs} \ A \ r \ p \ \text{HO} \ \text{cfg}\}
\]

**unfolding** \textit{SHOmsgVectors-def} \text{HOrcvdMsgs-def} by auto

With coordinators

**definition** \textit{CHOnextConfig} where
\[
\text{CHOnextConfig} \ A \ r \ \text{cfg} \ \text{HO} \ \text{coord} \ \text{cfg}' \equiv
\hspace{1cm}
\text{CSHOnextConfig} \ A \ r \ \text{cfg} \ \text{HO} \ \text{coord} \ \text{cfg}'
\]

**lemma** \textit{CHOnextConfig-eq}:
\[
\text{CHOnextConfig} \ A \ r \ \text{cfg} \ \text{HO} \ \text{coord} \ \text{cfg}' =
\hspace{1cm}
(\forall p. \ \text{CnextState} \ A \ r \ p \ (\text{cfg} \ p) \ (\text{HOrcvdMsgs} \ A \ r \ p \ (\text{HO} \ p) \ \text{cfg})
\hspace{1cm}
(\text{coord} \ p \ (\text{cfg}' \ p)))
\]
by (auto simp: \text{CHOnextConfig-def} \ CSHOnextConfig-def \ SHOmsgVectors-HO)

**definition** \textit{CHORun} where
\[
\text{CHORun} \ A \ \rho \ \text{HOs} \ \text{coords} \equiv \ \text{CSHORun} \ A \ \rho \ \text{HOs} \ \text{HOs} \ \text{coords}
\]

**lemma** \textit{CHORun-eq}:
\[
\text{CHORun} \ A \ \rho \ \text{HOs} \ \text{coords} =
\hspace{1cm}
(\text{CHOinitConfig} \ A \ (\rho \ 0) \ (\text{coords} \ 0))
\land (\forall r. \ \text{CHOnextConfig} \ A \ r \ (\rho \ r) \ (\text{HOs} \ r) \ (\text{coords} \ (\text{Suc} \ r)) \ (\rho \ (\text{Suc} \ r)))
\]
by (auto simp: \text{CHORun-def} \ CSHORun-def \ CHOinitConfig-def \ CHOnextConfig-def)

Without coordinators

**definition** \textit{HOnextConfig} where
lemma $\text{HOnextConfig-eq}$:
\[
\text{HOnextConfig} A \ r \ \text{cfg} \ \text{HO} \ \text{cfg}' = \\
(\forall p. \text{nextState} A \ r \ p (\text{cfg} \ p) (\text{HOrcvdMsgs} A \ r \ p (\text{HO} \ p) (\text{cfg} \ (\text{cfg}' \ p))) \\
\text{by (auto simp: HOnextConfig-def SHOnextConfig-eq SHOmsgVectors-HO)}
\]

definition $\text{HORun}$ where
\[
\text{HORun} A \ \rho \ \text{HOs} \equiv \text{SHORun} A \ \rho \ \text{HOs} \ \text{HOs}
\]

lemma $\text{HORun-eq}$:
\[
\text{HORun} A \ \rho \ \text{HOs} = \\
(\text{HOinitConfig} A (\rho \ 0) \wedge (\forall r. \text{HOnextConfig} A \ r (\rho \ r) (\text{HOs} \ r) (\rho \ (\text{Suc} \ r)))) \\
\text{by (auto simp: HORun-def SHORun-eq HOnextConfig-def)}
\]
The following derived proof rules are immediate consequences of the definition of $\text{CHORun}$; they simplify automatic reasoning.

lemma $\text{CHORun-0}$:
\[
\text{assumes CHORun} A \ \rho \ \text{HOs} \ \text{coords} \\
\text{and } (\forall \text{cfg}. \text{CHOinitConfig} A \ \text{cfg} \ (\text{coords} \ 0) \Longrightarrow \ P \ \text{cfg} \\
\text{shows } P \ (\rho \ 0) \\
\text{using assms unfolding CHORun-eq by blast}
\]

lemma $\text{CHORun-Suc}$:
\[
\text{assumes CHORun} A \ \rho \ \text{HOs} \ \text{coords} \\
\text{and } (\forall \ r. \text{CHOnextConfig} A \ r (\rho \ r) (\text{HOs} \ r) (\text{coords} \ (\text{Suc} \ r)) (\rho \ (\text{Suc} \ r))) \\
\Longrightarrow \ P \ n \\
\text{shows } P \ n \\
\text{using assms unfolding CHORun-eq by blast}
\]

lemma $\text{CHORun-induct}$:
\[
\text{assumes run: CHORun} A \ \rho \ \text{HOs} \ \text{coords} \\
\text{and init: CHOinitConfig} A (\rho \ 0) (\text{coords} \ 0) \Longrightarrow \ P \ 0 \\
\text{and step: } (\forall r. [ P \ r; \text{CHOnextConfig} A \ r (\rho \ r) (\text{HOs} \ r) (\text{coords} \ (\text{Suc} \ r)) \\
(\rho \ (\text{Suc} \ r))] \Longrightarrow \ P \ (\text{Suc} \ r)) \\
\text{shows } P \ n \\
\text{using run unfolding CHORun-eq by (induct n, auto elim: init step)}
\]

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 +
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 fine-grained 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 eas-
ier 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 fault-tolerant 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.

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

type-synonym ('pst, 'proc, 'msg) fgrun = nat => ('pst, 'proc, 'msg) config

definition fg-init-config where
  fg-init-config A (config::('pst, 'proc, 'msg) config) (coord::'proc coord) ≡

\end{verbatim}
\[ \text{round config} = (\lambda p. 0) \]
\[ \land (\forall p. \text{CinitState} A p (\text{state config} p) (\text{coord} p)) \]
\[ \land \text{sent config} = (\lambda p. \{\}) \]
\[ \land \text{rcvd config} = (\lambda p q. \text{None}) \]
\[ \land \text{network config} = \{\} \]

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 \( \text{sendMsg} \) function. The effect of the transition is to add \( q \) to the \( \text{sent} \) component of the configuration and the message quadruple to the \( \text{network} \) component.

**definition fg-send-msg**

\[
\begin{align*}
\text{fg-send-msg} \ A \ p \ q \ & \text{config} \ \text{config}' \\
& \equiv \\
& q \notin (\text{sent config} p) \\
& \land \text{config}' = \text{config} \} \\
& \land \text{sent} := (\text{sent config})(p := (\text{sent config} p) \cup \{q\}), \\
& \land \text{network} := \text{network config} \cup \\
& \{(p, \text{round config} p, q, \\
& \text{sendMsg} A (\text{round config} p) p q (\text{state config} p))\} \} \\
\end{align*}
\]

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 \( \text{HO} \) 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 \( \text{SHO} \) set (parameter \( \text{SHO} \)), then an arbitrary value \( m' \) is received instead of \( m \).

**definition fg-rcv-msg**

\[
\begin{align*}
\text{fg-rcv-msg} \ p \ q \ & \text{HO} \ \text{SHO} \ \text{config} \ \text{config}' \equiv \\
& \exists m m'. (q, (\text{round config} p), p, m) \in \text{network config} \\
& \land q \in \text{HO} \\
& \land \text{rcvd} := (\text{rcvd config})(p := (\text{rcvd config} p)(q := \\
& \text{if } q \in \text{SHO} \text{ then Some } m \text{ else Some } m')), \\
& \land \text{network} := \text{network config} - \{(q, (\text{round config} p), p, m)\} \} \\
\end{align*}
\]

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 $HO$). The local state is updated according to the algorithm's $C_{nextState}$ 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

$fg$-$local$ $A$ $p$ $HO$ $crd$ $config$ $config'$ $\equiv$

$\quad$ $sent$ $config$ $p$ $=$ $UNIV$

$\land$ $\text{dom } (rcvd$ $config$ $p)$ $=$ $HO$

$\land$ $(\exists s$. $C_{nextState}$ $A$ $(round$ $config$ $p)$ $p$ $(state$ $config$ $p)$ $(rcvd$ $config$ $p)$ $crd$ $s$

$\land$ $config'$ $=$ $config$ $\langle$

$\quad$ $\text{round }$ $:= (round$ $config) (p$ $:= Suc$ $(round$ $config$ $p))$, 

$\quad$ $\text{state }$ $:= (state$ $config) (p$ $:= s)$, 

$\quad$ $\text{sent }$ $:= (sent$ $config) (p$ $:= \{\})$, 

$\quad$ $\text{rcvd }$ $:= (rcvd$ $config) (p$ $:= \lambda q. \text{None} \rangle$ $\rangle$

The next-state relation for process $p$ is just the disjunction of the above three types of transitions.

**definition** $fg$-$next-config$ where

$fg$-$next-config$ $A$ $p$ $HO$ $SHO$ $crd$ $config$ $config'$ $\equiv$

$(\exists q$. $fg$-$send-msg$ $A$ $p$ $q$ $config$ $config'$ $\lor$

$(\exists q$. $fg$-$rcv-msg$ $p$ $q$ $HO$ $SHO$ $config$ $config'$ $\lor$

$fg$-$local$ $A$ $p$ $HO$ $crd$ $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** $fg$-$run$ where

$fg$-$run$ $A$ $rho$ $HO$s $SHOs$ $coords$ $\equiv$

$fg$-$init-config$ $A$ $(rho$ $0)$ $(coords$ $0)$

$\land$ $(\forall i$. $\exists p$. $fg$-$next-config$ $A$ $p$

$\quad$ $(HO$s $(round$ $(rho$ $i)$ $p)$ $p)$

$\quad$ $(SHOs$ $(round$ $(rho$ $i)$ $p)$ $p)$

$\quad$ $(coords$ $(round$ $(rho$ $(Suc$ $i))$ $p)$ $p)$

$\quad$ $(rho$ $i)$ $(rho$ $(Suc$ $i))$ $)$

$\land$ $(\forall p$ $r$. $\exists n$. $\text{round } (rho$ $n)$ $p$ $=$ $r$)

The following function computes at which “time point” (index in the fine-grained 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.

**definition** $fg$-$start-round$ where

$fg$-$start-round$ $rho$ $p$ $r$ $\equiv$ LEAST $(n$ $\cdot$ $\text{nat})$. $\text{round } (rho$ $n)$ $p$ $=$ $r$

16
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* \( \rho: \text{fg-run} A \rho \text{HOs} \text{SHOs} \text{coords} \)

*And* \( \text{rd}: \text{round} (\rho (\text{Suc} n)) p = \text{round} (\rho n) p \)

*Shows* \( \text{state} (\rho (\text{Suc} n)) p = \text{state} (\rho n) p \)

**Proof**

- From \( \rho \) have \( \exists p'. \text{fg-next-config} A p' (\text{HOs} (\text{round} (\rho n) p') p') \)
  
  (SHOs (round (\rho n) p') p')
  
  (coords (round (\rho (\text{Suc} n)) p') p')
  
  (\rho n) (\rho (\text{Suc} n))

   *by* (auto simp: fg-run-def)

   *With* \( \text{rd} \) show \( \text{thesis} \)

   *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* \( \rho: \text{fg-run} A \rho \text{HOs} \text{SHOs} \text{coords} \)

*And* \( n: n \leq m \)

*Shows* \( \text{round} (\rho n) p \leq \text{round} (\rho m) p \)

**Proof**

- From \( n \) obtain \( k \) where \( k: m = n + k \)

   *by* (auto simp: le-iff-add)

   { Fix \( i \)

     Have \( \text{round} (\rho n) p \leq \text{round} (\rho (n+i)) p \) (is \( \text{?P} i \))

     *Proof* (induct \( i \))

     *Show* \( \text{?P} 0 \) *by* simp

     Next

     Fix \( j \)

     Assume \( \text{ih}: \text{?P} j \)

     From \( \rho \) have \( \exists p'. \text{fg-next-config} A p' (\text{HOs} (\text{round} (\rho (n+j)) p') p') \)

     (SHOs (round (\rho (n+j)) p') p')

     (coords (round (\rho (\text{Suc} (n+j))) p') p')

     (\rho (n+j)) (\rho (\text{Suc} (n+j)))

     *by* (auto simp: fg-run-def)

     Hence \( \text{round} (\rho (n+j)) p \leq \text{round} (\rho (n + \text{Suc} j)) p \)

     *by* (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)

     *With* \( \text{ih} \) show \( \text{?P} (\text{Suc} j) \) *by* auto

**Qed**

} With \( k \) show \( \text{thesis} \) *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`
- $rd$: `round ($\rho$ $m$) $p$ = `round ($\rho$ $n$) $p$`

**Shows**
- `state ($\rho$ $m$) $p$ = `state ($\rho$ $n$) $p$`

**Proof**

```isar`

```

Since every process executes every round, function `fg-startRound` is well-defined. 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`: 18
assumes \( fg\text{-}run \ A \ rho \ HOs \ SHOs \ coords \)
shows \( \text{round} \ (\rho \ (fg\text{-}start\text{-}round \ \rho \ p \ r)) \ p = r \)
using assms by (auto simp: \( fg\text{-}run\text{-}def \ \\text{fg\text{-}start\text{-}round}\text{-}def \ \text{intro} \ \text{LeastI-ex} \)

lemma \( fg\text{-}start\text{-}round\text{-}smallest \):
assumes \( \text{round} \ (\rho \ k) \ p = r \)
shows \( fg\text{-}start\text{-}round \ \rho \ p \ r \leq \ k \ (k::\text{nat}) \)
using assms unfolding \( fg\text{-}start\text{-}round\text{-}def \) by (rule Least-le)

lemma \( fg\text{-}start\text{-}round\text{-}later \):
assumes \( \rho : \text{fg\text{-}run} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ \text{coords} \)
and \( r : \text{round} \ (\rho \ n) \ p = r \ \text{and} \ r' : r < r' \)
shows \( n < fg\text{-}start\text{-}round \ \rho \ p \ r' \ (\text{is} < ?\text{start}) \)
proof (rule contr)
  assume \( \neg \ ?\text{thesis} \)
  hence \( \text{start} : ?\text{start} \leq n \) by simp
  from \( \rho \) this have \( \text{round} \ (\rho \ (fg\text{-}start\text{-}round \ \rho \ p \ r)) \ p = r \) by (rule \( fg\text{-}start\text{-}round \))
  from \( \rho \) this \( r \) have \( \text{fg\text{-}start\text{-}round} \ \rho \ p \ 0 \leq 0 \) by (rule \( fg\text{-}start\text{-}round\text{-}smallest \))
  thus \( ?\text{thesis} \) by simp
qed

lemma \( fg\text{-}start\text{-}round\text{-}0 \):
assumes \( \rho : \text{fg\text{-}run} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ \text{coords} \)
shows \( fg\text{-}start\text{-}round \ \rho \ p \ 0 = 0 \)
proof
  from \( \rho \) this have \( \text{round} \ (\rho \ 0) \ p = 0 \) by (auto simp: \( fg\text{-}run\text{-}def \ \text{fg\text{-}init\text{-}config}\text{-}def \))
  hence \( fg\text{-}start\text{-}round \ \rho \ p \ 0 \leq 0 \) by (rule \( fg\text{-}start\text{-}round\text{-}smallest \))
  thus \( ?\text{thesis} \) by simp
qed

lemma \( fg\text{-}start\text{-}round\text{-}strict\text{-}mono \):
assumes \( \rho : \text{fg\text{-}run} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ \text{coords} \)
shows \( \text{strict\text{-}mono} \ (fg\text{-}start\text{-}round \ \rho \ p) \)
proof
  fix \( r \ r' \)
  assume \( r : (r::\text{nat}) < r' \)
  from \( \rho \) have \( \text{round} \ (\rho \ (fg\text{-}start\text{-}round \ \rho \ p \ r)) \ p = r \) by (rule \( fg\text{-}start\text{-}round \))
  from \( \rho \) this \( r \) show \( fg\text{-}start\text{-}round \ \rho \ p \ r < fg\text{-}start\text{-}round \ \rho \ p \ r' \) by (rule \( fg\text{-}start\text{-}round\text{-}later \))
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 \( fg\text{-}same\text{-}round\text{-}same\text{-}state \), this implies that the local state of process \( p \) is the same at all these configurations.

lemma \( fg\text{-}round\text{-}between\text{-}start\text{-}rounds \):
assumes \( \rho : \text{fg\text{-}run} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ \text{coords} \)
and \( 1 : \text{fg\text{-}start\text{-}round} \ \rho \ p \ r \leq n \)
and \( 2 : n < \text{fg\text{-}start\text{-}round} \ \rho \ p \ (\text{Suc} \ r) \)
shows \( \text{round} (\rho n) p = r \) (is \( \#rd = r \))

**proof** (rule antisym)

from 1 have \( \text{round} (\rho (\text{fg-start-round} \rho p r)) p \leq \#rd \)

by (rule \( \text{fg-round-numbers-increase} [\text{OF} \rho] \))

thus \( r \leq \#rd \) by (simp add: \( \text{fg-start-round} [\text{OF} \rho] \))

next

show \( \#rd \leq r \)

**proof** (rule ccontr)

assume \( \neg \)\( \text{thesis} \)

hence \( \text{Suc} r \leq \#rd \) by simp

hence \( \text{fg-start-round} \rho p (\text{Suc} r) \leq \text{fg-start-round} \rho p \#rd \)

by (rule \( \rho \) THEN \( \text{fg-start-round-strict-mono}, \) THEN \( \text{strict-mono-mono}, \) THEN \( \text{monoD} \))

also have ... \( \leq n \) by (auto intro: \( \text{fg-start-round-smallest} \))

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** \( \text{fg-local-transition-from-round} \):

**assumes** \( \rho: \text{fg-run} A \rho \text{HOs SHOs coords} \)

obtains \( n \) where \( \text{round} (\rho n) p = r \)

and \( \text{fg-start-round} \rho p (\text{Suc} r) = \text{Suc} n \)

and \( \text{fg-local} A p (\text{HOs} r p) (\text{coords} (\text{Suc} r) p) (\rho n) (\rho (\text{Suc} n)) \)

**proof** –

have \( \text{fg-start-round} \rho p (\text{Suc} r) \neq 0 \) (is \( \#start \neq 0 \))

**proof**

assume contr: \( \#start = 0 \)

from \( \rho \) have \( \text{round} (\rho \#start) p = \text{Suc} r \) by (rule \( \text{fg-start-round} \))

with contr \( \rho \) show False by (auto simp: \( \text{fg-run-def} \) \( \text{fg-init-config-def} \))

qed

then obtain \( n \) where \( n: \#start = \text{Suc} n \) by (auto simp: \( \text{gr0-conv-Suc} \))

with \( \text{fg-start-round}[\text{OF} \rho, \text{of} \ p \text{ Suc} r] \)

have 0: \( \text{round} (\rho (\text{Suc} n)) p = \text{Suc} r \) by simp

have 1: \( \text{round} (\rho n) p = r \)

**proof** (rule \( \text{fg-round-between-start-rounds}[\text{OF} \rho] \))

have \( \text{fg-start-round} \rho p r < \text{fg-start-round} \rho p (\text{Suc} r) \)

by (rule \( \text{fg-start-round-strict-mono}[\text{OF} \rho, \) THEN \( \text{strict-monoD}]) \) simp

with \( n \) show \( \text{fg-start-round} \rho p r \leq n \) by simp

next

from \( n \) show \( n < \#start \) by simp

**qed**

from \( \rho \) obtain \( p' \) where

\( \text{fg-next-config} A p' (\text{HOs} (\text{round} (\rho n) p') p') \)

(\( \text{SHOs} (\text{round} (\rho n) p') p' \))

(\( \text{coords} (\text{round} (\rho (\text{Suc} n)) p') p' \))

(\( \rho n \) (\( \rho (\text{Suc} n) \))

20
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 \( m: (p,r,q,m) \in \text{network} (\rho n) \) (is ?msg \( n \))
shows \( m = \text{sendMsg} A r p q (\text{state} (\rho (fg\text{-start-round} \rho p r) p)) \)
using \( m \) proof (induct \( n \))
— the base case is trivial because the network is empty
assume \( ?\text{msg} 0 \) with \( \rho \) show \( \text{thesis} \)
by (auto simp: fg-run-def fg-init-config-def)
next
fix \( n \)
assume \( m': ?\text{msg} (\text{Suc} n) \) and \( \text{ih}: ?\text{msg} n \Longrightarrow \text{thesis} \)
from \( \rho \) obtain \( p' \) where
\( fg\text{-next-config} A p' (\text{HOs} (\text{round} (\rho n) p') p' \)
\( (\text{SHOs} (\text{round} (\rho n) p') p') \)
\( (\text{coords} (\text{round} (\rho (\text{Suc} n)) p') p') \)
\( (\rho n) (\rho (\text{Suc} n)) \)
(is \( fg\text{-next-config} -\ - ?\text{HO} ?\text{SHO} ?\text{crd} ?\text{cfg} ?\text{cfg}' \))
by (force simp: fg-run-def)
thus \( \text{thesis} \)
proof (auto simp: fg-next-config-def)

Only \( fg\text{-send-msg} \) transitions for process \( p \) are interesting, since all other transitions cannot add a message for \( p \), hence we can apply the induction hypothesis.

fix \( q' \)
assume \( \text{send: } fg\text{-send-msg} A p' q' ?cfg ?cfg' \)
show \( \text{thesis} \)
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 r p q (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 1 2 show ?thesis by simp
qed
next
fix q'
  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
  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 ∈ HOs (round (rho n) q) p
    (is p ∈ HOs (?rd 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 ih: ?rcvd n ⇒ ?P n
  — For the inductive step we distinguish the possible transitions
  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 ?P (Suc n)
proof (auto simp: fg-next-config-def)

Except for \( \text{fg-rcv-msg} \) steps of process \( q \), the proof is immediately reduced to the induction hypothesis.

fix \( q' \)
assume rcvmsg: \( \text{fg-rcv-msg} \) \( p' \) \( q' \) \( \text{HO} \) \( \text{SHO} \) \( \text{cfg} \) \( \text{cfg}' \)
hence \( \text{rd} \): \( \text{rd} (\text{Suc } n) = ?\text{rd } n \) by (auto simp: \( \text{fg-rcv-msg-def} \))
show \( ?P (\text{Suc } n) \)
proof (cases ?rcvd \( n \))
  case True
  with \( \text{ih rd} \) show \( \text{thesis} \) by simp
next
  case False
  with rcvd rcvmsg rd have \( \text{thesis} \) by (auto simp: \( \text{fg-rcv-msg-def} \))
qed

next
fix \( q' \)
assume fg-send-msg A \( p' \) \( q' \) \( \text{cfg} \) \( \text{cfg}' \)
with rcvd have \( \text{rcvd } n \) \( \text{and rd} (\text{Suc } n) = ?\text{rd } n \) by (auto simp: \( \text{fg-send-msg-def} \))
with \( \text{ih} \) show \( ?P (\text{Suc } n) \) by simp
qed

next
fix \( n \)
assume \( \text{fg-local} A \) \( p' \) \( \text{HO} \) \( \text{crd} \) \( \text{cfg} \) \( \text{cfg}' \)
with rcvd have \( \text{rcvd } n \) \( \text{and rd} (\text{Suc } n) = ?\text{rd } n \)
  in fact, \( p' = q \) is impossible because the \( \text{rcvd} \) field of \( p' \) is cleared
  by (auto simp: \( \text{fg-local-def} \))
with \( \text{ih} \) show \( ?P (\text{Suc } n) \) by simp
qed

lemma \( \text{fg-invariant2b} \):
assumes rho: \( \text{fg-run} A \) \( \text{rho} \) \( \text{HOs} \) \( \text{SHOs} \) coords
  and \( m \): \( \text{rcvd} (\text{rho } n) \) \( q \) \( p = \text{Some } m \) (is \( ?\text{rcvd } n \))
  and sho: \( p \in \text{SHOs} (\text{round} (\text{rho } n) \) \( q) \) (is \( p \in \text{SHOs} ( ?\text{rd } n) \) \( q) \)
shows \( m = \text{sendMsg A} (\text{?rd } n) \) \( p \) \( q \)
  (is \( ?P \) \( n \))
using \( m \) sho proof (induct \( n \))
  The base case is trivial because \( q \) has not received any message initially
assume \( ?\text{rcvd } 0 \) with rho show \( ?P \) \( 0 \)
  by (auto simp: \( \text{fg-run-def} \) \( \text{fg-init-config-def} \))
next
fix \( n \)
assume rcvd: \( ?\text{rcvd} (\text{Suc } n) \) \( \text{and p} \in \text{SHOs} ( ?\text{rd} (\text{Suc } n)) \) \( q \)
  and ih: \( ?\text{rcvd } n \Longrightarrow p \in \text{SHOs} ( ?\text{rd } n) \) \( q \Longrightarrow ?P \) \( n \)
  For the inductive step we again distinguish the possible transitions
from rho obtain \( p' \) where
  \( \text{fg-next-config} A \) \( p' \) (\( \text{HOs} (\text{round} (\text{rho } n) \) \( p') \) \( p' \))
  (\( \text{SHOs} (\text{round} (\text{rho } n) \) \( p') \) \( p' \))
Except for \( \text{fg-rcv-msg} \) steps of process \( q \), the proof is immediately reduced to the induction hypothesis.

\[
\text{except for } \text{fg-rcv-msg} \text{ steps of process } q, \text{ the proof is immediately reduced to the induction hypothesis.}
\]

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 HO, SHO, and coordinator assignments. By definition (and the fact that local states remain equal between \( fg\text{-}start\text{-}round \) 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

\[
coarse\text{-}run \rho r p \equiv state (\rho (fg\text{-}start\text{-}round \rho p r)) p
\]

**theorem** reduction:

- **assumes** \( \rho : fg\text{-}run A \) \( HOs \) \( SHOs \) \( coords \)
- **shows** \( CSHORun A \) \( (coarse\text{-}run \rho) \) \( HOs \) \( SHOs \) \( coords \)
  - \( is \) \( CSHORun - ?cr - - - \)
- **proof** \( (auto \ simp: CSHORun-def) \)
  - **from** \( \rho \) **show** \( CHOinitConfig A \) \( (?cr \ 0) \) \( (coords \ 0) \)
    - \( by (auto \ simp: \ fg\text{-}run-def \ fg\text{-}init\text{-}config-def \ CHOinitConfig-def \)
    - \( coarse\text{-}run-def \) \( fg\text{-}start\text{-}round-0 \) \( (OF \ rho) \)

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\text{-}local\text{-}transition\text{-}from\text{-}round] \ obtain** \( n \)
    - **where** \( n : \ \ round \ (\rho \ n) \ p = r \)
      - **and** \( start : \ \ fg\text{-}start\text{-}round \ \ rho \ p \) \( (Suc \ r) = Suc \ n \) \( is \ ?start = - \)
      - **and** \( loc : \ \ fg\text{-}local A \ p \) \( (HOs \ r \ p) \) \( (coords \ (Suc \ r) \ p) \) \( (rho \ n) \) \( (rho \ (Suc \ n)) \)
        - \( (is \ fg\text{-}local - - ?HO \ ?crd \ ?cfg \ ?cfg') \)
      - **by** \( blast \)
  - **have** \( cfg : \ ?cr \ r \ p = state \ ?cfg \ p \)
    - **unfolding** coarse-run-def **proof** \( (rule fg\text{-}same\text{-}round\text{-}same\text{-}state \ (OF \ rho)) \)
  - **from** \( n \) **show** \( round \ (\rho \ (fg\text{-}start\text{-}round \ rho \ p \ r)) \ p = round \ ?cfg \ p \)
    - **by** \( (simp \ add: fg\text{-}start\text{-}round \ (OF \ rho)) \)
  - **qed**
  - **from** \( start \) **have** \( cfg' : \ ?cr \ (Suc \ r) \ p = state \ ?cfg' \ p \)
    - **by** \( (simp \ add: coarse\text{-}run\text{-}def) \)
  - **have** \( rcvd : \ \ rcvd \ ?cfg \ p \in SHOmsgVectors A \) \( r \) \( p \) \( (?cr \ r) \) \( ?HO \) \( (SHOs \ r \ p) \)
    - **proof** \( (auto \ simp: SHOmsgVectors-def) \)
      - **fix** \( q \)
        - **assume** \( q \in \ ?HO \)
      - **with** \( n \) **loc** **show** \( \exists \ m. rcvd \ ?cfg \ p \ q = Some \ m \) **by** \( (auto \ simp: fg\text{-}local\text{-}def) \)
  - next
    - **fix** \( q \ m \)
      - **assume** \( rcvd \ ?cfg \ p \ q = Some \ m \)
    - **with** \( \rho \) \( n \) **show** \( q \in \ ?HO \) **by** \( (auto \ simp: fg\text{-}invariant2a) \)
  - next

25
3.4 Locally Similar Runs and Local Properties

We say that two sequences of configurations (vectors of process states) are \emph{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 \emph{local} if it is insensitive to local similarity.

\begin{definition}
locally-similar where
locally-similar (σ::nat ⇒ ’proc ⇒ ’pst) τ ≡
∀p::’proc. (λn. σ n p) ≈ (λn. τ n p)
\end{definition}

\begin{definition}
local-property where
local-property P ≡
∀σ τ. locally-similar σ τ → P σ → P τ
\end{definition}

Local similarity is an equivalence relation.

\begin{lemma}
locally-similar-refl: \textit{locally-similar} σ σ
by (simp add: locally-similar-def stutter-equiv-refl)
\end{lemma}

\begin{lemma}
locally-similar-sym: \textit{locally-similar} σ τ → \textit{locally-similar} τ σ
by (simp add: locally-similar-def stutter-equiv-sym)
\end{lemma}

\begin{lemma}
locally-similar-trans [trans]:
\textit{locally-similar} q σ → \textit{locally-similar} σ τ → \textit{locally-similar} q τ
by (force simp add: locally-similar-def elim: stutter-equiv-trans)
\end{lemma}

\begin{lemma}
local-property-eq:
local-property P = (∀σ τ. locally-similar σ τ → P σ = P τ)
by (auto simp: local-property-def dest: locally-similar-sym)
\end{lemma}

Consider any fine-grained run \textit{rho}. The projection of \textit{rho} to vectors of
process states is locally similar to the coarse-grained run computed from \( \rho \).

**Lemma** coarse-run-locally-similar:

**Assumes** \( \rho : \text{fg-run} A \rho \text{ HO}s \text{ SHO}s \text{ coords} \)

**Shows** locally-similar \((\text{state} \circ \rho) \) (coarse-run \( \rho \))

**Proof** (auto simp: locally-similar-def)

\[
\text{fix } p
\]

\[
\text{show } (\lambda n. \text{state} (\rho n) p) \approx (\lambda n. \text{coarse-run } \rho n \ p) \quad (\text{is } ?\text{fgr} \approx ?\text{cgr})
\]

**Proof** (rule stutter-equivI)

\[
\text{show } \text{stutter-sampler} \ (\text{fg-start-round } \rho \ p) \ ?\text{fgr}
\]

**Proof** (auto simp: stutter-sampler-def)

\[
\text{from } \rho \text{ show } \text{fg-start-round } \rho \ p \ 0 = 0
\]

by (rule fg-start-round-0)

next

\[
\text{show } \text{strict-mono} \ (\text{fg-start-round } \rho \ p)
\]

by (rule fg-start-round-strict-mono[\( \text{OF } \rho \)])

next

\[
\text{fix } r \ n
\]

\[
\text{assume } \text{fg-start-round } \rho \ p \ r < n \text{ and } n < \text{fg-start-round } \rho \ p \ (\text{Suc } r)
\]

\[
\text{with } \rho \text{ have } \text{round} \ (\rho n) \ p = \text{round} \ (\rho \ (\text{fg-start-round } \rho \ p \ r)) \ p
\]

by (simp add: fg-start-round-fg-round-between-start-runs)

next

\[
\text{show } ?\text{fgr} \circ \text{fg-start-round } \rho \ p = ?\text{cgr} \circ \text{id}
\]

by (auto simp: coarse-run-def)

qed

Therefore, in order to verify a local property \( P \) for a fine-grained run over given HO, 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** \( \rho : \text{fg-run} A \rho \text{ HO}s \text{ SHO}s \text{ coords} \)

and \( P : \text{local-property } P \)

and coarse-correct:

\[
\bigwedge \rho. \ [ \text{CSHORun} A \ crho \text{ HO}s \text{ SHO}s \text{ coords}; \ crho \ 0 = \text{state} \ (\rho \ 0)] \implies P \ crho
\]

**Shows** \( P \ (\text{state} \circ \rho) \)

**Proof**

\[
\text{have } \text{coarse-run } \rho \ 0 = \text{state} \ (\rho \ 0)
\]

by (rule ext, simp add: coarse-run-def fg-start-round-0[\( \text{OF } \rho \)])
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 σ τ
and val: \( \forall n. \text{dec} (\sigma n p) = \text{Some} v \implies v \in \text{range vals} \)
and dec: \( \text{dec} (\tau n p) = \text{Some} v \)
shows \( v \in \text{range vals} \)
proof –
from sim have \( (\lambda r. \sigma r p) \approx (\lambda r. \tau r p) \) by (simp add: locally-similar-def)
then obtain m where σ m p = τ 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 σ τ
and val: \( \forall n. \text{dec} (\sigma n p) = \text{Some} w \implies w = v \)
and dec: \( \text{dec} (\tau n p) = \text{Some} w \)
shows \( w = v \)
proof –
from sim have \( (\lambda r. \sigma r p) \approx (\lambda r. \tau r p) \) by (simp add: locally-similar-def)
then obtain m where σ m p = τ 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 σ τ
and agr: \( \forall m n. \left[ \text{dec} (\sigma m p) = \text{Some} v; \text{dec} (\sigma n q) = \text{Some} w \right] \implies v = w \)
and v: dec (\tau m p) = Some v and w: \text{dec} (\tau n q) = Some w
shows \( v = w \)
proof –
from sim have \( (\lambda r. \sigma r p) \approx (\lambda r. \tau r p) \) by (simp add: locally-similar-def)
then obtain m' where m' = τ m p by (rule stutter-equiv-element-left)
from sim have \( (\lambda r. \sigma r q) \approx (\lambda r. \tau r q) \) by (simp add: locally-similar-def)
then obtain n' where n' = τ n q by (rule stutter-equiv-element-left)
from sym[OF m'] sym[OF n'] v w show v = w by (auto elim: agr)
qed

lemma termination-is-local:
assumes sim: locally-similar σ τ
and trm: \text{dec} (\sigma m p) = \text{Some} v
shows \( \exists n. \text{dec} (\tau n p) = \text{Some} v \)
proof

from sim have \((\lambda r. \sigma r p) \approx (\lambda r. \tau r p)\) by (simp add: locally-similar-def)
then obtain \(n\) where \(\sigma m p = \tau n p\) 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 \(\sigma\ \tau\ n\ p\ v\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall n\ p\ v.\ \text{dec} (\sigma\ n\ p) = \text{Some} v\ \longrightarrow v \in \text{range vals}\)
and \(\text{dec} (\tau\ n\ p) = \text{Some} v\)
thus \(v \in \text{range vals}\) by (blast intro: integrity-is-local)
next
fix \(\sigma\ \tau\ m\ n\ p\ q\ v\ w\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall m\ n\ p\ q\ v\ w.\ \text{dec} (\sigma\ m\ p) = \text{Some} v \land \text{dec} (\sigma\ n\ q) = \text{Some} w\ \longrightarrow v = w\)
and \(\text{dec} (\tau\ m\ p) = \text{Some} v\ \text{and} \ \text{dec} (\tau\ n\ q) = \text{Some} w\)
thus \(v = w\) by (blast intro: agreement-is-local)
next
fix \(\sigma\ \tau\ p\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall p.\ \exists m\ v.\ \text{dec} (\sigma\ m\ p) = \text{Some} v\)
thus \(\exists n\ w.\ \text{dec} (\tau\ n\ p) = \text{Some} w\) 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 \(\sigma\ \tau\ n\ p\ v\ w\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall n\ p\ v\ w.\ \text{dec} (\sigma\ n\ p) = \text{Some} v\ \longrightarrow w = v\)
and \(\text{dec} (\tau\ n\ p) = \text{Some} w\)
thus \(w = v\) by (blast intro: validity-is-local)
next
fix \(\sigma\ \tau\ m\ n\ p\ q\ v\ w\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall m\ n\ p\ q\ v\ w.\ \text{dec} (\sigma\ m\ p) = \text{Some} v \land \text{dec} (\sigma\ n\ q) = \text{Some} w\ \longrightarrow v = w\)
and \(\text{dec} (\tau\ m\ p) = \text{Some} v\ \text{and} \ w.\ \text{dec} (\tau\ n\ q) = \text{Some} w\)
thus \(v = w\) by (blast intro: agreement-is-local)
next
fix \(\sigma\ \tau\ p\)
assume locally-similar \(\sigma\ \tau\)
and \(\forall p.\ \exists m\ v.\ \text{dec} (\sigma\ m\ p) = \text{Some} v\)
thus \(\exists n\ w.\ \text{dec} (\tau\ n\ p) = \text{Some} w\) 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 $\text{crd}: \text{card } E < \text{card } S + \text{card } T$
  and $s: S \subseteq E$ and $t: T \subseteq E$ and $e: \text{finite } E$
  shows $S \cap T \neq \{\}$
proof (clarify)
  assume contra: $S \cap T = \{\}$
  from $s$ $t$ $e$ have finite $S$ and finite $T$ by (auto simp: finite-subset)
  with $\text{crd}$ $\text{contra}$ have $\text{card } E < \text{card } (S \cup T)$ by (auto simp add: card-Un-Int)
  moreover
  from $s$ $t$ $e$ have $\text{card } (S \cup T) \leq \text{card } E$ by (simp add: card-mono)
  ultimately
  show False by simp
qed

lemma abs-majoritiesE:
  assumes $\text{crd}: \text{card } E < \text{card } S + \text{card } T$
  and $s: S \subseteq E$ and $t: T \subseteq E$ and $e: \text{finite } E$
  obtains $p$ where $p \in S$ and $p \in T$
proof
  from $\text{assms}$ have $S \cap T \neq \{\}$ by (rule abs-majorities-intersect)
  then obtain $p$ where $p \in S \cap T$ by blast
  with that show ?thesis by auto
qed

Special case: both sets $S$ and $T$ are majorities.

lemma abs-majoritiesE':
  assumes Smaj: $\text{card } S > (\text{card } E) \text{ div } 2$ and Tmaj: $\text{card } T > (\text{card } E) \text{ div } 2$
  and $s: S \subseteq E$ and $t: T \subseteq E$ and $e: \text{finite } E$
  obtains $p$ where $p \in S$ and $p \in T$
proof (rule abs-majoritiesE[OF - s t e])
  from Smaj Tmaj show $\text{card } E < \text{card } S + \text{card } T$ 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:
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.

typedcl Proc — the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)

abbreviation
\[ N \equiv \text{card} (\text{UNIV}::\text{Proc set}) \]

The state of each process consists of two fields: \( x \) holds the current value proposed by the process and \( \text{decide} \) 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 OTR-initState where
  OTR-initState p st ≡ 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 HOV :: (Proc ⇒ 'val option) ⇒ 'val ⇒ Proc set where
  HOV msgs v ≡ { q . msgs 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 MFR :: (Proc ⇒ 'val option) ⇒ 'val ⇒ bool where
  MFR msgs v ≡ ∀ w. card (HOV msgs w) ≤ card (HOV msgs v)

lemma MFR-exists: ∃ v. MFR msgs v
proof –
  let ?cards = { card (HOV msgs v) | v . True }
  let ?mfr = Max ?cards
  have ∀ v. card (HOV msgs v) ≤ N by (auto intro: card-mono)
  hence ?cards ⊆ { 0 .. N } by auto
  hence fin: finite ?cards by (metis atLeast0AtMost finite-atMost finite-subset)
  hence ?mfr ∈ ?cards by (rule Max-in) auto
  then obtain v where v: ?mfr = card (HOV msgs v) by auto
  have MFR msgs v
    proof (auto simp: MFR-def)
      fix w
      from fin have card (HOV msgs w) ≤ ?mfr by (rule Max-ge) auto
      thus card (HOV msgs w) ≤ 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 fre-
quently received values are among the received messages.

lemma MFR-in-msgs:
  assumes HOs m p ≠ {} and v: MFR (HOrcvdMsgs OTR-M m p (HOs m p) (rho m)) v
  (is MFR ?msgs v)
  shows ∃ q ∈ HOs m p. v = the (?msgs q)
proof –

32
from \( HO \) obtain \( q \) where \( q \in HOs \) 
by auto 
with \( v \) have \( HOV \ ?msgs \ (the \ (\ ?msgs \ q)) \neq {} \) 
by (auto simp: HO-def HOrcvdMsgs-def) 
hence \( HOp \): \( 0 < \text{card} \ (HOV \ ?msgs \ (the \ (\ ?msgs \ q))) \) 
by auto 
also from \( v \) have \( \ldots \leq \text{card} \ (HOV \ ?msgs \ v) \) 
by (simp add: MFR-def) 
finally have \( HOV \ ?msgs \ v \neq {} \) 
by auto 
thus \( \vdots \) 
by (auto simp: HO-def HOrcvdMsgs-def) 
qed 

TwoThirds \( msgs \ v \) holds if value \( v \) has been received from more than \( \frac{2}{3} \) of all processes.

**Definition**: TwoThirds where 
\[
\text{TwoThirds} \ \text{msgs} \ v \equiv (2 \ast N) \div 3 < \text{card} \ (HO msgs \ v)
\]

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 \( \frac{2}{3} \) 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 \( \frac{2}{3} \) of all processes. If \( p \) hasn’t heard from more than \( \frac{2}{3} \) of all processes, the state remains unchanged. (Note that \( \text{Some} \) 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 
\[
\text{OTR-nextState} \ r \ p \ (st::(\alphav::linorder) pstate) \ msgs \ st' \equiv 
\begin{align*}
\text{if} & \ (2 \ast N) \div 3 < \text{card} \ \{q . \ msgs \ q \neq \text{None}\} \\
\text{then} & \ st' = (|x = \text{Min} \ \{v . \ MFR \ msgs \ v\}, \\
\text{decide} & = (\text{if} \ (\exists v. \ \text{TwoThirds} \ msgs \ v) \\
\text{then} & \ \text{Some} \ (\epsilon \ v . \ \text{TwoThirds} \ msgs \ v) \\
\text{else} & \ \text{decide} \ st) \\
\text{else} & \ st' = st
\end{align*}
\]

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**: OTR-sendMsg where 
\[
\text{OTR-sendMsg} \ r \ p \ q \ st \equiv x \ st
\]

### 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 OTR-commPerRd where**

\[ \text{OTR-commPerRd HOrs} \equiv \text{True} \]

**definition OTR-commGlobal where**

\[ \text{OTR-commGlobal HOs} \equiv \forall r. \exists r0 \Pi. r0 \geq r \land (\forall p. \text{HOs } r0 \ p = \Pi) \land \text{card } \Pi > (2N) \text{ div } 3 \]

### 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 \( \text{crd} \) arguments of the initial- and next-state predicates are unused.

**definition OTR-HOMachine where**

\[
\text{OTR-HOMachine} = \\
\quad () \text{CinitState} = (\lambda p \text{ st crd. OTR-initState } p \text{ st}), \\
\quad \text{sendMsg} = \text{OTR-sendMsg}, \\
\quad \text{CnextState} = (\lambda r p \text{ st msgs crd st'} \text{. OTR-nextState } r \ p \text{ st msgs st'}), \\
\quad \text{HOcommPerRd} = \text{OTR-commPerRd}, \\
\quad \text{HOcommGlobal} = \text{OTR-commGlobal} \]

**abbreviation OTR-M \equiv OTR-HOMachine::(Proc, '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 \( \text{decide} \) 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**

\[
\text{VInv rho n} \equiv \\
\quad \text{let xinit} = (\text{range } (x \circ (\text{rho } 0))) \\
\quad \text{in range } (x \circ (\text{rho } n)) \subseteq \text{xinit} \\
\quad \land \text{range } (\text{decide } \circ (\text{rho } n)) \subseteq \{\text{None}\} \cup (\text{Some } \text{xinit})
\]
**lemma** \texttt{vinv-invariant}:

\texttt{assumes run:HORun OTR-M rho HOs}
\texttt{shows VInv rho n}

\texttt{proof (induct n)}

\texttt{from run show VInv rho 0}

\texttt{by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def initState-def OTR-initState-def VInv-def image-def)}

\texttt{next}

\texttt{fix m}

\texttt{assume ih: VInv rho m}

\texttt{let ?xinit = range (x ◦ (rho 0))}

\texttt{have range (x ◦ (rho (Suc m))) ⊆ ?xinit}

\texttt{proof (clarsimp)}

\texttt{fix p}

\texttt{from run}

\texttt{have nxt: OTR-nextState m p (rho m p)}

\texttt{(HOrcvdMsgs OTR-M m p (HOs m p) (rho m))}

\texttt{(rho (Suc m) p)}

\texttt{(is OTR-nextState - - ?st msgs ?st')}

\texttt{by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)}

\texttt{show x ?st' ∈ ?xinit}

\texttt{proof (cases (2*N) div 3 < card (HOs m p))}

\texttt{case True}

\texttt{hence HO: HOs m p ≠ {} by auto}

\texttt{let ?MFRs = {v. MFR ?msgs v}}

\texttt{have Min ?MFRs ∈ ?MFRs}

\texttt{proof (rule Min-in)}

\texttt{from HO have ?MFRs ⊆ (the ∘ ?msgs)’(HOs m p)}

\texttt{by (auto simp: image-def intro: MFR-in-msgs)}

\texttt{thus finite ?MFRs by (auto elim: finite-subset)}

\texttt{next}

\texttt{from MFR-exists show ?MFRs ≠ {} by auto}

\texttt{qed}

\texttt{with HO have ∃ q ∈ HOs m p. Min ?MFRs = the (?msgs q)}

\texttt{by (intro MFR-in-msgs) auto}

\texttt{hence ∃ q ∈ HOs m p. Min ?MFRs = x (rho m q)}

\texttt{by (auto simp: HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)}

\texttt{moreover}

\texttt{from True nxt have x ?st' = Min ?MFRs}

\texttt{by (simp add: OTR-nextState-def HOrcvdMsgs-def)}

\texttt{ultimately}

\texttt{show ?thesis using ih by (auto simp: VInv-def image-def)}

\texttt{next}

\texttt{case False}

\texttt{with nxt ih show ?thesis}

\texttt{by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def Let-def)}

\texttt{qed}

\texttt{qed}
\[
\text{moreover have } \forall p. \text{ decide } ((\rho \circ (\text{Suc } m)) \circ p) \in \{\text{None}\} \cup \{\text{Some } \vec{?xinit}\} \\
\text{proof} \\
\text{fix } p \\
\text{from run have nxt: } \text{OTR-nextState } m \circ p (\rho \circ m \circ p) \Rightarrow (\text{HOrcvdMsgs } OTR-M \circ m \circ p (\text{HOs } m \circ p) (\rho \circ m)) \\
\Rightarrow (\rho \circ (\text{Suc } m) \circ p) \\
\text{by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)} \\
\text{show decide } \vec{?st}' \in \{\text{None}\} \cup \{\text{Some } \vec{?xinit}\} \\
\text{proof (cases } (2 \times N) \div 3 < \text{card } \{q. \text{msgs } q \neq \text{None}\}) \\
\text{assume HO: } (2 \times N) \div 3 < \text{card } \{q. \text{msgs } q \neq \text{None}\} \\
\text{show } ?\text{thesis} \\
\text{proof (cases } \exists v. \text{TwoThirds } ?\text{msgs } v) \\
\text{case True} \\
\text{let } ?\text{dec } = v \circ \text{TwoThirds } ?\text{msgs } v \\
\text{from True have TwoThirds } ?\text{msgs } ?\text{dec by (rule someI-ex)} \\
\text{hence HOV } ?\text{msgs } ?\text{dec } \neq \{\} \text{ by (auto simp add: TwoThirds-def)} \\
\text{then obtain } q \text{ where } x (\rho \circ m \circ q) = ?\text{dec } \\
\text{by (auto simp: HOV-def HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)} \\
\text{from sym[OF this] nxt ih show } ?\text{thesis} \\
\text{by (auto simp: OTR-nextState-def VInv-def image-def)} \\
\text{next} \\
\text{case False} \\
\text{with HO nxt ih show } ?\text{thesis} \\
\text{by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def image-def)} \\
\text{qed} \\
\text{next} \\
\text{case False} \\
\text{with nxt ih show } ?\text{thesis} \\
\text{by (auto simp: OTR-nextState-def VInv-def image-def)} \\
\text{qed} \\
\text{hence range } (\text{decide } \circ (\rho \circ (\text{Suc } m))) \subseteq \{\text{None}\} \cup \{\text{Some } \vec{?xinit}\} \text{ by auto} \\
\text{ultimately show VInv } \rho \circ (\text{Suc } m) \text{ by (auto simp: VInv-def image-def)} \\
\text{qed} \\
\text{Integrity is an immediate consequence.} \\
\text{theorem } \text{OTR-integrity: } \\
\text{assumes } \text{run: } \text{HORun } OTR-M \rho \text{ HOs and dec: } \text{decide } (\rho \circ n \circ p) = \text{Some } v \\
\text{shows } \exists q. \text{ v } = x (\rho \circ 0 \circ q) \\
\text{proof} - \\
\text{let } \vec{?xinit } = \text{range } (x \circ (\rho \circ 0)) \\
\text{from run have VInv } \rho \circ n \text{ by (rule vine-invariant)} \\
\text{hence range } (\text{decide } \circ (\rho \circ n)) \subseteq \{\text{None}\} \cup \{\text{Some } \vec{?xinit}\} \text{ by (auto simp: VInv-def Let-def)} \\
\text{qed}
hence \( \text{decide} ((\rho \ n) \ p) \in \{\text{None}\} \cup (\text{Some} \ ' \ ?xinit) \)
by (auto simp: image-def)
with \( \text{dec} \) show \( ?\text{thesis} \) by auto
qed

5.5 Proof of Agreement

The following lemma \( A1 \) 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 \( \neg ((2 \ast N) \text{ div } 3 < \text{card} \{ q. (\text{HOrcvdMsgs OTR-M n p} (\text{HOs n p}) (\rho n)) q \neq \text{None}\}) \)
shows \( \rho (\text{Suc n}) \ p = \rho n \ p \)
using assms
by (auto simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def OTR-nextState-def)

lemma nextState-decide:
assumes run: HORun OTR-M rho HOs
and \( \text{chg: decide} (\rho (\text{Suc n}) \ p) \neq \text{decide} (\rho n \ p) \)
shows TwoThirds (\text{HOrcvdMsgs OTR-M n p} (\text{HOs n p}) (\rho n))
(\text{the} (\text{decide} (\rho (\text{Suc n}) \ p)))
proof –
from run \( \text{chg} \) have TwoThirds (\text{HOrcvdMsgs OTR-M n p} (\text{HOs n p}) (\rho n)) (\rho (\text{Suc n}) \ p)
by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
with \( \text{chg} \) show \( ?\text{thesis} \) by (auto simp: OTR-nextState-def elim: someI)
qed

lemma A1:
assumes run: HORun OTR-M rho HOs
and \( \text{dec: decide} (\rho (\text{Suc n}) \ p) = \text{Some} \ v \)
and \( \text{chg: decide} (\rho (\text{Suc n}) \ p) \neq \text{decide} (\rho n \ p) \ (\text{is decide} ?st' \neq \text{decide} ?st) \)
shows \( (2 \ast N) \text{ div } 3 < \text{card} \{ q. x (\rho n \ q) = v \} \)
proof –
from run \( \text{chg} \)
have TwoThirds (\text{HOrcvdMsgs OTR-M n p} (\text{HOs n p}) (\rho n))
(\text{the} (\text{decide} ?st'))
(is TwoThirds ?msgs -)
by (rule nextState-decide)
with \( \text{dec} \) have TwoThirds ?msgs v by simp
hence \( (2 \ast N) \text{ div } 3 < \text{card} \{ q. ?msgs q = \text{Some} \ v \} \)
by (simp add: TwoThirds-def HOV-def)
moreover
have \{ q . \text{msgs} q = \text{Some} v \} \subseteq \{ q . x (\rho n q) = v \}

by (auto simp: OTR-HOMachine-def OTR-sendMsg-def HOrcvdMsgs-def)

hence \text{card} \{ q . \text{msgs} q = \text{Some} v \} \leq \text{card} \{ q . x (\rho n q) = v \}

by (simp add: card-mono)

ultimately

show \?thesis by simp

done

The following lemma \textit{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 \(v\).

\textbf{Lemma A2:}
\begin{itemize}
  \item assumes \textit{run}: \text{HORun} \text{ OTR-M} \text{ rho HO}\text{S}
  \item and \textit{HO}: \((2*\text{N}) \text{ div 3} \prec \text{card} \{ q . \text{HOrcvdMsgs} \text{ OTR-M} \text{ n p} (\text{HO}\text{S} \text{ n p}) (\rho n q) \neq \text{None} \} \)
  \item and \textit{maj}: \((2*\text{N}) \text{ div 3} \prec \text{card} \{ q . x (\rho n q) = v \} \)
  \item shows \(x (\rho (\text{Suc n} \text{ p}) = v \)
\end{itemize}

\begin{proof}
  from \textit{run}
  \begin{proof}
    have \textit{nxt}: \text{OTR-nextState} \text{ n p} (\rho n p)
      \begin{proof}
        \begin{proof}
          (\text{HOrcvdMsgs} \text{ OTR-M} \text{ n p} (\text{HO}\text{S} \text{ n p}) (\rho n))
          \begin{proof}
            (\rho (\text{Suc n} \text{ p}))
          \end{proof}
        \end{proof}
      \end{proof}
    \end{proof}
    \begin{proof}
      \text{is} \text{ OTR-nextState} - - \?st \?msgs ?st'
    \end{proof}
    by (simp add: \text{HORun-eq} \text{ HOnextConfig-eq} \text{ OTR-HOMachine-def} \text{ nextState-def})
    \begin{proof}
      \text{let} \?HO\text{Others} = \bigcup \{ \text{HOV} \?msgs w | w . w \neq v \}
      \begin{proof}
        \begin{proof}
          \text{processes from which} \text{ p} \text{ received values different from} v
        \end{proof}
      \end{proof}
      \begin{proof}
        \text{have} \textit{w}: \text{card} \?HO\text{Others} \leq \text{N div 3}
      \end{proof}
      \begin{proof}
        \text{proof}
        \begin{proof}
          \begin{proof}
            \text{have} \text{card} \?HO\text{Others} \leq \text{card} (\text{UNIV} - \{ q . x (\rho n q) = v \})
          \end{proof}
          \begin{proof}
            \text{by (auto simp: HO-def HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def}
          \end{proof}
        \end{proof}
      \end{proof}
      \begin{proof}
        \text{intro: card-mono)
        \text{also have} \ldots = \text{N} - \text{card} \{ q . x (\rho n q) = v \}
      \end{proof}
      \begin{proof}
        \text{by (auto simp: card-Diff-subset)
        \text{also from} \textit{maj} \text{ have} \ldots \leq \text{N div 3} \text{ by auto}
      \end{proof}
      \text{finally show} \?thesis .
    \end{proof}
  \end{proof}
\end{proof}
\end{proof}

have \textit{hov}: \text{HOV} \?msgs v = \{ q . \?msgs q \neq \text{None} \} - \?HO\text{Others}

by (auto simp: HO-def) blast

have \textit{othHO}: \?HO\text{Others} \subseteq \{ q . \?msgs q \neq \text{None} \}

by (auto simp: HO-def)

Show that \(v\) has been received from more than \(N/3\) processes.

from \textit{HO} have \text{N div 3} \prec \text{card} \{ q . \?msgs q \neq \text{None} \} - (\text{N div 3})

by auto

also from \textit{w} \text{HO} have \ldots \leq \text{card} \{ q . \?msgs q \neq \text{None} \} - \text{card} \?HO\text{Others}
by auto
also from hov othHO have \ldots = card (HOV ?msgs v)
  by (auto simp: card-Diff-subset)
finally have HOV: \( N \div 3 < \text{card} (HOV \ ?msgs v) \).

All other values are received from at most \( N/3 \) processes.

have \( \forall w. \ w \neq v \longrightarrow \text{card} (HOV \ ?msgs w) \leq \text{card} \ ?HOVothers \)
  by (force intro: card-mono)
with \( w \) have cardw: \( \forall w. \ w \neq v \longrightarrow \text{card} (HOV \ ?msgs w) \leq N \div 3 \) by auto

In particular, \( v \) is the single most frequently received value.

with HOV have MFR ?msgs v by (auto simp: MFR-def)

moreover
have \( \forall w. \ w \neq v \longrightarrow \neg (MFR \ ?msgs w) \)
proof (auto simp: MFR-def not-le)
  fix \( w \)
  assume \( w \neq v \)
  with cardw HOV have card (HOV ?msgs w) < card (HOV ?msgs v) by auto
  thus \( \exists v. \ \text{card} (HOV \ ?msgs w) < \text{card} (HOV \ ?msgs v) \).
qed

ultimately
have mfrv: \( \{ w . \ MFR \ ?msgs w \} = \{ v \} \) by auto

have \( \text{card} \ \{ q . \ ?msgs q = \text{Some} \ v \} \leq \text{card} \ \{ q . \ ?msgs q \neq \text{None} \} \)
  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: \( (2*N) \div 3 < \text{card} \ \{ q . \ (\text{rho} n q) = v \} \) (is twothird n)
  shows twothird (n+k)
proof (induct k)
  from n show twothird (n+0) by simp
next
  fix m
  assume m: twothird (n+m)
  have \( \forall q. \ \text{x} (\text{rho} (n+m) q) = v \longrightarrow \text{x} (\text{rho} (n + \text{Suc} m) q) = v \)
proof (rule+)
    fix q
    assume q: \( \text{x} (\text{rho} (n+m) q) = v \)
    let ?msgs = HORcvdMsgs OTR-M (n+m) q (HOS (n+m) q) (rho (n+m))
    show \( \text{x} (\text{rho} (n + \text{Suc} m) q) = v \)
proof (cases (2*N) div 3 < card \ \{ q . ?msgs q \neq \text{None} \})
    case True
    show...
from m have \((2\times N) \div 3 < \text{card } \{ q . x (\rho (n+m) q) = v \}\) 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
hence \(\text{card } \{ q . x (\rho (n+m) q) = v \} \leq \\text{card } \{ q . x (\rho (n + \text{Suc } m) q) = v \}\) by (auto intro: card-mono)
with m show ?twothird (\(n + \text{Suc } m\)) 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* \(\text{run: HORun OTR-M rho HO}\)
*and* \(\text{dec: decide (rho n p) = Some } v \) (is \(\text{?dec } n\))
*shows* \(\forall k. (2\times N) \div 3 < \text{card } \{ q . x (\rho (n+k) q) = v \}\)
(is \(\forall k. \text{?twothird } (n+k)\))

*using dec proof (induct n)*
— The base case is trivial since no process has decided

*assume* \(\text{?dec 0 with run show } \forall k. \text{?twothird } (0+k)\)
*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* \(ih: \text{?dec } m \implies \forall k. \text{?twothird } (m+k)\) and \(m: \text{?dec } (\text{Suc } m)\)
*show* \(\forall k. \text{?twothird } ((\text{Suc } m) + k)\)

*proof*
fix \(k\)
*have* \(\text{?twothird } (m + \text{Suc } k)\)

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 \(A1\) and \(A3\).

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

The Agreement property follows easily from lemma \(A4\): 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** \( \text{run: HORun OTR-M rho HO} \)
- **And** \( p: \text{decide (rho n p) = Some v} \)
- **And** \( p': \text{decide (rho (n+k) p') = Some w} \)
- **Shows** \( v = w \)

**Proof**

- From \( \text{run p} \)
  - Have \( (2*N) \mod 3 < \text{card } \{ q. \text{ x (rho (n+k) q) = v} \} \) (is - < card ?V)
    - By (blast dest: A4)
  - Moreover
    - From \( \text{run p'} \)
      - Have \( (2*N) \mod 3 < \text{card } \{ q. \text{ x (rho ((n+k)+0) q) = w} \} \) (is - < card ?W)
        - By (blast dest: A4)
  - Ultimately
    - Have \( N < \text{card } ?V + \text{card } ?W \) by auto
  - Then obtain proc where proc \( \in ?V \cap ?W \) by (auto dest: majorities-intersect)
  - Thus ?thesis by auto

**Theorem OTR-agreement:**

- **Assumes** \( \text{run: HORun OTR-M rho HO} \)
- **And** \( p: \text{decide (rho n p) = Some v} \)
- **And** \( p': \text{decide (rho m p') = Some w} \)
- **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 \( \text{run p p'} \) 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 \( \text{run p p'} \) 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** \( \text{run}: \text{HORun} \) \( \text{OTR-M} \) \( \text{rho} \) \( \text{HOs} \)
- **And** \( \text{commG}: \text{HOcommGlobal} \) \( \text{OTR-M} \) \( \text{HOs} \)
- **Shows** \( \exists r v. \text{decide} (\text{rho} r p) = \text{Some} v \)

**Proof** –
- **From** \( \text{commG} \) obtain \( r0 \) \( II \) where
  - \( \forall q. \text{HOs} r0 q = \text{II} \) and \( \text{pic: card} \ II > (2N) \) \( \text{div} \) \( 3 \)
- **By** (auto simp: \( \text{OTR-HOMachine-def} \) \( \text{OTR-commGlobal-def} \))
- **Let** \( \forall q. \text{HOrtvdMsgs} r0 q \)
- **From** \( \text{run} \) \( \text{pi} \) have \( \forall q. \text{HOrtvdMsgs} r0 q = \) \( \text{HOrcvdMsgs-def} \) \( \text{OTR-sendMsg-def} \)
  **Then obtain** \( \mu \) where \( \forall q. \text{HOrtvdMsgs} r0 q = \) \( \mu \)
  **Moreover**
- **From** \( \text{pi pic} \) have \( \forall q. (2N) \) \( \text{div} \) \( 3 < \) \( \text{card} \) \( \{ \text{HOrtvdMsgs} r0 q \} \) \( \neq \) \( \text{None} \)
  **By** (auto simp: \( \text{HORun-eq} \) \( \text{HOnextConfig-eq} \) \( \text{HOrcvdMsgs-def} \) \( \text{OTR-HOMachine-def} \) \( \text{nextState-def} \) \( \text{OTR-nextState-def} \))
- **Ultimately**
  **Have** \( \forall q. x (\text{rho} (\text{Suc} r0) q) = \text{Min} \{ v. \text{MFR} (\text{HOrcvdMsgs} r0 q) \} \)
  **Then obtain** \( v \) where \( \forall q. x (\text{rho} (\text{Suc} r0) q) = v \)

**Have** \( P: \forall k. \forall q. x (\text{rho} (\text{Suc} r0+k) q) = v \)
**Proof**
- **Fix** \( k \)
- **Show** \( \forall q. x (\text{rho} (\text{Suc} r0+k) q) = v \)
  **Proof** (induct \( k \))
  - **From** \( v \) **Show** \( \forall q. x (\text{rho} (\text{Suc} r0+0) q) = v \)
  - **Next**
    - **Fix** \( q \)
    - **Show** \( x (\text{rho} (\text{Suc} r0 + \text{Suc} k) q) = v \)
      **Proof** (cases \( (2N) \) \( \text{div} \) \( 3 < \) \( \text{card} \) \( \{ p. \text{HOrcvdMsgs} q (\text{Suc} r0 + k) p \neq \text{None} \} \))
        - **Case** \( \text{True} \)
          **Have** \( N > 0 \)
          **By** (rule finite-UNIV-card-ge-0) simp
          **With** \( \text{ih} \)
          **Have** \( \forall q. x (\text{rho} (\text{Suc} r0 + k) p) = v \)
          **By** auto
          **With** \( \text{True run show } ? \text{thesis by } (\text{auto elim: A2}) \)
        - **Next**
          - **Case** \( \text{False} \)
            **With** \( \text{run } \text{ih} \) **Show** \( ? \text{thesis by } (\text{auto dest: nextState-change}) \)
  - **Qed**

**Qed**
from commG obtain r0' Π'
  where r0': r0' ≥ Suc r0
  and p< : q. HOs r0' q = Π'
  and p<': card Π' > (2*N) div 3
  by (force simp: OTR-HOMachine-def OTR-commGlobal-def)
from r0' P have v' ∨ q. x (rho r0' q) = v by (auto simp: le-iff-add)

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 ◦ (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 (λ q. undefined)
  and commG: HOcommGlobal OTR-M HOs
  shows consensus (λ p. x (state (rho 0) p)) decide (state ◦ rho)
  (is consensus ?inits - -)
  proof (rule local-property-reduction[OF run consensus-is-local])
    fix crun
6 Verification of the Uniform Voting 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 \( \text{proc} \) 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

\( N \equiv \text{card} (\text{UNIV}:\text{Proc set}) \) — 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

\( nSteps \equiv 2 \)

definition phase where phase \((r::\text{nat})\) \(\equiv\ r \div nSteps\)

definition step where step \((r::\text{nat})\) \(\equiv\ r \mod nSteps\)

The following record models the local state of a process.

record \( \text{val \ pstate} = \)

\( x :: \text{\textquote{\text{val}}} \) — current value held by process

\( vote :: \text{\textquote{\text{val \ option}}} \) — value the process voted for, if any

\( decide :: \text{\textquote{\text{val \ option}}} \) — 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 ≡ ∃z v. m = ValVote z v
```

```
definition isVal where isVal m ≡ ∃v. m = 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 | getval (Val v) = z
```

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

```
definition UV-initState where UV-initState p st ≡ (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 msgRcvd where — processes from which some message was received msgRcvd (msgs::Proc ↦ 'val msg) = {q. msgs q ≠ None}
```

```
definition smallestValRcvd where smallestValRcvd (msgs::Proc ↦ ('val::linorder) msg) ≡ Min {v. ∃q. 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 send0 r p q st ≡ Val (x st)
```

```
definition next0 where next0 r p st (msgs::Proc ↦ ('val::linorder) msg) st' ≡ (∃v. (∀q ∈ msgRcvd msgs. msgs q = Some (Val v)) ∧ st' = st ( ⟨vote := Some v, x := smallestValRcvd msgs⟩)) ∨ (netv. ∀q ∈ msgRcvd msgs. msgs q = Some (Val v)) ∧ st' = st ( ⟨x := smallestValRcvd msgs⟩)
```

45
In step 1, each process sends its current $x$ and $vote$ values.

**definition send1 where**
\[send1\ r\ p\ q\ \mathit{st} \equiv ValVote\ (x\ \mathit{st})\ (vote\ \mathit{st})\]

**definition valVoteRcvd where**
— processes from which values and votes were received
\[\text{valVoteRcvd}\ (msgs ::\ \text{Proc} \rightarrow \text{'val.msg}) \equiv \{q . \exists v.\ msgs\ q = \text{Some}\ (ValVote\ v\ None)\}\]

**definition smallestValNoVoteRcvd where**
\[\text{smallestValNoVoteRcvd}\ (msgs ::\ \text{Proc} \rightarrow \text{'val.linorder})\ msg) \equiv \text{Min}\ \{v.\ \exists q.\ msgs\ q = \text{Some}\ (ValVote\ v\ None)\}\]

**definition someVoteRcvd where**
— set of processes from which some vote was received
\[\text{someVoteRcvd}\ (msgs ::\ \text{Proc} \rightarrow \text{'val.msg}) \equiv \{q . q \in \text{msgRcvd}\ \mathit{msgs} \land \text{isValVote}\ (the\ (msgs\ q)) \land \text{getvote}\ (the\ (msgs\ q)) \neq \text{None}\}\]

**definition identicalVoteRcvd where**
\[\text{identicalVoteRcvd}\ (msgs ::\ \text{Proc} \rightarrow \text{'val.msg})\ v \equiv \forall q \in \text{msgRcvd}\ \mathit{msgs}.\ \text{isValVote}\ (the\ (msgs\ q)) \land \text{getvote}\ (the\ (msgs\ q)) = \text{Some}\ v\]

**definition x-update where**
\[x\text{-update}\ \mathit{st}\ \mathit{msgs}\ \mathit{st}' \equiv (\exists q \in \text{someVoteRcvd}\ \mathit{msgs}.\ x\ \mathit{st}' = \text{the}\ (\text{getvote}\ (\text{the}\ (\mathit{msgs}\ q))))\]
\[\lor\ \text{someVoteRcvd}\ \mathit{msgs} = \{}\ \land\ x\ \mathit{st}' = \text{smallestValNoVoteRcvd}\ \mathit{msgs}\]

**definition dec-update where**
\[\text{dec-update}\ \mathit{st}\ \mathit{msgs}\ \mathit{st}' \equiv (\exists v.\ \text{identicalVoteRcvd}\ \mathit{msgs}\ v \land \text{decide}\ \mathit{st}' = \text{Some}\ v)\]
\[\lor\ \text{¬}(\exists v.\ \text{identicalVoteRcvd}\ \mathit{msgs}\ v) \land \text{decide}\ \mathit{st}' = \text{decide}\ \mathit{st}\]

**definition next1 where**
\[next1\ r\ p\ q\ \mathit{st} \equiv x\text{-update}\ \mathit{st}\ \mathit{msgs}\ \mathit{st}'\]
\[\land\ \text{dec-update}\ \mathit{st}\ \mathit{msgs}\ \mathit{st}'\]
\[\land\ \text{vote}\ \mathit{st}' = \text{None}\]

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

**definition UV-sendMsg where**
\[\text{UV-sendMsg}\ (r::\text{nat}) \equiv \text{if step}\ r = 0\ \text{then}\ \text{send0}\ r\ \text{else}\ \text{send1}\ r\]

**definition UV-nextState where**
\[\text{UV-nextState}\ r \equiv \text{if step}\ r = 0\ \text{then}\ \text{next0}\ r\ \text{else}\ \text{next1}\ r\]

46
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**

\[
UV-commPerRd HOrs \equiv \forall p q. \exists pq. \ pq \in HOrs p \cap HOrs q
\]

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**

\[
UV-commGlobal HOs \equiv \exists r. \ \forall p q. \ HOs r p = HOs r q
\]

6.3 The *UniformVoting* 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 *UniformVoting* is not a coordinated algorithm.

**definition UV-HOMachine where**

\[
UV-HOMachine = (\lambda \ CinitState = (\lambda p st crd. UV-initState p st),
sendMsg = UV-sendMsg,
C nextState = (\lambda r p st msgs crd st. UV-nextState r p st msgs st'),
HOcommPerRd = UV-commPerRd,
HOcommGlobal = UV-commGlobal
)
\]

**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 \( \exists \, pq. \, pq \in \text{msgRcvd} \left( \text{HOrcvdMsgs UV-M r p} \left( \text{HOs r p} \right) \left( \text{rho r} \right) \right) \)
\& \( \exists \, pq. \, pq \in \text{msgRcvd} \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \)
\& \( \left( \text{HOrcvdMsgs UV-M r p} \left( \text{HOs r p} \right) \left( \text{rho r} \right) \right) \)
\& \( \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \)
pq = \( \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \)
pq
using assms by (auto simp: UV-HOMachine-def UV-commPerRd-def HOrcvdMsgs-def UV-sendMsg-def send0-def send1-def msgRcvd-def)

When executing step 0, the minimum received value is always well defined.

lemma minval-step0:
assumes com: HOcommPerRd UV-M (HOs r) and s0: step r = 0
shows smallestValRcvd \( \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \)
\in \( \left\{ v. \, \exists \, p. \, \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \right\} p = \text{Some} \left( \text{Val} v \right) \)
(is smallestValRcvd ?msgs \in ?vals)
unfolding smallestValRcvd-def proof (rule Min-in)
have \( ?vals \subseteq \text{getval} \left( \left( \text{the o ?msgs} \right) \left( \text{HOs r q} \right) \right) \)
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 \( \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \) = \{\}
shows smallestValNoVoteRcvd \( \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \)
\in \( \left\{ v. \, \exists \, p. \, \left( \text{HOrcvdMsgs UV-M r q} \left( \text{HOs r q} \right) \left( \text{rho r} \right) \right) \right\} p = \text{Some} \left( \text{ValVote v None} \right) \)
(is smallestValNoVoteRcvd ?msgs \in ?vals)
unfolding smallestValNoVoteRcvd-def proof (rule Min-in)
have ?vals \subseteq getval \left( \left( \text{the o ?msgs} \right) \left( \text{HOs r q} \right) \right) \)
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 s1 nov show ?vals \neq \{\} by (auto simp: msgRcvd-def HOrcvdMsgs-def someVoteRcvd-def isValVote-def UV-HOMachine-def UV-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' = 0`
- **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 UV-M rho HOs`
  - and `com: ∀ 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'`
  - **Assume** `r: r = Suc r'`
  - Let `msgs = HOrcvdMsgs UV-M r' p (HOs r' p) (rho r')`
  - From `run` have `nxtState UV-M r' p (rho r' p) ?msgs (rho (Suc r') p)`
    - by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
  - With `vote r` have `nxt0: nxt0 r' p (rho r' p) ?msgs (rho r p) and s0: step r' = 0`
    - by (auto simp: nextState-def UV-HOMachine-def UV-nextState-def next1-def)
  - From `run s0` have `vote (rho r' p) = None` by (rule `reset-vote`)
  - With `vote nxt0`
    - Have `idv: ∀ q ∈ msgRcvd ?msgs. ?msgs q = Some (Val v)`
      - and `x: x (rho r p) = smallestValRcvd ?msgs`
      - by (auto simp: nxt0-def)
  - Moreover
    - From `com` obtain `q where q ∈ msgRcvd ?msgs`
by (force dest: some-common-msg)

with idv have \{x. \exists qq. ?msgs qq = 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 Irreversibility, Agreement and Integrity

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

lemma decide-step:
  assumes \( \text{run: HORun UV-M rho HOs} \)
  and \( \text{decide: decide (rho (Suc r) p) \neq decide (rho r p)} \)
  shows \( \text{step r = 1} \)
proof –
let \(?msgs = \text{HOrcvdMsgs UV-M r p (HOs r p) (rho r)}\)
from \(\text{assms have nextState UV-M r p (rho r p) ?msgs (rho (Suc r) p)}\)
  by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
with \(\text{decide: show ?thesis}\)
  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 \( \text{run: HORun UV-M rho HOs} \)
  and \( \text{decide: decide (rho (Suc r) p) \neq decide (rho r p)} \)
  shows \( \text{decide (rho (Suc r) p) \neq None} \)
proof –
let \(?msgs = \text{HOrcvdMsgs UV-M r p (HOs r p) (rho r)}\)
from \(\text{assms have s1: step r = 1 by (rule decide-step)}\)
with \(\text{run have nextState UV-M r p (rho r p) ?msgs (rho (Suc r) p)}\)
  by (auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq
    nextState-def UV-nextState-def)
with \(\text{decide: show ?thesis}\)
  by (auto simp: nextState-def UV-HOMachine-def
    UV-nextState-def next0-def step-def)
qed

If some process \( p \) votes for \( v \) at some round \( r \), then any message that \( p \)
rеceived in \( r \) was holding \( v \) as a value.

lemma msgs-unanimity:
  assumes \( \text{run: HORun UV-M rho HOs} \)
  and \( \text{vote: vote (rho (Suc r) p) = Some v} \)
  and \( \text{q: q \in msgRcvd (HOrcvdMsgs UV-M r p (HOs r p) (rho r))} \)
  (is - \in msgRcvd \ ?msgs)
  shows \( \text{getval (the (?msgs q)) = v} \)

50
proof
have $s_0$: step $r = 0$
proof (rule ccontr)
assume step $r \neq 0$
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 $s_0$ have nxt: next0 $r$ $p$ ($rho$ $r$ $p$) ?msgs ($rho$ (Suc $r$) $p$)
  by (auto simp: UV-HOMachine-def 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 $rho$ $HO$s
  and com: $\forall r. HOcommPerRd UV-M (HO$s $r$)
  and $p$: vote ($rho$ $r$ $p$) = Some $v$
  and $q$: vote ($rho$ $r$ $q$) = 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)
next
fix $r'$
assume $r$: $r = Suc r'$
let ?msgs $p$ = HORcvdMsgs UV-M $r'$ $p$ ($HO$s $r'$ $p$) ($rho$ $r'$)
from com obtain $pq$
  where ?msgs $p$ $pq$ = ?msgs $q$ $pq$
    and $sm$p: $pq$ $\in$ msgRcvd (?msgs $p$) and $sm$q: $pq$ $\in$ msgRcvd (?msgs $q$)
  by (force dest: some-common-msg)
moreover
from run $p$ $smp$ $r$ have getval (the (?msgs $p$ $pq$)) = $v$
  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 $rho$ $HO$s
  and com: $\forall r. HOcommPerRd UV-M (HO$s $r$)
and decide: decide (\(\rho (\text{Suc } r) \ p\)) \neq \text{decide} (\(\rho \ r\ p\))
and decval: decide (\(\rho (\text{Suc } r) \ p\)) = Some \(v\)
shows \(x (\rho (\text{Suc } r) \ q) = v\)

proof —
let \(?msgs p' = \text{HOrcvdMsgs UV-M } r \ p' (\text{HOs } r \ p') (\rho r)\)
from run decide have \(s_1: \text{step } r = 1\) by (rule decide-step)
from run have \(\text{nextState UV-M } r \ p\) (\(?msgs\) \((\rho (\text{Suc } r) \ p)\)) by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} \text{nextState-def})
with \(s_1\) have \(\text{next1 } r \ p\) (\(?msgs\) \((\rho (\text{Suc } r) \ p)\)) by (auto simp: \text{UV-HOMachine-def} \text{nextState-def} \text{UV-nextState-def})
from \text{run} have \(\text{nextState UV-M } r \ q\) (\(?msgs\) \((\rho (\text{Suc } r) \ q)\)) by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} \text{nextState-def})
with \(s_1\) have \(\text{next1 } q\) (\(?msgs\) \((\rho (\text{Suc } r) \ q)\)) (\(?msgs\) \((\rho (\text{Suc } r) \ q)\)) by (auto simp: \text{UV-HOMachine-def} \text{nextState-def} \text{UV-nextState-def})

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: \(\text{HORun UV-M }\rho \text{ HOs}\)
and \(\text{comm: } \forall r. \text{HOcommPerRd UV-M } (\text{HOs } r)\)
and \(\text{x: } \forall p. x (\rho r p) = v\)
shows \(x (\rho (\text{Suc } r) \ q) = v\)

proof —
let \(?msgs = \text{HOrcvdMsgs UV-M } r\ q\ (\text{HOs } r \ q) (\rho r)\)
from \text{comm obtain } \(p\) where \(p: p \in \text{msgRecvd } ?msgs\)
by (force dest: some-common-msg)
from run have \(\text{nextState UV-M } r \ q\) (\(?msgs\) \((\rho (\text{Suc } r) \ q)\)) by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} \text{nextState-def})
hence \(?msgs\) \((\rho (\text{Suc } r) \ q) \land \text{step } r = 0\)
\(\lor \text{next1 } r \ q\) (\(?msgs\) \((\rho (\text{Suc } r) \ q) \land \text{step } r \neq 0\))
is \(?\text{next0} \lor \text{next1}\)
by (auto simp: \text{UV-HOMachine-def} \text{nextState-def} \text{UV-nextState-def})
thus \( ?\text{thesis} \)

proof

assume \( \text{nxt0} : \text{nxt0} \)

hence \( x (\rho (\text{Suc } r) q) = \text{smallestValRcvd } ?\text{msgs} \)

by (auto simp: \text{nxt0-def})

moreover

from \( \text{nxt0 } x \) have \( \forall p \in \text{msgRcvd } ?\text{msgs} . \ ?\text{msgs } p = \text{Some } (\text{Val } v) \)

by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def} \text{msgRcvd-def} \text{send0-def})

from this \( p \) have \( \{ x . \exists p . \ ?\text{msgs } p = \text{Some } (\text{Val } x) \} = \{ v \} \)

by (auto simp: \text{msgRcvd-def})

hence \( \text{smallestValRcvd } ?\text{msgs} = v \)

by (auto simp: \text{smallestValRcvd-def})

ultimately

show \( ?\text{thesis} \) by simp

next

assume \( \text{nxt1} : \text{nxt1} \)

show \( ?\text{thesis} \)

proof (cases someVoteRcvd \( ?\text{msgs} = \{} \) )

case True

with \( \text{nxt1 } x \text{ True} \) have \( x (\rho (\text{Suc } r) q) = \text{smallestValNoVoteRcvd } ?\text{msgs} \)

by (auto simp: \text{nxt1-def} \text{x-update-def})

moreover

from \( \text{nxt1 } x \text{ True} \) have \( \forall p \in \text{msgRcvd } ?\text{msgs} . \ ?\text{msgs } p = \text{Some } (\text{ValVote } v \text{ None}) \)

by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def} \text{msgRcvd-def} \text{send1-def} \text{someVoteRcvd-def} \text{isValVote-def})

from this \( p \) have \( \{ x . \exists p . \ ?\text{msgs } p = \text{Some } (\text{ValVote } x \text{ None}) \} = \{ v \} \)

by (auto simp: \text{msgRcvd-def})

hence \( \text{smallestValNoVoteRcvd } ?\text{msgs} = v \)

by (auto simp: \text{smallestValNoVoteRcvd-def})

ultimately show \( ?\text{thesis} \) by simp

next

case False

with \( \text{nxt1 } \text{ obtain } p' v' \text{ where} \)

\( p' : \ ?\text{msgs isValVote } (\text{the } (?\text{msgs } p')) \)

getvote (the (\( ?\text{msgs } p' \)) = Some \( v' \) (\( \rho (\text{Suc } r) q) = v' \))

by (auto simp: \text{someVoteRcvd-def} \text{nxt1-def} \text{x-update-def})

with \( \text{nxt1 } \text{ have } x (\rho (\text{Suc } r) q) = x (\rho r p') \)

by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def} \text{msgRcvd-def} \text{send1-def} \text{isValVote-def} \text{x-vote-eq[OF run comm]})

with \( x \) show \( ?\text{thesis} \) by auto

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 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 \( r+k \)) \( q \)) = \( v \)
proof (induct \( k \) arbitrary: \( q \))
  fix \( q \)
  from decide-equals-x[OF assms] show \( x \) (\( \rho \) (Suc \( r+0 \)) \( q \)) = \( v \) by simp
next
  fix \( k \) \( q \)
  assume \( \forall q. \ x \) (\( \rho \) (Suc \( r+k \)) \( q \)) = \( v \)
  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 \( \exists m<n. \) decide (\( \rho \) (Suc \( m \)) \( p \)) \( \neq \) decide (\( \rho \) \( m \) \( p \))
\( \land \) decide (\( \rho \) (Suc \( m \)) \( p \)) = Some \( v \)
proof
  let \( \text{dec} \ k = \text{decide} \ (\rho \ k \ p) \)
  have \( \forall m<n. \ \text{dec} \ (\text{Suc} \ m) \neq \text{dec} \ (\text{Suc} \ m) \neq \text{Some} \ v \)
    \( \rightarrow \) \( \text{dec} \ n \neq \text{Some} \ v \)
    \( \rightarrow \text{dec} \ (\text{Suc} \ n) \neq \text{Some} \ v \)
    (is \( \text{P} \) \( n \) is \( \text{A} \) \( n \) \( \rightarrow \cdot \))
proof (induct \( n \))
  from run show \( \text{P} \ 0 \)
    by (auto simp: HORun-eq UV-HOMachine-def HOinitConfig-eq
     initState-def UV-initState-def)
next
  fix \( n \)
  assume ih: \( \text{P} \) \( n \) thus \( \text{P} \) (Suc \( n \)) by force
qed
with \( \text{dec} \) show \( \text{thesis} \) 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: \( \forall r. \ HOcommPerRd \ UV-M \ (HOs \ r) \)
shows \( \exists q. \ x \) (\( \rho \) \( r \) \( p \)) = \( x \) (\( \rho \) \( 0 \) \( q \))
proof (induct \( r \) arbitrary: \( p \))
  fix \( p \)
  show \( \exists q. \ x \) (\( \rho \) \( 0 \) \( p \)) = \( x \) (\( \rho \) \( 0 \) \( q \)) by auto
next

54
fix r p
assume ih: \( \forall p'. \exists q. x (\rho r p') = x (\rho 0 q) \)
let \( run \) have nextState UV-M r p (rho 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) \( \land \) step r = 0
  \lor next1 r p (rho r p) ?msgs (rho (Suc r) p) \( \land \) step r \( \neq \) 0
    (is \( ?nxt0 \lor ?nxt1 \))
  by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
thus \( \exists q. x (\rho (Suc r) p) = x (\rho 0 q) \)
proof
  assume \( ?nxt0 \)
  hence \( x (\rho (Suc r) p) = \) smallestValRcvd ?msgs
    by (auto simp: next0-def)
also with \( com \) \( ?nxt1 \) True have ...
    \( \in \{ v . \exists q. \) ?msgs q = Some (Val v)\}
    by (intro minval-step1) auto
also with \( ?nxt1 \) True have ...
    \( \in \{ x (\rho r q) | q . q \in msgRcvd ?msgs \} \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
      msgRcvd-def send0-def)
finally obtain \( q \) where \( x (\rho (Suc r) p) = x (\rho r q) \) 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 ...
    \( \in \{ v . \exists q. \) ?msgs q = Some (ValVote v None)\}
    by (intro minval-step1) auto
  also with \( ?nxt1 \) True have ...
    \( = \{ x (\rho r q) | q . q \in msgRcvd ?msgs \} \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
      someVoteRcvd-def isValVote-def msgRcvd-def send1-def)
finally obtain \( q \) where \( x (\rho (Suc r) p) = x (\rho r q) \) 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 isValVote-def msgRcvd-def send1-def)
with \( run \) have x (rho (Suc r) p) = x (rho r q)
  by (rule x-vote-eq)
with \( ih \) show \(?thesis\) by auto

theorem uv-integrity:
assumes run: HORun UV-M rho HOs
and com: \( \forall r. \) HOcommPerRd UV-M (HOs r)
and dec: decide (rho r p) = Some v
shows \( \exists q. v = x (rho 0 q) \)

proof –
from run dec obtain k where
decide (rho (Suc k) p) \( \neq \) 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 decdep: decide (rho (Suc r) p) \( \neq \) decide (rho r p)
and decvalp: decide (rho (Suc r) p) = Some v
and decdepq: decide (rho (Suc (r+k)) q) \( \neq \) decide (rho (r+k) q)
and decvalq: decide (rho (Suc (r+k)) q) = Some w
shows v = w

proof –
from run com decdep decvalp have x (rho (Suc r+k) q) = v
by (rule safety-argument)
moreover
from run com decdepq decvalq have x (rho (Suc (r+k)) q) = w
by (rule decide-equals-x)
ultimately
show ?thesis by simp
qed

theorem uv-agreement:
assumes run: HORun UV-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) ≠ decide (rho l q)
decide (rho (Suc l) q) = Some w
by (auto dest: decided-then-past-decision)

show ?thesis

proof (cases k ≤ 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 ≤ 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 UV-M rho HOs
    and com: ∀ 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 ¬ ?thesis
    with run ih obtain w where w: decide (rho (m + Suc n) p) = Some w
    by (auto dest!: decidenonnull)
    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-zeq:
  assumes run: HORun UV-M rho HOs
    and com: ∀ r. HOcommPerRd UV-M (HOs r)
    and hoeq: HOs r p = HOs r q
shows \( x \ (\rho \ (\text{Suc} \ r) \ p) = x \ (\rho \ (\text{Suc} \ r) \ q) \)

proof

let \(?msgs\ p = \text{HOrcvdMsgs} \text{ UV-M} \ r \ p \ (\text{HOs} \ r \ p) \ (\rho \ r)\)

from.hoq. have msgeq: \(?msgs \ p = \?msgs \ q\)
  by (auto simp: \text{UV-HOMachine-def} \text{ HOrcvdMsgs-def} \text{ UV-sendMsg-def} \text{ send0-def} \text{ send1-def})

show \(?thesis\)

proof (cases step \(r = 0\))

  case True
  with run
  have \(\forall p. \text{next0} \ r \ p \ (\rho \ r \ p) \ (\?msgs \ p) \ (\rho \ (\text{Suc} \ r) \ p) \ (\text{is} \ p) \ ?\text{nxt0} \ p\)
    by (force simp: \text{UV-HOMachine-def} \text{ HORun-eq} \text{ HOnextConfig-eq} \text{ nextState-def} \text{ UV-nextState-def})

  hence \(?\text{nxt0} \ p \ ?\text{nxt0} \ q\) by auto

  with msgeq show \(?thesis\) by (auto simp: \text{next0-def})

next

assume \(stp: \text{step} \ r \neq 0\)

with run

have \(\forall p. \text{next1} \ r \ p \ (\rho \ r \ p) \ (\?msgs \ p) \ (\rho \ (\text{Suc} \ r) \ p) \ (\text{is} \ p) \ ?\text{nxt1} \ p\)
  by (force simp: \text{UV-HOMachine-def} \text{ HORun-eq} \text{ HOnextConfig-eq} \text{ nextState-def} \text{ UV-nextState-def})

hence \(x\text{-update} \ (\rho \ r \ p) \ (\?msgs \ p) \ (\rho \ (\text{Suc} \ r) \ p)\)

\(x\text{-update} \ (\rho \ r \ q) \ (\?msgs \ q) \ (\rho \ (\text{Suc} \ r) \ q)\)

by (auto simp: \text{next1-def})

with msgeq have

\(x' : x\text{-update} \ (\rho \ r \ p) \ (\?msgs \ p) \ (\rho \ (\text{Suc} \ r) \ p)\)

\(x\text{-update} \ (\rho \ r \ q) \ (\?msgs \ p) \ (\rho \ (\text{Suc} \ r) \ q)\)

by auto

show \(?thesis\)

proof (cases \text{someVoteRcvd} \ (?msgs \ p) = \{\})

  case True
  with \(x'\) show \(?thesis\)
    by (auto simp: \text{x-update-def})

next

case False

with \(x'\) \text{stp} obtain \(qp \ qq\) where

vote \((\rho \ r \ qp)\) = Some \((x \ (\rho \ (\text{Suc} \ r) \ p)\)\) and
vote \((\rho \ r \ qq)\) = Some \((x \ (\rho \ (\text{Suc} \ r) \ q)\)\)

by (force simp: \text{UV-HOMachine-def} \text{ HOrcvdMsgs-def} \text{ UV-sendMsg-def} \text{ x-update-def} \text{ someVoteRcvd-def} \text{ isValVote-def} \text{ msgRcvd-def} \text{ send1-def})

with run com show \(?thesis\) by (rule \text{vote-agreement})

qed

qed

We now prove that \text{UniformVoting} terminates.

theorem \text{uv-termination}:
assumes run: HORun UV-M \rho HO\_s
and commR: \forall r. HO\_comm\_Per\_Rd UV-M (HO\_s r)
and commG: HO\_comm\_Global UV-M HO\_s
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 r0: \forall q. HO\_s r0 q = HO\_s r0 p
  by (force simp: UV-HOMachine-def UV-commGlobal-def)
let \?v = x (\rho (Suc r0) p)
from run commR r0 have xs: \forall q. x (\rho (Suc r0) q) = ?v
  by (auto dest: hoeq-xeq)

Now obtain a round where all votes agree.

def r' \equiv if step (Suc r0) = 0 then Suc r0 else Suc (Suc r0)
have stp': step r' = 0
  by (simp add: r'\_def step-def mod-Suc)
have x': \forall q. x (\rho r' q) = ?v
proof (auto simp: r'\_def)
  fix q
  from xs show x (\rho (Suc r0) q) = ?v ..
next
  fix q
  from run commR xs show x (\rho (Suc (Suc r0)) q) = ?v
    by (rule same-x-stable)
qed

have vote': \forall q. vote (\rho (Suc r') q) = Some ?v
proof
  fix q
  let ?msgs = HO\_rcvd\_msgs UV-M r' q (HO\_s r' q) (\rho r')
from run stp' have next0 r' q (\rho r' q) ?msgs (\rho (Suc r') q)
    by (force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq
      nextState-def UV-nextState-def)
moreover
from stp' x' have \forall q' \in msgRcvd ?msgs. ?msgs q' = 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 \?msgs' = HO\_rcvd\_msgs UV-M \?r'' p (HO\_s \?r'' p) (\rho \?r'')
from stp' have stp'': step \?r'' = 1
by \((\text{simp add: step-def mod-Suc})\)

with \text{run have next1 \(\rho''\)} \(\rho\) \(\rho''\) \(\text{msgs}'\) \(\rho\) \(\text{Suc} \(\rho''\)) \(\text{p}\)
by \((\text{auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def UV-nextState-def})\)

moreover
from \text{stp'' vote' have identicalVoteRcvd \(\text{msgs}'\) \(v\)}
by \((\text{auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def send1-def identicalVoteRcvd-def isValVote-def})\)

moreover
from \text{commR have msgRcvd \(\text{msgs}'\) \(\neq\)} \{\}
by \((\text{force dest: some-common-msg})\)
ultimately
have \(\text{decide} (\rho\) (\text{Suc} \(\rho''\)) \(\text{p}\) = \text{Some} \(\text{v}\)
by \((\text{force simp: next1-def dec-update-def identicalVoteRcvd-def msgRcvd-def isValVote-def})\)

thus \(\text{thesis by blast}\)

qed

6.7 \textbf{UniformVoting Solves Consensus}

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

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

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

\textbf{theorem} \text{uv-consensus-fg}:
\textbf{assumes} \text{run: fg-run UV-M \(\rho\) HOs HOs (\(\lambda r q. \text{undefined}\) )}
and \text{commR: \(\forall\) \(r\). HOcommPerRd UV-M (HOs \(r\))}
and \text{commG: HOcommGlobal UV-M HOs}
\textbf{shows} \text{consensus} (\(\lambda p x (\text{state} (\rho 0) \text{p})\)) \text{decide} (\text{state} \circ \rho\)
(is consensus ?inits - -)
\textbf{proof} \((\text{rule local-property-reduction[OF run consensus-is-local]})\)

fix \text{crun}
\textbf{assume} \text{crun: CSHORun UV-M crun HOs HOs (\(\lambda r q. \text{undefined}\) )}
and \text{init: crun 0 = state (\rho 0)\)
from \text{crun have HORun UV-M crun HOs}
by \((\text{unfold HORun-def SHORun-def})\)
from this \text{commR commG have consensus} (\(x \circ (\text{crun 0})\)) \text{decide crun}
by \((\text{rule uv-consensus})\)
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 ≡ card (UNIV::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) ≡ r div 4
definition step where step (r::nat) ≡ r mod 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
  commit :: 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
  coordΦ :: 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.

```
declaration isValStamp where isValStamp m ≡ ∃v ts. m = ValStamp v ts

declaration isVote where isVote m ≡ ∃v. m = Vote v

declaration isAck where isAck m ≡ m = Ack
```

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.

```
declaration LV-initState where
  LV-initState p st crd ≡
  vote st = None
  ∧ ¬(commit st)
  ∧ ¬(ready st)
  ∧ timestamp st = 0
  ∧ decide st = None
  ∧ coordΦ 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.

— processes from which values and timestamps were received
definition `valStampsRcvd` where
\[
valStampsRcvd (msgs :: Proc \rightarrow 'val msg) \equiv \\
\{ q . \exists v ts. msgs q = Some (ValStamp v ts)\}
\]

definition `highestStampRcvd` where
\[
highestStampRcvd msgs \equiv \\
\text{Max}\{ ts . \exists q v. (msgs :: Proc \rightarrow 'val msg) q = Some (ValStamp v ts)\}
\]

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 \(commit\) field to true.

definition `send0` where
\[
send0 r p q st \equiv \\
\text{if } q = \text{coord}\Phi st \text{ then } \text{ValStamp}(x\ st)(\text{timestamp}\ st) \text{ else Null}
\]

definition `next0` where
\[
next0 r p st msgs crd st' \equiv \\
\text{if } p = \text{coord}\Phi st \land \text{commit}\ st \text{ then } \text{Vote}(the\ (\text{vote}\ st)) \text{ else Null}
\]

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
\[
send1 r p q st \equiv \\
\text{if } p = \text{coord}\Phi st \land \text{commit}\ st \text{ then } \text{Vote}(the\ (\text{vote}\ st)) \text{ else Null}
\]

definition `next1` where
\[
next1 r p st msgs crd st' \equiv \\
\text{if } msgs\ (\text{coord}\Phi st) \neq \text{None} \land \text{isVote}\ (the\ (msgs\ (\text{coord}\Phi st))) \text{ then } st' = st (\{ x := \text{val}\ (the\ (msgs\ (\text{coord}\Phi st))), \text{timestamp} := \text{Suc}(phase\ r) \}) \text{ else } st' = st
\]

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
\[
send2 r p q st \equiv \\
\text{if } \text{timestamp}\ st = \text{Suc}(phase\ r) \land q = \text{coord}\Phi st \text{ then } \text{Ack} \text{ else Null}
\]

— processes from which an acknowledgement was received

definition `acksRcvd` where
acksRcvd \( (\text{msgs} :: \text{Proc} \rightarrow \text{val msg}) \equiv \{ q . \text{msgs} q \neq \text{None} \land \text{isAck} (\text{the} (\text{msgs} q)) \}\)

**definition** next2 where

\[
\text{next2} r p s t \text{msgs crd} s t' \equiv \\
\text{if } p = \text{coord} \Phi s t \land \text{card} (\text{acksRcvd} \text{msgs}) > N \text{ div } 2 \\
\text{then } s t' = s (\{ \text{ready} := \text{True} \}) \\
\text{else } s t' = s
\]

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 \text{ready} and \text{comm} fields to false. All processes reset the coordinators as indicated by the parameter of the operator.

**definition** send3 where

\[
\text{send3} r p q s t \equiv \\
\text{if } p = \text{coord} \Phi s t \land \text{ready} st \text{ then } \text{Vote} (\text{the} (\text{vote} st)) \text{ else Null}
\]

**definition** next3 where

\[
\text{next3} r p s t \text{msgs crd} s t' \equiv \\
\text{(if } \text{msgs} (\text{coord} \Phi s t) \neq \text{None} \land \text{isVote} (\text{the} (\text{msgs} (\text{coord} \Phi s t))) \\
\text{then } \text{decide} s t' = \text{decide} s) \\
\text{else } \text{decide} s t' = \text{decide} s \\
\land (\text{if } p = \text{coord} \Phi s t \\
\text{then } \lnot (\text{ready} s t') \land \lnot (\text{commt} s t') \\
\text{else } \text{ready} s t' = \text{ready} st \land \text{commt} s t' = \text{commt} st) \\
\land x s t' = x st \\
\land \text{vote} s t' = \text{vote} st \\
\land \text{timestamp} s t' = \text{timestamp} st \\
\land \text{coord} \Phi s t' = \text{crd}
\]

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

\[
\text{LV-sendMsg} (r::\text{nat}) \equiv \\
\text{if } \text{step} r = 0 \text{ then } \text{send0} r \\
\text{else if } \text{step} r = 1 \text{ then } \text{send1} r \\
\text{else if } \text{step} r = 2 \text{ then } \text{send2} r \\
\text{else } \text{send3} r
\]

**definition** LV-nextState :: nat \(\rightarrow\) Proc \(\rightarrow\)'val pstate \(\rightarrow\)(Proc \(\rightarrow\)'val msg) \(\rightarrow\) Proc \(\rightarrow\)'val pstate \(\rightarrow\) bool

\[
\text{where} \\
\text{LV-nextState} r \equiv \\
\text{if } \text{step} r = 0 \text{ then } \text{next0} r \\
\text{else if } \text{step} r = 1 \text{ then } \text{next1} r \\
\text{else if } \text{step} r = 2 \text{ then } \text{next2} r \\
\text{else } \text{next3} r
\]

64
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 $ph$ 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 $ph$, 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**

$LV\text{-commPerRd}$ where

$LV\text{-commPerRd} \ r \ (HO::Proc \ HO) \ (coord::Proc \ coord) \equiv True$

**definition**

$LV\text{-commGlobal}$ where

$LV\text{-commGlobal} \ HOs \ coords \equiv$

$\exists ph::nat. \exists c::Proc.$

$(\forall p. \ coords \ (4*ph) \ p = c)$

$\land \ \text{card} \ (\text{HOs} \ (4*ph) \ c) > N \ \text{div} \ 2$

$\land \ \text{card} \ (\text{HOs} \ (4*ph+2) \ c) > N \ \text{div} \ 2$

$\land (\forall p. \ c \in \text{HOs} \ (4*ph+1) \ p \cap \text{HOs} \ (4*ph+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** $LV\text{-CHO\text{Machine}}$ where

$LV\text{-CHO\text{Machine}} \equiv$

$(\mid \ \text{CinitState} = LV\text{-initState},$

$\text{sendMsg} = LV\text{-sendMsg},$

$\text{CnextState} = LV\text{-nextState},$

$CHO\text{commPerRd} = LV\text{-commPerRd},$

$CHO\text{commGlobal} = LV\text{-commGlobal} \ )$

**abbreviation**

$LV\text{-M} \equiv (LV\text{-CHO\text{Machine}}::(\text{Proc, } \text{val pstate, } \text{val msg}) \ \text{CHO\text{Machine}})$

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 . ∃ q v. (msgs::Proc → '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 ≠ {}
  obtains p v where msgs p = Some (ValStamp v (highestStampRcvd msgs))
proof –
  let ?ts = {ts . ∃ q v. msgs q = Some (ValStamp v ts)}
  from nempty have ?ts ≠ {} by (auto simp add: valStampsRcvd-def)
  with timeStampsRcvdFinite have highestStampRcvd msgs ∈ ?ts
  unfolding highestStampRcvd-def by (rule Max-in)
  then obtain p v 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 ≤ highestStampRcvd 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 LV-induct:
  assumes run: CHORun LV-M rho HOs coords
  and init: ∀ p. CinitState LV-M p (rho 0 p) (coords 0 p) ⇒ P 0
and step0: \begin{align*}
\forall r. \\
&\quad \text{step } r = 0; \ P r; \ \text{phase } (\text{Suc } r) = \text{phase } r; \ \text{step } (\text{Suc } r) = 1; \\
&\quad \forall p. \ \text{next0 } r \ p (\rho r p) \\
&\qquad (\text{HOrcvdMsgs } \text{LV-M } r \ p (\text{HOS } r \ p) (\rho r)) \\
&\qquad (\text{coords } (\text{Suc } r) \ p) \\
&\qquad (\rho (\text{Suc } r \ p)) \\
&\implies P (\text{Suc } r)
\end{align*}

and step1: \begin{align*}
\forall r. \\
&\quad \text{step } r = 1; \ P r; \ \text{phase } (\text{Suc } r) = \text{phase } r; \ \text{step } (\text{Suc } r) = 2; \\
&\quad \forall p. \ \text{next1 } r \ p (\rho r p) \\
&\qquad (\text{HOrcvdMsgs } \text{LV-M } r \ p (\text{HOS } r \ p) (\rho r)) \\
&\qquad (\text{coords } (\text{Suc } r) \ p) \\
&\qquad (\rho (\text{Suc } r \ p)) \\
&\implies P (\text{Suc } r)
\end{align*}

and step2: \begin{align*}
\forall r. \\
&\quad \text{step } r = 2; \ P r; \ \text{phase } (\text{Suc } r) = \text{phase } r; \ \text{step } (\text{Suc } r) = 3; \\
&\quad \forall p. \ \text{next2 } r \ p (\rho r p) \\
&\qquad (\text{HOrcvdMsgs } \text{LV-M } r \ p (\text{HOS } r \ p) (\rho r)) \\
&\qquad (\text{coords } (\text{Suc } r) \ p) \\
&\qquad (\rho (\text{Suc } r \ p)) \\
&\implies P (\text{Suc } r)
\end{align*}

and step3: \begin{align*}
\forall r. \\
&\quad \text{step } r = 3; \ P r; \ \text{phase } (\text{Suc } r) = \text{Suc } (\text{phase } r); \ \text{step } (\text{Suc } r) = 0; \\
&\quad \forall p. \ \text{next3 } r \ p (\rho r p) \\
&\qquad (\text{HOrcvdMsgs } \text{LV-M } r \ p (\text{HOS } r \ p) (\rho r)) \\
&\qquad (\text{coords } (\text{Suc } r) \ p) \\
&\qquad (\rho (\text{Suc } r \ p)) \\
&\implies P (\text{Suc } r)
\end{align*}

shows \( P n \)

proof (rule CHORun-induct[OF run])
assume CHOinitConfig LV-M \((\rho 0) \ (\text{coords } 0)\)
thus \( P 0 \) by (auto simp add: CHOinitConfig-def init)

next
fix \( r \)
assume ih: \( P r \)
and nzt: CHOnextConfig LV-M \((\rho r) \ (\text{HOS } r)\)
\((\text{coords } (\text{Suc } r) \ (\rho (\text{Suc } r)))\)

have \( \text{step } r \in \{0,1,2,3\} \) by (auto simp add: step-def)
thus \( P (\text{Suc } r) \)
proof auto
assume stp: \( \text{step } r = 0 \)

hence \( \text{step } (\text{Suc } r) = 1 \)
by (auto simp add: step-def mod-Suc)
with ih nzt stp show \( \text{?thesis} \)
by (intro step0)
  (auto simp: LV-CHOMachine-def CHOnextConfig-eq
   LV-nextState-def LV-sendMsg-def phase-Suc)

next
assume stp: \( \text{step } r = \text{Suc } 0 \)
hence \( \text{step} (\text{Suc} \ r) = 2 \)
  by (auto simp add: step-def mod-Suc)
with \( \text{ih} \ \text{nxt} \ \text{stp} \) show \( \text{\#thesis} \)
  by (intro step1)
    (auto simp: LV-CHOMachine-def \ CHOnextConfig-eq
     LV-nextState-def LV-sendMsg-def phase-Suc)

next
assume \( \text{stp} \): \( \text{step} \ r = 2 \)
hence \( \text{step} (\text{Suc} \ r) = 3 \)
  by (auto simp add: step-def mod-Suc)
with \( \text{ih} \ \text{nxt} \ \text{stp} \) show \( \text{\#thesis} \)
  by (intro step2)
    (auto simp: LV-CHOMachine-def \ CHOnextConfig-eq
     LV-nextState-def LV-sendMsg-def phase-Suc)

next
assume \( \text{stp} \): \( \text{step} \ r = 3 \)
hence \( \text{step} (\text{Suc} \ r) = 0 \)
  by (auto simp add: step-def mod-Suc)
with \( \text{ih} \ \text{nxt} \ \text{stp} \) show \( \text{\#thesis} \)
  by (intro step3)
    (auto simp: LV-CHOMachine-def \ CHOnextConfig-eq
     LV-nextState-def LV-sendMsg-def phase-Suc)

qed

The following rule similarly establishes a property of two successive configurations of a run by case distinction on the step that was executed.

lemma \( \text{LV-Suc} \):
  assumes \( \text{run} \): \( \text{CHORun} \ \text{LV-M} \ \text{rho} \ \text{HOs coords} \)
  and \( \text{step0} \): \[ \begin{align*}
    & \text{step} \ r = 0; \ 
    & \text{step} (\text{Suc} \ r) = 1; \ 
    & \text{phase} (\text{Suc} \ r) = \text{phase} \ r; \\
    & \forall \ p. \ \text{next0} \ r \ p \ (\text{rho} \ r \ p) \\
    & \ (\text{HOrcvdMsgs} \ \text{LV-M} \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r)) \\
    & \ (\text{coords} \ (\text{Suc} \ r) \ p \ (\text{rho} \ (\text{Suc} \ r) \ p)) \\
\end{align*}\]
  \( \Rightarrow \) \( \text{P} \ r \)
  and \( \text{step1} \): \[ \begin{align*}
    & \text{step} \ r = 1; \ 
    & \text{step} (\text{Suc} \ r) = 2; \ 
    & \text{phase} (\text{Suc} \ r) = \text{phase} \ r; \\
    & \forall \ p. \ \text{next1} \ r \ p \ (\text{rho} \ r \ p) \\
    & \ (\text{HOrcvdMsgs} \ \text{LV-M} \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r)) \\
    & \ (\text{coords} \ (\text{Suc} \ r) \ p \ (\text{rho} \ (\text{Suc} \ r) \ p)) \\
\end{align*}\]
  \( \Rightarrow \) \( \text{P} \ r \)
  and \( \text{step2} \): \[ \begin{align*}
    & \text{step} \ r = 2; \ 
    & \text{step} (\text{Suc} \ r) = 3; \ 
    & \text{phase} (\text{Suc} \ r) = \text{phase} \ r; \\
    & \forall \ p. \ \text{next2} \ r \ p \ (\text{rho} \ r \ p) \\
    & \ (\text{HOrcvdMsgs} \ \text{LV-M} \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r)) \\
    & \ (\text{coords} \ (\text{Suc} \ r) \ p \ (\text{rho} \ (\text{Suc} \ r) \ p)) \\
\end{align*}\]
  \( \Rightarrow \) \( \text{P} \ r \)
  and \( \text{step3} \): \[ \begin{align*}
    & \text{step} \ r = 3; \ 
    & \text{step} (\text{Suc} \ r) = 0; \ 
    & \text{phase} (\text{Suc} \ r) = \text{Suc} \ (\text{phase} \ r); \\
    & \forall \ p. \ \text{next3} \ r \ p \ (\text{rho} \ r \ p) \\
    & \ (\text{HOrcvdMsgs} \ \text{LV-M} \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r)) \\
    & \ (\text{coords} \ (\text{Suc} \ r) \ p \ (\text{rho} \ (\text{Suc} \ r) \ p)) \\
\end{align*}\]
  \( \Rightarrow \) \( \text{P} \ r \)
shows \( P \ r \)

\[ \begin{align*}
\text{proof -} \\
\text{from } \text{run} \\
\text{have } \text{nxt: CHOnextConfig LV-M } r \ (\rho r) \ (Hos r) \ (coords \ (Suc r)) \ (rho \ (Suc r)) \\
\text{by } (auto \ simp \ add: CHORun-eq) \\
\text{have } \text{step } r \in \{0,1,2,3\} \text{ by } (auto \ simp \ add: \text{step-def}) \\
\text{thus } P \ r \\
\end{align*} \]

\[ \begin{align*}
\text{proof } (auto) \\
\text{assume } \text{stp: step } r = 0 \\
\text{hence } \text{step } (Suc \ r) = 1 \\
\text{by } (auto \ simp \ add: \text{step-def mod-Suc}) \\
\text{with } \text{nxt stp show } ?\text{thesis} \\
\text{by } (intro \ step0) \\
\text{(auto simp: LV-CHOMachine-def CHOnextConfig-eq \ LV-nextState-def LV-sendMsg-def phase-Suc)} \\
\end{align*} \]

next

\[ \begin{align*}
\text{assume } \text{stp: step } r = \text{Suc } 0 \\
\text{hence } \text{step } (Suc \ r) = 2 \\
\text{by } (auto \ simp \ add: \text{step-def mod-Suc}) \\
\text{with } \text{nxt stp show } ?\text{thesis} \\
\text{by } (intro \ step1) \\
\text{(auto simp: LV-CHOMachine-def CHOnextConfig-eq \ LV-nextState-def LV-sendMsg-def phase-Suc)} \\
\end{align*} \]

next

\[ \begin{align*}
\text{assume } \text{stp: step } r = 2 \\
\text{hence } \text{step } (Suc \ r) = 3 \\
\text{by } (auto \ simp \ add: \text{step-def mod-Suc}) \\
\text{with } \text{nxt stp show } ?\text{thesis} \\
\text{by } (intro \ step2) \\
\text{(auto simp: LV-CHOMachine-def CHOnextConfig-eq \ LV-nextState-def LV-sendMsg-def phase-Suc)} \\
\end{align*} \]

next

\[ \begin{align*}
\text{assume } \text{stp: step } r = 3 \\
\text{hence } \text{step } (Suc \ r) = 0 \\
\text{by } (auto \ simp \ add: \text{step-def mod-Suc}) \\
\text{with } \text{nxt stp show } ?\text{thesis} \\
\text{by } (intro \ step3) \\
\text{(auto simp: LV-CHOMachine-def CHOnextConfig-eq \ LV-nextState-def LV-sendMsg-def phase-Suc)} \\
\end{align*} \]

qed

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

**lemma** \( LV\text{-induct'}: \)

\[ \begin{align*}
\text{assumes run: CHORun LV-M \rho Hos coords} \\
\end{align*} \]
and init: CinitState LV-M p (rho 0 p) (coords 0 p) \Rightarrow P p 0

and step0: \forall r. [ step r = 0; P p r; phase (Suc r) = phase r; step (Suc r) = 1;
next0 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p (Suc r)

and step1: \forall r. [ step r = 1; P p r; phase (Suc r) = phase r; step (Suc r) = 2;
next1 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p (Suc r)

and step2: \forall r. [ step r = 2; P p r; phase (Suc r) = phase r; step (Suc r) = 3;
next2 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p (Suc r)

and step3: \forall r. [ step r = 3; P p r; phase (Suc r) = Suc (phase r); step (Suc r) = 0;
next3 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p (Suc r)

shows P p n.
by \text{(rule \text{LV-induct}(OF run))}
(auto intro: init step0 step1 step2 step3)

lemma LV-Suc':
assumes run: CHORun LV-M rho \text{HOs} coords
and step0: [ step r = 0; step (Suc r) = phase r;
next0 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p r

and step1: [ step r = 1; step (Suc r) = phase r;
next1 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p r

and step2: [ step r = 2; step (Suc r) = phase r;
next2 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p r

and step3: [ step r = 3; step (Suc r) = 0; phase (Suc r) = Suc (phase r);
next3 r p (rho r p)

(\text{HOrcvdMsgs LV-M} r p (\text{HOs} r p) (\text{rho} r))
(coords (Suc r) p) (rho (Suc r) p) ]
\Rightarrow P p r

shows P p r
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) ≤ (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) ≤ timestamp (rho (Suc n) p)
(is ?P p n is ?ts ≤ -)
proof (rule LV-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 ≤ phase n
  using LV-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 ≤ n
shows timestamp (rho m p) ≤ timestamp (rho n p)
(is ?ts m ≤ -)
proof –
from le obtain k where k: n = m+k
  by (auto simp add: le-iff-add)
have ?ts m ≤ ?ts (m+k) (is ?P k)
proof (induct k)
  case 0 show ?P 0 by simp
next
  fix k
  assume ih: ?P k
from run have ?ts (m+k) ≤ ?ts (m + Suc k)
    by (auto simp add: LV-timestamp-increasing)
with \( \texttt{th} \) show \( ?P \ (\text{Suc} \ k) \) by simp
qed

with \( k \) show \( \texttt{thesis} \) by simp
qed

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

**definition** \( \texttt{procsBeyondTS} \ where \)

\[
\texttt{procsBeyondTS} \ ts \ \texttt{cfg} \equiv \{ \ p . \ ts \leq \text{timestamp} \ (\texttt{cfg} \ p) \}
\]

Since timestamps grow monotonically, so does the set of processes that are beyond a certain bound.

**lemma** \( \texttt{procsBeyondTS-monotonic} \):

**assumes** \( \texttt{run} \): \( \texttt{CHORun LV-M \rho HOs \ coords} \)

**and** \( p \): \( p \in \texttt{procsBeyondTS} \ ts \ (\rho \ m) \) **and** \( \texttt{le} \): \( m \leq n \)

**shows** \( p \in \texttt{procsBeyondTS} \ ts \ (\rho \ n) \)

**proof**

from \( p \) have \( ts \leq \text{timestamp} \ (\rho \ m \ p) \) (is - \( \leq \) ?ts \( m \))
by (simp add: \( \texttt{procsBeyondTS-def} \))
moreover
from \( \texttt{run \ le} \) have \( ?ts \ m \leq ?ts \ n \) by (rule \( \texttt{LV-timestamp-monotonic} \))
ultimately show \( \texttt{thesis} \)
by (simp add: \( \texttt{procsBeyondTS-def} \))
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** \( \texttt{notStep3EqualCoord} \):

**assumes** \( \texttt{run} \): \( \texttt{CHORun LV-M \rho HOs \ coords} \) **and** \( \texttt{stp} \): \( \texttt{step} \ r \neq 3 \)

**shows** \( \texttt{coordΦ} \ (\rho \ (\text{Suc} \ r) \ p) = \texttt{coordΦ} \ (\rho \ r \ p) \) (is \( \texttt{?P \ p \ r} \))
by (rule \( \texttt{LV-Suc}[^{\text{OF \ run, where \ P=?P}}] \))
(auto simp: \( \texttt{stp \ next0-def next1-def next2-def} \))

**lemma** \( \texttt{coordinators} \):

**assumes** \( \texttt{run} \): \( \texttt{CHORun LV-M \rho HOs \ coords} \)

**shows** \( \texttt{coordΦ} \ (\rho \ r \ p) = \texttt{coords} \ (4*\text{phase} \ r) \) \( p \)

**proof**

let \( ?r0 = (4*\text{phase} \ r) - 1 \)
let \( ?r1 = (4*\text{phase} \ r) \)

have \( \texttt{coordΦ} \ (\rho \ ?r1 \ p) = \texttt{coords} \ ?r1 \ p \)

**proof** (cases \( \texttt{phase} \ r \geq 0 \))

case \( \texttt{False} \)

72
hence \( \text{phase } r = 0 \) by auto
with run show \(?\text{thesis}\)
by (auto simp: LV-CHOMachine-def CHORun-eq CHOinitConfig-def LV-initState-def)

next
case True
hence \( \text{step } (\text{Suc } ?r0) = 0 \) by (auto simp: step-def)
hence \( \text{step } ?r0 = 3 \) by (auto simp: mod-Suc step-def)
moreover
from run
have \( \text{LV-nextState } ?r0 p (\rho ?r0 p) \)
\( (\text{HOrcvdMsgs } \text{LV-M } ?r0 p (\text{HOs } ?r0 p) (\rho ?r0 p)) \)
\( (\text{coords } (\text{Suc } ?r0) p) (\rho (\text{Suc } ?r0) p) \)
by (auto simp: LV-CHOMachine-def CHORun-eq CHOnextConfig-eq)
ultimately
have \( \text{nxt: next3 } ?r0 p (\rho ?r0 p) \)
\( (\text{HOrcvdMsgs } \text{LV-M } ?r0 p (\text{HOs } ?r0 p) (\rho ?r0 p)) \)
\( (\text{coords } (\text{Suc } ?r0) p) (\rho (\text{Suc } ?r0) p) \)
by (auto simp: LV-CHOMachine-def CHORun-eq CHOnextConfig-eq)
ultimately
have \( \text{coord}\Phi (\rho (\text{Suc } ?r0) p) = \text{coords } (\text{Suc } ?r0) p \)
by (auto simp: next3-def)
with True show \(?\text{thesis}\) by auto
qed

Commit status only changes at steps 0 and 3.

\textbf{lemma notStep03EqualCommit \[\text{rule-format}\]:}
\textbf{assumes run: CHORun LV-M rho HOs coords}
\textbf{shows step } r \neq 0 \land \text{step } r \neq 3 \rightarrow \text{commt } (\rho (\text{Suc } r) p) = \text{commt } (\rho r p) \)
\( (\text{is } ?P p r) \)
by (rule LV-Suc"[OF run, where \( P=\?P\)]")
(\text{auto simp: notStep0-def next1-def next2-def next3-def})
Timestamps only change at step 1.

**Lemma notStep1EqualTimestamp** [rule-format]:

**Assumes** run: CHORun LV-M rho HOs coords

**Shows** step r \( \neq 1 \) \( \rightarrow \) timestamp \((\rho (Suc r) p) = \) timestamp \((\rho r p)\)

\(\text{is } ?P r p\)

**By** (rule LV-Suc[\(OF\) 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 \( \neq 1 \) \( \rightarrow \) \(x (\rho (Suc r) p) = x (\rho r p)\) (is \(\rho r p\))

**By** (rule LV-Suc[\(OF\) run, \where \(P=?P\)])

(auto simp: next0-def next1-def next2-def next3-def)

A process \(p\) has its commit 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: commit \((\rho r p)\)

**And** conds: \(1 \leq \text{step } r\); \(\text{coord} \Phi (\rho r p) = p\); \(\text{vote} (\rho r p) \neq \text{None}\); \(\text{card} \{q . \text{coord} \Phi (\rho r q) = p\} > N \div 2\)

\(\Rightarrow A\)

**Shows** \(A\)

**Proof**

- **Have** commit \((\rho r p) \rightarrow \)

  \(1 \leq \text{step } r\)

  \& \(\text{coord} \Phi (\rho r p) = p\)

  \& \(\text{vote} (\rho r p) \neq \text{None}\)

  \& \(\text{card} \{q . \text{coord} \Phi (\rho r q) = p\} > N \div 2\)

\(\text{is } ?P r p \text{ - } \rightarrow ?R r\)

**Proof** (rule LV-induct[\(OF\) run, \where \(P=?P\)])

— the only interesting step is step 0

**Fix** \(n\)

**Assume** next: 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: ¬ commt (rho n p) by simp

with nxt cm'

have coordΦ (rho n p) = p
  ∧ vote (rho (Suc n) p) ≠ 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))
  ⊆ {q. coordΦ (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)))
  ≤ card {q. coordΦ (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 cons, 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 cons: \[ 2 \leq \text{step} r; \]
  \[
  \text{commt} (\text{rho} \ r \ (\text{coord}\Phi (\text{rho} \ r \ p)));
  \]
  \[
  x (\text{rho} \ r \ p) = \text{the} \ (\text{vote} (\text{rho} \ r \ (\text{coord}\Phi (\text{rho} \ r \ p))));
  \]
  \[
  \[ \Rightarrow \] A

shows A

proof

let ?ts n = timestamp (rho n p)
let ?crd n = coordΦ (rho n p)

have ?ts r = Suc (phase r) \[ \Rightarrow \]
  \[
  2 \leq \text{step} r
  \]
\[ \wedge \text{commt} (\rho r (?\text{crd } r)) \]
\[ \wedge x (\rho r p) = \text{the} (\rho r (?\text{crd } r)) \]
\[ \text{is } ?Q p r \text{ is } \rightarrow ?R r \]

proof (rule \text{LV-induct}[\text{OF run, where } P=\text{?Q}])

-- The assertion is trivially true initially because the timestamp is 0.
assume \text{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 post-state cannot be current (cf. lemma \text{LV-timestamp-bounded}).

fix \( n \)
assume \( \text{stp'}: \text{step (Suc n)} = 1 \)
with run \text{LV-timestamp-bounded}[\text{where } n=\text{Suc n}]
have \( ?ts (\text{Suc n}) \leq \text{phase (Suc n)} \) by auto
thus \( ?Q p (\text{Suc n}) \) by simp
next

Step 1 establishes the assertion by definition of the transition relation.

fix \( n \)
assume \( \text{stp}: \text{step } n = 1 \) and \( \text{stp'}: \text{step (Suc n)} = 2 \)
and \( \text{ph}: \text{phase (Suc n)} = \text{phase } n \)
and \( \text{nxt}: \text{next1 } n p (\text{HOrcvdMsgs LV-M n p (HOs n p) (rho n)}) (\text{coords (Suc n) p}) (\text{rho (Suc n) p}) \)
show \( ?Q p (\text{Suc n}) \)
proof
assume \( \text{ts}: ?ts (\text{Suc n}) = \text{Suc (phase (Suc n))} \)
from run \( \text{stp} \) \text{LV-timestamp-bounded}[\text{where } n=n]
have \( ?ts n \leq \text{phase } n \) by auto
moreover
from run \( \text{stp} \)
have \( \text{vote } (\rho (\text{Suc n}) (?\text{crd (Suc n)})) = \text{vote } (\rho n (?\text{crd n})) \)
by (auto simp: notStep3EqualCoord notStep0EqualVote)
moreover
from run \( \text{stp} \)
have \( \text{commt } (\rho (\text{Suc n}) (?\text{crd (Suc n)})) = \text{commt } (\rho n (?\text{crd n})) \)
by (auto simp: notStep3EqualCoord notStep03EqualCommit)
moreover
note \( \text{ts} \) \( \text{nxt} \) \( \text{stp} \) \( \text{stp'} \) \( \text{ph} \)
ultimately
show \( ?R (\text{Suc n}) \)
by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def
next1-def send1-def isVote-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 \( \text{stp}: \text{step } n = 2 \) and \( \text{stp}' : \text{step } (\text{Suc } n) = 3 \)

and \( \text{ph}: \text{phase } (\text{Suc } n) = \text{phase } n \)

and \( \text{ih}: \forall Q p n \)

and \( \text{nxt}: \text{next2 } n p (\text{rho } n p) (\text{HOrcvdMsgs LV-M } n p (\text{HOs } n p) (\text{rho } n p)) (\text{coords } (\text{Suc } n) p) (\text{rho } (\text{Suc } n) p) \)

show \( \exists Q p (\text{Suc } n) \)

proof

assume \( ts: \exists ts (\text{Suc } n) = \text{Suc } (\text{phase } (\text{Suc } n)) \)

from run \( \text{stp} \)

have \( vt: \text{vote } (\text{rho } (\text{Suc } n) \ (\forall\text{crd } (\text{Suc } n))) = \text{vote } (\text{rho } n \ (\forall\text{crd } n)) \)

by (auto simp add: notStep3EqualCoord notStep0EqualVote)

from run \( \text{stp} \)

have \( \text{cnt}: \text{commit } (\text{rho } (\text{Suc } n) \ (\forall\text{crd } (\text{Suc } n))) = \text{commit } (\text{rho } n \ (\forall\text{crd } n)) \)

by (auto simp add: notStep3EqualCoord notStep03EqualCommit)

with \( vt \ ts \ ph \ stp' \ ih \ nxt \)

show \( \exists R (\text{Suc } n) \)

by (auto simp add: next2-def)

qed

next

The assertion is trivially preserved by step 3 because the timestamp in the post-state cannot be current (cf. lemma \( \text{LV-timestamp-bounded} \)).

fix \( n \)

assume \( \text{stp}' : \text{step } (\text{Suc } n) = 0 \)

with run \( \text{LV-timestamp-bounded}[\text{where } n=\text{Suc } n] \)

have \( \exists ts (\text{Suc } n) \leq \text{phase } (\text{Suc } n) \) by auto

thus \( \exists Q p (\text{Suc } n) \) by simp

qed

with \( ts \) show \( \exists \text{thesis} \) by (intro \( \text{conds} \)) auto

qed

If a process \( p \) has its \textit{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.

\textbf{lemma \( \text{readyE} \)}:

\textbf{assumes} run: \( \text{CHORun LV-M rho HOs coords and rdy: ready } (\text{rho } r p) \)

and \( \text{conds}: \ [ \text{step } r = 3; \ \text{coord}\Phi (\text{rho } r p) = p; \) \)

\( \text{card } \{ q : \text{coord}\Phi (\text{rho } r q) = p \land \text{timestamp } (\text{rho } r q) = \text{Suc } (\text{phase } r) \} > N \div 2 \)

\textbf{shows} \( P \)

\textbf{proof} –

\textbf{let} \( \exists qs n = \{ q : \text{coord}\Phi (\text{rho } n q) = p \land \text{timestamp } (\text{rho } n q) = \text{Suc } (\text{phase } n) \} \)
have \( \text{ready} \ (\rho \ r \ p) \rightarrow \)

\[ \text{step} \ r = 3 \]
\[ \land \ \text{coord} \Phi \ (\rho \ r \ p) = p \]
\[ \land \ \text{card} \ (\mathbb{Q} s) > N \div 2 \]

(is \( \mathbb{Q} p \ r \rightarrow \mathbb{Q} p \ r \))

\textbf{proof} (rule \( \text{LV-induct}[\text{OF run, where } P=\mathbb{Q}] \))

— the interesting case is step 2

\textbf{fix} \ n

\textbf{assume} \( \text{stp}: \text{step} \ n = 2 \) \textbf{and} \( \text{stp}': \text{step} \ (\text{Suc} \ n) = 3 \)

\textbf{and} \( \text{ih}: \ \forall q \ p \ n \ \text{and} \ \text{ph}: \ \text{phase} \ (\text{Suc} \ n) = \text{phase} \ n \)

\textbf{and} \( \text{nxt}: \text{next2} \ n \ p \ (\rho \ n \ p) \ (\text{HOr}c\text{vdM}sgs \ \text{LV-M} \ n \ p \ (\text{H}Os \ n \ p) \ (\rho \ n)) \)

\( \text{coords} \ (\text{Suc} \ n \ p) \ (\text{rho} \ (\text{Suc} \ n) \ p) \)

\textbf{show} \( \mathbb{Q} p \ (\text{Suc} \ n) \)

\textbf{proof} (clarify)

\textbf{fix} \ q

\textbf{assume} \ q: \ q \in \mathbb{Q} \ p \ (\text{Suc} \ n)

\textbf{with} \( \text{stp} \)

\textbf{have} \ n: \ \text{coord} \Phi \ (\rho \ n \ q) = p \ \land \ \text{timestamp} \ (\rho \ n \ q) = \text{Suc} \ (\text{phase} \ n) \)

\textbf{by} (auto simp: \( \text{LV-CHOMachine-def} \ \text{HOr}c\text{vdM}sgs-def \ \text{LV-sendMsg-def} \ \text{a}cks\text{Rcvd-def} \ \text{send2-def} \ \text{is}\text{Ack-def} \)

\textbf{with} \( \text{run} \ \text{stp} \ \text{ph} \)

\textbf{show} \( \text{coord} \Phi \ (\rho \ (\text{Suc} \ n) \ q) = p \)

\( \land \ \text{timestamp} \ (\rho \ (\text{Suc} \ n) \ q) = \text{Suc} \ (\text{phase} \ (\text{Suc} \ n)) \)

\textbf{by} (simp add: \( \text{notStep3EqualCoord} \ \text{notStep1EqualTimestamp} \)

\textbf{qed}

\textbf{hence} \( \mathbb{Q} \ p \ (\text{Suc} \ n) \)

\textbf{by} (intro \( \text{card-mono} \) \( \text{auto} \)

\textbf{with} \( \text{stp}' \) \( \text{coord} \ \text{a}cks\text{Rcvd} \) \textbf{show} \( \mathbb{R} p \ (\text{Suc} \ n) \)

\textbf{by} \( \text{auto} \)

\textbf{qed}

— the remaining steps are all solved trivially

\textbf{qed} (auto simp: \( \text{LV-CHOMachine-def} \ \text{LV-initState-def} \ \text{next0-def} \ \text{next1-def} \ \text{next3-def} \)

\textbf{with} \( \text{rdy} \) \textbf{show} \( \text{thesis} \) \textbf{by} (blast intro: \( \text{conds} \)

\textbf{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 \textit{ready} and \textit{commit} bits set.

\textbf{Lemma} \textit{decisionE}:
\begin{itemize}
\item \textbf{assumes} \textit{run}: \textit{CHORun} \textit{LV-M} \textit{rho} \textit{HOs coords}
\item \textbf{and} \textit{dec}: \textit{decide} (\textit{rho}\ (\textit{Suc}\ \textit{r}) \ \textit{p}) \neq \textit{decide} (\textit{rho}\ \textit{r}\ \textit{p})
\item \textbf{and} \textit{conds}: \]
\begin{itemize}
\item \textit{step} \textit{r} = 3;
\item \textit{decide} (\textit{rho}\ (\textit{Suc}\ \textit{r}) \ \textit{p}) = \textit{Some} (\textit{the} (\textit{vote} (\textit{rho}\ \textit{r}\ (\textit{coord}\Phi (\textit{rho}\ \textit{r}\ \textit{p})))));
\item \textit{ready} (\textit{rho}\ \textit{r}\ (\textit{coord}\Phi (\textit{rho}\ \textit{r}\ \textit{p}))); \textit{commit} (\textit{rho}\ \textit{r}\ (\textit{coord}\Phi (\textit{rho}\ \textit{r}\ \textit{p})))
\end{itemize}
\end{itemize}
\[ \implies P \]
\textbf{proof} –
\begin{itemize}
\item \textbf{let} \(?cfg\ = \textit{rho}\ \textit{r})
\item \textbf{let} \(?cfg'\ = \textit{rho}\ (\textit{Suc}\ \textit{r})
\item \textbf{let} \(?crd\ p\ = \textit{coord}\Phi (\textit{cfg}\ p)
\item \textbf{let} \(?dec'\ = \textit{decide} (\textit{cfg}'\ p)
\end{itemize}

Except for the assertion about the \textit{commit} field, the assertion can be proved directly from the next-state relation.

\begin{itemize}
\item \textbf{have} \textit{1}: \textit{step} \textit{r} = 3
\item \textbf{and} \textit{dec'} = \textit{Some} (\textit{the} (\textit{vote} (\textit{cfg}\ (\textit{crd}\ \textit{p}))))
\item \textbf{and} \textit{ready} (\textit{cfg}\ (\textit{crd}\ \textit{p}))
\item \textbf{(is \textit{Q} \textit{p} \textit{r})}
\item \textbf{proof} (\textbf{rule} \textit{LV-Suc}[OF \textit{run}, \textbf{where} \textit{P=}\textit{Q}])
\item \textbf{—} for step 3, we prove the thesis by expanding the relevant definitions
\item \textbf{assume} \textit{next3} \textit{r} \textit{p} (\textit{cfg} \ \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{p} \textit{r}) \textit{cfg})
\item \textbf{(coords} \textit{(Suc} \textit{r}) \textit{p} \textit{(cfg}' \textit{p})
\item \textbf{and} \textit{step} \textit{r} = 3
\item \textbf{with} \textit{dec} \textbf{show} \textit{thesis} \textbf{by} (\textit{auto simp: next3-def send3-def isVote-def LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def})
\item \textbf{next} – the other steps don’t change the decision
\item \textbf{assume} \textit{next0} \textit{r} \textit{p} (\textit{cfg} \ \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{p} \textit{r}) \textit{cfg})
\item \textbf{(coords} \textit{(Suc} \textit{r}) \textit{p} \textit{(cfg}' \textit{p})
\item \textbf{with} \textit{dec} \textbf{show} \textit{thesis} \textbf{by} (\textit{auto simp: next0-def})
\item \textbf{next}
\item \textbf{assume} \textit{next1} \textit{r} \textit{p} (\textit{cfg} \ \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{p} \textit{r}) \textit{cfg})
\item \textbf{(coords} \textit{(Suc} \textit{r}) \textit{p} \textit{(cfg}' \textit{p})
\item \textbf{with} \textit{dec} \textbf{show} \textit{thesis} \textbf{by} (\textit{auto simp: next1-def})
\item \textbf{next}
\item \textbf{assume} \textit{next2} \textit{r} \textit{p} (\textit{cfg} \ \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{p} \textit{r}) \textit{cfg})
\item \textbf{(coords} \textit{(Suc} \textit{r}) \textit{p} \textit{(cfg}' \textit{p})
\item \textbf{with} \textit{dec} \textbf{show} \textit{thesis} \textbf{by} (\textit{auto simp: next2-def})
\item \textbf{qed}
\item \textbf{hence} \textit{ready} (\textit{cfg}\ (\textit{crd}\ \textit{p})) \textbf{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.

\[
\text{with } \text{run} \\
\text{have } \text{card } \{ q . \ ?\text{crd } q = \ ?\text{crd } p \land \text{timestamp } (\ ?\text{cfg } q) = \text{Suc } (\text{phase } r) \} > N \div 2 \text{ by (rule readyE)} \\
\text{— Hence there is at least one such process . . .} \\
\text{hence card } \{ q . \ ?\text{crd } q = \ ?\text{crd } p \land \text{timestamp } (\ ?\text{cfg } q) = \text{Suc } (\text{phase } r) \} \neq 0 \\
\text{by arith} \\
\text{then obtain } q \text{ where } ?\text{crd } q = ?\text{crd } p \land \text{timestamp } (\ ?\text{cfg } q) = \text{Suc } (\text{phase } r) \text{ by auto} \\
\text{— . . . and by a previous lemma the coordinator must have committed.} \\
\text{with } \text{run have } \text{commit } (\ ?\text{cfg } (\ ?\text{crd } p)) \text{ by (auto elim: currentTimestampE)} \\
\text{with } I \text{ show } \text{?thesis by (blast intro: cons)} \\
\text{qed}
\]

### 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** `lo-integrityInvariant`:

**Assumes** run: `CHORun LV-M rho HOs coords`

**And inv**: `\[ range (x \circ (\rho n)) \subseteq range (x \circ (\rho 0)) ;
\quad range (vote \circ (\rho n)) \subseteq \{None\} \cup \text{Some } \cdot \text{range } (x \circ (\rho 0)) ;
\quad range (decide \circ (\rho n)) \subseteq \{None\} \cup \text{Some } \cdot \text{range } (x \circ (\rho 0)) \]` 

\[\Rightarrow A\]

**Shows** A

**Proof**

- **let** `\?x0 = range (x \circ (\rho 0))`
- **let** `\?x0opt = \{None\} \cup \text{Some } \cdot \?x0`
- **have** `range (x \circ (\rho n)) \subseteq \?x0`
  \quad \land `range (vote \circ (\rho n)) \subseteq \?x0opt`
  \quad \land `range (decide \circ (\rho n)) \subseteq \?x0opt`
- **is** `?Inv n is ?X n \land ?Vote n \land ?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** `vt: ?Vote n` **and dec: ?Decide n`

Proof of first conjunct

- **have** `x: ?X (Suc n)`
  - **proof (clarsimp)**
  - **fix** `p`
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 show ?thesis by auto
next
case False
with stp nxt have cnt: commt (rho n (coordΦ (rho n p)))
and xp: x (rho (Suc n) p) = the (vote (rho n (coordΦ (rho n p))))
by (auto simp: next1-def LV-CHOMachine-def HOrcvdMsgs-def
LV-sendMsg-def send1-def isVote-def)
from run cnt have vote (rho n (coordΦ (rho n p))) ≠ None
by (rule commitE)
moreover
from vt have vote (rho n (coordΦ (rho n p))) ∈ ?v0opt
by (auto simp add: image-def)
moreover
note xp
ultimately
show ?thesis by (force simp add: image-def)
qed
— the other steps don’t change x
next
assume step n = 0
with run have x (rho (Suc n) p) = x (rho n p)
by (simp add: notStep1EqualX)
with x 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 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 show ?thesis by auto
qed
qed

Proof of second conjunct

have vt': ?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 \rightarrow v \in \?x0 (is \?P p n)
proof (rule LV-Suc'[where P=?P])
-- here only step0 is of interest
assume stp: step n = 0
and 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)
show \?thesis
proof (cases rho (Suc n) p = rho n p)
case True
from vt 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 (rho 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 st 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 st have vote (rho n p) \in \?x0opt
  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 st have vote (rho n p) \in \?x0opt

82
ultimately

show ?thesis by auto

qed

with v show ∃ q. v = x (rho 0 q) by auto

qed

Proof of third conjunct

have dec': ?Decide (Suc n)

proof (clarsimp simp: image-def)

fix p v

assume v: decide (rho (Suc n) p) = Some v

show ∃ q. 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 = coordΦ (rho n p)

from False run

have d': decide (rho (Suc n) p) = Some (the (vote (rho n ?crd)))

and cmt: commit (rho n ?crd)

by (auto elim: decisionE)

from vt have vtc: vote (rho n ?crd) ∈ ?x0opt

by (auto simp: image-def)

from run cmt have vote (rho n ?crd) ≠ None

by (rule commitE)

with d' v vtc show ?thesis by auto

qed

qed

from x' vt' dec' show ?thesis by simp

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 ∃ q. v = x (rho 0 q)

proof –

from run have decide (rho n p) ∈ {None} ∪ Some ′ (range (x ◦ (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** \(\text{decisionThenMajorityBeyondTS}^{\text{TH}}\):

- **assumes** \(\text{run: CHORun LV-M rho HOs coords}\)
- **and** \(\text{dec: decide (rho (Suc r) p) \neq decide (rho r p)}\)
- **shows** \(\text{card (procsBeyondTS (Suc (phase r)) (rho r)) > } N \text{ div } 2\)
- **using** \(\text{run dec proof (rule decisionE)}\)

Lemma \(\text{decisionE}\) tells us that we are at step 3 and that the coordinator is ready.

Now, lemma \(\text{readyE}\) implies that a majority of processes have a recent timestamp.

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

No two different processes have their \text{commit} flag set at any state.

**Lemma** \(\text{committedProcsEqual}\):

- **assumes** \(\text{run: CHORun LV-M rho HOs coords}\)
- **and** \(\text{cmt: commit (rho r p) and cmt': commit (rho r p')}\)
- **shows** \(p = p'\)

**proof**

- **from** \(\text{run cmt have card } \{q . \text{coord\Phi (rho r q) = p}\} > N \text{ div } 2\)
  - **by** (blast elim: commitE)
- **moreover**
- **from** \(\text{run cmt' have card } \{q . \text{coord\Phi (rho r q) = p'}\} > N \text{ div } 2\)
  - **by** (blast elim: commitE)
- **ultimately**
- **obtain** \(q\) **where** \(\text{coord\Phi (rho r q) = p and p'} = \text{coord\Phi (rho r q)}\)
  - **by** (auto elim: majoritiesE')
- **thus** \(?thesis by simp\)

No two different processes have their \text{ready} flag set at any state.

**Lemma** \(\text{readyProcsEqual}\):
assumes run: CHORun LV-M rho HOs coords
and rdg: ready (rho r p) and rdg': ready (rho r p')
shows p = p'
proof —
  let ?C p = {q. coordΦ (rho r q) = p ∧ timestamp (rho r q) = Suc (phase r)}
  from run rdg have card (?C p) > N div 2
    by (blast elim: readyE)
  moreover
  from run rdg' have card (?C p') > N div 2
    by (blast elim: readyE)
  ultimately
  obtain q where coordΦ (rho r q) = p and p' = coordΦ (rho r q)
    by (auto elim: majoritiesE')
  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 ts, then p votes for
a value held by some process whose timestamp is beyond ts.

lemma commitThenVoteRecent:
  assumes run: CHORun LV-M rho HOs coords
  and maj: card (procsBeyondTS ts (rho r)) > N div 2
  and cmt: commt (rho r p)
  shows ∃q ∈ procsBeyondTS ts (rho r). vote (rho r p) = Some (x (rho r q))
    (is ?Q r)
proof —
  let ?bynd n = procsBeyondTS ts (rho n)
  have card (?bynd r) > N div 2 ∧ commt (rho r p) —→ ?Q r (is ?P p r)
  proof (rule LV-induct[OF run])
next0 establishes the property
  fix n
  assume stp: step n = 0
    and nxt: ∀q. next0 n q (rho n q) (HOrcvdMsgs LV-M n q (HOs n q) (rho n))
    (coords (Suc n) q) (rho (Suc n) q)
    (is ∀ q. ?nxt q)
  from nxt have nxp: ?nxp 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 ¬ commt (rho n p) by (auto elim: commitE)
      with nxp ct obtain q v 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 ?bynd (Suc n) and q2': q' \in valStampsRcvd ?msgs
by (rule majoritiesE')
have timestamp (\rho n q') \leq 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') = ts'
  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 ?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
We now prove that \textit{next1} preserves the property. Observe that \textit{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.

\textbf{fix} \ n
\textbf{assume}\ stp: step \ n = 1 \\
\textbf{and} \ nxt: \forall \ q. \ next1 \ n \ q \ (\rho \ n \ q) \\
(\text{\textit{HOrcvdMsgs}} \ \text{LV-M} \ n \ q \ (\text{\textit{HOs}} \ n \ q) \ (\rho \ n)) \\
(\text{coords} \ (\text{Suc} \ n) \ q) \\
(\rho \ (\text{Suc} \ n) \ q) \\
(is \ \forall \ q. \ \exists \ nxt \ q) \\
\textbf{and} \ ih: \ ?P \ p \ n
\textbf{from} \ \textbf{nxt} \ \textbf{have} \ \textbf{nxp}: \ ?nxt \ p \
\textbf{show} \ ?P \ p \ (\text{Suc} \ n)
\textbf{proof} \ (\text{clarify})
\textbf{assume} \ mj': \ \text{card} \ (\text{\textit{bynd}} \ (\text{Suc} \ n)) > N \ \text{div} \ 2 \\
\textbf{and} \ ct': \ \text{commit} \ (\rho \ (\text{Suc} \ n) \ p) \\
\textbf{from} \ \textbf{run} \ \textbf{stp} \ \textbf{ct'} \ \textbf{have} \ \textbf{ct}: \ \text{commit} \ (\rho \ n \ p) \\
by \ (\text{\textbf{simp add: notStep03EqualCommit}})
\textbf{from} \ \textbf{run} \ \textbf{stp} \ \textbf{have} \ \textbf{vote'}: \ \text{vote} \ (\rho \ (\text{Suc} \ n) \ p) = \text{vote} \ (\rho \ n \ p) \\
by \ (\text{\textbf{simp add: notStep0EqualVote}})
\textbf{show} \ ?Q \ (\text{Suc} \ n)
\textbf{proof} \ (\text{cases} \ \exists \ q \in \text{\textit{bynd}} \ (\text{Suc} \ n). \ \rho \ (\text{Suc} \ n) \ q \neq \rho \ n \ q)
\textbf{case} \ True
\textbf{in this case the property holds because} \ q \ \text{updates its} \ x \ \text{field to the vote} \\
\textbf{then obtain} \ q \ \textbf{where} \\
q1: \ q \in \text{\textit{bynd}} \ (\text{Suc} \ n) \ \textbf{and} \ q2: \ \rho \ (\text{Suc} \ n) \ q \neq \rho \ n \ q \
\textbf{from} \ \textbf{nxt} \ \textbf{have} \ ?nxt \ q \
\textbf{with} \ q2 \ \textbf{stp}
\textbf{have} \ x': \ x \ (\rho \ (\text{Suc} \ n) \ q) = \text{the} \ (\text{vote} \ (\rho \ n \ (\text{coord} \ \Phi \ (\rho \ n \ q)))) \\
\textbf{and} \ \text{coord: commit} \ (\rho \ n \ (\text{coord} \ \Phi \ (\rho \ n \ q))) \\
bym (\text{auto simp: next1-def send1-def LV-CHOMachine-def HOrcvdMsgs-def } \\
\text{LV-sendMsg-def isVote-def})
\textbf{from} \ \textbf{run} \ \textbf{ct} \ \textbf{have} \ \textbf{vote}: \ \text{vote} \ (\rho \ n \ p) \neq \text{None} \\
bym (\text{rule commitE})
\textbf{from} \ \textbf{run} \ \textbf{coord} \ \textbf{ct} \ \textbf{have} \ \text{\textit{coord}} \ (\rho \ n \ q) = p \\
bym (\text{rule committedProcsEqual})
\textbf{with} \ q1 \ x' \ \textbf{vote} \ \textbf{vote'} \ \textbf{show} \ ?\text{thesis} \ \textbf{by} \ \textbf{auto}
\textbf{next}
\textbf{case} \ False
\textbf{if no relevant process moves then} \ \textit{procsBeyondTS} \ \text{doesn’t change and we invoke the} \ \text{induction hypothesis} \\
\textbf{hence} \ \textit{bynd}: \ ?\text{bynd} \ (\text{Suc} \ n) = ?\text{bynd} \ n

87
proof (auto simp: procsBeyondTS-def)
fix r
assume ts: ts ≤ timestamp (rho n r)
from run have timestamp (rho n r) ≤ timestamp (rho (Suc n) r)
  by (simp add: LV-timestamp-monotonic)
with ts show ts ≤ timestamp (rho (Suc n) r) by simp
qed

with mj' have mj: card (?bynd n) > N div 2 by simp
with ct ih obtain q where
  q ∈ ?bynd n and vote (rho n p) = Some (x (rho n q))
  by blast
with vote' bynd False show ?thesis by auto
qed

next

step2 preserves the property, via the induction hypothesis.

fix n
assume stp: step n = 2
  and nxt: ∀ 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 ∀ q. ?nxt q)
  and ih: ?P p n
from nxt have nxp: ?nxt p ..
show ?P p (Suc n)
proof (clarify)
assume mj': card (?bynd (Suc n)) > N div 2
  and ct': commit (rho (Suc n) p)
from run stp ct' have ct: commit (rho n p)
  by (simp add: notStep03EqualCommit)
from run stp have vote': vote (rho (Suc n) p) = vote (rho n p)
  by (simp add: notStep03EqualVote)
from run stp have ∀ q. 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 ∀ 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 commit flag cannot
be set.

qed (auto elim: commitE[OF run])
with maj cmt show ?thesis by simp
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** \( \text{XOfTimestampBeyondDecision} \):

**Assumes**
- \( \text{run}: \text{ CHORun LV-M rho HOs coords} \)
- \( \text{dec}: \text{ decide (rho (Suc r) p) \neq \text{ decide (rho r p)}} \)

**Shows**
- \( \forall q \in \text{procsBeyondTS (Suc (phase r)) (rho (r+k)). x (rho (r+k) q) = \text{ the (decide (rho (Suc r) p))}} \)

(is \( \forall q \in \text{bynd k}. - = ?v \text{ is ?P p k} \))

**Proof** (induct \( k \))

- Base step
  - **Show** \( ?P p 0 \)
  - **Proof** (clarify)
    - Fix \( q \)
    - Assume \( q \in \text{bynd 0} \)

Use preceding lemmas about the decision value and the \( x \) field of processes with fresh timestamps.

- From \( \text{run dec} \)
  - Have \( \text{stp: step r = 3} \)
  - And \( v: \text{ decide (rho (Suc r) p) = Some (the (vote (rho r (coord\Phi (rho r p)))))} \)
  - And \( \text{cmt: commit (rho r (coord\Phi (rho r p)))} \)
  - By (auto elim: decisionE)

From \( \text{stp LV-timestamp-bounded[OF run, where n=r]} \)

- Have \( \text{timestamp (rho r q) \leq Suc (phase r) by simp} \)
- With \( q \) have \( \text{timestamp (rho r q) = Suc (phase r)} \)
  - By (simp add: procsBeyondTS-def)

With \( \text{run} \)

- Have \( x: x (rho (r+0) q) = \text{ the (vote (rho r (coord\Phi (rho r q))))} \)
  - And \( \text{cmt': commit (rho r (coord\Phi (rho r q)))} \)
  - By (auto elim: currentTimestampE)

From \( \text{run cmt cmt'} \)

- Have \( \text{coord\Phi (rho r p) = coord\Phi (rho r q)} \)
  - By (rule committedProcsEqual)

With \( x v \) show \( x (rho (r+0) q) = ?v \) by simp

Qed

Next

- Induction step
  - Fix \( k \)
  - Assume \( \text{ih: ?P p k} \)
  - Show \( ?P p (Suc k) \)

Proof (clarify)

- Fix \( q \)
  - Assume \( q \in \text{bynd (Suc k)} \)
  - Distinguish the kind of transition—only \( \text{step1} \) is interesting

- Have \( x (rho (Suc (r + k)) q) = ?v \) (is \( ?X q (r+k) \))

Proof (rule LV-Suc[\( \text{OF run, where P=X} \)])

- Assume \( \text{stp: step (r + k) = 1} \)
and \( \text{nxt: next1 (r+k) q (\rho (r+k)) q} \)
\( (\text{HOrcvdMsgs LV-M (r+k) q (HOs (r+k)) q (\rho (r+k)))} \)
\( (\text{coords (Suc (r+k)) q}) \)
\( (\rho (\text{Suc (r+k)}))) \)

show \(?\text{thesis}\)
proof (cases \(\rho (\text{Suc (r+k)}) = \rho (r+k)\) q)
case True
with q ih show \(?\text{thesis}\) by (auto simp: procsBeyondTS-def)
next
case False
from run dec have card (\(?\text{bynd 0}\) > N div 2
by (simp add: decisionThenMajorityBeyondTS)
moreover
have \(?\text{bynd 0} \subseteq \?\text{bynd k}\)
by (auto elim: procsBeyondTS-monotonic[OF run])
hence card (\(?\text{bynd 0}\) ≤ card (\(?\text{bynd k}\)
by (auto intro: card-mono)
ultimately
have maj: card (\(?\text{bynd k}\) > N div 2 by simp
let \(?\text{crd} = \text{coord}_\Phi (\rho (r+k) q)\)
from False stp nxt have
cmt: \text{commit} (\rho (r+k) \?\text{crd}) and
\( x: x (\rho (\text{Suc (r+k)}) q) = \text{the} \ (\rho (r+k) \?\text{crd}) \)
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' \?\text{bynd k}
and q2': vote (\rho (r+k) \?\text{crd}) = Some (x (\rho (r+k) q'))
by (blast dest: commitThenVoteRecent)
with x ih show \(?\text{thesis}\) by auto
qed
next
— all other steps hold by induction hypothesis

assume step (r+k) = 0
with run have x: x (\rho (\text{Suc (r+k)}) q) = x (\rho (r+k) q)
and ts: \text{timestamp} (\rho (\text{Suc (r+k)}) q) = \text{timestamp} (\rho (r+k) q)
by (auto simp: notStep1EqualX notStep1EqualTimestamp)
from ts q have q \?\text{bynd k}
by (auto simp: procsBeyondTS-def)
with x ih show \(?\text{thesis}\) by auto
next

assume step (r+k) = 2
with run have x: x (\rho (\text{Suc (r+k)}) q) = x (\rho (r+k) q)
and ts: \text{timestamp} (\rho (\text{Suc (r+k)}) q) = \text{timestamp} (\rho (r+k) q)
by (auto simp: notStep1EqualX notStep1EqualTimestamp)
from ts q have q \?\text{bynd k}
by (auto simp: procsBeyondTS-def)
with x ih show \(?\text{thesis}\) by auto
next
assume step \( (r+k) = 3 \)
with run have \( x \equiv x (\rho (\text{Suc} (r+k))) q = x (\rho (r+k)) q \)
and \( ts \equiv \text{timestamp} (\rho (\text{Suc} (r+k))) q = \text{timestamp} (\rho (r+k)) q \)
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
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 \( \text{run}: \text{CHORun} \text{ LV-M} \rho \text{HOs coords} \)
and \( p \equiv \text{decide} (\rho (\text{Suc} (r)) p) \neq \text{decide} (\rho (r) p) \)
and \( q \equiv \text{decide} (\rho (\text{Suc} (r+k)) q) \neq \text{decide} (\rho (r+k) q) \)
shows \( \text{decide} (\rho (\text{Suc} (r+k)) q) = \text{decide} (\rho (\text{Suc} (r)) p) \)
proof –
let \( ?bynd k = \text{procsBeyondTS} (\text{Suc} (\text{phase} r)) (\rho (r+k)) \)
let \( ?qcrd = \text{coord}(\Phi) (\rho (r+k)) q \)
from \( \text{run p} \) have \( \text{notNone}: \text{decide} (\rho (\text{Suc} (r)) p) \neq \text{None} \)
by (auto elim: decisionE)
— process \( q \) decides on the vote of its coordinator
from \( \text{run q} \) have \( \text{dec}: \text{decide} (\rho (\text{Suc} (r+k)) q) = \text{Some} (\text{the} (\text{vote} (\rho (r+k)) ?qcrd))) \)
and \( \text{cmt}: \text{commit} (\rho (r+k)) ?qcrd \)
by (auto elim: decisionE)
— that vote is the \( x \) field of some process \( q' \) with a recent timestamp
from \( \text{run p} \) have \( \text{card} (?bynd 0) > N \text{ div } 2 \)
by (simp add: decisionThenMajorityBeyondTS)
moreover
from \( \text{run have} ?bynd 0 \subseteq ?bynd k \)
by (auto elim: procsBeyondTS-monotonic)
hence \( \text{card} (?bynd 0) \leq \text{card} (?bynd k) \)
by (auto intro: card-mono)
ultimately
have \( \text{maj}: \text{card} (?bynd k) > N \text{ div } 2 \) by simp
from \( \text{run maj cmt} \) obtain \( q' \)
where \( q'1: q' \in ?bynd k \)
and \( q'2: \text{vote} (\rho (r+k)) ?qcrd = \text{Some} (x (\rho (r+k)) q') \)
by (auto dest: commitThenVoteRecent)
— the \( x \) field of process \( q' \) is the value \( p \) decided on
from \( \text{run p} q' \)
have \( x (\rho (r+k)) q' = \text{the} (\text{decide} (\rho (\text{Suc} (r)) p)) \)
by (auto dest: XORTimestampBeyondDecision)
— which proves the assertion
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** $\exists m < n. \, \text{decide (rho (Suc m) p) \neq decide (rho m p)}$
and $\text{decide (rho (Suc m) p) = Some v}$

**proof**

let $\mathbf{dec \; k} = \text{decide (rho k p)}$

have ($\forall m < n. \, \mathbf{dec \; (Suc m)} \neq \mathbf{dec \; m} \rightarrow \mathbf{dec \; (Suc m)} \neq \mathbf{Some \; v}$)

(is $P \; n$ is $\forall \; n \rightarrow -$)

**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 $\mathbf{ih: \; ?P \; n}$
show $?P \; (\text{Suc} \; n)$

**proof** (clarify)

assume $p$: $\forall A \; (\text{Suc} \; n) \, \text{and v: \; \mathbf{dec \; (Suc \; n)} = \mathbf{Some \; v}$
from $p$ have $?A \; n$ by simp

with $\mathbf{ih}$ have $?\mathbf{dec \; n} \neq \mathbf{Some \; v}$ by simp

moreover

from $p$

have $?\mathbf{dec \; (Suc \; n)} \neq ?\mathbf{dec \; n} \rightarrow ?\mathbf{dec \; (Suc \; n)} \neq \mathbf{Some \; v}$ by simp

ultimately

have $?\mathbf{dec \; (Suc \; n)} \neq \mathbf{Some \; v}$ by auto

with $v$ show False by simp

qed

**with** $\mathbf{dec \; show \; ?thesis \; by \; auto}$

**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

$n1$: $n < m$ 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 $\forall i. \text{decide} \ (\rho \ (\text{Suc} \ (n+i)) \ p) = \text{Some} \ v$ (is $\forall \ i. \ ?\text{dec} \ i$)

proof
fix $i$
show $\ ?\text{dec} \ i$
proof (induct $i$)
from $n3$ show $\ ?\text{dec} \ 0$ by simp
next
fix $j$
assume $ih: \ ?\text{dec} \ j$
show $\ ?\text{dec} \ (\text{Suc} \ j)$
proof (rule ccontr)
assume $ctr: \neg \ (\ ?\text{dec} \ (\text{Suc} \ j))$
with $ih$
have $\text{decide} \ (\rho \ (\text{Suc} \ (n + \text{Suc} \ j)) \ p) \neq \text{decide} \ (\rho \ (n + \text{Suc} \ j) \ p)$
by simp
with run $n2$
have $\text{decide} \ (\rho \ (\text{Suc} \ (n + \text{Suc} \ j)) \ p) = \text{decide} \ (\rho \ (\text{Suc} \ n) \ p)$
by (rule laterProcessDecidesSameValue)
with $ctr \ n3$ show False by simp
qed
qed

moreover
from $n1$ obtain $j$ where $m+k = \text{Suc} (n+j)$
by (auto dest: less-imp-Suc-add)
ultimately
show $\ ?\text{thesis}$ by auto
qed

theorem lv-agreement:
assumes run: \text{CHORun} \ \text{LV-M} \ \rho \ \text{HOs} \ \text{coords}
and $p$: $\text{decide} \ (\rho \ m \ p) = \text{Some} \ v$
and $q$: $\text{decide} \ (\rho \ n \ q) = \text{Some} \ w$
shows $v = w$
proof
from run $p$ obtain $k$
where $k1$: $\text{decide} \ (\rho \ (\text{Suc} \ k) \ p) \neq \text{decide} \ (\rho \ k \ p)$
and $k2$: $\text{decide} \ (\rho \ (\text{Suc} \ k) \ p) = \text{Some} \ v$
by (auto dest: decisionNonNullThenDecided)
from run $q$ obtain $l$
where $l1$: $\text{decide} \ (\rho \ (\text{Suc} \ l) \ q) \neq \text{decide} \ (\rho \ l \ q)$
and $l2$: $\text{decide} \ (\rho \ (\text{Suc} \ l) \ q) = \text{Some} \ w$
by (auto dest: decisionNonNullThenDecided)
show $\ ?\text{thesis}$
proof (cases $k \leq l$)
case True
then obtain $m$ where $m$: $l = k+m$ by (auto simp: le-iff-add)
from run $k1 \ l1 \ m$
have $\text{decide} \ (\rho \ (\text{Suc} \ l) \ q) = \text{decide} \ (\rho \ (\text{Suc} \ k) \ p)$

93
by (auto elim: laterProcessDecidesSameValue)
with \( k2 \) \( l2 \) show \( ?\)thesis by simp

next

case False
hence \( l \leq k \) by simp
then obtain \( m \) where \( m = k + m \) by (auto simp: le-iff-add)
from run \( l1 \) \( k1 \) \( m \)
have decide \((\rho \Suc k)\) = decide \((\rho \Suc l)\) q
by (auto elim: laterProcessDecidesSameValue)
with \( l2 \) \( k2 \) show \( ?\)thesis by simp
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 \( run: \text{CHORun LV-M } \rho \text{HOs coords} \)
and \( \text{commG:CHOcommGlobal LV-M HOs coords} \)
shows \( \exists r. \forall p. \text{decide } (\rho r p) \neq \text{None} \)

**Proof:**

The communication predicate implies the existence of a “successful” phase \( ph \), coordinated by some process \( c \) for all processes.

from \( \text{commG obtain } ph \) \( c \)
where \( c: \forall p. \text{coords } (\rho \Suc ph) p = c \)
and \( \text{maj0: card } (\text{HOs } (\rho \Suc ph) c) > N \div 2 \)
and \( \text{maj2: card } (\text{HOs } (\rho \Suc ph+2) c) > N \div 2 \)
and \( \text{rcv1: } \forall p. c \in \text{HOs } (\rho \Suc ph+1) p \)
and \( \text{rcv3: } \forall p. c \in \text{HOs } (\rho \Suc ph+3) p \)
by (auto simp: LV-CHOMachine-def LV-commGlobal-def)

let \( \gamma 0 = \rho \Suc ph \)
let \( \gamma 1 = \text{Suc } \gamma 0 \)
let \( \gamma 2 = \text{Suc } \gamma 1 \)
let \( \gamma 3 = \text{Suc } \gamma 2 \)
let \( \gamma 4 = \text{Suc } \gamma 3 \)

Process \( c \) is the coordinator of all steps of phase \( ph \).

from \( \text{run } c \) have \( c: \forall p. \text{coord} (\rho \gamma r p) = c \)
by (auto simp add: phase-def coordinators)

with \( \text{run } c1: \forall p. \text{coord} (\rho \gamma 1 r p) = c \)
by (auto simp add: step-def mod-Suc notStep3EqualCoord)

with \( \text{run } c2: \forall p. \text{coord} (\rho \gamma 2 r p) = c \)
by (auto simp add: step-def mod-Suc notStep3EqualCoord)
with run have c3: ∀ p. coordΦ (rho ?r3 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 ph and therefore commits during the transition at the end of step 0.

have 1: commt (rho ?r1 c) (is ?P c (4∗ph))
proof (rule LV-Suc"[OF run, where P=??], auto simp: step-def)
  assume next0 ?r c (rho ?r) (HOrcvdMsgs LV-M ?r c (HOs ?r2 c) (rho ?r))
  (coords (Suc ?r) c (rho (Suc ?r) c)
  with c' 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: ∀ p. timestamp (rho ?r2 p) = Suc ph
proof
  fix p
  let ?msgs = HOrcvdMsgs LV-M ?r1 p (HOs ?r1 p) (rho ?r1)
  let ?crd = coordΦ (rho ?r1 p)
  from run 1 c1 rcv1
  have cnd: ?msgs ?crd ≠ None ∧ isVote (the (?msgs ?crd))
    by (auto elim: commitE simp: step-def LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def send1-def isVote-def)
  show timestamp (rho ?r2 p) = Suc ph (is ?P p (Suc (4∗ph)))
  proof (rule LV-Suc"[OF run, where 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=??], 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: ∀ p. decide (rho ?r4 p) ≠ None
proof
fix p
let ?crd = coord\Phi (rho ?r3 p)
from run 3 c3 rcv3
have cnd: ?msgs ?crd \neq None \land isVote (the (?msgs ?crd))
  by (auto elim: readyE
      simp: step-def mod-Suc LV-CHOMachine-def HOrcvdMsgs-def
            LV-sendMsg-def send3-def isVote-def numeral-3-eq-3)
show decide (rho ?r4 p) \neq None (is P p (Suc (Suc (Suc (Suc (Suc ph))))))
proof (rule LV-Suc[\OF 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 ?r4 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 (rho 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 (\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 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 o (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 \( U_{T,E,\alpha} \) Consensus Algorithm

Algorithm \( U_{T,E,\alpha} \) 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.

typed

\begin{align*}
\text{typedecl} & \quad \text{Proc} — \text{the set of processes} \\
\text{axiomatization} & \quad \text{where} \quad \text{Proc-finite: OFCLASS(Proc, finite-class)} \\
\text{instance} & \quad \text{Proc :: finite by} (\text{rule Proc-finite})
\end{align*}

abbreviation

\( \text{N} \equiv \text{card (UNIV::Proc set)} \quad \text{— number of processes} \)

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

\( \text{nSteps} \equiv 2 \)

definition phase where \( \text{phase (r::nat)} \equiv r \text{ div nSteps} \)

definition step where \( \text{step (r::nat)} \equiv r \text{ mod nSteps} \)

lemma phase-zero \([\text{simp}: \text{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 ≡
(vote st = None) ∧ (decide st = None)

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

locale ute-parameters =
fixes α :: nat and T :: nat and E :: nat
assumes majE: $2*E \geq N + 2*\alpha$
and majT: $2*T \geq N + 2*\alpha$
and EltN: $E < N$
and TltN: $T < N$

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 ⇒ Proc ⇒ Proc ⇒ 'val pstate ⇒ 'val msg
where
send0 r p q st ≡ Val (x st)

definition
next0 :: nat ⇒ Proc ⇒ Proc ⇒ 'val pstate ⇒ (Proc ⇒ 'val msg option) ⇒ 'val pstate ⇒ bool
where
next0 r p st msgs st' ≡
(∃ v. card {q. msgs q = Some (Val v)} > T ∧ st' = st (\ vote := Some v ))
∧ ¬(∃ v. card {q. msgs q = Some (Val v)} > T) ∧ st' = st (\ vote := None )

In step 1, each process sends its current vote.
If it receives more than α 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 ⇒ Proc ⇒ Proc ⇒ 'val pstate ⇒ 'val msg
where
send1 r p q st ≡ Vote (vote st)

definition
next1 :: nat ⇒ Proc ⇒ Proc ⇒ 'val pstate ⇒ (Proc ⇒ 'val msg option) ⇒ 'val pstate ⇒ bool
where
next1 r p st msgs st' ≡
( (∃ v. card {q. msgs q = Some (Vote (Some v))} > α ∧ x st' = v)
∧ ¬(∃ v. card {q. msgs q = Some (Vote (Some v))} > α)
∧ x st' = undefined )
∧ (∃ v. card {q. msgs q = Some (Vote (Some v))} > E ∧ decide st' = Some v)
∧ ¬(∃ v. card {q. msgs q = Some (Vote (Some v))} > E)
∧ decide st' = decide st )
∧ vote st' = 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 ⇒ Proc ⇒ Proc ⇒ 'val pstate ⇒ 'val msg
where
Ute-sendMsg (r::nat) ≡ if step r = 0 then send0 r else send1 r

definition
Ute-nextState :: nat ⇒ Proc ⇒ 'val pstate ⇒ (Proc ⇒ 'val msg option) ⇒ 'val pstate ⇒ bool
where
Ute-nextState r ≡ if step r = 0 then next0 r else next1 r
8.2 Communication Predicate for $U_{T,E,\alpha}$

Following [3], we now define the communication predicate for the $U_{T,E,\alpha}$ 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** $Ute\text{-commPerRd}$

where

$$Ute\text{-commPerRd} \equiv \forall p. \text{card}(\text{HOrs} p - \text{SHOrs} p) \leq \alpha$$

$$\land \text{card}(\text{SHOrs} p \cap \text{HOrs} p) > N + 2*\alpha - E - 1$$

$$\land \text{card}(\text{SHOrs} p \cap \text{HOrs} p) > T$$

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** $Ute\text{-commGlobal}$

where

$$Ute\text{-commGlobal} \equiv \exists \Phi. (\text{let } r = \text{Suc}(nSteps*\Phi)$$

$$\text{in } (\exists \pi. \forall p. \pi = HOs (Suc r) p \land \pi = SHOs (Suc r) p)$$

$$\land (\forall p. \text{card}(SHOs (Suc r) p \cap HOs (Suc r) p) > T)$$

$$\land (\forall p. \text{card}(SHOs (Suc (Suc r)) p \cap HOs (Suc (Suc r)) p) > E))$$

8.3 The $U_{T,E,\alpha}$ Heard-Of Machine

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

**definition** $Ute\text{-SHOMachine}$

where

$$Ute\text{-SHOMachine} = []$$
\[ CinitState = (\lambda p \text{ st crd. } Ute-initState p \text{ st}), \]
\[ sendMsg = Ute-sendMsg, \]
\[ CnextState = (\lambda r p st msgs \text{ crd st}. Ute-nextState r p st msgs st'), \]
\[ SHOcommPerRd = Ute-commPerRd, \]
\[ SHOcommGlobal = Ute-commGlobal \]
\]

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 \( \text{nxt: nextState Ute-M r p (\rho r p) \mu (\rho \text{ (Suc r) p})} \)
and \( \text{vote (\rho \text{ (Suc r) p}) \neq \text{None}} \)
shows \( \text{step r = 0} \)
proof (rule ccontr)
assume \( \text{sr: step r \neq 0} \)
with assms have \( \text{vote (\rho \text{ (Suc r) p}) = \text{None}} \)
  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 \( \text{run: SHORun Ute-M \rho HOs SHOs} \)
and \( \text{d1: decide (\rho r p) \neq \text{Some v}} \)
and \( \text{d2: decide (\rho \text{ (Suc r) p}) = \text{Some v}} \)
shows \( \text{step r \neq 0} \)
proof
assume \( \text{sr: step r = 0} \)
from run obtain \( \mu \text{ where Ute-nextState r p (\rho r p) \mu (\rho \text{ (Suc r) p})} \)
  unfolding Ute-SHOMachine-def nextState-def SHORun-eq SHOnextConfig-eq
  by force
with sr have \( \text{next0 r p (\rho r p) \mu (\rho \text{ (Suc r) p})} \)
  unfolding Ute-nextState-def by auto
hence \( \text{decide (\rho r p) = decide (\rho \text{ (Suc r) p})} \)
  by (auto simp:next0-def)
with \(d1\) \(d2\) show \(\text{False by auto}\)
qed

lemma unique-majority-E:
assumes majv: card \(\{qq::\text{Proc. } F qq = \text{Some } m\}\) > \(E\)
and majw: card \(\{qq::\text{Proc. } F qq = \text{Some } m'\}\) > \(E\)
shows \(m = m'\)
proof –
from majv majw majE
have card \(\{qq::\text{Proc. } F qq = \text{Some } m\}\) > \(N \div 2\)
    and card \(\{qq::\text{Proc. } F qq = \text{Some } m'\}\) > \(N \div 2\)
    by auto
then obtain \(qq\)
    where \(qq \in \{qq::\text{Proc. } F qq = \text{Some } m\}\)
        and \(qq \in \{qq::\text{Proc. } F qq = \text{Some } m'\}\)
        by (rule majoritiesE')
    thus \(?thesis\) by auto
qed

lemma unique-majority-E-\(\alpha\):
assumes majv: card \(\{qq::\text{Proc. } F qq = \text{m}\}\) > \(E - \alpha\)
and majw: card \(\{qq::\text{Proc. } F qq = \text{m}'\}\) > \(E - \alpha\)
shows \(m = m'\)
proof –
from majE alpha-\(\text{lt-N}\) majv majw
have card \(\{qq::\text{Proc. } F qq = \text{m}\}\) > \(N \div 2\)
    and card \(\{qq::\text{Proc. } F qq = \text{m}'\}\) > \(N \div 2\)
    by auto
then obtain \(qq\)
    where \(qq \in \{qq::\text{Proc. } F qq = \text{m}\}\)
        and \(qq \in \{qq::\text{Proc. } F qq = \text{m}'\}\)
        by (rule majoritiesE')
    thus \(?thesis\) by auto
qed

lemma unique-majority-T:
assumes majv: card \(\{qq::\text{Proc. } F qq = \text{Some } m\}\) > \(T\)
and majw: card \(\{qq::\text{Proc. } F qq = \text{Some } m'\}\) > \(T\)
shows \(m = m'\)
proof –
from majT majv majw
have card \(\{qq::\text{Proc. } F qq = \text{Some } m\}\) > \(N \div 2\)
    and card \(\{qq::\text{Proc. } F qq = \text{Some } m'\}\) > \(N \div 2\)
    by auto
then obtain \(qq\)
    where \(qq \in \{qq::\text{Proc. } F qq = \text{Some } m\}\)
        and \(qq \in \{qq::\text{Proc. } F qq = \text{Some } m'\}\)
        by (rule majoritiesE')
    thus \(?thesis\) by auto

102
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 p (rho (Suc r) p)

and mup: mu p ∈ SHOmsgVectors Ute-M r p (rho r) (HO p) (SHO p)

and nxtp: nextState Ute-M r q (rho r q) mu q (rho (Suc r) q)

and muq: mu q ∈ SHOmsgVectors Ute-M r q (rho r) (HO q) (SHO q)

and vp: vote (rho (Suc r) p) = Some vp

and vq: vote (rho (Suc r) q) = Some vq

**shows** vp = vq

using assms proof –

have gtn: card {qq. sendMsg Ute-M r qq p (rho r qq) = Val vp} + card {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq} > N

proof –

have {qq. sendMsg Ute-M r qq p (rho r qq) = Val vp} > T - α ∧ card {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq} > T - α

(is card ?vrcvd p - ?ahop <= ?vrcvd q - ?ahop)

by (auto simp: SHOmsgVectors-def)


moreover

from nxtp vp have stp: step r = 0 by (auto simp: vote-step)

from mup

have {qq. mu p qq = Some (Val vp)} - (HO p - SHO p)

<= {qq. sendMsg Ute-M r qq p (rho r qq) = Val vp}

(is ?vrcvd p - ?ahop <= ?vrcvd q)

by (auto simp: SHOmsgVectors-def)


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 vp have card ?vrcvd p > T

unfolding next0-def by auto

moreover

from mup

have {qq. mu q qq = Some (Val vq)} - (HO q - SHO q)

<= {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq}

(is ?vrcvd q - ?ahoq <= ?vrcvd q)

by (auto simp: SHOmsgVectors-def)


moreover

from nxtp stp have next0 r q (rho r q) mu q (rho (Suc r) q)

by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)

with vq have card {qq. mu q qq = Some (Val vq)} > T
by (unfold next0-def, auto)
moreover
from usafe have card ?ahop ≤ α and card ?ahoq ≤ α
  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 ≠ vq
  have ∀ qq. 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} ∩ {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq} = {}
    by auto
  with gtn
  have card ({qq. sendMsg Ute-M r qq p (rho r qq) = Val vp}
    ∪ {qq. sendMsg Ute-M r qq 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}
    ∪ {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq}) ≤ N
    by (auto simp: card-mono)
  ultimately
  show False by auto
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) ≠ Some v
  and d2: decide (rho (Suc r) p) = Some v
  shows card {q. sendMsg Ute-M r q p (rho r q) = Vote (Some v)} > E − α
proof
  from run obtain μp
    where nxt:nextState Ute-M r p (rho r p) μp (rho (Suc r) p)
    and ∀ qq. qq ∈ HOs r p ↔ μp qq ≠ None
    and ∀ qq. qq ∈ SHOs r p ∩ HOs r p
      → μp qq = Some (sendMsg Ute-M r qq p (rho r qq))
  104
unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq SHOmsgVectors-def by blast

hence \( \{ q. \mu pp qq = \text{Some } (\text{Vote } (\text{Some } v))\} - (\text{HOs } r p - \text{SHOs } r p) \)
\( \subseteq \{ q. \text{sendMsg } Ute-M \text{ } r \text{ } q \text{ } p \text{ } (\rho h \text{ } r \text{ } qq) = \text{Vote } (\text{Some } v)\}\)

(is \( \text{?vrcvdp} - \text{?ahop} \subseteq \text{?vsentp} \)) by auto

hence card (\( \text{?vrcvdp} - \text{?ahop} \)) \( \leq \) card \( \text{?vsentp} \)
and card (\( \text{?vrcvdp} - \text{?ahop} \)) \( \geq \) card \( \text{?vrcvdp} - \text{card ?ahop} \)
by (auto simp: card-mono diff-card-le-card-Diff)

hence card \( \text{?vsentp} \) \( \geq \) card \( \text{?vrcvdp} - \text{card ?ahop} \)
by auto

moreover
from unsafe have card (\( \text{HOs } r p - \text{SHOs } r p \)) \( \leq \) \( \alpha \)
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)

moreover
from run \( \text{d1 d2} \) have \( \text{step } r \neq 0 \) by (rule decide-step)
with \( \text{next1 } r \text{ } p \text{ } (\rho h \text{ } r \text{ } p) \mu p \text{ } (\rho h \text{ } (\text{Suc } r) \text{ } p) \)
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)

with run \( \text{d1 d2} \) have card \( \{ q. \mu pp qq = \text{Some } (\text{Vote } (\text{Some } v))\} \) \( \geq \) \( E \)
unfolding next1-def by auto
ultimately
show \( \text{?thesis} \) 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 \( pp \) correctly receives more than \( \alpha \) such messages.

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

moreover
from unsafe
have card (\( \text{SHOs } (\text{Suc } r) \text{ } pp \cap \text{HOs } (\text{Suc } r) \text{ } pp) \)) \( > \) \( N + 2*\alpha - E - 1 \)
by (auto simp: Ute-SHOMachine-def step-def Ute-commPerRd-def)
ultimately
show \( \text{?thesis} \) using threshold by auto
qed

moreover
have card (\( \text{?msgs pp v} \) + card (\( \text{SHOs } (\text{Suc } r) \text{ } pp \cap \text{HOs } (\text{Suc } r) \text{ } pp) \))
\( = \) card (\( \text{?msgs pp v} \cup (\text{SHOs } (\text{Suc } r) \text{ } pp \cap \text{HOs } (\text{Suc } r) \text{ } pp) \))
If more than \(E - \alpha\) messages holding \(v\) are sent to \(p\) at some round \(r\), then any process \(pp\) will set its \(x\) to value \(v\) in \(r\).

**Lemma common-x-argument-2:**

**Assumes run:** \(\text{SHORun \ Ute-M \ rho \ HOs \ SHOs}\)

**And unsafe:** \(\forall \, r. \, \text{SHOcommPerRd \ Ute-M \ (HOs \ r) \ (SHOs \ r)}\)

**And nxtpp:** \(\text{nextState \ Ute-M \ (Suc \ r) \ pp \ (rho \ (Suc \ r) \ pp)} \ 
\mu_{pp \ rho \ Suc \ r} \ pp \)

**And mupp:** \(\mu_{pp} \in \text{SHOmsgVectors \ Ute-M \ (Suc \ r) \ pp \ (rho \ (Suc \ r)) \ HOs \ (Suc \ r) \ pp} \)

**And threshold:** \(\text{card} \ \{q. \ sendMsg \ Ute-M \ (Suc \ r) \ q \ p \ (rho \ (Suc \ r) \ q) \ 
= \text{Vote} \ (Some \ v)\} > E - \alpha\)

**Shows** \(x \ (rho \ (Suc \ r) \ pp) = v\)

**Proof**

**Assume sr:** \(\text{step} \ (Suc \ r) = 0\)

**Hence** \(\forall \, q. \, \text{sendMsg \ Ute-M \ (Suc \ r) \ q \ p \ (rho \ (Suc \ r) \ q) \ 
= \text{Val} \ (x \ (rho \ (Suc \ r) \ q))}\)

**By** \(\text{auto simp: Ute-SHOMachine-def \ Ute-sendMsg-def send0-def}\)

**Moreover**

**From threshold obtain qq where**

\(\text{sendMsg \ Ute-M \ (Suc \ r) \ qq \ p \ (rho \ (Suc \ r) \ qq) = \text{Vote} \ (Some \ v)}\)

**By force**

**Ultimately**

**Show** \(False\) by **simp**

**Qed**

**Have va:** \(\text{card} \ \{\,qq. \ \mu_{pp \ qq} = \text{Some} \ (\text{Vote} \ (\text{Some} \ v))\} > \alpha\)

**Is card (\text{?msgs} \ v) > \alpha\)

**Proof**

**From mupp**

**Have SHOs \ (Suc \ r) \ pp \ \cap \ \text{HOs} \ (Suc \ r) \ pp \ 
\subseteq \ \{\,qq. \ \mu_{pp \ qq} = \text{Some} \ (\text{sendMsg} \ Ute-M \ (Suc \ r) \ qq \ pp \ (rho \ (Suc \ r) \ qq))\}\}

**Unfolding SHOmsgVectors-def by auto**

**Moreover**

**Hence (\text{?msgs} \ v) \supseteq (\text{?sent} \ pp \ v) \ \cap \ (\text{SHOs} \ (Suc \ r) \ pp \ \cap \ \text{HOs} \ (Suc \ r) \ pp)\)

**By auto**

**Hence** \(\text{card} \ \{\,\text{?msgs} \ v\}\)

\(\geq \ \text{card} \ ((\text{?sent} \ pp \ v) \ \cap \ (\text{SHOs} \ (Suc \ r) \ pp \ \cap \ \text{HOs} \ (Suc \ r) \ pp))\)
by (auto intro: card-mono)
moreover
from unsafe threshold
have alph:card (((?sent pp v) ∩ (SHOs (Suc r) pp ∩ HOs (Suc r) pp)) > α
  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) µpp (rho (Suc (Suc r)) pp)
  by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
ultimately
obtain w where wa:card (?msgs w) > α and xw:x (rho (Suc (Suc r)) pp) = w
  unfolding next1-def by auto
have v = w
proof –
  note unsafe
moreover
obtain qv where qv ∈ SHOs (Suc r) pp and µpp qv = Some (Vote (Some v))
proof –
  have ¬ (?msgs v ⊆ HOs (Suc r) pp − SHOs (Suc r) pp)
  proof
    assume ?msgs v ⊆ HOs (Suc r) pp − SHOs (Suc r) pp
    hence card (?msgs v) ≤ card ((HOs (Suc r) pp) − (SHOs (Suc r) pp))
      by (auto simp: card-mono)
  moreover
  from unsafe
  have card (HOs (Suc r) pp − SHOs (Suc r) pp) ≤ α
    by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
  moreover
  note va
  ultimately
  show False by auto
qed
then obtain qv
where qv ∉ HOs (Suc (Suc r)) pp
  and qv:µpp qv = Some (Vote (Some v))
  by auto
with mupp have qv ∈ SHOs (Suc r) pp
  unfolding SHOmsgVectors-def by auto
with qv 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 qw where
\(qw \in \text{SHOs} (\text{Suc } r) \text{ pp and } \mu \text{pp } qw = \text{Some } (\text{Vote } (\text{Some } w))\)

**proof**
- **have** \(\neg (\text{msgs } w \subseteq \text{HOs} (\text{Suc } r) \text{ pp } - \text{SHOs} (\text{Suc } r) \text{ pp})\)
- **proof**
  - **assume** \(\text{msgs } w \subseteq \text{HOs} (\text{Suc } r) \text{ pp } - \text{SHOs} (\text{Suc } r) \text{ pp}\)
  - **hence** \(\text{card } (\text{msgs } w) \leq \text{card } (\text{HOs} (\text{Suc } r) \text{ pp } - \text{SHOs} (\text{Suc } r) \text{ pp})\)
    - by (auto simp: card-mono)
  - **moreover**
    - from usafe **have** \(\text{card } (\text{HOs} (\text{Suc } r) \text{ pp } - \text{SHOs} (\text{Suc } r) \text{ pp}) \leq \alpha\)
      - by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
  - **moreover**
    - note \(wa\)
    - **ultimately**
      - **show** \(\text{False}\) by auto

**qed**

**then obtain** \(qw\)
- **where** \(qw \notin \text{HOs} (\text{Suc } r) \text{ pp } - \text{SHOs} (\text{Suc } r) \text{ pp}\)
  - **and** \(qw:: \mu \text{pp } qw = \text{Some } (\text{Vote } (\text{Some } w))\)
    - by auto
- **with** \(mupp\) **have** \(qw \in \text{SHOs} (\text{Suc } r) \text{ pp}\)
  - **unfolding** \(\text{SHOmsgVectors-def}\) by auto
  - **with** \(qw\) that **show** \(?\text{thesis}\) by auto

**qed**

**with** \(stp\) \(mupp\) **have** \(\text{vote } (\rho (\text{Suc } r) \text{ qw}) = \text{Some } w\)
- **by** (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send1-def)

**moreover**
- **from** \(\text{run\ obtain}\ \muqv \\muqw\)
  - **where** \(\text{nextState } Ute-M r \text{ qw } ((\rho (\text{Suc } r) \text{ qw}) \quad \muqv (\rho (\text{Suc } r) \text{ qw})\)
    - **and** \(\muqv \in \text{SHOmsgVectors } Ute-M r \quad \text{qv } ((\rho (\text{Suc } r) \text{ qw})\quad (\text{SHOs } r \text{ qw})\)
      - **and** \(\text{nextState } Ute-M r \text{ qw } ((\rho (\text{Suc } r) \text{ qw}) \text{ \muqv (\rho (\text{Suc } r) \text{ qw})}\)
        - **and** \(\muqv \in \text{SHOmsgVectors } Ute-M r \quad \text{qw } ((\rho (\text{Suc } r) \text{ qw}) \quad (\text{SHOs } r \text{ qw})\)
          - **by** (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq) blast
    - **ultimately**
      - **show** \(?\text{thesis}\) using usafe by (auto dest: common-vote)

**qed**

**with** \(xw\) **show** \(x (\rho (\text{Suc } (\text{Suc } r)) \text{ pp}) = v\) by auto

**qed**

Inductive argument for the agreement and validity theorems.

**lemma** safety-inductive-argument:
- **assumes** \(\text{run: SHORun } Ute-M \rho \text{ HOs SHOs}\)
- **and** \(\text{comm: } \forall r. \text{SHOcommPerRd } Ute-M (\text{HOs } r) (\text{SHOs } r)\)
- **and** \(\text{sh: } E - \alpha < \text{card } \{q. \text{sendMsg } Ute-M r' q p (\rho (\text{Suc } r') q) = \text{Vote } (\text{Some } v)\}\)
- **and** \(\text{stp1: step } r' = \text{Suc } 0\)
- **shows** \(E - \alpha < \text{card } \{q. \text{sendMsg } Ute-M (\text{Suc } (\text{Suc } r')) q p (\rho (\text{Suc } (\text{Suc } r')) q) = \text{Vote } (\text{Some } v)\}\)

108
proof –
from \( \text{stp1 have } r' > 0 \) by (auto simp: step-def)
with \( \text{stp1 obtain } r \) where \( \forall r', r'' \subseteq \text{Suc } r \) and \( \text{step:step } (\text{Suc } r) = \text{Suc } 0 \)
by (auto dest: gr0-implies-Suc)

have \( \forall pp. \, x \, (\rho \, (\text{Suc } (\text{Suc } r)) \, pp) = v \)
proof
fix \( pp \)
from \( \text{run obtain } \mu pp \)
where \( \mu pp \in \text{SHOmsgVectors } \text{Ute-M } r' \, pp \, (\rho \, (\text{Suc } r')) \, (\text{SHOs } r') \, pp \)
and \( \text{nextState } \text{Ute-M } r' \, pp \, (\rho \, (\text{Suc } r')) \, pp \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with \( \text{run comm ih rr' show } x \, (\rho \, (\text{Suc } (\text{Suc } r)) \, pp) = v \)
by (auto dest: common-x-argument-2)
qed

with \( \text{run stpr} \)
have \( \forall pp \, p. \, \text{sendMsg } \text{Ute-M } (\text{Suc } (\text{Suc } r)) \, pp \, p \, (\rho \, (\text{Suc } (\text{Suc } r)) \, pp) = \text{Val } v \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
\text{Ute-sendMsg-def send0-def mod-Suc step-def})

with \( \text{rr'} \)
have \( \forall p \, \mu pp', \, \mu pp' \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \, p \, (\rho \, (\text{Suc } r')) \)
\( \text{(HOs } (\text{Suc } r') \, p \, (\text{SHOs } (\text{Suc } r') \, p) \)
\( \subseteq \{ q. \, \mu pp' \, q = \text{Some } (\text{Val } \text{v}) \} \)
by (auto simp: SHOmsgVectors-def)

hence \( \forall p \, \mu pp', \, \mu pp' \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \, p \, (\rho \, (\text{Suc } r')) \)
\( \text{(HOs } (\text{Suc } r') \, p \, (\text{SHOs } (\text{Suc } r') \, p) \)
\( \subseteq \text{card } \{ q. \, \mu pp' \, q = \text{Some } (\text{Val } \text{v}) \} \)
by (auto simp: card-mono)
moreover
from \( \text{comm have } \forall p. \, T < \text{card } (\text{SHOs } (\text{Suc } r') \, p \, \cap \, \text{HOs } (\text{Suc } r') \, p) \)
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
ultimately
have \( \forall vT. \forall p \, \mu pp', \, \mu pp' \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \, p \, (\rho \, (\text{Suc } r')) \)
\( \text{(HOs } (\text{Suc } r') \, p \, (\text{SHOs } (\text{Suc } r') \, p) \)
\( \Rightarrow \, T < \text{card } \{ q. \, \mu pp' \, q = \text{Some } (\text{Val } \text{v}) \} \)
by (auto dest: less-le-trans)

show \( \text{thesis} \)
proof –
have \( \forall pp. \, \text{vote } ((\rho \, (\text{Suc } (\text{Suc } r'))) \, pp) = \text{Some } v \)
proof
fix \( pp \)
from \( \text{run obtain } \mu pp \)
where \( \text{nextState } \text{Ute-M } (\text{Suc } r') \, pp \, (\rho \, (\text{Suc } r') \, pp) \, \mu pp \)
\( (\rho \, (\text{Suc } (\text{Suc } r')) \, pp) \)
and \( \mu pp \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \, pp \, (\rho \, (\text{Suc } r')) \)
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** \( \exists \, m < n . \, \text{decide (rho (Suc m) p) \neq \text{decide (rho m p)}} \wedge \text{decide (rho (Suc m) p) = Some v} \)

**Proof**

- let \( ?\text{dec k} = \text{decide ((rho k) p)} \)
- have \( (\forall \, m < n . \, ?\text{dec (Suc m)} \neq ?\text{dec (Suc m) \neq Some v}) \) \rightarrow \( ?\text{dec n} \neq \text{Some v} \)
  - (is ?P n is ?A n \rightarrow -)
- proof (induct n)
  - from run show \( ?\text{P} 0 \)
    - by (auto simp: Ute-SHOMachine-def SHORun-eq HOinitConfig-eq initState-def Ute-initState-def)

**Next**

- fix \( n \)
- assume \( ih: ?\text{P} n \) thus \( ?\text{P} (\text{Suc} n) \) by force

**Qed**

with \( \text{dec} \) show \( ?\text{thesis} \) 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:∀ $r$. SHOcommPerRd Ute-M (HOs $r$) (SHOs $r$)
and $dv_1$ : decide ($\rho$ (Suc $r$) $p_1$) = Some $v_1$
and $dn_2$ : decide ($\rho$ ($r + k$) $p_2$) ≠ Some $v_2$
and $dv_2$ : decide ($\rho$ (Suc ($r + k$)) $p_2$) = Some $v_2$

**shows** $v_2 = v_1$

**proof**

- from run $dv_1$ obtain $r_1$
  where $r r_1 < $ Suc $r$
  and $dn_1$ : decide ($\rho$ $r_1$ $p_1$) ≠ Some $v_1$
  and $dv_1'$ : decide ($\rho$ (Suc $r_1$) $p_1$) = Some $v_1$
  by (auto dest: decisionNonNullThenDecided)

- from $r r_1$ obtain $s$ where $r r_1$:Suc $r = $ Suc ($r_1 + s$)
  by (auto dest: less-imp-Suc-add)
- then obtain $k'$ where $kk':r + k = r_1 + k'$
  by auto
  with $dn_2$ $dv_2$
  have $dn_2'$ : decide ($\rho$ ($r_1 + k'$) $p_2$) ≠ Some $v_2$
  and $dv_2'$ : decide ($\rho$ (Suc ($r_1 + k'$)) $p_2$) = Some $v_2$
  by auto

- from run $dn_1$ $dv_1'$ $dn_2'$ $dv_2'$
  have $rs_0$ : step $r_1 = $ Suc $0$
  and $rks_0$ : step ($r_1 + k'$) = Suc $0$
  by (auto simp: mod-Suc step-def dest: decide-step)

  have $step (r_1 + k') = step (step r_1 + k')$
  unfolding $step-def$ by (rule mod-add-left-eq)
  with $rs_0$ $rks_0$ have step $k' = 0$
  by (auto simp: step-def mod-Suc)
  then obtain $k''$ where $k' = k''*nSteps$
  by (auto simp: step-def)
  with $dn_2'$ $dv_2'$
  have $dn_2''$:decide ($\rho$ ($r_1 + k''*nSteps$) $p_2$) ≠ Some $v_2$
  and $dv_2'':$decide ($\rho$ (Suc ($r_1 + k''*nSteps$)) $p_2$) = Some $v_2$
  by auto

- from $rs_0$
  have $stp:step (r_1 + k''*nSteps) = $ Suc $0$
  unfolding $step-def$ by auto

  have $inv:card \{ q. sendMsg Ute-M (r_1 + k''*nSteps) q p_1 (\rho (r_1 + k''*nSteps)) q \}$
  = Vote (Some $v_1$) > $E - \alpha$

  **proof** (induct $k''$
  from $stp$
  have $step (r_1 + 0*nSteps) = $ Suc $0$
  by (auto simp: step-def)
  from run comm $dn_1$ $dv_1'$
  show $card \{ q. sendMsg Ute-M (r_1 + 0*nSteps) q p_1 (\rho (r_1 + 0*nSteps)) q \}$
= Vote (Some v1)} > E - α

by (intro decide-with-threshold-E) auto

next

fix k''

assume ih: E - α <

\[ \text{card } \{ q. \text{sendMsg Ute-M (r1 + k''*nSteps)} q p1 (\rho (r1 + k''*nSteps)) q \} = Vote (Some v1) \]

from rs0 have steps: step (r1 + k''*nSteps) = Suc 0

by (auto simp: step-def)

with run comm ih

have E - α <

\[ \text{card } \{ q. \text{sendMsg Ute-M (Suc (Suc (r1 + k''*nSteps))) q p1 (\rho (Suc (Suc (r1 + k''*nSteps))) q) = Vote (Some v1)} \}

by (rule safety-inductive-argument)

thus E - α <

\[ \text{card } \{ q. \text{sendMsg Ute-M (r1 + Suc k'' * nSteps) q p1 (\rho (r1 + Suc k'' * nSteps) q) = Vote (Some v1)} \}

by auto

qed

moreover

from run

have ∀ q. \text{sendMsg Ute-M (r1 + k''*nSteps) q p1 (\rho (r1 + k''*nSteps) q) = sendMsg Ute-M (r1 + k''*nSteps) q p2 (\rho (r1 + k''*nSteps) q) by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def step-def send0-def send1-def)}

moreover

from run comm dn2'' dv2''

have E - α <

\[ \text{card } \{ q. \text{sendMsg Ute-M (r1 + k''*nSteps) q p2 (\rho (r1 + k''*nSteps) q) = Vote (Some v2)} \}

by (auto dest: decide-with-threshold-E)

ultimately

show v2 = v1 by (auto dest: unique-majority-E-α)

qed

The Agreement property is an immediate consequence of the two preceding lemmas.

\textbf{theorem ute-agreement:}

\textbf{assumes} run: SHORun Ute-M rho HOs SHOs

\textbf{and} comm: \forall r. \text{SHOCommPerRd Ute-M (HOs r) (SHOs r)}

\textbf{and} p: decide (\rho m p) = Some v

\textbf{and} q: decide (\rho n q) = Some w

\textbf{shows} v = w

\textbf{proof} –

from run p obtain k

\textbf{where} k1: decide (\rho (Suc k) p) \neq decide (\rho k p)
and \( k_2 \): decide \((\rho (\text{Suc} k)) p\) = Some \( v \)
by (auto dest: decisionNonNullThenDecided)

from run \( q \) obtain \( l \)
 where \( l_1 \): decide \((\rho (\text{Suc} l)) q\) \( \neq \) decide \((\rho l q)\)
 and \( l_2 \): decide \((\rho (\text{Suc} l)) q\) = Some \( w \)
by (auto dest: decisionNonNullThenDecided)

show \(?thesis\)
proof (cases \( k \leq l \))
  case True
  then obtain \( m \) where \( m \): \( l = k + m \)
by (auto simp add: le_iff_add)
from run \( k_1 l_1 l_2 m \)

have \( w = v \)
by (auto elim!: laterProcessDecidesSameValue)
thus \(?thesis\) by simp

next
  case False
  hence \( l \leq k \)
by simp
then obtain \( m \)
where \( m \): \( k = l + m \)
by (auto simp add: le_iff_add)
from run \( k_2 l_2 k_1 k_2 m \)

show \(?thesis\)
by (auto elim!: laterProcessDecidesSameValue)
qed

qed

Main lemma for the proof of the Validity property.

lemma validity-argument:
assumes run: \( \text{SHORun Ute-M} \ \rho \ \text{HOs} \ \text{SHOs} \)
and comm: \( \forall \ r. \ \text{SHOcommPerRd Ute-M} (\text{HOs} r) (\text{SHOs} r) \)
and init: \( \forall \ x. ((\rho 0) p) = v \)
and dw: \( \text{decide} (\rho r p) = \text{Some} \ w \)
and stp: \( \text{step} r' = \text{Suc} 0 \)
shows \( \text{card} \{ q. \ \text{sendMsg Ute-M} r' q p (\rho r q) = \text{Vote} (\text{Some} v)\} > \alpha - E \)
proof –
from stp obtain \( k \) where stp: \( r' = \text{Suc} 0 + k * nSteps \)
  unfolding step-def using mod-Suc mod-eqD by blast
moreover
have \( E - \alpha < \)
  \( \text{card} \{ q. \ \text{sendMsg Ute-M} (\text{Suc} 0 + k*nSteps) q p ((\rho (\text{Suc} 0 + k*nSteps)) q) = \text{Vote} (\text{Some} v)\} \)
proof (induct \( k \))
  have \( \forall \ pp. \ \text{vote} ((\rho (\text{Suc} 0)) pp) = \text{Some} \ v \)
  proof
    fix \( pp \)
    from run obtain \( \mu pp \)
      where \( \text{nextpp:nextState Ute-M} 0 pp (\rho 0 pp) \mu pp (\rho (\text{Suc} 0) pp) \)
      and \( \mu pp: \mu pp \in \text{SHOmsgVectors Ute-M} 0 pp (\rho 0) (\text{HOs} 0 pp) (\text{SHOs} 0 pp) \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
  have \( \text{majv:card} \{ q. \ \mu pp q = \text{Some} (\text{Val} v)\} > T \)
  proof –
from run init have \( \forall q. \text{sendMsg Ute-M 0 q pp (rho 0 q)} = \text{Val v} \)
   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 \( \cap \) HOs 0 pp) > T
   by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)

moreover
from mupp have SHOs 0 pp \( \cap \) HOs 0 pp \( \subseteq \) \{q. \mu pp q = \text{Some (sendMsg Ute-M 0 q pp (rho 0 q))}\}
   by (auto simp: Ute-SHOMachine-def)

hence \( \text{card (SHOs 0 pp \( \cap \) HOs 0 pp)} \leq \text{card (q. \mu pp q = \text{Some (sendMsg Ute-M 0 q pp (rho 0 q))}}\}
   by (auto simp: card-mono)

ultimately
show \( \text{thesis} \) by (auto simp: less-le-trans)

qed

moreover
from nxtpp have next0 0 pp ((\( \rho \) 0) pp) \( \mu \) pp (\( \rho \) (Suc 0) pp)
   by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def step-def)

ultimately
obtain w where majw:card \{q. \mu pp q = Some (Val w)} > T
   by (auto simp: next0-def)

with votew show vote ((\( \rho \) (Suc 0)) pp) = Some v by simp

with run
have card \{q. \text{sendMsg Ute-M (Suc 0) q p (rho (Suc 0) q)} = \text{Vote (Some v)}\} = N
   by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
   Ute-sendMsg-def send1-def)

thus \( E - \alpha < \)

    \[ \text{card (q. \text{sendMsg Ute-M (Suc 0 + 0 * nSteps) q p (rho (Suc 0 + 0 * nSteps) q}) = Vote (Some v)}\] 

    using majE EltN by auto

next
fix k
assume ih:E - \alpha < 

    \[ \text{card (q. \text{sendMsg Ute-M (Suc 0 + k * nSteps) q p (rho (Suc 0 + k * nSteps) q}) = Vote (Some v)}\] 

have step (Suc 0 + k * nSteps) = Suc 0
   by (auto simp: mod-Suc step-def)

from run comm ih this
have E - \alpha < 

    \[ \text{card (q. \text{sendMsg Ute-M (Suc (Suc 0 + k * nSteps))) q p}}\]
\[(\text{rho} \ (\text{Suc} \ (\text{Suc} \ 0 + k \ast n\text{Steps}))) \ q)\]

\[\text{by (rule safety-inductive-argument)}\]

thus \(E - \alpha <\)

\[\text{card} \ \{q. \text{sendMsg} \ Ute-M \ (\text{Suc} \ 0 + \text{Suc} \ k \ast n\text{Steps}) \ q \ p \ (\text{rho} \ (\text{Suc} \ 0 + \text{Suc} \ k \ast n\text{Steps}) \ q)\] = \(\text{Vote} \ (\text{Some} \ v)\) \ by simp

qed

ultimately show \(\bar{\text{thesis}}\) by simp

qed

The following theorem shows the Validity property of algorithm \(\text{UT}_E,\alpha\).

**Theorem ute-validity:**

assumes \(\text{run}:\ \text{SHORun} \ Ute-M \ \text{rho} \ \text{HOs} \ \text{SHOs}\)

and \(\text{comm}: \forall r. \ \text{SHOcommPerRd} \ Ute-M \ (\text{HOs} \ r) \ (\text{SHOs} \ r)\)

and \(\text{init}: \forall p, x. \ (\text{rho} \ 0 \ p) = v\)

and \(\text{dw}: \text{decide} \ (\text{rho} \ r \ p) = \text{Some} \ w\)

shows \(v = w\)

proof

from \(\text{run} \ \text{dw} \ \text{obtain} \ r1\)

where \(\text{dnr1}: \text{decide} \ ((\text{rho} \ r1) \ p) \neq \text{Some} \ w\)

and \(\text{dwr1}: \text{decide} \ ((\text{rho} \ (\text{Suc} \ r1)) \ p) = \text{Some} \ w\)

by (force dest: decisionNonNullThenDecided)

with \(\text{run} \ \text{have} \ \text{step} \ r1 \neq 0 \ \text{by (rule decide-step)}\)

hence \(\text{step} \ r1 = \text{Suc} \ 0\) by (simp add: step-def modSuc)

with \(\text{assms}\)

have \(E - \alpha <\)

\[\text{card} \ \{q. \text{sendMsg} \ Ute-M \ r1 \ q \ p \ (\text{rho} \ r1 \ q) = \text{Vote} \ (\text{Some} \ v)\}\]

by (rule validity-argument)

moreover

from \(\text{run} \ \text{comm} \ \text{dnr1} \ \text{dwr1}\)

have \(\text{card} \ \{q. \text{sendMsg} \ Ute-M \ r1 \ q \ p \ (\text{rho} \ r1 \ q) = \text{Vote} \ (\text{Some} \ w)\} > E - \alpha\)

by (auto dest: decide-with-threshold-E)

ultimately

show \(v = w\) by (auto dest: unique-majority-E-\(\bar{\alpha}\))

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 \(\text{run}:\ \text{SHORun} \ Ute-M \ \text{rho} \ \text{HOs} \ \text{SHOs}\)

and \(\text{comm}: \ \text{SHOcommPerRd} \ Ute-M \ (\text{HOs} \ r) \ (\text{SHOs} \ r)\)

and \(\text{stp}: \text{step} \ (\text{Suc} \ r) = \text{Suc} \ 0\)

and \(\pi: \forall p. \ \text{HOs} \ (\text{Suc} \ r) \ p = \text{SHOs} \ (\text{Suc} \ r) \ p\)
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** \( \text{run}: \text{SHORun Ute-M \( \rho \) HOs SHOs} \)

**and** \( \text{comm}: \text{SHOcommPerRd Ute-M (HOs r) (SHOs r)} \)

**and** \( \pi: \forall p. \pi_0 = \text{HOs (Suc r) p} \land \pi_0 = \text{SHOs (Suc r) p} \)

**obtains** \( v \) where \( \wedge p. \mu p \ q. \)

\[
\begin{aligned}
\exists q \in \text{SHOs (Suc (Suc r)) p} \cap \text{HOs (Suc (Suc r)) p}; \\
\mu' \in \text{SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r)))} \\
(\text{HOs (Suc (Suc r)) p} \cap \text{SHOs (Suc (Suc r)) p}) \\
\Rightarrow \mu' q = (\text{Some (Val v)})
\end{aligned}
\]

**Proof** —

**From** \( \pi \) have \( \text{hosho:} \forall p. \text{SHOs (Suc r) p} = \text{SHOs (Suc r) p} \cap \text{HOs (Suc r) p} \)

**By** \( \text{simp} \)

**Have** \( \wedge p. \ q. x \ (\text{rho (Suc (Suc r)) p}) = x \ (\text{rho (Suc (Suc r)) q}) \)

**Proof** —

**Fix** \( p \ q \)

**From** \( \text{run} \) obtain \( \mu p \)

**Where** \( \text{nxt}: \text{nextState Ute-M (Suc r) p (rho (Suc r) p)} \)

\( \mu p \ (\text{rho (Suc (Suc r)) p}) \)

**And** \( \mu: \mu p \in \text{SHOmsgVectors Ute-M (Suc r) p (rho (Suc r))} \)

\( (\text{HOs (Suc r) p} \cap \text{SHOs (Suc r) p}) \)

**By** \( \text{auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq} \)

**From** \( \text{run} \) obtain \( \mu q \)

**Where** \( \text{nxtq}: \text{nextState Ute-M (Suc r) q (rho (Suc r) q)} \)

\( \mu q \ (\text{rho (Suc (Suc r)) q}) \)

**And** \( \muq: \mu q \in \text{SHOmsgVectors Ute-M (Suc r) q (rho (Suc r))} \)

\( (\text{HOs (Suc r) q} \cap \text{SHOs (Suc r) q}) \)

**By** \( \text{auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq} \)

**Have** \( \forall qq. \mu p qq = \mu q qq \)

**Proof** —

**Fix** \( qq \)

**Show** \( \mu p qq = \mu q qq \)

**Proof** \( \text{(cases } \mu p qq = \text{None}) \)

**Case** \( \text{False} \)

**With** \( \muu \pi \) have \( 1:qq \in \text{SHOs (Suc r) p} \) and \( 2:qq \in \text{SHOs (Suc r) q} \)

**Unfolding** \( \text{SHOmsgVectors-def} \) **By** \( \text{auto} \)

**From** \( \muu \pi \) \( 1 \)

**Have** \( \mu p qq = \text{Some (sendMsg Ute-M (Suc r) qq p (rho (Suc r) qq))} \)

**Unfolding** \( \text{SHOmsgVectors-def} \) **By** \( \text{auto} \)

**Moreover**

**From** \( \muq \pi \) \( 2 \)

**Have** \( \mu q qq = \text{Some (sendMsg Ute-M (Suc r) qq q (rho (Suc r) qq))} \)

**Unfolding** \( \text{SHOmsgVectors-def} \) **By** \( \text{auto} \)
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 ∉ HOs (Suc r) p unfolding SHOmsgVectors-def by auto
with π muq have µ qq = None unfolding SHOmsgVectors-def by auto
with True show ?thesis by simp
qed
qed

hence vsets: ∀ v. { qq. µ p qq = Some (Vote (Some v))}
= { qq. µq qq = Some (Vote (Some v))}
by auto

show x (ρ (Suc (Suc r)) p) = x (ρ (Suc (Suc r)) q)

proof (cases ∃ v. α < card { qq. µ p qq = Some (Vote (Some v))}, clarify)
fix v
assume vp: α < card { qq. µ p qq = Some (Vote (Some v))} by auto
with run comm stp π nxt mu have x (ρ (Suc (Suc r)) p) = v
  by (auto dest: set-x-from-vote)

moreover
from vsets vp have α < card { qq. µq qq = Some (Vote (Some v))} by auto
with run comm stp π nxtq muq have x (ρ (Suc (Suc r)) q) = v
  by (auto dest: set-x-from-vote)

ultimately
show x (ρ (Suc (Suc r)) p) = x (ρ (Suc (Suc r)) q)
by auto

next
assume nov: ¬ (∃ v. α < card { qq. µ p qq = Some (Vote (Some v))})
with nxt stp have x (ρ (Suc (Suc r)) p) = undefined
  by (auto simp: Ute-SHOMachine-def nextState-def
    Ute-nextState-def next1-def)

moreover
from vsets nov have ¬ (∃ v. α < card { qq. µq qq = Some (Vote (Some v))}) by auto
with nxtq stp have x (ρ (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 where ∃ q. x (ρ (Suc (Suc r)) q) = v by blast

moreover
from stp have step (Suc (Suc r)) = 0
by (auto simp: step-def mod-Suc)
hence \( \wedge p \, \mu p' \, q \).
\[
[q \in SHOs (Suc (Suc r)) \, p \cap HOs (Suc (Suc r)) \, p; \\
\mu p' \in SHOmsgVectors Ute-M (Suc (Suc r)) \, p (\rho (Suc (Suc r))) \\
(HOs (Suc (Suc r)) \, p) (SHOs (Suc (Suc r)) \, p) \\
] \implies \mu p' \, q = \text{Some} (\text{Val} (x (\rho (Suc (Suc r)) \, q))) \\
\text{by} (\text{auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send0-def})
\]
ultimately
have \( \wedge p \, \mu p' \, q \).
\[
[q \in SHOs (Suc (Suc r)) \, p \cap HOs (Suc (Suc r)) \, p; \\
\mu p' \in SHOmsgVectors Ute-M (Suc (Suc r)) \, p (\rho (Suc (Suc r))) \\
(HOs (Suc (Suc r)) \, p) (SHOs (Suc (Suc r)) \, p) \\
] \implies \mu p' \, q = (\text{Some} (\text{Val} v)) \\
\text{by auto}
\]
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 \( \mu p \in SHOmsgVectors Ute-M (Suc (Suc r)) \, p (\rho (Suc (Suc r))) \\
(HOs (Suc (Suc r)) \, p) (SHOs (Suc (Suc r)) \, p) \)
and \( \text{nxtq: nextState Ute-M r q (\rho r q) \, \mu q (\rho (Suc r) \, q) \) \)
and \( \text{vq: vote (\rho (Suc r) \, q) = Some v} \)
and \( \text{qsho: q \in SHOs (Suc r) \, p \cap HOs (Suc r) \, p} \)
shows \( \mu p \, q = \text{Some} (\text{Vote (Some v)}) \)
proof –
from \( \text{nxtq vq have step r = 0 by (auto simp: vote-step)} \)
with \( \text{map qsho have \mu p \, q = Some (Vote (\text{vote (\rho (Suc r) \, q)}) \))} \)
\text{by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def step-def send1-def mod-Suc)}
with \( \text{vq show \mu p \, q = Some (\text{Vote (Some v)}) by auto} \)
qed

We now prove the Termination property.

theorem ute-termination:
assumes \( \text{run: SHORun Ute-M \, \rho \, HOs SHOs} \)
and \( \text{commR: \forall r. \, SHOcommPerRd Ute-M (HOs \, r) (HOs \, r)} \)
and \( \text{commG: \, SHOcommGlobal Ute-M \, HOs \, SHOs} \)
shows \( \exists v \, r. \, \text{decide (\rho r \, p) = Some v} \)
proof –
from \( \text{commG} \)
\text{obtain \Phi \, \pi \, r0} \\
\text{where \( rr: r0 = \text{Suc (nSteps \ast \Phi)} \) } \\
\text{and \( \pi: \forall p. \, \pi = HOs r0 \, p \wedge \pi = HOs r0 \, p \) } \\
\text{and \( t: \forall p. \, \text{card (HOs (Suc r0) \, p \cap HOs (Suc r0) \, p)) > T} \) } \\
\text{and \( e: \forall p. \, \text{card (HOs (Suc (Suc r0)) \, p \cap HOs (Suc (Suc r0)) \, p)) > E} \) } \\
\text{by (auto simp: Ute-SHOMachine-def Ute-commGlobal-def Let-def)}
from \( \text{rr have stp:step r0 = Suc t by (auto simp: step-def)} \)
obtain \( w \) where \( v o t e w \vdash \forall p. \ (v o t e \ (r h o \ (S u c \ (S u c \ r 0)) \ p)) = \text{Some} \ w \)

proof
  have \( a b c \vdash \forall w. \ v o t e \ (r h o \ (S u c \ (S u c \ r 0))) \ p) = \text{Some} \ w \)
  proof
    fix \( p \)
    from \( r u n \ s t p \) obtain \( \mu p \)
      where \( n x t \cdot n e x t S t a t e \ U t e - M \ (S u c \ r 0) \ p \ (r h o \ (S u c \ r 0)) \ \mu p \ (r h o \ (S u c \ (S u c \ r 0))) \ p) \)
      and \( m u p : \mu p \in S H O m s g V e c t o r s \ U t e - M \ (S u c \ r 0) \ p \ p (r h o \ (S u c \ r 0))) \ (H O s \ (S u c \ r 0) \ p) \ (S H O s \ (S u c \ r 0) \ p) \)
      by \( (a u t o \ s i m p : \text{Ute-SHOMachine-def} \ S H O R u n - e q \ S H O n e x t C o n f i g - e q) \)
  have \( \exists v. \ T < \text{card} \ \{q q. \ \mu p \ q q = \text{Some} \ (V a l \ v)\} \)
  proof
    from \( t \) have \( \text{card} \ (S H O s \ (S u c \ r 0)) \ p \cap H O s \ (S u c \ r 0)) \ p) > T .. \)
    moreover
    from \( r u n \ c o m m R \ s t p \ \pi \ \rho r \)
    obtain \( v \) where
      \( \forall p \ \mu p : \mu p \in S H O m s g V e c t o r s \ U t e - M \ (S u c \ r 0) \ p \ p (r h o \ (S u c \ r 0))) \ (H O s \ (S u c \ r 0) \ p) \ (S H O s \ (S u c \ r 0) \ p) \)
      \( \mu p \ q \Rightarrow \mu p \ q \ q = \text{Some} \ (V a l \ v) \)
      using \( \text{termination-argument-1} \) by \( b l a s t \)
    with \( m u p \) obtain \( v \) where
      \( \forall q q. \ q q \in S H O s \ (S u c \ r 0)) \ p \cap H O s \ (S u c \ r 0)) \ p) \Rightarrow \mu p \ q q = \text{Some} \ (V a l \ v) \)
      by \( a u t o \)
      hence \( S H O s \ (S u c \ r 0)) \ p \cap H O s \ (S u c \ r 0)) \ p) \subseteq \{q q. \ \mu p \ q q = \text{Some} \ (V a l \ v)\} \)
      by \( a u t o \)
      hence \( \text{card} \ \{q q. \ \mu p \ q q = \text{Some} \ (V a l \ v)\} \)
      \( \leq \text{card} \ \{q q. \ \mu p \ q q = \text{Some} \ (V a l \ v)\} \)
      by \( (a u t o \ i n t r o : \text{card-mono}) \)
    ultimately
    have \( T < \text{card} \ \{q q. \ \mu p \ q q = \text{Some} \ (V a l \ v)\} \) by \( a u t o \)
    thus \( ? t h e s i s \) by \( a u t o \)
  qed
with \( s t p \) obtain \( w \) where
  \( \forall q q. \ q q \in S H O s \ (S u c \ r 0)) \ p \cap H O s \ (S u c \ r 0)) \ p) \Rightarrow \mu p \ q q = \text{Some} \ (V a l \ v) \)
  by \( (a u t o \ s i m p : \text{Ute-SHOMachine-def} \ n e x t S t a t e - d e f \ U t e - n e x t S t a t e - d e f \ n e x t S t e p - d e f \ m o d - S u c \ n e x t 0 - d e f) \)
  qed
then obtain \( q q \) \( w \) where \( q q w : v o t e \ (r h o \ (S u c \ (S u c \ r 0))) \ q q) = \text{Some} \ w \)
  by \( b l a s t \)
have \( \forall p p. \ v o t e \ (r h o \ (S u c \ (S u c \ r 0))) \ p p) = \text{Some} \ w \)
proof
fix \( p p \)
from \( a b c \) obtain \( w p \) where \( p w p : v o t e \ (\ (r h o \ (S u c \ (S u c \ r 0))) \ p p) = \text{Some} \ w p \)
by blast
from run obtain μpp μqq
  where nxtq: nextState Ute-M (Suc r0) pp (rho (Suc r0)) pp
       μpp (rho (Suc (Suc r0))) pp
  and mup: μpp ∈ SHOMsgVectors Ute-M (Suc r0) pp (rho (Suc r0))
      (HOs (Suc r0)) pp (SHOs (Suc r0)) pp
  and nxtq: nextState Ute-M (Suc r0) qq (rho (Suc r0)) qq
       μqq (rho (Suc (Suc r0))) qq
  and mup: μqq ∈ 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 wp = w
by (auto dest: common-vote)
with pwp show vote ((rho (Suc (Suc r0))) pp) = Some w
by auto
qed
with that show ?thesis by auto
qed

from run obtain μp'
  where nxtq: nextState Ute-M (Suc (Suc r0)) p (rho (Suc (Suc r0))) p
       μp' (rho (Suc (Suc (Suc r0)))) p
  and mup': μp' ∈ SHOMsgVectors Ute-M (Suc (Suc r0)) p (rho (Suc (Suc r0)))
       (HOs (Suc (Suc r0))) p (SHOs (Suc (Suc r0))) p
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
have ∀qq, qq ∈ SHOs (Suc (Suc r0)) p ∩ HOs (Suc (Suc r0)) p
  ⇒ μp' qq = Some (Vote (Some w))
proof –
  fix qq
  assume qqsho: qq ∈ SHOs (Suc (Suc r0)) p ∩ HOs (Suc (Suc r0)) p
from run obtain μqq where
  nxtqq: nextState Ute-M (Suc r0) qq (rho (Suc r0)) qq
       μqq (rho (Suc (Suc r0))) qq
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
from commR mup' nxtqq votew qqsho show μp' qq = Some (Vote (Some w))
by (auto dest: termination-argument-2)
qed
hence SHOs (Suc (Suc r0)) p ∩ HOs (Suc (Suc r0)) p
  ⊆ {qq, μp' qq = Some (Vote (Some w))}
by auto
hence wsho: card (SHOs (Suc (Suc r0)) p ∩ HOs (Suc (Suc r0)) p)
  ≤ card {qq, μp' qq = Some (Vote (Some w))}
by (auto simp: card-mono)

from stp have step (Suc (Suc r0)) = Suc θ
unfolding step-def by auto
with nxtp have next1 (Suc (Suc r0)) p (rho (Suc (Suc r0))) p μp'
  (rho (Suc (Suc (Suc r0)))) p
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
moreover
from e have E < card (SHOs (Suc (Suc r0))) p ∩ HOs (Suc (Suc r0)) p
  by auto
with wsho have majv: card {qq. pp' qq = Some (Vote (Some w))} > E
  by auto
ultimately
show thesis by (auto simp: next1-def)
qed

8.7 $U_{T,E,\omega}$ Solves Weak Consensus

Summing up, all (coarse-grained) runs of $U_{T,E,\omega}$ 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: \forall r. SHOcommPerRd Ute-M (HOs r) (SHOs r)
  and commG: SHOcommGlobal Ute-M HOs SHOs
shows weak-consensus (x ◦ (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 ◦ (crun 0)) decide crun
  by (rule ute-weak-consensus)
with init show weak-consensus ?inits decide crun
  by (simp add: o-def)
qed

end — context ute-parameters
9 Verification of the $A_{T,E,\alpha}$ Consensus algorithm

Algorithm $A_{T,E,\alpha}$ is presented in [3]. Like $U_{T,E,\alpha}$, 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 $U_{T,E,\alpha}$, allowing the algorithm to be adapted to the characteristics of different systems. $A_{T,E,\alpha}$ 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.

```plaintext
typedesc Proc — the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
```

abbreviation

$$N \equiv \text{card (UNIV::Proc set)}$$ — number of processes

The following record models the local state of a process.

```plaintext
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.

```plaintext
definition Ate-initState where
  Ate-initState p st \equiv (decide st = None)
```

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

```plaintext
locale ate-parameters =
  fixes $\alpha :: \text{nat}$ and $T :: \text{nat}$ and $E :: \text{nat}$
  assumes TNaE:T \geq 2*(N + 2*\alpha - E)
  and TltN:T < N
  and EltNE < N

begin
```

123
The following are consequences of the assumptions on the parameters.

**Lemma** $\text{majE}$: $2 \cdot (E - \alpha) \geq N$
*using* $TNaE TltN$ *by* auto

**Lemma** $\text{EgtE}$: $E > \alpha$
*using* $\text{majE EltN}$ *by* auto

**Lemma** $\text{Tge2a}$: $T \geq 2 \cdot \alpha$
*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 $\text{OneThirdRule}$, 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** $\text{mostOftenRcvd}$ where

$$\text{mostOftenRcvd} \,(\text{msgs :: Proc} \Rightarrow \text{val option}) \equiv \{v. \forall w. \text{card}\{qq. \text{msgs} qq = \text{Some} \, w\} \leq \text{card}\{qq. \text{msgs} qq = \text{Some} \, v\}\}$$

**Definition** $\text{Ate-sendMsg}$ :: $\text{nats} \Rightarrow \text{Procs} \Rightarrow \text{Procs} \Rightarrow \text{pstate} \Rightarrow \text{pstate}$

*where*

$\text{Ate-sendMsg} \, r \, p \, q \, st \equiv x \, st$

**Definition** $\text{Ate-nextState}$ :: $\text{nats} \Rightarrow \text{Procs} \Rightarrow (\text{val} :: \text{linorder}) \Rightarrow \text{pstate} \Rightarrow \text{pstate} \Rightarrow \text{bool}$

*where*

$\text{Ate-nextState} \, r \, p \, st \, msgs \, st' \equiv x \, st$$

(if card $\{q. \text{msgs} q \neq \text{None}\} > T$
then $x \, st' = \text{Min} \,(\text{mostOftenRcvd} \, msgs)$
else $x \, st' = \, x \, st$)
∧ ( (\exists v. \text{card}\{q. \text{msgs} q = \text{Some} \, v\} > E \land \text{decide} \, st' = \text{Some} \, v\)
∨ ¬(\exists v. \text{card}\{q. \text{msgs} q = \text{Some} \, v\} > E)
∧ \text{decide} \, st' = \text{decide} \, st\)

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

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

**Definition** $\text{Ate-commPerRd}$ where

$\text{Ate-commPerRd} \, HOrs \, SHOrs \equiv \forall p. \text{card}\,(\text{HOrs} \, p - \text{SHOrs} \, p) \leq \alpha$

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 SHOs ≡
(\( \forall r \ p \ \exists r' > r. \ \text{card} (\text{HOs} \ r' \ p) > T \))
\& (\( \forall r \ p \ \exists r' > r. \ \text{card} (\text{SHOs} \ r' \ p \cap \text{HOs} \ r' \ p) > E \))
\& (\( \forall r. \ \exists r' > r. \ \exists \pi_1 \ \pi_2. \)
\text{card} \( \pi_1 \) > \( E - \alpha \)
\& \text{card} \( \pi_2 \) > \( T \)
\& (\( \forall p \in \pi_1. \ \text{HOs} \ r' \ p = \pi_2 \land \text{SHOs} \ r' \ p \cap \text{HOs} \ r' \ p = \pi_2 \))

9.3 The \( A_{T,E,\alpha} \) Heard-Of Machine

We now define the non-coordinated SHO machine for the \( A_{T,E,\alpha} \) algorithm by assembling the algorithm definition and its communication-predicate.

definition Ate-SHOMachine where
Ate-SHOMachine = (\( \text{Proc} \text{HO} \Rightarrow \text{Proc} \text{HO} \Rightarrow \text{bool} \))

abbreviation
Ate-M ≡ (Ate-SHOMachine::(\text{Proc},'val::linorder pstate,'val) \text{SHOMachine})

end — locale ate-parameters

end

theory AteProof
imports AteDefs ../Reduction
begin

context ate-parameters
begin

end

end
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 vp: decide (rho (Suc r) p) = Some v
**shows** card \{qq, sendMsg Ate-M r qq p (rho r qq) = v\} > $E - \alpha$

**proof**

from run obtain $\mu p$
  where $\mu p \in$ SHOmsgVectors Ate-M r p (rho r) (HOs r p) (SHOs r p)
  and nxt: nextState Ate-M r p (rho r p) $\mu p$ (rho (Suc r) p)
by (auto simp: SHORun-eq SHOnextConfig-eq)
from $\mu p$

have \{qq, $\mu p$ qq = Some v\} - (HOs r p - SHOs r p)
  $\subseteq$ \{qq, sendMsg Ate-M r qq p (rho r qq) = v\}
(is $\notvrcvd - ?ahop \subseteq ?vsentp)$
by (auto simp: SHOmsgVectors-def)

hence card ($\notvrcvd - ?ahop$) $\leq$ card $?vsentp$
  and card ($\notvrcvd - ?ahop$) $\geq$ card $\notvrcvd - card ?ahop$
by (auto simp: card-mono diff-card-le-card-Diff)

hence card $?vsentp$ $\geq$ card $\notvrcvd - card ?ahop$ by auto

moreover

from nxt nvp vp have card $\notvrcvd > E$
  by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)

moreover

from comm have card (HOs r p - SHOs 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) $\mu q$ ((rho (Suc r) q) q)
and muq: $\mu q \in$ SHOmsgVectors Ate-M r q (rho r) (HOs r q) (SHOs r q)
and esent: card \{qq, sendMsg Ate-M r qq q (rho r qq) = v\} $> E - \alpha$
(is card $?vrcvd > -$)
**shows** card \{qq, $\mu q$ qq $\neq$ Some v\} $\cap$ HOs r q $\leq$ $N + 2*\alpha - E$

**proof**

from nxt muq

have \{qq, $\mu q$ qq $\neq$ Some v\} $\cap$ HOs r q $-$ (HOs r q - SHOs r q)
  $\subseteq$ \{qq, sendMsg Ate-M r qq q (rho r qq) $\neq$ v\}
(is $\notvrcvd - ?ahop \subseteq ?vsent\notvrcvd$)
unfolding \texttt{SHOmsgVectors-def} by \texttt{auto}

hence \(\text{card} \ ?\text{notvsent} \geq \text{card} \ (\text{?notvrcvd} - \ ?\text{aho})\)

and \(\text{card} \ (\text{?notvrcvd} - \ ?\text{aho}) \geq \text{card} \ ?\text{notvrcvd} - \text{card} \ ?\text{aho}\)

by (\textit{auto simp: card-mono diff-card-le-card-Diff})

moreover

from \texttt{comm} have \(\text{card} \ ?\text{aho} \leq \alpha\)

by (\textit{auto simp: Ate-SHOMachine-def Ate-commPerRd-def})

moreover

have \(1: \text{card} \ ?\text{notvsent} + \text{card} \ ?\text{vsent} = \text{card} \ (\text{?notvsent} \cup \ ?\text{vsent})\)

by (\textit{subst card-Un-Int}) \texttt{auto}

have \(?\text{notvsent} \cup ?\text{vsent} = (\text{UNIV::Proc set})\)

by (\texttt{auto simp})

hence \(\text{card} \ (\text{?notvsent} \cup \ ?\text{vsent}) = N\)

by (\texttt{simp})

with \(1\) \texttt{vsent}

have \(\text{card} \ ?\text{notvsent} \leq N - (E + 1 - \alpha)\)

by (\texttt{auto})

ultimately

show \(?\text{thesis}\) using \(\texttt{EltN Egta}\) by \texttt{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 \(r\).

\textbf{lemma} \textit{mostOftenRcvd-v}:

\textbf{assumes} \texttt{comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)}

\textbf{and} \texttt{nxt: nextState Ate-M r q \ ((rho (Suc r)) q)}

\textbf{and} \texttt{muq: \mu \in SHOmsgVectors Ate-M r q \ (HOs r q)} (\texttt{SHOs r q})

\textbf{and} \texttt{threshold-T: card \{qq. \mu q qq \neq None\} > T}

\textbf{and} \texttt{threshold-E: card \{qq. sendMsg Ate-M r qq q \ (rho r qq) = v\} > E - \alpha}

shows \texttt{mostOftenRcvd \mu q = \{v\}}

\textbf{proof} –

from \texttt{muq} have \texttt{hodef:HOs r q = \{qq. \mu q qq \neq None\}}

unfolding \texttt{SHOmsgVectors-def} by \texttt{auto}

from \texttt{comm nxt muq threshold-E}

have \(\text{card} \ (\{qq. \mu q qq \neq \text{Some } v\} \cap \text{HOs r q}) \leq N + 2 \alpha - E\)

(is \texttt{card \ ?heardnotv} \leq -)

by (\textit{rule other-values-received})

moreover

have \(\text{card} \ ?\text{heardnotv} \geq T + 1 - \text{card} \ \{qq. \mu q qq = \text{Some } v\}\)

proof –

from \texttt{muq}

have \texttt{?heardnotv = (HOs r q) - \{qq. \mu q qq = \text{Some } v\}}

and \(\{qq. \mu q qq = \text{Some } v\} \subseteq \text{HOs r q}\)

unfolding \texttt{SHOmsgVectors-def} by \texttt{auto}

hence \(\text{card} \ ?\text{heardnotv} = \text{card} \ (\text{HOs r q}) - \text{card} \ \{qq. \mu q qq = \text{Some } v\}\)

by (\texttt{auto simp: card-Diff-subset})

with \texttt{hodef threshold-T} show \(?\text{thesis}\) by \texttt{auto}

qed

ultimately

have \(\text{card} \ \{qq. \mu q qq = \text{Some } v\} > \text{card} \ ?\text{heardnotv}\)

using \texttt{TNaE} by \texttt{auto}
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:**

**Assumptions:**
- $\text{run} : \text{SHORun Ate-M rho HOs SHOs}$
- $\text{comm} : \text{SHOcommPerRd Ate-M (HOs (r+k)) (SHOs (r+k))}$
- $\text{ih} : \text{card} \{qq. x (\rho (r + k)) qq = v\} > E - \alpha$

**Shows:**
- $\text{card} \{qq. x (\rho (r + Suc k)) qq = v\} > E - \alpha$

**Proof:**

1. From $\text{ih}$
   - Have $\forall pp. \text{card} \{qq. \text{sendMsg Ate-M (r + k) qq pp (\rho (r + k)) qq = v}\} > E - \alpha$
     - By (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)

2. Fix $qq$
   - Assume $kv : x (\rho (r + k)) qq = v$
   - From $\text{run}$ obtain $\mu qq$
     - Where $nxt : \text{nextState Ate-M (r + k) qq (\rho (r + k)) qq \mu qq ((\rho (Suc (r + k))) qq)$
       - And $\mu qq \in \text{SHOmsgVectors Ate-M (r + k) qq (\rho (r + k)) (\rho (Suc (r + k))) qq (\rho (r + k)) qq (HOs (r + k)) qq (SHOs (r + k)) qq}$
     - By (auto simp: SHORun-eq SHOnextConfig-eq)

3. Have $x (\rho (r + Suc k)) qq = v$

**Proof**

1. **Cases** $\text{card} \{pp. \mu qq pp \neq None\} > T$
   - **Case** $True$
     - With $\text{comm} nxt muq thrE$ have $\text{mostOftenRcvd} \mu qq = \{v\}$
     - 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 $x (\rho (r + Suc k)) qq = v$ by simp

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; \( \forall r. \) SHOcommPerRd (Ate-M::(Proc, 'val::linorder pstate, 'val) SHOMachine)
  \[ (HOs \ r) (SHOs \ r) \]
- **and** d1: decide (rho r p) \( \neq \) Some \( v \)
- **and** d2: decide (rho (Suc r) p) = Some \( v \)
- **and** qupdate\(x\): x (rho (r + Suc k) q) \( \neq \) x (rho (r + k) q)
- **shows** x (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 \( \mu q \)
    - **where** nxt: nextState Ate-M (r+k) q (rho (r+k) q) \( \mu q \) (rho (r + Suc k) q)
    - **and** muq: \( \mu q \in \) SHOmsgVectors Ate-M (r+k) q (rho (r+k))
      \[ (HOs \ (r+k)) (SHOs \ (r+k)) q \]
    - **by** (auto simp: SHORun-eq SHOnextConfig-eq)
  - **moreover**
    - **from** nxt qupdate\(x\)
      - **have** threshold-T: card \{ qq. \( \mu q \) qq \( \neq \) None \} > \( T \)
        - **and** xsmall: x (rho (r + Suc k) q) = Min (mostOftenRcvd \( \mu q \))
          - **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-M r 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 + Suc k) qq) = \( v \) \}
                - **by** (auto dest: common-x-induct)
  - **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 where m < n
and decide (rho m p) \( \neq \) Some v
and decide (rho (Suc m) p) = Some v

**Proof**
- let \( \text{?dec } k = \text{decide (rho k p) } \)
- have \((\forall m < n. \text{?dec (Suc m) } \neq \text{?dec (Suc m) } \neq \text{Some v}) \rightarrow \text{?dec n } \neq \text{Some v)\)

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: \( \forall r. \text{SHOcommPerRd Ate-M (HOs r) (SHOs r) } \)
and initv: \( \forall q. x (\text{rho } 0 q) = v \)
and dp: decide (rho r p) = Some w
shows \( w = v \)

**Proof**
- \{ 
  \begin{align*}
  \text{fix } r \\
  \text{have } \forall qq. \text{sendMsg Ate-M } r \text{ qq } p (\text{rho } r \text{ qq}) = v \\
  \text{proof (induct } r) \\
  \text{from run initv show } \forall qq. \text{sendMsg Ate-M } 0 \text{ qq } p (\text{rho } 0 \text{ qq}) = v \\
  \text{by (auto simp: SHORun-eq SHOnextConfig-eq Ate-SHOMachine-def Ate-sendMsg-def)}
  \end{align*}
\}
next
fix r
assume \( \forall qq. \text{sendMsg Ate-M r qq p (\(\rho\) r qq)} = v \)

have \(\forall qq, x (\(\rho\) (Suc r) qq) = v\)
proof
fix qq
from run obtain \(\mu qq\)
where nxt: \(\text{nextState Ate-M r qq (\(\rho\) r qq) \(\mu qq\) (\(\rho\) (Suc r) qq)}\)
and mu: \(\mu qq \in \text{SHOmsgVectors Ate-M r qq (\(\rho\) r qq)}\) (SHOs r qq)
by (auto simp: SHORun-eq SHOnextConfig-eq)

from nxt
have \((\text{card } \{pp. \mu qq pp \neq None\} > T \land x (\(\rho\) (Suc r) qq) = \text{Min (mostOftenRcvd \(\mu qq\)})\) 
\(\forall (\text{card } \{pp. \mu qq pp \neq None\} \leq T \land 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: \(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: \(T < \text{card } \{pp. \mu qq pp \neq None\}\)
and xsmall: \(x (\(\rho\) (Suc r) qq) = \text{Min (mostOftenRcvd \(\mu qq\)})\)

have \((\text{card } \{pp. \exists w. w \neq v \land \mu qq pp = \text{Some w}\} \leq T \text{ div } 2\) 
proof --
from comm have 1: \(\text{card (HOs r qq - SHOs r qq)} \leq \alpha\)
by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
moreover
from mu ih
have \(\text{SHOs r qq \cap HOs r qq} \subseteq \{pp. \mu qq pp = \text{Some v}\}\)
and \(\text{HOs r qq} = \{pp. \mu qq pp \neq None\}\)
by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def Ate-sendMsg-def)

hence \(\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = \text{Some v}\}\) 
\(\subseteq \text{HOs r qq - SHOs r qq}\)
by auto

hence \(\text{card (\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = \text{Some v}\})}\) 
\(\leq \text{card (HOs r qq - SHOs r qq)}\)
by (auto simp:card-mono)
ultimately
have \((\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = \text{Some v}\})\) \(\leq T \text{ div } 2\)
using Tge2a by auto
moreover
have \(\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = \text{Some v}\}\) 
\(= \{pp. \exists w. w \neq v \land \mu qq pp = \text{Some w}\}\) by auto
ultimately
show \(?thesis\) by simp

131
qed
moreover
have \( \{ pp. \mu qq pp \neq \text{None} \} \)
  \( = \{ pp. \mu qq pp = \text{Some } v \} \cup \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} \)
and \( \{ pp. \mu qq pp = \text{Some } v \} \cap \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} = \)
\[
\emptyset
\]
by auto
hence \( \text{card} \{ pp. \mu qq pp \neq \text{None} \} \)
  \( = \text{card} \{ pp. \mu qq pp = \text{Some } v \} + \text{card} \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} \)

\( = \text{Some } w \)
by (auto simp: card-Un-Int)
moreover
note \text{threshold-T}
ultimately
have \( \text{card} \{ pp. \mu qq pp = \text{Some } v \} > \text{card} \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} \)
by auto
moreover
\(
\{ \text{fix } w \\
\text{assume } w \neq v \\
\text{hence } \{ pp. \mu qq pp = \text{Some } w \} \subseteq \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} \\
\text{by auto} \\
\text{hence } \text{card} \{ pp. \mu qq pp = \text{Some } w \} \leq \text{card} \{ pp. \exists w. w \neq v \land \mu qq pp = \text{Some } w \} = \text{Some } w \\
\text{by (auto simp: card-mono)} \\
\}
\)
ultimately
have \( \forall w. w \neq v \implies \text{card} \{ pp. \mu qq pp = \text{Some } w \} < \text{card} \{ pp. \mu qq pp = \text{Some } v \} \)
by force
hence \( \forall w. \text{card} \{ pp. \mu qq pp = \text{Some } v \} \leq \text{card} \{ pp. \mu qq pp = \text{Some } w \} \)
  \( \implies w = v \)
by force
with \( \exists z \) have \( \text{mostOftenRcvd } \mu qq = \{ v \} \)
by (force simp: mostOftenRcvd-def)
with \( x \) small show \( x (\rho (\text{Suc } r) qq) = v \) by auto
qed
qed
thus \( \forall qq. \text{sendMsg } Ate-M (\text{Suc } r) qq p (\rho (\text{Suc } r) qq) = v \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
qed
}

note \( P = \text{this} \)

from \( \text{run dp obtain } rp \)
where \( rp : rp < r \) decide \( (\rho rp p) \neq \text{Some } w \)
  decide \( (\rho (\text{Suc } rp) p) = \text{Some } w \)
by (rule decisionNonNilThenDecided)

132
from run obtain \( \mu p \)
where nxt: nextState Ate-M rp p (rho rp p) \( \mu p \) (rho (Suc rp) p)
and mu: \( \mu p \in SHOmsgVectors Ate-M rp p \) (HOs rp p) (SHOs rp p)
by (auto simp: SHORun-eq SHOnextConfig-eq)

\[
\begin{align*}
\{ \\
& \text{fix } w \\
& \text{assume } w: w \neq v \\
& \text{from } \text{comm have } \text{card } (HOs rp p - SHOs rp p) \leq \alpha \\
& \quad \text{by } (\text{auto simp: Ate-SHOMachine-def Ate-commPerRd-def}) \\
& \text{moreover} \\
& \text{from } \text{mu P have } \text{SHOs rp p} \cap \text{HOs rp p} \subseteq \{pp. \mu p pp = \text{Some } v\} \\
& \quad \text{by } (\text{auto simp: SHOmsgVectors-def}) \\
& \text{hence } \{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some } v\} \\
& \quad \subseteq \text{HOs rp p} - \text{SHOs rp p} \\
& \quad \text{by } auto \\
& \text{hence } \text{card } (\{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some } v\}) \\
& \quad \leq \text{card } (\text{HOs rp p} - \text{SHOs rp p}) \\
& \quad \text{by } (\text{auto simp: card-mono}) \\
& \text{ultimately} \\
& \text{have } \text{card } (\{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some } v\}) < E \\
& \quad \text{using } Egta \text{ by } auto \\
& \text{moreover} \\
& \text{from } w \text{ have } \{pp. \mu p pp = \text{Some } w\} \\
& \quad \subseteq \{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some } v\} \\
& \quad \text{by } auto \\
& \text{hence } \text{card } \{pp. \mu p pp = \text{Some } w\} \\
& \quad \leq \text{card } (\{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some } v\}) \\
& \quad \text{by } (\text{auto simp: card-mono}) \\
& \text{ultimately} \\
& \text{have } \text{card } \{pp. \mu p pp = \text{Some } w\} < E \text{ by } simp \\
& \text{hence } PP: \forall w. \text{card } \{pp. \mu p pp = \text{Some } w\} \geq E \implies w = v \text{ by } force \\
\end{align*}
\]

from rp nxt mu have card \{q. \mu p q = \text{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 same 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 vp: decide (rho (Suc r) p) = Some v
and nwq: decide (rho r q) |\neq| Some w
and wq: decide (rho (Suc r) q) = Some w
shows w = v

proof –
  have gtn: card {qq. sendMsg Ate-M r qq p (rho r qq) = v} + card {qq. sendMsg Ate-M r qq q (rho r qq) = w} > N
  proof –
  from run comm nvp vp
  have card {qq. sendMsg Ate-M r qq p (rho r qq) = v} > E - \alpha
  by (rule decide-sent-msgs-threshold)
  moreover
  from run comm nwq wq
  have card {qq. sendMsg Ate-M r qq q (rho r qq) = w} > E - \alpha
  by (rule decide-sent-msgs-threshold)
  ultimately
  show ?thesis using majE by auto
qed

show ?thesis
proof (rule ccontr)
  assume vv:w \neq v
  have \forall qq. sendMsg Ate-M r qq p (rho r qq) = sendMsg Ate-M r qq q (rho r qq)
  by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
  with vv
  have {qq. sendMsg Ate-M r qq p (rho r qq) = v} \cap {qq. sendMsg Ate-M r qq q (rho r qq) = w} = {}
  by auto
  with gtn
  have card {{qq. sendMsg Ate-M r qq p (rho r qq) = v}} \cup {{qq. sendMsg Ate-M r qq q (rho r qq) = w}} > N
  by (auto simp: card-Un-Int)
  moreover
  have card {{qq. sendMsg Ate-M r qq p (rho r qq) = v}} \cup {{qq. sendMsg Ate-M r qq q (rho r qq) = 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) |\neq| Some v
and \(d1\): decide \((\rho \ Suc \ r) \ p\) = Some \(v\)
and \(nd2\): decide \((\rho \ (r+k) \ q)\) \(\neq\) Some \(w\)
and \(d2\): decide \((\rho \ (Suc \ (r+k)) \ q)\) = Some \(w\)
shows \(w = v\)

**proof** (rule ccontr)
assume \(vdifw: w \neq v\)
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: \(\{qq. \text{sendMsg} \ Ate-M \ r \ qq \ (\rho \ r \ qq) = v\}\)
\(\cap \{qq. \text{sendMsg} \ Ate-M \ (r+k) \ qq \ (\rho \ (r+k) \ qq) = w\} = \{\}\)
(is \(?sentv \cap \?sentw = \{\}\))
proof (rule ccontr)
assume \(\neg \?thesis\)
then obtain \(qq\)
where \(xrv: x \ (\rho \ r \ qq) = v\) and \(rkw: x \ (\rho \ (r+k) \ qq) = w\)
by (auto simp: Ate-SHM-def Ate-sendMsg-def)
have \(\exists k' < k. x \ (\rho \ (r + k') \ qq) \neq w \land x \ (\rho \ (r + Suc \ k') \ qq) = w\)
proof (rule ccontr)
assume \(f: \neg \?thesis\)
{
fix \(k'\)
assume \(kk': k' < k\) hence \(x \ (\rho \ (r + k') \ qq) \neq w\)
proof (induct \(k'\))
from \(xrv \ vdifw\)
show \(x \ (\rho \ (r + 0) \ qq) \neq w\) by simp
next
fix \(k'\)
assume \(ih: k' < k \implies x \ (\rho \ (r + k') \ qq) \neq w\)
and \(ksk': Suc \ k' < k\)
from \(ksk'\) have \(k' < k\) by simp
with \(ih \ f\) show \(x \ (\rho \ (r + Suc \ k') \ qq) \neq w\) by auto
qed
}
with \(f\) have \(\forall k' < k. \ (\rho \ (r + Suc \ k') \ qq) \neq w\) by auto
moreover
from \(kgt0\) have \(k - 1 < k\) and \(kk: Suc \ (k - 1) = k\) by auto
ultimately
have \(x \ (\rho \ (r + Suc \ (k - 1)) \ qq) \neq w\) by blast
with \(rkw \ kk\) show False by simp
qed
then obtain \(k'\)
where \(k' < k\)

135
and \( w: x \ (\rho \ (r + \text{Suc} \ k') \ qq) = w \)
and \( \text{updax:} \ x \ (\rho \ (r + \text{Suc} \ k') \ qq) \neq x \ (\rho \ (r + k') \ qq) \)
by auto

from run comm nd1 d1 updax
have \( x \ (\rho \ (r + \text{Suc} \ k') \ qq) = v \) by (rule common-x)
with \( \text{wdiw} \) show False by simp

qed

from run comm nd1 d1 have \( \text{sentv}: \text{card} \ ?\text{sentv} > E - \alpha \)
by (auto dest: decide-sent-msgs-threshold)
from run comm nd2 d2 have \( \text{card} \ ?\text{sentw} > E - \alpha \)
by (auto dest: decide-sent-msgs-threshold)
with \( \text{sentv majE} \) have \( (\text{card} \ ?\text{sentv}) + (\text{card} \ ?\text{sentw}) > N \)
by simp
with \( \text{1 vdifw} \) have \( 2: \text{card} \ (\text{?sentv} \cup \text{?sentw}) > N \)
by (auto simp: card-Un-Int)
have \( \text{card} \ (\text{?sentv} \cup \text{?sentw}) \leq N \)
by (auto simp: card-mono)
with \( \text{2 show False by simp} \)

qed

The Agreement property is now an immediate consequence.

```latex
theorem ate-agreement:
  assumes \( \text{run: SHORun Ate-M rho HOs SHOs} \)
  and \( \text{comm: \forall~r. SHOcommPerRd Ate-M (HOs r) (SHOs r)} \)
  and \( p: \text{decide} \ (\rho \ m \ p) = \text{Some} \ v \)
  and \( q: \text{decide} \ (\rho \ n \ q) = \text{Some} \ w \)
  shows \( w = v \)
proof
  from run p obtain \( k \) where
  \( k: k < m \text{ decide} \ (\rho \ k \ p) \neq \text{Some} \ v \text{ decide} \ (\rho \ (\text{Suc} \ k) \ p) = \text{Some} \ v \)
  by (rule decisionNonNullThenDecided)
from run q obtain \( l \) where
  \( l: l < n \text{ decide} \ (\rho \ l \ q) \neq \text{Some} \ w \text{ decide} \ (\rho \ (\text{Suc} \ l) \ q) = \text{Some} \ w \)
  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 \( k \ l \) show ?thesis
  by (auto dest: laterProcessDecidesSameValue)
next
  case False
  hence \( l \leq k \) by simp
  then obtain \( i \) where \( m: k = l + i \) by (auto simp add: le-iff-add)
  with run comm \( k \ l \) show ?thesis
  by (auto dest: laterProcessDecidesSameValue)
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 at-termination:**
- Assumes `run: SHORun Ate-M rho HOs SHOs`
- and `commR: ∀ r. (SHOcommPerRd::((Proc, 'val::linorder pstate, 'val) SHOMachine) ⇒ (Proc HO r)⇒ (Proc HO r)⇒ bool)`
- and `commG: SHOcommGlobal Ate-M HOs SHOs`
- Shows `∃ r v. decide (rho r p) = Some v`

**Proof** –
- From `commG` obtain `r' π1 π2`
  - where `π1: card π1 > E − α`
  - and `π2: card π2 > T`
  - and `hosho: ∀ p ∈ π1. (HOs r' p = π2 ∧ SHOs r' p ∩ HOs r' p = π2)`
  - by (auto simp: Ate-SHOMachine-def Ate-commGlobal-def)

Obtain `v` where
- `P1: ∀ pp. card {qq. sendMsg Ate-M (Suc r') qq pp (rho (Suc r') qq) = v} > E − α`

**Proof** (clarify)
- Fix `p q`
- Assume `p: p ∈ π1 and q: q ∈ π1`

- From `run` obtain `μp`
  - where `nxtp: nextState Ate-M r' p (rho r' p) μp (rho (Suc r') p)`
  - and `mup: μp ∈ SHOmsgVectors Ate-M r' p (rho r' p) (SHOs r' p)`
  - by (auto simp: SHORun-eq SHOnextConfig-eq)

- From `run` obtain `μq`
  - where `nxtq: nextState Ate-M r' q (rho r' q) μq (rho (Suc r') q)`
  - and `muq: μq ∈ SHOmsgVectors Ate-M r' q (rho r' q) (SHOs r' q)`
  - by (auto simp: SHORun-eq SHOnextConfig-eq)

- From `mup muq p q`
  - Have `{qq. μq qq ≠ None} = HOs r' q`
  - and `2:{qq. μq qq = Some (sendMsg Ate-M r' qq q (rho r' qq))} ⊇ SHOs r' q ∩ HOs r' q`
  - and `{qq. μp qq ≠ None} = HOs r' p`
  - and `4:{qq. μp qq = Some (sendMsg Ate-M r' qq p (rho r' qq))} ⊇ SHOs r' p ∩ HOs r' p`
  - by (auto simp: SHOmsgVectors-def)
- With `p q hosho`
have \( \pi_2 = \{qq. \mu qq \neq \text{None}\} \)
and \( \pi_2 = \{qq. \mu pp \neq \text{None}\} \) by auto
from \( p q \) hosho 2
have \( bb: \{qq. \mu qq = \text{Some} (sendMsg \text{Ate-M } r' qq q (\rho r' qq))\} \supseteq \pi_2 \)
by auto
from \( p q \) hosho 4
have \( dd: \{qq. \mu pp qq \neq \text{None}\} \)
by auto
have \( \text{Min} (\text{mostOftenRcvd } \mu) = \text{Min} (\text{mostOftenRcvd } \mu p) \)
proof –
have \( \forall qq. \mu p qq = \text{None} \)
  by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
with \( aa bb cc dd \)
have \( \forall qq. \mu pp qq = \mu qq \)
by force
moreover
from \( aa bb cc dd \)
have \( \forall qq. \mu pp qq = \text{None} \)
by auto
hence \( \forall qq. \mu pp qq = \mu qq \)
by blast
ultimately
have \( \forall qq. \mu p qq = \mu qq \)
by auto
ultimately
have \( \forall qq. \mu p qq = \mu qq \)
by blast
thus \( \text{thesis} \)
by (auto simp: mostOftenRcvd-def)
qed
with \( \pi_1 \)
show \( \forall p \in \pi_1. x (\rho (\text{Suc } r') p) = x (\rho (\text{Suc } r') q) \)
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
qed
then obtain \( v \) where \( P v: \forall p \in \pi_1. x (\rho (\text{Suc } r') p) = v \)
by blast
{ 
fix \( pp \)
from \( P v \) have \( \forall p \in \pi_1. \text{sendMsg } \text{Ate-M } (\text{Suc } r') p pp (\rho (\text{Suc } r') p) = v \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
hence \( \text{card } \pi_1 \leq \text{card} \{qq. \text{sendMsg } \text{Ate-M } (\text{Suc } r') qq pp (\rho (\text{Suc } r') qq) = v\} \)
by (auto intro: card-mono)
with \( \pi a \)
have \( E - \alpha < \text{card} \{qq. \text{sendMsg } \text{Ate-M } (\text{Suc } r') qq pp (\rho (\text{Suc } r') qq) = v\} \)
by simp
}
with that show \( \text{thesis} \) by blast
qed
proof (induct k)
  from P1 show ?P 0 by simp
next
  fix k
  assume ih: ?P k
  from commR
  have (SHOcommPerRd:((Proc, 'val::linorder pstate, 'val) SHOMachine)
           ⇒ (Proc HO) ⇒ (Proc HO) ⇒ bool)
      Ate-M (HOs (Suc r' + k)) (SHOs (Suc r' + k)) ..
  moreover
  from ih have E - α < card \{qq. x (rho (Suc r' + k) qq) = v\}
      by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
  ultimately
  have E - α < card \{qq. x (rho (Suc r' + Suc k) qq) = v\}
      by (rule common-x-induct[OF run])
  thus ?P (Suc k)
      by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
  qed
}

note P2 = this

\{
  fix k pp
  assume ppupdatex: x (rho (Suc r' + Suc k) pp) \neq x (rho (Suc r' + k) pp)

  from commR
  have (SHOcommPerRd:((Proc, 'val::linorder pstate, 'val) SHOMachine)
           ⇒ (Proc HO) ⇒ (Proc HO) ⇒ bool)
      Ate-M (HOs (Suc r' + k)) (SHOs (Suc r' + k)) ..
  moreover
  from run obtain \mu pp
      where nxt:nextState Ate-M (Suc r' + k) pp (rho (Suc r' + k) pp) \mu pp
           (rho (Suc r' + Suc k) pp)
      and mu: \mu pp \in SHOMsgVectors Ate-M (Suc r' + k) pp (rho (Suc r' + k))
           (HOs (Suc r' + k) pp) (SHOs (Suc r' + k) pp)
      by (auto simp: SHORun-eq SHOnextConfig-eq)
  moreover
  from nxt ppupdatex
  have threshold-T: card \{qq. \mu pp qq \neq None\} > T
      and zsmall: x (rho (Suc r' + Suc k) pp) = Min (mostOftenRcvd \mu pp)
      by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
  moreover
  from P2
  have E - α < card \{qq. sendMsg Ate-M (Suc r' + k) qq pp (rho (Suc r' + k) qq) = v\}
      by (auto simp: mostOftenRcvd-def)
  ultimately
  have mostOftenRcvd \mu pp = \{v\} by (auto dest!: mostOftenRcvd-v)
  with zsmall
  have x (rho (Suc r' + Suc k) pp) = v by simp

  139
\[
\forall \rho. \exists k. x (\rho (\text{Suc } r' + \text{Suc } k)) = v
\]

**Proof**

- **Note** $P3 = \text{this}$
- **Have** $P4: \forall \rho. \exists k. x (\rho (\text{Suc } r' + \text{Suc } k)) = v$

**Proof**

- Fix $\rho$
- From \textit{commG} have $\exists r'' > r'. \ \text{card} (\text{HOs } r'' \rho) > T$
  - By (auto simp: \textit{Ate-SHOMachine-def} \textit{Ate-commGlobal-def})
- Then obtain $k$ where $\text{Suc } r' + k > r'$ and $t: \text{card} (\text{HOs } (\text{Suc } r' + k) \rho) > T$
  - By (auto: \textit{less-imp-Suc-add})
- Moreover
  - From \textit{run} obtain $\mu$
    - Where $\text{nxt: nextState Ate-M } (\text{Suc } r' + k) \rho (\text{Suc } r' + k) \mu$ \rho (\text{Suc } r' + k)) \rho$
    - And $\mu: \mu \in \text{SHOmsgVectors Ate-M } (\text{Suc } r' + k) \rho (\text{Suc } r' + k)) \rho$
      - (\text{HOs } (\text{Suc } r' + k) \rho) \rho$
    - By (auto simp: \textit{SHORun-eq} \textit{SHOnextConfig-eq})
- Moreover
  - Have $x (\rho (\text{Suc } r' + k) \rho) = v$
    - By (auto: \textit{Ate-SHOMachine-def} \textit{nextState-def} \textit{Ate-nextState-def})
- Ultimately
  - Have $\text{mostOftenRcvd } \mu = \{v\}$
    - Using $\text{nxt } \mu$ by (auto dest: \textit{mostOftenRcvd-v})
    - With $\text{xsmall show thesis by auto}$
- QED

**Have** $P5a: \forall \rho. \exists rr. \forall k. x (\rho (\text{Suc } r' + \text{Suc } k)) = v$

**Proof**

- Fix $\rho$
- From $P4$ obtain $rr$ where
  - $x (\rho (\text{Suc } r' + \text{Suc } rr)) = v$ (\textit{is x (rho ?rr pp) = v})
by blast
have \( \forall k. x (\rho (r + k)) pp = v \)
proof
  fix \( k \)
  show \( x (\rho (r + k)) pp = v \)
  proof (induct \( k \))
  from \( xrrv \) show \( x (\rho (\preceq + 0)) pp = v \) by simp
  next
  fix \( k \)
  assume \( \text{ih} : x (\rho (r + k)) pp = v \)
  obtain \( k' \) where \( r + k' = ?rr + k \) by auto
  show \( x (\rho (r + Suc k)) pp = v \)
  proof (rule ccontr)
    assume \( nv : x (\rho (r + Suc k)) pp \neq v \)
    with \( r + k' \) \text{ih}
    have \( x (\rho (Suc (r + Suc k')) pp) \neq x (\rho (Suc r' + k')) pp \)
    by (simp add: ac-simps)
    hence \( x (\rho (Suc r' + Suc k')) pp = v \) by (rule P3)
    with \( r + k' \) \text{nv show False by (simp add: ac-simps)}
  qed
  qed
  qed
  thus \( \exists r. \forall k. x (\rho (r + k)) pp = v \) by blast
qed

from \( P5a \) have \( \exists F. \forall pp k. x (\rho (F pp + k)) pp = v \) by (rule choice)
then obtain \( R : (Proc \Rightarrow \text{nat}) \)
  where \( \text{imgR} : R \cdot (\text{UNIV} :: \text{Proc}) \neq {} \)
  and \( R : \forall pp k. x (\rho (R pp + k)) pp = v \)
  by blast
def \( rr \equiv \text{Max} (R \cdot \text{UNIV}) \)

have \( P5 : \forall r' > rr. \forall pp. x (\rho r' pp) = v \)
proof (clarify)
  fix \( r' pp \)
  assume \( r' : r' > rr \)
  hence \( r' > R pp \) by (auto simp: rr-def)
  then obtain \( i \) where \( r' = R pp + i \)
  by (auto dest: less-imp-Suc-add)
  with \( R \) show \( x (\rho r' pp) = v \) by auto
qed

from \( \text{commG} \) have \( \exists r' > rr. \text{card} (\text{SHOs} r' p \cap \text{HOs} r' p) > E \)
  by (auto simp: \text{Ate-SHOMachine-def} \text{Ate-commGlobal-def})
with \( P5 \) obtain \( r' \)
  where \( r' > rr \)
  and \( \text{card} (\text{SHOs} r' p \cap \text{HOs} r' p) > E \)
  and \( \forall pp. \text{sendMsg Ate-M} r' pp (\rho r' pp) = v \)
  by (auto simp: \text{Ate-SHOMachine-def} \text{Ate-sendMsg-def})
moreover
from run obtain \( \mu p \)
where \( \text{nxt: nextState Ate-M r'} p (\rho r' p) \mu p (\rho (Suc r') p) \)
and \( \mu: \mu p \in SHOmsgVectors Ate-M r' p (\rho r' p) (HOs r' p) \)
by (auto simp: SHORun-eq SHOnextConfig-eq)

from \( \mu \)
have \( \text{card} (HOs r' p \cap SHOs r' p) \leq \text{card} \{ q. \mu p q = \text{Some} (sendMsg Ate-M r' q p (\rho r' q)) \}
by (auto simp: SHOmsgVectors-def intro: card-mono)

ultimately
have \( \text{threshold-E: card} \{ q. \mu p q = \text{Some} v \} > E \) by auto
with \( \text{nxt} \) show \(?\),thesis
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)

qed

9.8 \( A_{T,E,\alpha} \) Solves Weak Consensus

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

theorem ate-weak-consensus:
assumes \( \text{run: SHORun Ate-M \rho HOs SHOs} \)
and \( \text{commR: } \forall r. \text{SHOcommPerRd Ate-M (HOs r) (SHOs r)} \)
and \( \text{commG: SHOcommGlobal Ate-M HOs SHOs} \)
shows \( \text{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 \( \text{run: fg-run Ate-M \rho HOs SHOs (\lambda q. undefined)} \)
and \( \text{commR: } \forall r. \text{SHOcommPerRd Ate-M (HOs r) (SHOs r)} \)
and \( \text{commG: SHOcommGlobal Ate-M HOs SHOs} \)
shows \( \text{weak-consensus \( (\lambda p. x (\text{state (\rho 0) p})) \) decide \( \text{state \circ \rho) \)} \)
(is weak-consensus ?inits - -)
proof (rule local-property-reduction[OF run weak-consensus-is-local!])
fix \( \text{crun} \)
assume \( \text{crun: CSHORun Ate-M crun HOs SHOs (\lambda q. undefined)} \)
and \( \text{init: crun 0 = state (\rho 0)} \)
from \( \text{crun} \) have \( \text{SHORun Ate-M crun HOs SHOs} \) by (unfold SHORun-def)
from this \( \text{commR commG} \)
have \( \text{weak-consensus \( (x \circ (\text{crun 0})) \) decide \( \text{crun} \)} \)
by (rule ate-weak-consensus)
with \( \text{init} \) show \( \text{weak-consensus \?inits decide \( \text{crun} \)} \)
by (simp add: o-def)

qed
10 Verification of the $\text{EIGByz}_f$ Consensus Algorithm

Lynch [12] presents $\text{EIGByz}_f$, 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 $\text{'proc}$ 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
$N \equiv \text{card (UNIV::Proc set)}$ — 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 $f :: \text{nat}$
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 $[p,q] \mapsto \text{Some v}$ 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 = \{xs::Proc list. length xs \leq Suc f \land \text{distinct xs}\}
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 \( \text{Label} \subseteq \{ \text{xs . set xs} \subseteq (\text{UNIV::Proc set}) \land \text{length xs} \leq \text{Suc f} \} \)
  by (auto simp: Label-def)
moreover
have \( \text{finite} \{ \text{xs . set xs} \subseteq (\text{UNIV::Proc set}) \land \text{length xs} \leq \text{Suc f} \} \)
  by (rule finite-lists-length-le) auto
ultimately
show \( ?\text{thesis} \) by (auto elim: finite-subset)
qed

lemma finite-UNIV-Label: finite \((\text{UNIV::Label set})\)
proof
- from finite-Label have finite \((\text{Abs-Label} \cdot \text{Label})\) by simp
moreover
{ 
  fix \( l :: \text{Label} \)
  have \( l \in \text{Abs-Label} \cdot \text{Label} \)
    by (rule Abs-Label-cases) auto
}
hence \((\text{UNIV::Label set}) = (\text{Abs-Label} \cdot \text{Label})\) by auto
ultimately show \( ?\text{thesis} \) by simp
qed

lemma finite-Label-set \([\text{iff}]\): finite \((S :: \text{Label set})\)
using finite-UNIV-Label by (auto intro: finite-subset)

Utility functions on labels.
definition root-node where
  root-node \( \equiv \text{Abs-Label} \cdot [] \)
definition length-lbl where
  length-lbl \( l \equiv \text{length} \,(\text{Rep-Label} l) \)
lemma length-lbl \([\text{intro}]\): length-lbl \( l \leq \text{Suc f} \)
  unfolding length-lbl-def using Label-def Rep-Label by auto
definition is-leaf where
  is-leaf \( l \equiv \text{length-lbl} \, l = \text{Suc f} \)
definition last-lbl where
  last-lbl \( l \equiv \text{last} \,(\text{Rep-Label} l) \)
definition butlast-lbl where
  butlast-lbl \( l \equiv \text{Abs-Label} \,(\text{butlast} \,(\text{Rep-Label} l)) \)
definition set-lbl where
  set-lbl \( l = \text{set} \,(\text{Rep-Label} l) \)

The children of a non-leaf label are all possible extensions of that label.
**definition** children where
children l ≡
  if is-leaf l
  then {}
  else { Abs-Label (Rep-Label l @ [p]) | p . p /∈ set-lbl l }

### 10.2 Model of the Algorithm

The following record models the local state of a process.

```plaintext
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 ≡
  (∀ l. (vals st l = None) = (l ≠ root-node))
  ∧ decide st = None

**type-synonym** 'val Msg = Label ⇒ '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 ≡ 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' = (λ l .
    if length-lbl l = Suc r ∧ msgs (last-lbl l) ≠ None
    then (the (msgs (last-lbl l))) (butlast-lbl l)
    else if length-lbl l = Suc r ∧ msgs (last-lbl l) = None then None
    else vals st l)

**definition** next-main where
next-main r p st msgs st' ≡ extend-vals r p st msgs st' ∧ decide st' = 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.

```haskell
fun fixupval :: 'val option ⇒ 'val where
  fixupval None = undefined
  | fixupval (Some v) = v
```

```haskell
definition has-majority :: 'val ⇒ ('a ⇒ 'val) ⇒ 'a set ⇒ bool where
  has-majority v g S ≡ card {e ∈ S. g e = v} > (card S) div 2
```

```haskell
definition check-newvals :: 'val pstate ⇒ bool where
  check-newvals st ≡ ∀l. is-leaf l ∧ newvals st l = fixupval (vals st l)
  ∨ ¬(is-leaf l) ∧
      ( (∃w. has-majority w (newvals st) (children l) ∧ newvals st l = w)
    ∨ (¬(∃w. has-majority w (newvals st) (children l))
       ∧ newvals st l = undefined))
```

```haskell
definition next-end where
  next-end r p st msgs st' ≡
    extend-vals r p st msgs st'
  ∧ check-newvals st'
  ∧ decide st' = Some (newvals st' root-node)
```

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

```haskell
definition EIG-nextState where
  EIG-nextState r ≡
    if r < f then next-main r
    else if r = f then next-end r
    else (λp st msgs st'. st' = st)
```

### 10.3 Communication Predicate for EIGByz

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

```haskell
definition SKr :: Proc HO ⇒ Proc HO ⇒ Proc set where
  SKr HO SHO ≡ { q . ∀p. q ∈ HO p ∩ SHO p}
```

The secure kernel SK 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.

```haskell
definition SK :: (nat ⇒ Proc HO) ⇒ (nat ⇒ Proc HO) ⇒ Proc set where
  SK HOs SHOs ≡ {q. ∀r. q ∈ SKr (HOs 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** \(EIG\text{-commPerRd}\) where

\[
EIG\text{-commPerRd} \text{ HO SHO} \equiv \text{card} (SK \text{ HO SHO}) > (N + f) \div 2
\]

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** \(EIG\text{-commGlobal}\) where

\[
EIG\text{-commGlobal} \text{ HOs SHOs} \equiv \text{card} (SK \text{ HOs SHOs}) \geq N - f
\]

The above communication predicates differ from Lynch’s presentation of \(EIGByz_f\). 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 \(EIG\text{-commGlobal}\) is satisfied. The standard correctness proof assumes that \(N > 3f\), and therefore \(N - f > (N + f) \div 2\). Since moreover, for any \(r\), we obviously have

\[
\left( \bigcap_{p \in \Pi, r' \in \mathbb{N}} SHO(p, r') \right) \subseteq \left( \bigcap_{p \in \Pi} SHO(p, r) \right),
\]

it follows that any execution of \(EIGByz_f\) where \(N > 3f\) also satisfies \(EIG\text{-commPerRd}\) at any round. The standard correctness hypotheses thus imply our communication predicates.

However, our proof shows that \(EIGByz_f\) 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 \(EIG\text{-commPerRd}\) always holds. Also, \(EIG\text{-commGlobal}\) 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 \(EIGByz_f\) 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 \(EIGByz_f\) Heard-Of Machine

We now define the non-coordinated SHO machine for \(EIGByz_f\) by assembling the algorithm definition and its communication-predicate.

**definition** \(EIG\text{-SHOMachine}\) where

\[
EIG\text{-SHOMachine} = ()
\]

\[
\text{CinitState} = (\lambda p \text{ st crd. } EIG\text{-initState } p \text{ st}),
\]
sendMsg = EIG-sendMsg,
CnextState = (λ r p st msgs crd st . EIG-nextState r p st msgs st'),
SHOcommPerRd = EIG-commPerRd,
SHOcommGlobal = EIG-commGlobal

abbreviation EIG-M ≡ (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: ¬(is-leaf l)
  shows length-lbl l ≤ f
  using l length-lbl[of l] by (simp add: is-leaf-def)

lemma nil-is-Label: [] ∈ 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: ¬(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 ≠ root-node
  shows butlast (Rep-Label l) ∈ Label
proof –
  have Rep-Label l ≠ []
  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

end
  by (auto simp: Label-def elim: distinct-butlast)
qed

The label of a child is well-formed.

lemma Rep-Label-append:
  assumes l: ~(is-leaf l)
  shows (Rep-Label l @ [p] ∈ Label) = (p ∉ set-lbl l)
    (is ?lhs = ?rhs is (?l' ∈ -) = -)
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' ≤ 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' by (simp add: set-lbl-def)
ultimately
  show ?lhs by (simp add: Label-def)
qed

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

lemma children-length:
  assumes l ∈ children h
  shows length-lbl l = Suc (length-lbl h)
  using label-children[of l] by (auto simp: length-lbl-def)

The root node is never a child.

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

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

\textbf{lemma \textit{children-butlast-lbl}}:
\begin{itemize}
\item \textbf{assumes} \(c \in \text{children } l\)
\item \textbf{shows} \(\text{butlast-lbl } c = l\)
\end{itemize}
using \textit{label-children}[\textit{OF \textit{assms}}]
by (auto simp: \textit{butlast-lbl-def} \textit{Reps-Label-inverse})

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

\textbf{lemma \textit{root-iff-no-child}}: \((l = \text{root-node}) = (\forall l'. l \notin \text{children } l')\)

\textbf{proof}
\begin{itemize}
\item \textbf{assume} \(l = \text{root-node}\)
\item \textbf{thus} \(\forall l'. l \notin \text{children } l'\) by (auto elim: \textit{children-not-root})
\end{itemize}

\textbf{next}
\begin{itemize}
\item \textbf{assume} \(\text{rhs} : \forall l'. l \notin \text{children } l'\)
\item \textbf{show} \(l = \text{root-node}\)
\begin{itemize}
\item \textbf{proof} (rule \textit{rev-exhaust}[of \textit{Rep-Label } l])
\item \textbf{assume} \(\text{Rep-Label } l = []\)
\item \textbf{hence} \(\text{Rep-Label } l = \text{Rep-Label } \text{root-node}\) by simp
\item \textbf{thus} \(\text{?thesis}\) by (simp only: \textit{Rep-Label-inject})
\end{itemize}
\item \textbf{next}
\begin{itemize}
\item \textbf{fix } l' \(q\)
\item \textbf{assume} \(l' : \text{Rep-Label } l = l' @ [q]\)
\item \textbf{let} \(l'' = \text{Abs-Label } l'\)
\item \textbf{from} \(\text{Rep-Label}[\textit{of } l] \ l' \text{ have } l' \in \text{Label}\) by (simp add: \textit{Label-def})
\item \textbf{hence} \(\text{repl': } \text{Rep-Label } l'' = l'\) by (rule \textit{Abs-Label-inverse})
\end{itemize}
\item \textbf{from} \(\text{Rep-Label}[\textit{of } l] \ l' \text{ have } l' @ [q] \in \text{Label}\) by (simp add: \textit{Label-def})
\item \textbf{with} \(l' \text{ have} \text{Rep-Label } l = \text{Rep-Label } (\text{Abs-Label } (l' @ [q]))\)
\item \textbf{by} (simp add: \textit{Abs-Label-inverse})
\item \textbf{hence} \(l = \text{Abs-Label } (l' @ [q])\) by (simp add: \textit{Rep-Label-inject})
\item \textbf{moreover}
\begin{itemize}
\item \textbf{from} \(\text{Rep-Label}[\textit{of } l] \ l' \text{ have } \text{length } l' < \text{Suc } f q \notin \text{set } l'\)
\item \textbf{by} (auto simp: \textit{Label-def})
\end{itemize}
\item \textbf{moreover}
\begin{itemize}
\item \textbf{note} \(\text{repl'}\)
\item \textbf{ultimately have} \(l \in \text{children } ?l''\)
\item \textbf{by} (auto simp: \textit{children-def is-leaf-def length-lbl-def set-lbl-def})
\item \textbf{with} \(\text{rhs}\) \textbf{show} \(\text{?thesis}\) by blast
\end{itemize}
\end{itemize}
\end{proof}
\end{proof}

\textbf{qed}

\textbf{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 \(l\).

\textbf{lemma \textit{children-last-set}}:
\begin{itemize}
\item \textbf{assumes} \(l : \neg(\text{is-leaf } l)\)
\item \textbf{shows} \(\text{last-lbl } ' (\text{children } l) = \text{UNIV} - \text{set-lbl } l\)
\end{itemize}
proof
  show last-lbl '(children l) ⊆ UNIV − set-lbl l
    by (auto dest: label-children simp: last-lbl-def)
next
  show UNIV − set-lbl l ⊆ last-lbl '(children l)
proof (auto simp: image-def)
    fix p
    assume p: p /∈ set-lbl l
    with l have c: Abs-Label (Rep-Label l @ [p]) ∈ children l
      by (auto simp: children-def)
    with Rep-Label-append[OF l] p
    show ∃c ∈ 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'
  assume c: c ∈ children l and c': c' ∈ children l
  and eq: last-lbl c = last-lbl c'
  from c c' obtain p p'
    where p: Rep-Label c = Rep-Label l @ [p]
      and p': Rep-Label c' = Rep-Label l @ [p']
    by (auto dest!: label-children)
  from p p' eq have p = p' by (simp add: last-lbl-def)
  with p p' have Rep-Label c = Rep-Label c' by simp
  thus c = c' 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 ~is-leaf l
  shows card (children l) = N − (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
Suppose a non-root label \( l' \) 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** \( sho\text{-}correct\text{-}vals \):

**Assumes**  
- \( run \colon SHORun \ EIG\text{-}M \ rho \ HOs \ SHOs \)
- \( l' \colon l' \in \text{children } l \)
- \( \text{shop} \colon \text{last-lbl } l' \in \text{SHOs} \ (\text{length-lbl } l) \ p \cap \text{HOs} \ (\text{length-lbl } l) \)

**Shows**  
- \( \text{vals} \ (\rho \ (\text{len } l') \ p) \ l' = \text{vals} \ (\rho \ (\text{len } l) \ q) \ l \)

**Proof** -

- let \( \forall r = \text{len } l \)
- from \( run \) obtain \( \mu p \)
  - where \( \text{nxt} \colon \text{nextState } EIG\text{-}M \ ?r \ p \ (\rho \ ?r \ p) \ \mu p \ (\rho \ (\text{Suc } ?r) \ p) \)
  - and \( \text{msg} \colon \mu p \in \text{SHOmsgVectors} \ EIG\text{-}M \ ?r \ p \ (\rho \ ?r) \ (\text{HOs} \ ?r \ p) \ (\text{SHOs} \ ?r \ p) \)
  - by \( \text{(auto simp: EIG\text{-}SHOMachine-def SHORun-eq SHOnextConfig-eq)} \)
- with \( \text{shop} \)
  - have \( \text{msl } l' \text{ show } \text{?thesis} \)
  - by \( \text{(auto simp: EIG\text{-}SHOMachine-def SHORun-eq SHOnextConfig-eq)} \)

**QED**

A process fixes the value \( \text{vals } l \) of a label at state \( \text{length-lbl } l \), and then never modifies the value.

**Lemma** \( keep\text{-}vals \):

**Assumes**  
- \( run \colon SHORun \ EIG\text{-}M \ rho \ HOs \ SHOs \)

**Shows**  
- \( \text{vals} \ (\rho \ (\text{length-lbl } l + \ n) \ p) \ l = \text{vals} \ (\rho \ (\text{length-lbl } l) \ p) \ l \)
  - (is \( \forall v \ n = \ ?v \))

**Proof**  

- (induct \( n \))
- show \( \forall v \ 0 = \ ?v \) by simp

**Next**

- fix \( n \)
- assume \( ih \colon \forall v \ n = \ ?v \)
- let \( \forall r = \text{length-lbl } l + \ n \)
- from \( run \) obtain \( \mu p \)
  - where \( \text{nxt} \colon \text{nextState } EIG\text{-}M \ ?r \ p \ (\rho \ ?r \ p) \ \mu p \ (\rho \ (\text{Suc } ?r) \ p) \)
  - by \( \text{(auto simp: EIG\text{-}SHOMachine-def SHORun-eq SHOnextConfig-eq)} \)
- with \( ih \) show \( \forall (\text{Suc } n) = \ ?v \)
  - by \( \text{(auto simp: EIG\text{-}SHOMachine-def nextState-def EIG\text{-}nextState-def next-main-def next-end-def extend-vals-def)} \)
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**
- $r$: $l' \in \text{children } l$
- $\text{skr}: \text{last-lbl } l' \in \text{SKr (HOs (length-lbl l)) (SHOs (length-lbl l))}$

**Shows**
- $\text{vals (rho (length-lbl l') p) l'} = \text{vals (rho (length-lbl l') q) l'}$

**Using**
- $\text{assms unfolding SKr-def by (auto simp: sho-correct vals)}$

**Proof**

Suppose that $l$ is a non-root label whose last element was well heard by all processes at round $r$, and that $l'$ 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 $l'$ by any process $p$ are identical.

**Lemma Lynch-6-16-a:**

**Assumes**
- $r$: $l \in \text{children } t$
- $\text{skr}: \text{last-lbl } l \in \text{SKr (HOs (length-lbl t)) (SHOs (length-lbl t))}$
- $\text{srr}' : \text{last-lbl } l' \in \text{SKr (HOs (length-lbl l)) (SHOs (length-lbl l))}$

**Shows**
- $\text{vals (rho (length-lbl l') p) l'} = \text{vals (rho (length-lbl l') q) l}$

**Using**
- $\text{assms by (auto simp: SKr-def sho-correct vals)}$

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

**Lemma Lynch-6-16-c:**

**Assumes**
- $\text{commR (EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l)))}$
- $\text{is EIG-commPerRd (HOs ?r) -}$
- $\text{l: \neg(is-leaf l)}$

**Shows**
- $\text{card \{l' \in \text{children } l. last-lbl l' \in \text{SKr (HOs ?r) (SHOs ?r)}\} > card (\text{children } l) div 2}$
- $\text{(is card ?lhs > -)}$

**Proof**

- **Let** $\text{skr = SKr (HOs ?r) (SHOs ?r)}$

**Have**
- $\text{last-lbl' ?lhs = ?skr - set-lbl l}$

**Proof**

- **From** $\text{children-last-set[OF l]}$

**Show**
- $\text{last-lbl' ?lhs \subseteq ?skr - set-lbl l}$

- **By** (auto simp: children-length)

**Next**

- **Fix** $p$
- **Assume** $p: p \in ?skr p \notin set-lbl l$
- **With** $\text{children-last-set[OF l]}$

153
have \( p \in \text{last-lbl} \cdot \text{children} \ l \) by auto
with \( p \) have \( p \in \text{last-lbl} \cdot \text{?lhs} \)
by (auto simp: image-def children-length)

thus \( \text{?skr} - \text{set-lbl} \ l \subseteq \text{last-lbl} \cdot \text{?lhs} \) by auto
qed

moreover
from \( \text{last-lbl-inj-on-children[of l]} \)
have \( \text{inj-on last-lbl ?lhs} \) by (auto simp: inj-on-def)
ultimately
have \( \text{card} \ ?\text{lhs} = \text{card} (\text{skr} - \text{set-lbl} l) \) by (auto dest: card-image)
finally have \( \text{card} \ ?\text{lhs} \geq (\text{card} \ ?\text{skr}) - \text{?r} \)
using \( \text{card-set-lbl[of l]} \) by simp

moreover
from \( \text{commR} \) have \( \text{card} \ ?\text{skr} > (N + f) \) div 2
by (auto simp: EIG-commPerRd-def)
with \( \text{not-leaf-length[OF l,f]} \)
have \( (\text{card} \ ?\text{skr}) - \text{?r} > (N - \text{?r}) \) div 2 by auto
with \( \text{card-children[OF l]} \)
have \( (\text{card} \ ?\text{skr}) - \text{?r} > \text{card} (\text{children} l) \) div 2 by simp

ultimately show \( \text{?thesis} \) by simp
qed

If \( l \) is a non-leaf label such that all of its children corresponding to well-heard processes at round \( \text{length-lbl} \ l \) have a uniform \text{newvals} decoration at round \( f+1 \), then \( l \) itself is decorated with that same value.

lemma \text{newvals-skr-uniform}:
assumes \( \text{run: SHORun EIG-M \rho HOs SHOs} \)
and \( \text{commR: EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))} \)
and \( \text{notleaf: \neg(is-leaf l)} \)
and \( \text{unif: } \forall l'. l' \in \text{children} \ l; \)
\( \text{last-lbl l'} \in \text{SKr (HOs (length-lbl l)) (SHOs (length-lbl l))} \)
\( \Rightarrow \text{newvals (rho (Suc f) p) l'} = v \)
shows \( \text{newvals (rho (Suc f) p) l = v} \)
proof –
from \( \text{unif} \)
have \( \text{card} \ \{l' \in \text{children} \ l. \text{last-lbl l'} \in \text{SKr (HOs ?r)} (\text{SHOs ?r})\} \)
\( \leq \text{card} \ \{l' \in \text{children} \ l. \text{newvals (rho (Suc f) p) l'} = v\} \)
by (auto intro: card-mono)
with \( \text{lynch-6-16-c[OF HOs l SHOs, OF commR notleaf]} \)
have \( \text{maj: has-majority v (newvals (rho (Suc f) p)) (children l)} \)
by (simp add: has-majority-def)

from \( \text{run} \) have \( \text{check-newvals (rho (Suc f) p)} \)
A node whose label \( l \) ends with a process which is well heard at round \( \text{length-lbl} \ l \) will have its \text{newvals} field set (at round \( f+1 \)) to the “fixed-up” value given by \text{vals}.

\textbf{lemma lynch-6-16-d:}

\begin{itemize}
\item \textbf{assumes run:} \text{SHORun} \ EIG-M \ rho \ HOs \ SHOs
\item \textbf{and commR:} \( \forall \ r. \ \text{EIG-commPerRd} \ (\text{HOs} \ r) \ (\text{SHOs} \ r) \)
\item \textbf{and notroot:} \( l \in \text{children} \ t \)
\item \textbf{and skr:} \( \text{last-lbl} \ l \in \text{SKr} \ (\text{HOs} \ (\text{length-lbl} \ t)) \ (\text{SHOs} \ (\text{length-lbl} \ t)) \)
\item \textbf{shows} \text{newvals} \( (\rho \ (\text{Suc} \ f) \ p) \ l = \text{fixupval} \ (\text{vals} \ (\rho \ (\text{Suc} \ f) \ p) \ l) \)
\end{itemize}

\textbf{using} notroot skr \textbf{proof} (induct \( \text{Suc} \ f - (\text{?len} \ l) \) arbitrary: \( l \ t \))

\begin{itemize}
\item \textbf{fix} \( l \ t \)
\item \textbf{assume} \( 0 = \text{Suc} \ f - \text{?len} \ l \)
\item \textbf{with} \text{length-lbl}[of \ l] \textbf{have} \( \text{leaf} : \text{is-leaf} \ l \text{ by} \) (simp add: is-leaf-def)
\end{itemize}

\begin{itemize}
\item \textbf{from run have} \text{check-newvals} \( (\rho \ (\text{Suc} \ f) \ p) \)
\item \textbf{by} (auto simp: \text{EIG-SHOMachine-def} \text{SHORun-eq} \text{SHOnextConfig-eq} \text{nextState-def} \text{EIG-nextState-def} \text{next-end-def})
\end{itemize}

\textbf{with} leaf \textbf{show} \( \exists \! P \ l \)
\textbf{by} (auto simp: check-newvals-def is-leaf-def)

\begin{itemize}
\item \textbf{next}
\item \textbf{fix} \( k \ l \ t \)
\item \textbf{assume} \( \exists \ l' \ t': \[
\begin{array}{l}
[k = \text{Suc} \ f - \text{length-lbl} \ l'; l' \in \text{children} \ t'; \\
\text{last-lbl} \ l' \in \text{SKr} \ (\text{HOs} \ (\text{?len} \ t')) \ (\text{SHOs} \ (\text{?len} \ t'))]
\implies \exists \! P \ l'
\end{array}
\]
\item \textbf{and flk:} \( \text{Suc} \ k = \text{Suc} \ f - \text{?len} \ l \)
\item \textbf{and notroot:} \( l \in \text{children} \ t \)
\item \textbf{and skr:} \( \text{last-lbl} \ l \in \text{SKr} \ (\text{HOs} \ (\text{?len} \ t)) \ (\text{SHOs} \ (\text{?len} \ t)) \)
\end{itemize}

\textbf{let} \( \exists \! v = \text{fixupval} \ (\text{vals} \ (\rho \ (\text{Suc} \ f) \ p) \ l) \)
\textbf{from flk have} \( \neg \text{(is-leaf} \ l) \text{ by} \) (simp add: is-leaf-def)

\begin{itemize}
\item \{ 
\item \textbf{fix} \( l' \)
\item \textbf{assume} \( l' : l' \in \text{children} \ l \)
\item \textbf{and skr':} \( \text{last-lbl} \ l' \in \text{SKr} \ (\text{HOs} \ (\text{?len} \ l)) \ (\text{SHOs} \ (\text{?len} \ l)) \)
\end{itemize}
from run notroot skr l' skr′
have vals (rho (?len l') p) l' = vals (rho (?len l) p) l
by (rule lynch-6-16-a)

moreover
from flk l' have k = Suc f - ?len l' by (simp add: children-length)
from this l' skr' have ?P l' by (rule sh)
ultimately
have newvals (rho (Suc f) p) l' = ?v
using notroot l' 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**

\[ \text{common } \rho \ l \equiv \ \forall p \ q. \ \text{newvals} (\rho (\text{Suc } f) p) l = \text{newvals} (\rho (\text{Suc } f) q) l \]

The subtrees of a given label are all its possible extensions.

**definition subtrees where**

\[ \text{subtrees } h \equiv \{ \ l . \ \exists t. \ \text{Rep-Label} \ l = (\text{Rep-Label} \ h) @ t \ \} \]

**lemma children-in-subtree:**

assumes \( l \in \text{children } h \)
shows \( l \in \text{subtrees } h \)
using label-children[OF assms] by (auto simp: subtrees-def)

**lemma subtrees-refl [iff]:** \( l \in \text{subtrees } l \)
by (auto simp: subtrees-def)

**lemma subtrees-root [iff]:** \( l \in \text{subtrees root-node} \)
by (auto simp: subtrees-def)

**lemma subtrees-trans:**

assumes \( l'' \in \text{subtrees } l' \ \text{and} \ l' \in \text{subtrees } l \)
shows \( l'' \in \text{subtrees } l \)
using assms by (auto simp: subtrees-def)

**lemma subtrees-antisym:**

assumes \( l \in \text{subtrees } l' \ \text{and} \ l' \in \text{subtrees } l \)
shows \( l' = l \)
using assms by (auto simp: subtrees-def Rep-Label-inject)

**lemma subtrees-tree:**
assumes $l': l \in \text{subtrees } l'$ and $l'': l \in \text{subtrees } l''$
shows $l' \in \text{subtrees } l'' \lor l'' \in \text{subtrees } l'$
using assms proof (auto simp: subtrees-def append-eq-append-conv2)
fix $xs$
assume $\text{Rep-Label } l' \in \text{subtrees } l''$
by (rule sym)
thus $\exists ys. \text{Rep-Label } l' = \text{Rep-Label } l'' \in ys$. .
qed

lemma subtrees-cases:
assumes $l': l' \in \text{subtrees } l$
and self: $l' = l \implies P$
and child: $\forall c. [c \in \text{children} l \land l' \in \text{subtrees } c] \implies P$
shows $P$
proof
from $l'$ obtain $t$ where $t$: $\text{Rep-Label } l' = (\text{Rep-Label } l) @ t$
by (auto simp: subtrees-def)
have $l' = l \lor (\exists c \in \text{children } l. l' \in \text{subtrees } c)$
proof (cases $t$)
assume $t = []$
with $t$ show $\text{thesis}$ by (simp add: Rep-Label-inject)
next
fix $p$ $t'$
assume cons: $t = p \# t'$
from $\text{Rep-Label}[of l']$ $t$ have length (Rep-Label $l @ t) \leq \text{Suc } f$
by (auto simp: Label-def)
with cons have notleaf: $\neg(\text{is-leaf } l)$
by (auto simp: is-leaf-def length-lbl-def)
let $?c = \text{Abs-Label } (\text{Rep-Label } l @ [p])$
from $t$ cons $\text{Rep-Label}[of l']$ have $p$: $p \notin \text{set-lbl } l$
by (auto simp: Label-def set-lbl-def)
with notleaf have $c$: $?c \in \text{children } l$
by (auto simp: children-def)
moreover
from notleaf $p$ have $\text{Rep-Label } l @ [p] \in \text{Label}$
by (simp add: Rep-Label-append)
from $\text{Rep-Label } l @ [p]$ have $t$: $\forall c. ?c \in \text{subtrees } ?c$
by (auto simp: subtrees-def)
ultimately show $\text{thesis}$ by blast
qed
thus $\text{thesis}$ by (auto elim!: self child)
qed

lemma subtrees-leaf:
assumes $l$: is-leaf $l$ and $l': l' \in \text{subtrees } l$
shows $l' = l$
using \( l' \) proof (rule subtrees-cases)

fix \( c \)
assume \( c \in \text{children } l \) — impossible
with \( l \) show \( \text{thesis by (simp add: children-def)} \)
qed

lemma \textit{children-subtrees-equal}:
assumes \( c: c \in \text{children } l \) and \( c': c' \in \text{children } l \)
and \( \text{sub: } c' \in \text{subtrees } c \)
shows \( c' = c \)
proof —
from assms 
have \( \text{Rep-Label } c' = \text{Rep-Label } c \)
by (auto simp: subtrees-def dest!: label-children)
thus \( \text{thesis by (simp add: Rep-Label-inject)} \)
qed

A set \( C \) of labels is a \textit{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 \textit{subcovering} where
\( \text{subcovering } C \; l \equiv \forall s \in \text{subtrees } l. \; \text{is-leaf } s \rightarrow (\exists h \in C. \; h \in \text{subtrees } l \land s \in \text{subtrees } h) \)

A \textit{covering} is a subcovering w.r.t. the root node.

abbreviation \textit{covering} where
\( \text{covering } C \equiv \text{subcovering } C \; \text{root-node} \)

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 \textit{lynch-6-18-a}:
assumes \( \text{SHORun } EIG-M \; \rho \; \text{HOs SHOs} \)
and \( \forall r. \; \text{EIG-commPerRd } (\text{HOs } r) \; (\text{SHOs } r) \)
and \( l \in \text{children } t \)
and \( \text{last-lbl } t \in \text{SKr } (\text{HOs } (\text{length-lbl } t)) \; (\text{SHOs } (\text{length-lbl } t)) \)
shows \( \text{common } \rho \; l \)
using assms
by (auto simp: common-def lynch-6-16-d lynch-6-15
intro: arg-cong[where \( f=\text{fixupval} \)])

lemma \textit{lynch-6-18-b}:
assumes \( \text{run: } \text{SHORun } EIG-M \; \rho \; \text{HOs SHOs} \)
and \( \text{commG: } \text{EIG-commGlobal } \text{HOs SHOs} \)
and \( \text{commR: } \forall r. \; \text{EIG-commPerRd } (\text{HOs } r) \; (\text{SHOs } r) \)
shows \( \text{covering } \{l. \; \exists t. \; l \in \text{children } t \land \text{last-lbl } l \in (\text{SK } \text{HOs SHOs})\} \)
proof (clarsimp simp: subcovering-def)
fix \( l \)
assume \( \text{is-leaf } l \)
with \text{card-set-lbl}[of \( l \)] 
have \( \text{card } (\text{set-lbl } l) = \text{Suc } f \)
by \((\text{simp add: is-leaf-def})\)

with \(\text{commG have } N < \text{ card } (SK \text{ HOs SHOs}) + \text{ card } (\text{set-lbl } l)\)
by \((\text{simp add: EIG-commGlobal-def})\)

hence \(\exists q \in \text{ set-lbl } l . \ q \in SK \text{ HOs SHOs}\)
by \((\text{auto dest: majorities-intersect})\)

then obtain \(l_1 \ q \ l_2\) where \(l: \text{Rep-Label } l = (l_1 @ [q]) @ l_2\) and \(q \in SK \text{ HOs SHOs}\)

unfolding set-lbl-def by \((\text{auto intro: split-list-propE})\)

let \(?h = \text{Abs-Label } (l_1 @ [q])\)
from \(\text{Rep-Label[of } l\) lhave \(l_1 @ [q] \in \text{Label}\) by \((\text{simp add: Label-def})\)

hence \(?h \neq \text{root-node}\) by \(\text{auto}\)
then obtain \(t\) where \(?h \in \text{children } t\)
by \((\text{auto simp: root-iff-no-child})\)

moreover
from \(\text{reph q have } \text{last-lbl } ?h \in SK \text{ HOs SHOs}\) by \((\text{simp add: last-lbl-def})\)

moreover
from \(\text{reph } l\) have \(l \in \text{subtrees } ?h\) by \((\text{simp add: subtrees-def})\)

ultimately
show \(\exists h. (\exists t . h \in \text{children } t) \land \text{last-lbl } h \in SK \text{ HOs SHOs} \land l \in \text{subtrees } h\)
by \(\text{blast}\)

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 \(\text{cov: subcovering } C l\)
and \(l: l \notin C\)
and \(e: e \in \text{children } l\)
shows \(\text{subcovering } C e\)
proof \((\text{clarsimp simp: subcovering-def})\)
fix \(s\)
assume \(s: s \in \text{subtrees } e\) and \(\text{leaf: is-leaf } s\)
from \(s \text{ children-in-subtree[OF } e]\) have \(s \in \text{subtrees } l\)
by \((\text{rule subtrees-trans})\)
with \(\text{leaf cov obtain } h \text{ where } h: h \in C \ h \in \text{subtrees } l \ s \in \text{subtrees } h\)
by \((\text{auto simp: subcovering-def})\)

with \(l\) obtain \(e'\) where \(e': e' \in \text{children } l \ h \in \text{subtrees } e'\)
by \((\text{auto elim: subtrees-cases})\)
from \((s \in \text{subtrees } h) \ h \in \text{subtrees } e'\) have \(s \in \text{subtrees } e'\)
by \((\text{rule subtrees-trans})\)

with \(s\) have \(e \in \text{subtrees } e' \lor e' \in \text{subtrees } e\)
by \((\text{rule subtrees-tree})\)
with \(e e'\) have \(e' = e\)
by \((\text{auto dest: children-subtrees-equal})\)
with \(e' \) show \(\exists h \in C . h \in \text{subtrees } e \land s \in \text{subtrees } h\) by \(\text{blast}\)
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 $C l$
and com: $\forall l' \in C. \text{common rho l'}$

**Shows** common rho $l$

**Using** cov proof (induct Suc $f$ - length-lbl $l$ arbitrary: $l$)

**Proof**

1. Fix $l$
2. Assume $0: 0 = \text{Suc } f - \text{length-lbl } l$
   - and $C$: subcovering $C l$
   - From $0 \text{length-lbl}[of } l$ have is-leaf $l$
     - by (simp add: is-leaf-def)
   - With $C$ obtain $h$ where $h: h \in C \text{ h } \in \text{subtrees } l \text{ l } \in \text{subtrees } h$
     - by (auto simp: subcovering-def)
   - Hence $l \in C$ by (auto dest: subtrees-antisym)
   - With com show common rho $l$ ..

**Next**

1. Fix $k l$
2. Assume $k: \text{Suc } k = \text{Suc } f - \text{length-lbl } l$
   - and $C$: subcovering $C l$
   - and $\text{ih}: \forall l' \text{, } [k = \text{Suc } f - \text{length-lbl } l'; \text{subcovering } C l'] \implies \text{common rho } l'$
3. Show common rho $l$
4. Proof (cases $l \in C$)
   - Case True
     - With com show thesis ..
   - Next
     - Case False
       - With $C$ have $\forall e \in \text{children } l. \text{subcovering } C e$
         - by (blast intro: Lynch-6-19-a)
       - Moreover
         - From $k$ have $\forall e \in \text{children } l. k = \text{Suc } f - \text{length-lbl } e$
           - by (auto simp: children-length)
         - Ultimately
           - Have com-ch: $\forall e \in \text{children } l. \text{common rho } e$
             - by (blast intro: ih)
5. Show thesis
6. Proof (clarsimp simp: common-def)

1. Fix $p q$
2. From $k$ have notleaf: $\neg (\text{is-leaf } l)$ by (simp add: is-leaf-def)
3. Let $？r = \text{Suc } f$
4. From com-ch
5. Have $\forall e \in \text{children } l. \text{newvals } (\text{rho } ?r p) e = \text{newvals } (\text{rho } ?r q) e$
   - by (auto simp: common-def)
6. Hence $\forall w. \{ e \in \text{children } l. \text{newvals } (\text{rho } ?r p) e = w \}$
   - $= \{ e \in \text{children } l. \text{newvals } (\text{rho } ?r q) e = w \}$
   - by auto
7. Moreover

160
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
(∃ w. has-majority w (newvals (rho ?r p)) (children l)
 ∧ newvals (rho ?r p) l = w)
∨ ¬(∃ w. has-majority w (newvals (rho ?r p)) (children l))
 ∧ newvals (rho ?r p) l = undefined
(∃ w. has-majority w (newvals (rho ?r q)) (children l)
 ∧ newvals (rho ?r q) l = w)
∨ ¬(∃ 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: ∀ 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 ∈ children t last-lbl l ∈ SK HOs 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 μp
where EIG-nextState r p (rho r p) μp (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 ¬(r < f) r ≠ f
  with ih
  show decide (rho r p) = Some (newvals (rho (Suc f) p) root-node)
  by simp
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: ∀ r. 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: ∀ r. EIG-commPerRd (HOs r) (SHOs r)
  and initv: ∀ q. the (vals (rho 0 q) root-node) = v
  and dp: decide (rho r p) = Some w
  shows v = w
proof
  have v: ∀ q. vals (rho 0 q) root-node = Some v
  proof
    fix q
    from run have vals (rho 0 q) root-node ≠ 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 show vals (rho 0 q) root-node = Some v by simp
qed

let $\text{len} = \text{length-lbl}$
let $\text{Suc } f$

\{
  fix $l'$
  assume $l': l' \in \text{children root-node}$
  and $\text{skr: last-lbl } l' \in \text{SKr } (\text{HOs } 0) (\text{SHOs } 0)$
  with run $v$ have $\text{vals } (\rho \ (\text{len } l') \ p) \ l' = \text{Some } v$
  by (auto dest: sho-correct-val simp: SKr-def)

  moreover
  from run $\text{commR } l' \text{ skr}$
  have newvals $\text{(rho } ?r \ p) \ l' = \text{fixupval } (\text{vals } (\rho \text{(len } l') \ p) \ l')$
  by (auto intro: lynch-6-16-d)

  ultimately
  have newvals $\text{(rho } ?r \ p) \ l' = v$ by simp
\}
with run $\text{commR root-node-not-leaf}$
have newvals $\text{(rho } ?r \ p) \text{ root-node } = v$
  by (auto intro: newvals-skr-uniform)
with $\text{dp}$ show $\text{thesis}$ by (simp add: decide[OF run])

Termination is trivial for $\text{EIGByz}_f$.

\textbf{theorem} \textit{Termination}:
\textbf{assumes} $\text{SHORun EIG-M } \rho \text{ HO} \text{s SHO} \text{s}$
\textbf{shows} $\exists r \ v. \text{ decide } (\rho \ r \ p) = \text{Some } v$
\textbf{using} assms by (auto simp: decide)

10.8 \textit{EIGByz}_f \text{ Solves Weak Consensus}

Summing up, all (coarse-grained) runs of $\text{EIGByz}_f$ for HO and SHO collections that satisfy the communication predicate satisfy the Weak Consensus property.

\textbf{theorem} $\text{eig-weak-consensus}$:
\textbf{assumes} run: $\text{SHORun EIG-M } \rho \text{ HO} \text{s SHO} \text{s}$
\textbf{and} commR: $\forall r. \text{ EIG-commPerRd } (\text{HO} \text{s } r) (\text{SHO} \text{s } r)$
\textbf{and} commG: $\text{EIG-commGlobal } \text{HO} \text{s SHO} \text{s}$
\textbf{shows} weak-consensus $\lambda p. \text{ the } (\text{vals } (\rho \ 0 \ p) \text{ root-node}) \text{ decide rho}$
\textbf{unfolding} weak-consensus-def
\textbf{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**

- \(\text{run}: \text{fg-run EIG-M rho HOs SHOs (} \lambda r \ q. \ \text{undefined})\)
- and \(\text{commR: } \forall r. \ EIG\text{-commPerRd (HOs } r\) (SHOs } r)\)
- and \(\text{commG: } \text{EIG\text{-commGlobal HOs SHOs}\)

**Shows** \(\text{weak-consensus (} \lambda p. \ \text{the (vals (state (rho } 0\) p) root-node)}\))

\(\text{decide (state } \circ \rho)\)

(is weak-consensus ?inits - -)

**Proof**

(rule local-property-reduction[OF run weak-consensus-is-local])

fix \(\text{crun}\)

assume \(\text{crun: CSHORun EIG-M crun HOs SHOs (} \lambda r \ q. \ \text{undefined})\)

and \(\text{init: crun } 0 = \text{state (rho } 0)\)

from \(\text{crun}\) have \(\text{SHORun EIG-M crun HOs SHOs by (unfold SHORun-def)}\)

from this \(\text{commR commG}\)

have \(\text{weak-consensus (} \lambda p. \ \text{the (vals (crun } 0\) p) root-node}\)) \(\text{decide crun}\)

by (rule eig-weak-consensus)

with \(\text{init show weak-consensus ?inits decide crun}\)

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 relates 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 \cite{10}, but our proof here is an order of magnitude shorter, although we prove safety and liveness properties, whereas only safety was considered in \cite{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 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


