Verifying Fault-Tolerant Distributed Algorithms In The Heard-Of Model*

Henri Debrat$^1$ and Stephan Merz$^2$
$^1$ Université de Lorraine & LORIA
$^2$ Inria Nancy Grand-Est & LORIA
Villers-lès-Nancy, France

April 19, 2020

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

8.1 Model of the Algorithm ............................................. 97
8.2 Communication Predicate for $\mathcal{U}_{T,E,\alpha}$ .................. 100
8.3 The $\mathcal{U}_{T,E,\alpha}$ Heard-Of Machine ........................ 100
8.4 Preliminary Lemmas ................................................ 101
8.5 Proof of Agreement and Validity ................................. 105
8.6 Proof of Termination ................................................. 115
8.7 $\mathcal{U}_{T,E,\alpha}$ Solves Weak Consensus ....................... 122

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

9.1 Model of the Algorithm ............................................. 123
9.2 Communication Predicate for $\mathcal{A}_{T,E,\alpha}$ .................. 124
9.3 The $\mathcal{A}_{T,E,\alpha}$ Heard-Of Machine ........................ 125
9.4 Preliminary Lemmas ................................................ 126
9.5 Proof of Validity ...................................................... 130
9.6 Proof of Agreement .................................................. 133
9.7 Proof of Termination ................................................. 137
9.8 $\mathcal{A}_{T,E,\alpha}$ Solves Weak Consensus ....................... 142

10 Verification of the $EIGByz_f$ Consensus Algorithm

10.1 Tree Data Structure ................................................. 143
10.2 Model of the Algorithm ............................................. 145
10.3 Communication Predicate for $EIGByz_f$ ....................... 146
10.4 The $EIGByz_f$ Heard-Of Machine .............................. 147
10.5 Preliminary Lemmas ................................................ 148
10.6 Lynch’s Lemmas and Theorems ................................... 153
10.7 Proof of Agreement, Validity, and Termination ............... 162
10.8 $EIGByz_f$ Solves Weak Consensus .............................. 163

11 Conclusion .............................................................. 164
1 Introduction

We are interested in the verification of fault-tolerant distributed algorithms. The archetypical problem in this area is the Consensus problem that requires a set of distributed nodes to achieve agreement on a common value in the presence of faults. Such algorithms are notoriously hard to design and to get right. This is particularly true in the presence of asynchronous communication: the landmark theorem by Fischer, Lynch, and Paterson [9] shows that there is no algorithm solving the Consensus problem for asynchronous systems in the presence of even a single, permanent fault. Existing solutions therefore rely on assumptions of “partial synchrony” [8].

Different computational models, and different concepts for specifying the kinds and numbers of faults such algorithms must tolerate, have been introduced in the literature on distributed computing. This abundance of subtly different notions makes it very difficult to compare different algorithms, and has sometimes even led to misunderstandings and misinterpretations of what an algorithm claims to achieve. The general lack of rigorous, let alone formal, correctness proofs for this class of algorithms makes it even harder to understand the field.

In this contribution, we formalize in Isabelle/HOL the Heard-Of (HO) model, originally introduced by Charron-Bost and Schiper [7]. This model can represent algorithms that operate in communication-closed rounds, which is true of virtually all known fault-tolerant distributed algorithms. Assumptions on failures tolerated by an algorithm are expressed by communication predicates that impose bounds on the set of messages that are not received during executions. Charron-Bost and Schiper show how the known failure hypotheses from the literature can be represented in this format. The Heard-Of model therefore makes an interesting target for formalizing different algorithms, and for proving their correctness, in a uniform way. In particular, different assumptions can be compared, and the suitability of an algorithm for a particular situation can be evaluated.

The HO model has subsequently been extended [3] to encompass algorithms designed to tolerate value (also known as malicious or Byzantine) faults. In the present work, we propose a generic framework in Isabelle/HOL that encompasses the different variants of HO algorithms, including resilience to benign or value faults, as well as coordinated and non-coordinated algorithms.

A fundamental design decision when modeling distributed algorithm is to determine the unit of atomicity. We formally relate in Isabelle two definitions of runs: we first define “coarse-grained” executions, in which entire rounds are executed atomically, and then define “fine-grained” executions that correspond to conventional interleaving representations of asynchronous networks. We formally prove that every fine-grained execution corresponds
to a certain coarse-grained execution, such that every process observes the same sequence of local states in the two executions, up to stuttering. As a corollary, a large class of correctness properties, including Consensus, can be transferred from coarse-grained to fine-grained executions.

We then apply our framework for verifying six different distributed Consensus algorithms w.r.t. their respective communication predicates. The first three algorithms, One-Third Rule, 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 if-split-asm [split] — perform default perform case splitting on conditionals

2 Heard-Of Algorithms

2.1 The Consensus Problem

We are interested in the verification of fault-tolerant distributed algorithms. The Consensus problem is paradigmatic in this area. Stated informally, it assumes that all processes participating in the algorithm initially propose some value, and that they may at some point decide some value. It is required that every process eventually decides, and that all processes must decide the same value.

More formally, we represent runs of algorithms as $\omega$-sequences of configurations (vectors of process states). Hence, a run is modeled as a function of type $\text{nat} \Rightarrow \text{proc} \Rightarrow \text{pst}$ where type variables $\text{proc}$ and $\text{pst}$ 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 $\text{dec} : \text{pst} \Rightarrow \text{val option}$ 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.

**type-synonym**

\[
\texttt{(proc, pst)} \text{ run} = \texttt{nat} \Rightarrow \texttt{proc} \Rightarrow \texttt{pst}
\]

**definition**

\[
\text{consensus} :: (\texttt{proc} \Rightarrow \texttt{val}) \Rightarrow (\texttt{pst} \Rightarrow \texttt{val} \texttt{option}) \Rightarrow (\texttt{proc, pst}) \text{ run} \Rightarrow \texttt{bool}
\]

**where**

\[
\text{consensus vals dec rho} \equiv
\begin{aligned}
(\forall n p v. \, \text{dec} (\rho n p) = \text{Some } v \rightarrow v \in \text{range } vals) \\
\land (\forall m n p q w. \, \text{dec} (\rho m p) = \text{Some } v \land \text{dec} (\rho n q) = \text{Some } w \\
\quad \rightarrow v = w) \\
\land (\forall p. \exists n. \, \text{dec} (\rho n p) \neq \text{None})
\end{aligned}
\]

A variant of the Consensus problem replaces the Integrity requirement by

**Validity.** If all processes initially propose the same value \( v \) then every process may only decide \( v \).

**definition**

\[
\text{weak-consensus} \text{ where}
\]

\[
\text{weak-consensus vals dec rho} \equiv
\begin{aligned}
(\forall v. \, (\forall p. \, \text{vals } p = v) \rightarrow (\forall n p w. \, \text{dec} (\rho n p) = \text{Some } w \rightarrow w = v)) \\
\land (\forall m n p q w. \, \text{dec} (\rho m p) = \text{Some } v \land \text{dec} (\rho n q) = \text{Some } w \\
\quad \rightarrow v = w) \\
\land (\forall p. \exists n. \, \text{dec} (\rho n p) \neq \text{None})
\end{aligned}
\]

Clearly, \text{consensus} implies \text{weak-consensus}.

**lemma** \text{consensus-then-weak-consensus}:

**assumes** \text{consensus vals dec rho}

**shows** \text{weak-consensus vals dec rho}

**using** \text{assms by (auto simp: consensus-def weak-consensus-def image-def)}

Over Boolean values (“binary Consensus”), \text{weak-consensus} implies \text{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., \( \text{card (range vals)} \leq 2 \)).

**lemma** \text{binary-weak-consensus-then-consensus}:

**assumes** \text{bc: weak-consensus (vals::'proc ⇒ bool) dec rho}

**shows** \text{consensus vals dec rho}

**proof**

\[
\begin{aligned}
\{ \quad \text{Show the Integrity property, the other conjuncts are the same.} \\
\text{fix } n p v
\end{aligned}
\]
assume \( \text{dec} \cdot \text{dec} \ (\rho \ n \ p) = \text{Some} \ v \)

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

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

case True

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

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

next

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 \( \text{integrity} = \text{this} \)

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

unfolding \( \text{consensus-def weak-consensus-def} \) 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 \cite{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” \( \text{HO}(p,r) \) 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 \( \text{HO}(p,r) \) 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 \( \text{HO}(p,r) \) 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 \cite{3} adds a second collection of sets \( \text{SHO}(p,r) \subseteq \text{HO}(p,r) \) 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 \( \text{HO}(p,r) \setminus \text{SHO}(p,r) \) 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 $\text{HO}(p,r)$ and $\text{SHO}(p,r)$. 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 $\text{HO}(p,r)$, the definitions assume an external coordinator assignment such that $\text{coord}(p,r)$ 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 $\text{CinitState}$ such that $\text{CinitState} p \text{ st \ crd}$ is true precisely of the initial states $\text{st}$ of process $p$, assuming that $\text{crd}$ is the initial coordinator of $p$,
- a function $\text{sendMsg}$ where $\text{sendMsg} r p q \text{ st}$ yields the message that process $p$ sends to process $q$ at round $r$, given its local state $\text{st}$, and
- a predicate $\text{CnextState}$ where $\text{CnextState} r p \text{ st \ msgs \ crd \ st'}$ characterizes the successor states $\text{st'}$ of process $p$ at round $r$, given current state $\text{st}$, the vector $\text{msgs :: '\text{proc} \Rightarrow '\text{msg \ option}}$ of messages that $p$ received at round $r$ ($\text{msgs} q = \text{None}$ indicates that no message has been received from process $q$), and process $\text{crd}$ 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 $\text{sendMsg}$.

We represent an algorithm by a record as follows.

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

8
For non-coordinated HO algorithms, the coordinator argument of functions \( C_{\text{initState}} \) and \( C_{\text{nextState}} \) is irrelevant, and we define utility functions that omit that argument.

**definition** \( \text{isNCA} \)\( \text{Algorithm} \) where
\[
\text{isNCA} \text{Algorithm} \ \text{alg} \equiv
\forall p \ st \ crd \ crd'. \ C_{\text{initState}} \ \text{alg} \ p \ st \ crd = C_{\text{initState}} \ \text{alg} \ p \ st \ crd'
\land (\forall r \ p \ st \ msgs \ crd \ crd' \ st'. \ C_{\text{nextState}} \ \text{alg} \ r \ p \ st \ msgs \ crd \ st'
\quad = C_{\text{nextState}} \ \text{alg} \ r \ p \ st \ msgs \ crd' \ st')
\]

**definition** \( \text{initState} \) where
\[
\text{initState} \ \text{alg} \ p \ st \equiv C_{\text{initState}} \ \text{alg} \ p \ st \ \text{undefined}
\]

**definition** \( \text{nextState} \) where
\[
\text{nextState} \ \text{alg} \ r \ p \ st \ msgs \ st' \equiv C_{\text{nextState}} \ \text{alg} \ r \ p \ st \ msgs \ \text{undefined} \ st'
\]

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

**type-synonym**
\[
'\text{proc} \ \text{HO} = '\text{proc} \Rightarrow '\text{proc} \ \text{set}
\]

**type-synonym**
\[
'\text{proc} \ \text{coord} = '\text{proc} \Rightarrow '\text{proc}
\]

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

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

**definition** $\text{SHOmsgVectors}$ where

$\text{SHOmsgVectors} \text{ cfg} \text{ HOp} \text{ SHOp} \equiv$

$\{\mu. (\forall q. q \in \text{HOp} \leftrightarrow \mu q \neq \text{None})$

$\land (\forall q. q \in \text{SHOp} \cap \text{HOp} \rightarrow \mu q = \text{Some (sendMsg A r q p (cfg q)))}\}$

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

**definition** $\text{CSHOnextConfig}$ where

$\text{CSHOnextConfig} \text{ cfg} \text{ HOp} \text{ SHOp} \equiv$

$\forall p. \exists \mu \in \text{SHOmsgVectors} \text{ cfg} \text{ HOp} \text{ SHOp} \rightarrow \text{CnextState} A \text{ p} \text{ (cfg p)} \mu \text{ (coord p)} \text{ (cfg' p)}$

**definition** $\text{CSHORun}$ where

$\text{CSHORun} \text{ rho} \text{ SHOs} \text{ SHOs} \text{ coords} \equiv$

$\text{CHOinitConfig} \text{ rho} \text{ 0} \text{ coords} \equiv$

$\text{CnextConfig} A \text{ r} \text{ 0} \text{ (HOs r)} \text{ (SHOs r)} \text{ (coords (Suc r))}$

For non-coordinated algorithms, the $\text{coord}$ 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** $\text{HOinitConfig}$ where

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

**lemma** $\text{HOinitConfig-eq}$:

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

**by** (auto simp: $\text{HOinitConfig-def}$ $\text{CHOinitConfig-def}$ $\text{initState-def}$)

**definition** $\text{SHOnextConfig}$ where

$\text{SHOnextConfig} \text{ cfg} \equiv \text{CnextState} A \text{ p} \text{ (cfg p)} \mu \text{ (cfg' p)}$

**lemma** $\text{SHOnextConfig-eq}$:

$\forall p. \exists \mu \in \text{SHOmsgVectors} \text{ cfg} \text{ HOp} \text{ SHOp} \rightarrow \text{CnextState} A \text{ p} \text{ (cfg p)} \mu \text{ (cfg' p)}$

**by** (auto simp: $\text{HOinitConfig-def}$ $\text{SHOnextConfig-def}$ $\text{SHOmsgVectors-def}$ $\text{CnextState-def}$)
definition \textit{SHORun} where
\begin{align*}
\text{SHORun} A \rho \text{HOs} \text{SHOs} &\equiv \\
\text{CSHORun} A \rho \text{HOs} \text{SHOs} (\lambda r q. \text{undefined})
\end{align*}

lemma \textit{SHORun-eq}:
\begin{align*}
\text{SHORun} A \rho \text{HOs} \text{SHOs} = \\
(\text{HOinitConfig} A (\rho 0)) \\
\land (\forall r. \text{SHOnextConfig} A r (\rho r) (\text{HOs} r) (\text{SHOs} r) (\rho (\text{Suc} r)))
\end{align*}
by (auto simp: \textit{SHORun-def} \textit{CSHORun-def} \textit{HOinitConfig-def} \textit{SHOnextConfig-def})

Algorithms designed to tolerate benign failures are not subject to message
corruption, and therefore the \text{SHO} sets are irrelevant (more formally, each
\text{SHO} set equals the corresponding \text{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 \text{HO} sets.

definition \textit{HOrcvdMsgs} where
\begin{align*}
\text{HOrcvdMsgs} A r p \text{HO} \text{cfg} &\equiv \\
(\lambda q. \text{if } q \in \text{HO} \text{ then Some (sendMsg} A r q p \text{ (cfg} q) \text{) else None})
\end{align*}

lemma \textit{SHOmsgVectors-HO}:
\begin{align*}
\text{SHOmsgVectors} A r p \text{HO} \text{cfg} \text{HO} \text{HO} = \\
\{ \text{HOrcvdMsgs} A r p \text{HO} \text{cfg}\}
\end{align*}
unfolding \textit{SHOmsgVectors-def} \textit{HOrcvdMsgs-def} by auto

With coordinators

definition \textit{CHOnextConfig} where
\begin{align*}
\text{CHOnextConfig} A r \text{cfg} \text{HO} \text{coord} \text{cfg}' &\equiv \\
\text{CSHOnextConfig} A r \text{cfg} \text{HO} \text{coord} \text{cfg}'
\end{align*}

lemma \textit{CHOnextConfig-eq}:
\begin{align*}
\text{CHOnextConfig} A r \text{cfg} \text{HO} \text{coord} \text{cfg}' = \\
(\forall p. \text{CnextState} A r p (\text{cfg} p) (\text{HOrcvdMsgs} A r p (\text{HO} p) \text{cfg}) \\
(\text{coord} p) (\text{cfg}' p))
\end{align*}
by (auto simp: \textit{CHOnextConfig-def} \textit{CSHOnextConfig-def} \textit{SHOmsgVectors-HO})

definition \textit{CHORun} where
\begin{align*}
\text{CHORun} A \rho \text{HOs} \text{coords} &\equiv \\
\text{CSHORun} A \rho \text{HOs} \text{HOs} \text{coords}
\end{align*}

lemma \textit{CHORun-eq}:
\begin{align*}
\text{CHORun} A \rho \text{HOs} \text{coords} = \\
(\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)))
\end{align*}
by (auto simp: \textit{CHORun-def} \textit{CSHORun-def} \textit{CHOinitConfig-def} \textit{CHOnextConfig-def})

Without coordinators

definition \textit{HOnextConfig} where
\[ HOnextConfig \ A \ r \ cfg \ HO \ cfg' \equiv \ SHOnextConfig \ A \ r \ cfg \ HO \ HO \ cfg' \]

**Lemma** \(HOnextConfig-eq\):
\[ HOnextConfig \ A \ r \ cfg \ HO \ cfg' = (\forall \ p. \ nextState \ A \ r \ p \ (cfg \ p) \ (HOrcvdMsgs \ A \ r \ p \ (HO \ p) \ (cfg) \ (cfg' \ p))) \]
by (auto simp: HOnextConfig-def SHOnextConfig-eq SHOmsgVectors-HO)

**Definition** \(HORun\) where
\[ HORun \ A \ rho \ HOs \equiv \ SHORun \ A \ rho \ HOs \ HOs \]

**Lemma** \(HORun-eq\):
\[ HORun \ A \ rho \ HOs = \]
\[ (HOinitConfig \ A \ (\rho \ 0)) \land (\forall \ r. \ HOnextConfig \ A \ r \ (\rho \ r) \ (HOs \ r) \ (\rho \ (Suc \ r))) \]
by (auto simp: HORun-def SHORun-eq HOnextConfig-def)

The following derived proof rules are immediate consequences of the definition of \(CHORun\); they simplify automatic reasoning.

**Lemma** \(CHORun-0\):
assumes \(CHORun \ A \ rho \ HOs \ coords\)
and \(\bigwedge \ cfg. \ CHOinitConfig \ A \ cfg \ (coords \ 0) \implies P \ cfg\)
shows \(P \ (\rho \ 0)\)
using \(assms\) unfolding \(CHORun-eq\) by blast

**Lemma** \(CHORun-Suc\):
assumes \(CHORun \ A \ rho \ HOs \ coords\)
and \(\bigwedge \ r. \ CHOnextConfig \ A \ r \ (\rho \ r) \ (HOs \ r) \ (coords \ (Suc \ r)) \ (\rho \ (Suc \ r))\)
\(\implies P \ r\)
shows \(P \ n\)
using \(assms\) unfolding \(CHORun-eq\) by blast

**Lemma** \(CHORun-induct\):
assumes \(run: \ CHORun \ A \ rho \ HOs \ coords\)
and \(init: \ CHOinitConfig \ A \ (\rho \ 0) \ (coords \ 0) \implies P \ 0\)
and \(step: \bigwedge \ r. \ [ \ P \ r; \ CHOnextConfig \ A \ r \ (\rho \ r) \ (HOs \ r) \ (coords \ (Suc \ r)) \]
\((\rho \ (Suc \ r)) \implies P \ (Suc \ r)\)
shows \(P \ n\)
using \(run\) unfolding \(CHORun-eq\) by (induct \(n\), auto elim: init step)

Because algorithms will not operate for arbitrary \(HO\), \(SHO\), and coordinator assignments, these are constrained by a communication predicate. For convenience, we split this predicate into a per Round part that is expected to hold at every round and a global part that must hold of the sequence of (S)HO assignments and may thus express liveness assumptions.

In the parlance of \(\text{[7]}\), 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** \(\text{'proc, 'pst, 'msg} \ HOMachine = \text{'proc, 'pst, 'msg} \ CHOAlgorithm + \)
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 ex-
ample, 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.

record ('pst', 'proc', 'msg) config =
  round :: 'proc $\Rightarrow$ nat
  state :: 'proc $\Rightarrow$ 'pst
  sent :: 'proc $\Rightarrow$ 'proc set
  rcvd :: 'proc $\Rightarrow$ 'proc $\Rightarrow$ 'msg option
  network :: ('proc * nat * 'proc * 'msg) set

type-synonym ('pst,'proc,'msg) fgrun = nat $\Rightarrow$ ('pst,'proc,'msg) config

In an initial configuration for an algorithm, the local state of every process
satisfies the algorithm’s initial-state predicate, and all other components
have obvious default values.

definition fg-init-config where
  fg-init-config A (config::('pst,'proc,'msg) config) (coord::'proc coord) \equiv
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** \( \text{fg-send-msg} \) where

\[
\text{fg-send-msg} \ A \ p \ q \ \text{config config'} \equiv \\
q \not\in (\text{sent config } p) \\
\land \text{config'} = \text{config} \\
\land \text{sent} := (\text{sent config})(p := (\text{sent config } p) \cup \{q\}), \\
\land \text{network} := \text{network config} \cup \\
\{((p, \text{round config } p, q, \\
\text{sendMsg } A (\text{round config } p) p q (\text{state config } p))) \}
\]

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** \( \text{fg-rcv-msg} \) where

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

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 $C_{\text{nextState}}$ 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**

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

- $sent config p = UNIV$
- $\land \text{dom}(rcvd config p) = HO$
- $\land (\exists s. C_{\text{nextState}} A (\text{round config } p) p (\text{state config } p) (rcvd config p) crd s$
  - $\land config' = config []$
  - $\land \text{round} := (\text{round config})(p := \text{Suc} (\text{round config } p)),$
  - $\land \text{state} := (\text{state config})(p := s),$
  - $\land \text{sent} := (\text{sent config})(p := \{\}),$
  - $\land \text{rcvd} := (\text{rcvd config})(p := \lambda q. \text{None} [])$}

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**

$$fg-run A \rho HOs SHOs coords \equiv$$

- $fg-init-config A (\rho 0) (coords 0)$
- $\land (\forall i. \exists p. fg-next-config A p$
  - $\land HOs (\text{round } (\rho i) p) p)$
  - $\land SHOs (\text{round } (\rho i) p) p)$
  - $\land coords (\text{round } (\rho (\text{Suc } i)) p) p)$
  - $\land (\rho i) (\rho (\text{Suc } i))$)
- $\land (\forall p r. \exists n. \text{round } (\rho n) p = r)$

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**

$$fg-start-round \rho p r \equiv \text{LEAST } (n::\text{nat}). \text{round } (\rho n) p = r$$
3.2 Properties of the Fine-Grained Semantics

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

Process states change only when round numbers change during a fine-grained run.

**lemma fg-state-change:**

**assumes** rho: fg-run A rho HOs SHOs coords 
and rd: round (rho (Suc n)) p = round (rho n) p

**shows** state (rho (Suc n)) p = state (rho n) p

**proof**

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

by (auto simp: fg-run-def)

with rd show ?thesis
by (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)

qed

Round numbers never decrease.

**lemma fg-round-numbers-increase:**

**assumes** rho: fg-run A rho HOs SHOs coords and n: n \( \leq \) m

**shows** round (rho n) p \( \leq \) round (rho m) p

**proof**

from n obtain k where k: m = n+k by (auto simp: le-iff-add)

{ 
fix i have round (rho n) p \( \leq \) round (rho (n+i)) p (is ?P i)
proof (induct i)
show ?P 0 by simp
next
fix j
assume ih: ?P j
from rho have \( \exists p'. fg-next-config A p' (HOs (round (rho (n+j)) p') p') \)
(SHOs (round (rho (n+j)) p') p')
(coords (round (rho (Suc (n+j))) p') p')
(rho (n+j)) (rho (Suc (n+j)))

by (auto simp: fg-run-def)
hence round (rho (n+j)) p \( \leq \) round (rho (n + Suc j)) p
by (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)
with ih show ?P (Suc j) by auto

qed
}

with k show ?thesis by simp

qed

Combining the two preceding lemmas, it follows that the local states of
process $p$ at two configurations are the same if these configurations have the same round number.

**lemma** \( \text{fg\ same\ round\ same\ state} \):

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

**and** \( \text{rd} : \text{round}\ (\rho m) p = \text{round}\ (\rho n) p \)

**shows** \( \text{state}\ (\rho m) p = \text{state}\ (\rho n) p \)

**proof**

\[
\{ \\
  \text{fix}\ k\ i \\
  \text{have}\ \text{round}\ (\rho (k+i)) p = \text{round}\ (\rho k) p \\
  \implies \text{state}\ (\rho (k+i)) p = \text{state}\ (\rho k) p \\
  (\text{is } \text{?R i } \implies \text{?S i}) \\
  \text{proof } (\text{induct i}) \\
  \text{show } \text{?S } 0 \text{ by simp} \\
\}
\]

**next**

\[
\text{fix}\ j \\
\text{assume ih: } \text{?R } j \implies \text{?S } j \\
\text{and } r : \text{round}\ (\rho (k+Suc\ j)) p = \text{round}\ (\rho k) p \\
\text{from } \rho \text{ have 1: } \text{round}\ (\rho k) p \leq \text{round}\ (\rho (k+j)) p \\
\text{by } (\text{auto elim: fg-round-numbers-increase}) \\
\text{from } \rho \text{ have 2: } \text{round}\ (\rho (k+j)) p \leq \text{round}\ (\rho (k + Suc\ j)) p \\
\text{by } (\text{auto elim: fg-round-numbers-increase}) \\
\text{from 1 2 r have 3: } \text{round}\ (\rho (k+j)) p = \text{round}\ (\rho k) p \text{ by auto} \\
\text{with } r \text{ have round}\ (\rho (Suc\ (k+j))) p = \text{round}\ (\rho (k+j)) p \text{ by simp} \\
\text{with } \rho \text{ have state}\ (\rho (Suc\ (k+j))) p = \text{state}\ (\rho (k+j)) p \\
\text{by } (\text{auto elim: fg-state-change}) \\
\text{with 3 ih show } \text{?S } (Suc\ j) \text{ by simp} \\
\}
\]

**qed**

**note** \( \text{aux} = \text{this} \)

**show** \( \text{?thesis} \)

**proof** (cases \( n \leq m \))

**case** \( \text{True} \)

then obtain \( k \) where \( m = n+k \) by (auto simp: le-iff-add)

with \( r \) show \( \text{?thesis} \) by (auto simp: aux)

**next**

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

hence \( m \leq n \) by simp

then obtain \( k \) where \( n = m+k \) by (auto simp: le-iff-add)

with \( r \) show \( \text{?thesis} \) by (auto simp: aux)

**qed**

**qed**

Since every process executes every round, function \( \text{fg-startRound} \) is well-defined. We also list a few facts about \( \text{fg-startRound} \) that will be used to show that it is a “stuttering sampling function”, a notion introduced in the theories about stuttering equivalence.

**lemma** \( \text{fg-start-round} \):
assumes \( fg\text{-run} A \rho \text{ HOs SHOs coords} \)
shows \( \text{round} (\rho \ (fg\text{-start-round} \rho \ p \ r)) \ p = r \)
using assms by (auto simp: fg-run-def fg-start-round-def intro: LeastI-ex)

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

lemma \( fg\text{-start-round-later} \):
assumes \( \rho: \text{fg\text{-run} A \rho \text{ HOs SHOs coords} } \)
\( \text{and} \ r: \text{round} (\rho \ n) \ p = r \ \text{and} \ r' < r' \)
shows \( n < fg\text{-start-round} \rho \ p \ r' \ (\text{is} < ?\text{start}) \)
proof (rule ccontr)
assume \( \neg \ ?\text{thesis} \)
hence \( \text{start} \ < \ ?\text{start} \leq n \) by simp
from \( \rho\) this have \( \text{round} (\rho \ ?\text{start}) \ p \leq \text{round} (\rho \ n) \ p \)
by (rule fg-round-numbers-increase)
with \( r\) have \( r' \leq r \) by (simp add: fg-start-round[OF \( \rho\)])
with \( r'\) show False by simp
qed

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

lemma \( fg\text{-start-round-strict-mono} \):
assumes \( \rho: \text{fg\text{-run} A \rho \text{ HOs SHOs coords} } \)
shows \( \text{strict-mono} (fg\text{-start-round} \rho \ p) \)
proof
fix \( r \ r' \)
assume \( r: (r::\text{nat}) < r' \)
from \( \rho\) have \( \text{round} (\rho \ (fg\text{-start-round} \rho \ p \ r)) \ p = r \) by (rule fg-start-round)
from \( \rho\) this \( r\) show \( fg\text{-start-round} \rho \ p \ r < fg\text{-start-round} \rho \ p \ r' \)
by (rule fg-start-round-later)
qed

Process \( p \) is at round \( r \) at all configurations between the start of round \( r \) and the start of round \( r+1 \). By lemma \( fg\text{-same-round-same-state} \), this implies that the local state of process \( p \) is the same at all these configurations.

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

19
For any process $p$ and round $r$ there is some instant $n$ where $p$ executes a local transition from round $r$. In fact, $n+1$ marks the start of round $r+1$.

**Lemma fg-local-transition-from-round:**

assumes $\rho$: fg-run $A$ $\rho$ HOs SHOs coords

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

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 $\text{?start} \neq 0$)

proof

assume $\text{contr}: \text{?start} = 0$

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

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

qed

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

with $\text{fg-start-round}[\text{OF} \; \rho_0, \; \text{of} \; p \; \text{Suc} \; r]$

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

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

proof (rule fg-round-between-start-rounds[OF $\rho_0$])

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

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

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

next

from $n$ show $n < \text{?start} \; \text{by} \; \text{simp}$

qed

from $\rho$ obtain $p'$ where

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

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

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

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

by (force simp: fg-run-def)

hence fg-local A p (HOs r p) (coords (Suc r) p) (rho n) (rho (Suc n))

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

fix q

assume fg-send-msg A p' q ?cfg ?cfg'

— impossible because round changes

with 0 1 show ?thesis by (auto simp: fg-send-msg-def)

next

fix q


— impossible because round changes

with 0 1 show ?thesis by (auto simp: fg-rcv-msg-def)

next


with 0 1 show ?thesis by (cases p' = p) (auto simp: fg-local-def)

qed

with 1 n that show ?thesis by auto

qed

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

lemma fg-invariant1:

assumes rho: fg-run A rho HOs SHOs coords

and m: (p,r,q,m) \in network (rho n) (is \ ?msg n)

shows m = sendMsg A r p q (state (rho (fg-start-round rho p r)) p)

using m proof (induct n)

— the base case is trivial because the network is empty

assume \ ?msg 0 with rho show ?thesis

by (auto simp: fg-run-def fg-init-config-def)

next

fix n

assume m': \ ?msg (Suc n) and ih: \ ?msg n \imp \ ?thesis

from rho obtain p' where

fg-next-config A p' (HOs (round (rho n) p') p')

(SHOs (round (rho n) p') p')

(coords (round (rho (Suc n)) p') p')

(rho n) (rho (Suc n))

(is fg-next-config - - ?HO ?SHO ?crd ?cfg ?cfg')

by (force simp: fg-run-def)

thus ?thesis

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

Only fg-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'


show ?thesis
proof (cases ?msg n)
  case True
  with ih show ?thesis .
next
  case False
  with send m' have 1: p' = p round ?cfg p = r
      and 2: m = sendMsg A 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'
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) q
      (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 \texttt{fg-rcv-msg} steps of process \( q \), the proof is immediately reduced to the induction hypothesis.

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

next

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

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

23
(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 fg-rcv-msg steps of process q, the proof is immediately reduced to the
induction hypothesis.

fix q' 
hence rd: ?rd (Suc n) = ?rd n by (auto simp: fg-rcv-msg-def)
show ?P (Suc n)
proof (cases ?rcvd n)
case True
with ih p rd show ?thesis by simp
next
case False
from rcvmsg obtain m' m'' where
(q', round ?cfg p', p', m') ∈ network ?cfg
rcvd ?cfg' = (rcvd ?cfg)(p' := (rcvd ?cfg p')(q' :=
if q' ∈ ?SHO then Some m' else Some m'))
by (auto simp: fg-rcv-msg-def split del: if-split-asm)
with False rcvd p rd have (p, ?rd n, q, m) ∈ network ?cfg by auto
with rho rd show ?thesis by (auto simp: fg-invariant1)
qed
next
fix q' 
assume fg-send-msg A p' q' ?cfg ?cfg'
with rcvd have ?rcvd n and ?rd (Suc n) = ?rd n
by (auto simp: fg-send-msg-def)
with p ih show ?P (Suc n) by simp
next
with rcvd have ?rcvd n and ?rd (Suc n) = ?rd n
— in fact, p' = q is impossible because the rcvd field of p' is cleared
by (auto simp: fg-local-def)
with p ih show ?P (Suc n) by simp
qed
qed

3.3 From Fine-Grained to Coarse-Grained Runs

The reduction theorem asserts that for any fine-grained run rho there is a
coarse-grained run such that every process sees the same sequence of local
states in the two runs, modulo stuttering. In other words, no process can
locally distinguish the two runs.

Given fine-grained run rho, the corresponding coarse-grained run sigma is
defined as the sequence of state vectors at the beginning of every round. Notice in particular that the local states \( \sigma_r p \) and \( \sigma_r q \) of two different processes \( p \) and \( q \) appear at different instants in the original run \( \rho \). Nevertheless, we prove that \( \sigma \) is a coarse-grained run of the algorithm for the same HO, SHO, and coordinator assignments. By definition (and the fact that local states remain equal between \( \text{fg-start-round} \) instants), the sequences of process states in \( \rho \) and \( \sigma \) are easily seen to be stuttering equivalent, and this will be formally stated below.

**definition coarse-run where**

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

**theorem reduction:**

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

**shows** \( \text{CSHORun } A \left( \text{coarse-run } \rho \right) \text{ HOs SHOs coords} \)

(is \( \text{CSHORun - ?cr - - -} \))

**proof** (auto simp: CSHORun-def)

**from** \( \rho \) **show** \( \text{CHOinitConfig } A \left( \text{Suc r} \right) \text{ (HOs Suc r) (SHOs Suc r) (coords Suc r)} \)

(\( \text{Suc r} \))

**proof** (auto simp add: CSHOnextConfig-def)

**fix** \( p \)

**from** \( n \text{[THEN fg-local-transition-from-round]} \) **obtain** \( n \)

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

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

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

(is \( \text{fg-local - - ?HO ?crd ?cfg ?cfg} \'))

by blast

have \( \text{cfg: Suc r p = state ?cfg p} \)

unfolding coarse-run-def proof (rule \( \text{fg-same-round-same-state} \))

**from** \( n \) **show** \( \text{round } (\rho (\text{fg-start-round } \rho p r)) p = \text{round } ?cfg p \)

by (simp add: \( \text{fg-start-round} \))

qed

**from** \( \text{start have } \text{cfg': Suc r p = state ?cfg' p} \)

by (simp add: coarse-run-def)

**have** \( \text{rcvd: rcvd ?cfg p } = \text{SHOmsgVectors } A p (\text{Suc r}) \ \text{SHOs Suc r p} \)

**proof** (auto simp: \( \text{SHOmsgVectors-def} \))

**fix** \( q \)

**assume** \( q } = \text{SHO} \)

with \( n \) **show** \( \exists m. rcvd ?cfg p q = \text{Some m} \) by (auto simp: \( \text{fg-local-def} \))

**next**

**fix** \( q \ m \)

**assume** \( \text{rcvd ?cfg p q } = \text{Some m} \)

with \( \rho n \) **show** \( q } = \text{SHO} \) by (auto simp: \( \text{fg-invariant2a} \))

**next**
fix $q$

assume sho: $q \in \text{SHOs } r p$ and ho: $q \in \text{?HO}$

from ho n loc obtain $m$ where rcvd ?cfg $p q = \text{Some } m$
  by (auto simp: fg-local-def)

with rho n sho show rcvd ?cfg $p q = \text{Some } (\text{sendMsg } A r p (?cr r q))$
  by (auto simp: fg-invariant2b coarse-run-def)

qed

with rho n sho show rcvd ?cfg $p q = \text{Some } (\text{sendMsg } A r q p (?cr r q))$
  by (auto simp: fg-invariant2b coarse-run-def)

qed

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.

\textbf{definition} locally-similar \textbf{where}

\textit{locally-similar} ($\sigma::\text{nat} \Rightarrow '\text{proc} \Rightarrow '\text{pst}$) $\tau \equiv$
\hspace{1cm} $\forall p::'\text{proc}. (\lambda n. \sigma n p) \approx (\lambda n. \tau n p)$

\textbf{definition} local-property \textbf{where}

\textit{local-property} $P \equiv$
\hspace{1cm} $\forall \sigma \tau. \text{locally-similar } \sigma \tau \rightarrow P \sigma \rightarrow P \tau$

Local similarity is an equivalence relation.

\textbf{lemma} locally-similar-refl: locally-similar $\sigma \sigma$
  by (simp add: locally-similar-def stutter-equiv-refl)

\textbf{lemma} locally-similar-sym: locally-similar $\sigma \tau \Rightarrow$ locally-similar $\tau \sigma$
  by (simp add: locally-similar-def stutter-equiv-sym)

\textbf{lemma} locally-similar-trans [trans]:
  locally-similar $\sigma \sigma \Rightarrow$ locally-similar $\sigma \tau \Rightarrow$ locally-similar $\sigma \sigma$
  by (force simp add: locally-similar-def elim: stutter-equiv-trans)

\textbf{lemma} local-property-eq:
  local-property $P = (\forall \sigma \tau. \text{locally-similar } \sigma \tau \rightarrow P \sigma = P \tau)$
  by (auto simp: local-property-def dest: locally-similar-sym)

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

**lemma** coarse-run-locally-similar:

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

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

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

fix \( p \)

show \((\lambda n. \text{state } (\rho n) \ p) \approx (\lambda n. \text{coarse-run } \rho n \ p)\) (is \( \approx \)fg ≈ \approx cgr)

**proof** (rule stutter-equivI)

show stutter-sampler \((\text{fg-start-round } \rho \ p) \approx \text{fg}\)

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

from \( \rho \) show \( \text{fg-start-round } \rho \ p \ 0 = 0 \)

by (rule fg-start-round-0)

next

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

by (rule fg-start-round-strict-mono | OF \( \rho \))

next

fix \( r \ n \)

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

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

by (simp add: \( \text{fg-round-between-start-rounds} \))

with \( \rho \) show \( \text{state } (\rho n) \ p = \text{state } (\rho (\text{fg-start-round } \rho \ p \ r)) \ p \)

by (rule fg-same-round-same-state)

qed

next

show stutter-sampler id \( ?cgr \)

by (rule id-stutter-sampler)

next

show \( \approx \text{fg} \circ \text{fg-start-round } \rho \ p = \approx \text{cgr} \circ \text{id} \)

by (auto simp: coarse-run-def)

qed

qed

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

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

and coarse-correct:

\[ \forall crho. \ [ \text{CSHORun } A \ crho \ \text{HOs SHOs coords; crho } 0 = \text{state } (\rho 0) ] \Rightarrow P \ crho \]

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

**proof** –

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

by (rule ext, simp add: coarse-run-def \( \text{fg-start-round-0} [\text{OF } \rho] \))
from \( \rho \) \[ \text{THEN reduction} \] this
have \( P \) (coarse-run \( \rho \)) by (rule coarse-correct)
with coarse-run-locally-similar[OF \( \rho \)] \( P \)
show \( \text{thesis} \) 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 \textit{local-property-reduction}.

lemma \textit{integrity-is-local}:
assumes \( \text{sim}: \text{locally-similar} \ \sigma \ \tau \)
and \( \text{val}: \bigwedge n. \text{dec} (\sigma \ n \ p) = \text{Some} \ v \implies v \in \text{range vals} \)
and \( \text{dec}: \text{dec} (\tau \ n \ p) = \text{Some} \ v \)
shows \( v \in \text{range vals} \)
proof
  from \( \text{sim} \) have \( (\lambda r. \sigma \ r \ p) \approx (\lambda r. \tau \ r \ p) \) by (simp add: locally-similar-def)
  then obtain \( m \) where \( \sigma \ m \ p = \tau \ n \ p \) by (rule stutter-equiv-element-left)
  from \( \text{sym}[OF \ this] \) \( \text{dec} \) show \( \text{thesis} \) by (auto elim: \text{val})
qed

lemma \textit{validity-is-local}:
assumes \( \text{sim}: \text{locally-similar} \ \sigma \ \tau \)
and \( \text{val}: \bigwedge n. \text{dec} (\sigma \ n \ p) = \text{Some} \ w \implies w = v \)
and \( \text{dec}: \text{dec} (\tau \ n \ p) = \text{Some} \ w \)
shows \( w = v \)
proof
  from \( \text{sim} \) have \( (\lambda r. \sigma \ r \ p) \approx (\lambda r. \tau \ r \ p) \) by (simp add: locally-similar-def)
  then obtain \( m' \) where \( \sigma \ m' \ p = \tau \ n \ p \) by (rule stutter-equiv-element-left)
  from \( \text{sym}[OF \ this] \) \( \text{dec} \) show \( \text{thesis} \) by (auto elim: \text{val})
qed

lemma \textit{agreement-is-local}:
assumes \( \text{sim}: \text{locally-similar} \ \sigma \ \tau \)
and \( \text{agr}: \bigwedge m \ n. [\text{dec} (\sigma \ m \ p) = \text{Some} \ v; \text{dec} (\sigma \ n \ q) = \text{Some} \ w] \implies v = w \)
and \( v: \text{dec} (\tau \ m \ p) = \text{Some} \ v \) and \( w: \text{dec} (\tau \ n \ q) = \text{Some} \ w \)
shows \( v = w \)
proof
  from \( \text{sim} \) have \( (\lambda r. \sigma \ r \ p) \approx (\lambda r. \tau \ r \ p) \) by (simp add: locally-similar-def)
  then obtain \( m' \) where \( \sigma \ m' \ p = \tau \ m \ p \) by (rule stutter-equiv-element-left)
  from \( \text{sim} \) have \( (\lambda r. \sigma \ r \ q) \approx (\lambda r. \tau \ r \ q) \) by (simp add: locally-similar-def)
  then obtain \( n' \) where \( \sigma \ n' \ q = \tau \ n \ q \) by (rule stutter-equiv-element-left)
  from \( \text{sym}[OF \ m'] \) \( \text{sym}[OF \ n'] \) \( v \ w \) show \( v = w \) by (auto elim: \text{agr})
qed

lemma \textit{termination-is-local}:
assumes \( \text{sim}: \text{locally-similar} \ \sigma \ \tau \)
and \( \text{trm}: \text{dec} (\sigma \ m \ p) = \text{Some} \ v \)
shows \( \exists n. \text{dec} (\tau \ n \ p) = \text{Some} \ v \)
proof
tfrom 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 \(\text{trm}\) show \(\text{thesis}\) 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 \(\forall n p v. \text{dec} (\sigma n p) = \text{Some} v \rightarrow v \in \text{range vals}\)
  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 \(\forall m n p q v w. \text{dec} (\sigma m p) = \text{Some} v \land \text{dec} (\sigma n q) = \text{Some} w \rightarrow v = w\)
  and \(\text{dec} (\tau m p) = \text{Some} v\) 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 \(\forall n p v w. \text{dec} (\sigma n p) = \text{Some} v \rightarrow w \in \text{range vals}\)
  and \(\text{dec} (\tau n p) = \text{Some} v\)
  thus \(w \in \text{range vals}\) by (blast intro: validity-is-local)
next
  fix \(\sigma \tau m n p q v w\)
  assume locally-similar \(\sigma \tau\)
  and \(\forall m n p q v w. \text{dec} (\sigma m p) = \text{Some} v \land \text{dec} (\sigma n q) = \text{Some} w \rightarrow v = w\)
  and \(\text{dec} (\tau m p) = \text{Some} v\) 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

end

theory Majorities
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* $\text{card } E < \text{card } S + \text{card } T$ and $S \subseteq E$ and $T \subseteq E$ and $E$ finite

*Shows* $S \cap T \neq \emptyset$

*Proof* (clarify)

Assume contra: $S \cap T = \emptyset$

From $s$ $t$ $e$ have finite $S$ and finite $T$ by (auto simp: finite-subset)

With $\text{card } S \cap T = \emptyset$ have $\text{card } (S \cup T) \leq \text{card } E$ by (simp add: card-mono)

Ultimately

Show $\text{False}$ by simp

**Lemma abs-majoritiesE:**

*Assumes* $\text{card } E < \text{card } S + \text{card } T$ and $S \subseteq E$ and $T \subseteq E$ and $E$ finite

*Omits* $\text{card } S \cap T = \emptyset$

*Proof* (clarify)

Assume contra: $S \cap T = \emptyset$

From $s$ $t$ $e$ have finite $S$ and finite $T$ by (auto simp: finite-subset)

With $\text{card } S \cap T = \emptyset$ have $\text{card } (S \cup T) \leq \text{card } E$ by (simp add: card-mono)

Ultimately

Show $\text{False}$ by simp

We restate the above theorems for the case where the base type is finite (taking $E$ as the universal set).

**Lemma abs-majoritiesE’:**

*Assumes* $\text{card } S > (\text{card } E) \div 2$ and $\text{card } T > (\text{card } E) \div 2$

*Omits* $S \subseteq E$ and $T \subseteq E$ and $E$ finite

*Proof* (clarify)

Assume contra: $S \cap T = \emptyset$

From $s$ $t$ $e$ have finite $S$ and finite $T$ by (auto simp: finite-subset)

With $\text{card } S \cap T = \emptyset$ have $\text{card } (S \cup T) \leq \text{card } E$ by (simp add: card-mono)

Ultimately

Show $\text{False}$ by simp
5 Verification of the One-Third Rule Consensus Algorithm

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

5.1 Model of the Algorithm

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

typedecl Proc — the set of processes

axiomatization where Proc-finite: OFCLASS(Proc, finite-class)

instance Proc :: finite by (rule Proc-finite)

abbreviation

N ≡ card (UNIV::Proc set)

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

The initial value of field x is unconstrained, but no decision has been taken initially.

definition 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 HO: HOs m p ≠ {} 
    and v: MFR (HOrcvdMsgs OTR-M m p (HOs m p) (rho m)) v 
    (is MFR ?msgs v)
  shows 3 q ∈ HOs m p. v = the (?msgs q)
proof –

from HO obtain q where q: q ∈ HOs m p
  by auto
with v have HOV ?msgs (the (?msgs q)) ≠ {}
  by (auto simp: HO-def HOrcvdMsgs-def)
  hence HOp: 0 < card (HOV ?msgs (the (?msgs q)))
  by auto
also from v have ... ≤ card (HOV ?msgs v)
  by (simp add: MFR-def)
finally have HOV ?msgs v ≠ {}
  by auto
thus ?thesis
  by (auto simp: HO-def HOrcvdMsgs-def)
qed

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

**definition TwoThirds where**
TwoThirds msgs v ≡ (2∗N) div 3 < card (HOV msgs v)

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

**definition OTR-nextState where**

OTR-nextState r p (st::(val::linorder) pstate) msgs st' ≡
if (2∗N) div 3 < card {q. msgs q ≠ None}
then st' = (| x = Min {v . MFR msgs v},
  decide = (if (∃ v. TwoThirds msgs v)
    then Some (ε v. TwoThirds msgs v)
    else decide st) |)
else st' = st

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

**definition OTR-sendMsg where**

OTR-sendMsg r p q st ≡ 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 Π of processes where Π
contains more than two thirds of all processes. The “per-round” part of the communication predicate is trivial.

**definition** OTR-commPerRd **where**

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

**definition** OTR-commGlobal **where**

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

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

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

**definition** OTR-HOMachine **where**

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

**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 *decide* fields of the process states are limited to the *x* values present in the initial states since the algorithm does not introduce any new values.

**definition** VInv **where**

\[
VInv \rho \ n \equiv \\
\text{let } xinit = (\text{range } (x \circ \rho 0)) \text{ in } \text{range } (x \circ \rho n) \subseteq xinit \\
\land \text{range } (\text{decide} \circ \rho n) \subseteq \{\text{None}\} \cup (\text{Some } \ xinit)
\]

34
lemma vinv-invariant:
  assumes run: HORun OTR-M rho HOs
  shows VInv rho n
proof (induct n)
  from run show VInv rho 0
    by (simp add: HORun-eq HOinitConfig-eq OTR-HOMachine-def initState-def
             OTR-initState-def VInv-def image-def)
next
  fix m
  assume ih: VInv rho m
  let ?xinit = range (x ◦ (rho 0))
  have range (x ◦ (rho (Suc m))) ⊆ ?xinit
    proof (clarsimp cong del: image-cong-simp)
      fix p from run have nxt: OTR-nextState m p (rho m p)
        (HOrcvdMsgs OTR-M m p (HOs m p) (rho m))
        (rho (Suc m) p)
        (is OTR-nextState - - ?st ?msgs ?st')
      by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
      show x ?st' ∈ ?xinit
        proof (cases (2*N) div 3 < card (HOs m p))
          case True
          hence HO: HOs m p ≠ {} by auto
          let ?MFRs = {v. MFR ?msgs v}
          have Min ?MFRs ∈ ?MFRs
            proof (rule Min-in)
              from HO have ?MFRs ⊆ (the ◦ ?msgs)’(HOs m p)
              by (auto simp: image-def intro: MFR-in-msgs)
              thus finite ?MFRs by (auto elim: finite-subset)
            next
              from MFR-exists show ?MFRs ≠ {} by auto
            qed
          with HO have ∃ q ∈ HOs m p. Min ?MFRs = the (?msgs q)
            by (intro MFR-in-msgs auto)
          hence ∃ q ∈ HOs m p. Min ?MFRs = x (rho m q)
            by (auto simp: HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)
          moreover
            from True nxt have x ?st' = Min ?MFRs
              by (simp add: OTR-nextState-def HOrcvdMsgs-def)
          ultimately
            show ?thesis using ih by (auto simp: VInv-def image-def)
          next
            case False
            with nxt ih show ?thesis
              by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def Let-def)
          qed
        qed
  qed
moreover
have \( \forall p. \text{decide } ((\rho (\text{Suc } m)) p) \in \{\text{None}\} \cup (\text{Some } ?xinit) \)
proof
  fix \( p \)
  from \( \text{run} \)
  have \( \text{nxt: OTR-nextState } m \ p \ (\rho \ m \ p) \)
    \( (\text{HOrcvdMsgs } OTR-M \ m \ p \ (\text{HOs } m \ p) \ (\rho \ m)) \)
    \( (\rho \ (\text{Suc } m) \ p) \)
    \( (\text{is OTR-nextState } - \ - ?st \ ?msgs \ ?st') \)
    by (simp add: \( \text{HORun-eq} \ \text{HOnextConfig-eq} \ \text{OTR-HOMachine-def} \ \text{nextState-def} \))
  show \( \text{decide } ?st' \in \{\text{None}\} \cup (\text{Some } ?xinit) \)
  proof (cases \( (2 \times N) \div 3 < \text{card } \{q. \ ?msgs \ q \neq \text{None}\} \))
    assume \( \text{HO: } (2 \times N) \div 3 < \text{card } \{q. \ ?msgs \ q \neq \text{None}\} \)
    show \( ?\text{thesis} \)
      proof (cases \( \exists v. \ \text{TwoThirds } ?msgs \ v \))
        case \( \text{True} \)
        let \( ?\text{dec} = \epsilon \ v. \ \text{TwoThirds } ?msgs \ v \)
        from \( \text{True} \) have \( \text{TwoThirds } ?msgs \ ?\text{dec} \) by (rule \( \text{someI-ex} \))
        hence \( \text{HOV } ?msgs \ ?\text{dec} \neq \{\} \) by (auto simp add: \( \text{TwoThirds-def} \))
        then obtain \( q \) where \( x(\rho \ m \ q) = ?\text{dec} \)
          by (auto simp: \( \text{HOV-def} \ \text{HOrcvdMsgs-def} \ \text{OTR-HOMachine-def} \ \text{OTR-sendMsg-def} \))
        from \( \text{sym[OF this]} \) \( \text{nxt} \ \text{ih} \) show \( ?\text{thesis} \)
          by (auto simp: \( \text{OTR-nextState-def} \ \text{VInv-def} \ \text{image-def} \))
      next
        case \( \text{False} \)
        with \( \text{HO} \ \text{nxt} \ \text{ih} \) show \( ?\text{thesis} \)
          by (auto simp: \( \text{OTR-nextState-def} \ \text{VInv-def} \ \text{HOrcvdMsgs-def} \ \text{image-def} \))
      qed
    qed
  next
    case \( \text{False} \)
    with \( \text{nxt} \ \text{ih} \) show \( ?\text{thesis} \)
      by (auto simp: \( \text{OTR-nextState-def} \ \text{VInv-def} \ \text{image-def} \))
    qed
  qed
  hence \( \text{range } (\text{decide o } (\rho (\text{Suc } m))) \subseteq \{\text{None}\} \cup (\text{Some } ?xinit) \) by auto
ultimately
  show \( \text{VInv } \rho \ (\text{Suc } m) \) by (auto simp: \( \text{VInv-def} \ \text{image-def} \))
qed

Integrity is an immediate consequence.

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

36
hence decide \((\rho n, p) \in \{\text{None}\} \cup (\text{Some } \cdot \ ?\text{init})\)
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 \(\text{HORun OTR-M } \rho \text{ HOs}\)
- and \(\neg ((2* N) \div 3 < \text{card\{} {q. (HOrcvdMsgs OTR-M n p (HOs n p) (\rho n)) q \neq \text{None}}\})\)
- shows \(\rho (\text{Suc n}) p = \rho n p\)
- using \text{assms}
- by (auto simp add: \text{HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def OTR-nextState-def})

**Lemma** nextState-decide:
- assumes \(\text{run: HORun OTR-M } \rho \text{ HOs}\)
- and \(\text{chg: decide } (\rho (\text{Suc n}) p) \neq \text{decide } (\rho n p)\)
- shows \(\text{TwoThirds } (\text{HOrcvdMsgs OTR-M n p (HOs n p) (\rho n)) (the (decide (\rho (\text{Suc n}) p))})\)

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

**Lemma** \(A1:\)
- assumes \(\text{run: HORun OTR-M } \rho \text{ HOs}\)
- 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 \((2* N) \div 3 < \text{card } \{ q . x (\rho n q) = v \}\)

proof –
- from \(\text{run chg}\)
  have \(\text{TwoThirds } (\text{HOrcvdMsgs OTR-M n p (HOs n p) (\rho n)}))\)
  (the (decide ?st'))
  (is \(\text{TwoThirds } \text{msgs -}\))
  by (rule nextState-decide)
- with \(\text{dec}\) have \(\text{TwoThirds } \text{msgs v by simp}\)
- hence \((2* N) \div 3 < \text{card } \{ q . \text{msgs q = Some } v \}\)
  by (simp add: TwoThirds-def HOV-def)

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

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

\[\text{lemma} \ A2: \]
\[\text{assumes run:}\ Horun\ OTR-M\ \rho\ HOs\]
\[\text{and}\ \text{HO}: (2\ast N)\ \text{div} 3 <\text{card}\ \{q . \text{HOrcvdMsgs} OTR-M n p (HOs n p) (\rho n) q \neq \text{None} \}\]
\[\text{and}\ \text{maj}: (2\ast N)\ \text{div} 3 <\text{card}\ \{q . x (\rho n q) = v \}\]
\[\text{shows} x (\rho (Suc n) p) = v\]
\[\text{proof} –\]
\[\text{from}\ run\]
\[\text{have}\ \text{nxt:}\ OTR-nextState\ n\ p\ (\rho n p)\]
\[\text{HOrcvdMsgs\ OTR-M\ n\ p\ (HOs\ n\ p)\ (\rho n))}\]
\[\text{HO\ (Suc\ n)\ p}\]
\[\text{(is}\ OTR-nextState\ - - ?st \?msgs \?st')}\]
\[\text{by}\ (\text{simp add:}\ Horun-eq\ HOnextConfig-eq\ OTR-HOMachine-def\ nextState-def)\]
\[\text{let} \ ?HOVothers = \bigcup \{ \text{HOV} \ ?msgs w | w . w \neq v \}\]
\[\text{— processes from which}\ p\ \text{received values different from} v\]
\[\text{have}\ w:\ \text{card} \ ?HOVothers \leq N\ \text{div} 3\]
\[\text{proof} –\]
\[\text{have}\ \text{card} \ ?HOVothers \leq \text{card}\ (\text{UNIV} - \{ q . x (\rho n q) = v \})\]
\[\text{by}\ (\text{auto simp:}\ \text{HOV-def}\ \text{HOrcvdMsgs-def}\ \text{OTR-HOMachine-def}\ \text{OTR-sendMsg-def})\]
\[\text{intro:}\ \text{card-mono})\]
\[\text{also have} \ldots = N - \text{card} \{ q . x (\rho n q) = v \}\]
\[\text{by}\ (\text{auto simp:}\ \text{card-Diff-subset})\]
\[\text{also from}\ \text{maj}\ \text{have} \ldots \leq N\ \text{div} 3\ \text{by}\ \text{auto}\]
\[\text{finally show} \ ?thesis .\]
\[\text{qed}\]

\[\text{have}\ \text{hov:}\ \text{HOV} \ ?msgs v = \{ q . ?msgs q \neq \text{None} \} - \?HOVothers\]
\[\text{by}\ (\text{auto simp:}\ \text{HOV-def})\ \text{blast}\]

\[\text{have}\ \text{othHO}: \ ?HOVothers \subseteq \{ q . ?msgs q \neq \text{None} \}\]
\[\text{by}\ (\text{auto simp:}\ \text{HOV-def})\]

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

\[\text{from}\ \text{HO}\ \text{have} N\ \text{div} 3 < \text{card} \{ q . ?msgs q \neq \text{None} \} - (N\ \text{div} 3)\]
\[\text{by}\ \text{auto}\]
\[\text{also from}\ w\ \text{HO}\ \text{have} \ldots \leq \text{card} \{ q . ?msgs q \neq \text{None} \} - \text{card} \ ?HOVothers\]

38
by auto
also from hov othHO have \ldots = card (HOV ?msgs v)
by (auto simp: card-Diff-subset)
finally have HOV: \(N \div 3 < card (HOV ?msgs v)\).

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

\[
\begin{align*}
\text{have } & \forall w. w \neq v \longrightarrow card (HOV ?msgs w) \leq card \ ?HOV\text{others} \\
\text{by } & (\text{force intro: card-mono}) \\
\text{with } & w \text{ have cardw: } \forall w. w \neq v \longrightarrow card (HOV ?msgs w) \leq N \div 3 \text{ by auto}
\end{align*}
\]

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

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

qed

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

\[
\begin{align*}
\text{have } & \text{card } \{ q . \ ?msgs q = Some v \} \leq \text{card } \{ q . \ ?msgs q \neq None \} \\
\text{by } & (\text{auto intro: card-mono}) \\
\text{with } & HO mfrv nxt show \ ?\text{thesis by (auto simp: OTR-nextState-def)}
\end{align*}
\]

qed

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

lemma A3:
assumes run:HORun OTR-M rho HOs
and n: \((2*N) \div 3 < card \{ q . x (rho n q) = v \} \) (is \twothird n)
shows \twothird (n+k)
proof (induct k)
from n show \twothird (n+0) by simp
next
fix m
assume m: \twothird (n+m)
have \(\forall q. x (\rho (n+m) q) = v \longrightarrow x (\rho (n + Suc m) q) = v\)
proof (rule+)
fix q
assume q: x ((\rho (n+m)) q) = v
let \(\text{msgs} = \text{HORcvdMsgs OTR-M (n+m) q (HOs (n+m) q) (rho (n+m))}\)
show x (\rho (n + Suc m) q) = v
proof (cases (2*N) \div 3 < card \{ q . \ ?msgs q \neq None \})
case True

39
from m have \((2\times N) \div 3 < \text{card}\{q \cdot x (\rho (n+m) q) = v \}\) 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 \(\text{card}\{q \cdot x (\rho (n+m) q) = v\} \leq \text{card}\{q \cdot x (\rho (n + \text{Suc} m) q) = v\}\)

by (auto intro: card-mono)

with m show \(\text{twothird} (n + \text{Suc} m)\) 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 run: \(\text{HORun} \ OTR-M \ \rho \ \text{HO}\)
and \(\text{dec}: \text{decide} (\rho n p) = \text{Some } v\) (is \(\text{?dec } n\))
shows \(\forall k. (2\times N) \div 3 < \text{card}\{q \cdot x (\rho (n+k) q) = v\}\)
(is \(\forall k. \text{?twothird} (n+k)\))

using \(\text{dec}\) 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: \(\text{HORun-eq } \text{HOinitConfig-eq } \text{OTR-HOMachine-def}\)
\(\text{initState-def } \text{OTR-initState-def}\))

next

— For the inductive step, we assume that process \(p\) has decided on \(v\).

fix \(m\)

assume \(\text{ih}: \text{?dec } m \Rightarrow \forall k. \text{?twothird} (m+k)\) and \(\text{m: ?dec (Suc } m)\)

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 \(\text{ih}\) 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** \( \text{run: HORun OTR-M } \rho \text{ HOs} \)

**And** \( p: \text{decide } (\rho \ n \ p) = \text{Some } v \)

**And** \( p': \text{decide } (\rho \ (n+k) \ p') = \text{Some } w \)

**Shows** \( v = w \)

**Proof**

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

**QED**

**Theorem OTR-agreement:**

**Assumes** \( \text{run: HORun OTR-M } \rho \text{ HOs} \)

**And** \( p: \text{decide } (\rho \ n \ p) = \text{Some } v \)

**And** \( p': \text{decide } (\rho \ m \ p') = \text{Some } w \)

**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 \( ?\text{thesis by (auto elim: A5)} \)

- 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 \( ?\text{thesis ..} \)

**QED**

### 5.6 Proof of Termination

We now show that every process must eventually decide.

The idea of the proof is to observe that the communication predicate guarantees the existence of two uniform rounds where every process hears from the same two-thirds majority of processes. The first such round serves to ensure that all \( x \) fields hold the same value, the second round copies that
value into all decision fields.

Lemma A2 is instrumental in this proof.

**theorem** OTR-termination:
- **assumes** run: HORun OTR-M rho HOs
  and commG: HOcommGlobal OTR-M HOs
- **shows** \( \exists r \, v. \, \text{decide} (\text{rho} \ r \ p) = \text{Some} \ v \)

**proof** –
- from commG obtain \( r0 \) II where
  \( pi: \forall q. \, \text{HOs} \ r0 \ q = \Pi \) and \( pic: \text{card} \ \Pi > (2*N) \ \text{div} \ 3 \)
  by (auto simp: OTR-HOMachine-def OTR-commGlobal-def)
- let \( ?msgs \ q \ r = \text{HOrcvdMsgs} \ OTR-M \ r \ q \ (\text{HOs} \ r \ q) \ (\text{rho} \ r) \)

from run pi have \( \forall p \ q. \ ?msgs \ q \ r0 = ?msgs \ p \ r0 \)
by (auto simp: HORun-eq OTR-HOMachine-def HOrcvdMsgs-def OTR-sendMsg-def)
then obtain \( \mu \) where \( \forall q. \ ?msgs \ q \ r0 = \mu \) by auto
moreover
from \( pi \) pic have \( \forall p. \ (2*N) \ \text{div} \ 3 < \text{card} \ \{ q. \ ?msgs \ p \ r0 \ q \neq \text{None} \} \)
by (auto simp: HORun-eq HOnextConfig-eq HOrcvdMsgs-def)
with run have \( \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0) \ q) = \text{Min} \ \{ v. \ \text{MFR} \ (?msgs \ q \ r0) \ v \} \)
by (auto simp: HORun-eq HOnextConfig-eq OTR-HOMachine-def
nextState-def OTR-nextState-def)
ultimately
have \( \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0) \ q) = \text{Min} \ \{ v. \ \text{MFR} \ \mu \ v \} \) by auto
then obtain \( v \) where \( v : \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0) \ q) = v \) by auto

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 \( \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0+0) \ q) = v \) by simp
  **next**
  - fix \( k \)
  - assume \( \text{ih:} \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0 + k) \ q) = v \)
  - show \( \forall q. \ x \ (\text{rho} \ (\text{Suc} \ r0 + \text{Suc} k) \ q) = v \)
  **proof**
  - fix \( q \)
  - show \( x \ (\text{rho} \ (\text{Suc} \ r0 + \text{Suc} k) \ q) = v \)
  **proof** (cases \( (2*N) \ \text{div} \ 3 < \text{card} \ \{ p. \ ?msgs \ q \ (\text{Suc} \ r0 + k) \ p \neq \text{None} \} \))
    - case True
      - have \( N > 0 \) by (rule finite-UNIV-card-ge-0) simp
      - with \( \text{ih} \)
      - have \( (2*N) \ \text{div} \ 3 < \text{card} \ \{ p. \ x \ (\text{rho} \ (\text{Suc} \ r0 + k) \ p) = v \} \) by auto
      - with True run show \( \text{?thesis} \) by (auto elim: A2)
  **next**
  - case False
    - with run \( \text{ih} \) show \( \text{?thesis} \) by (auto dest: nextState-change)
  qed
qed
from \textit{commG} obtain $r^0' \Pi'$
\begin{itemize}
\item where $r^0': r^0' \geq \text{Suc} r^0$
\item and $\Pi': \forall q. \text{HOs} r^0' q = \Pi'$
\item and $\Pi': \text{card} \Pi' > (2 \times N) \div 3$
\end{itemize}
by (force simp: \textit{OTR-HOMachine-def} \textit{OTR-commGlobal-def})

from $r^0' P$ have $v' \forall q. x (\rho r^0' q) = v$ by (auto simp: le-iff-add)

from run have $\text{OTR-nextState} r^0' p (\rho r^0' p) (\textit{?msgs} \ p \ r^0') (\rho (\text{Suc} r^0') p)$
by (simp add: \textit{HORun-eq} \textit{HOnextConfig-eq} \textit{OTR-HOMachine-def} \textit{nextState-def})

moreover from $\Pi' \ \text{card} \Pi' < \text{card} \{q. (\textit{?msgs} \ p \ r^0') q \neq \text{None}\}$
by (auto simp: \textit{HOrcvdMsgs-def} \textit{OTR-sendMsg-def})

moreover from $\Pi' \ \text{card} \Pi' < \text{card} \{q. (\textit{?msgs} \ p \ r^0') q \neq \text{None}\}$
by (auto simp: \textit{HOrcvdMsgs-def} \textit{OTR-sendMsg-def})

ultimately have $\text{decide} (\rho (\text{Suc} r^0') p) = \text{Some} (\epsilon \ v. \text{TwoThirds} (\textit{?msgs} \ p \ r^0') v)$
by (auto simp: \textit{OTR-nextState-def})

thus $\text{thesis} \ by \ \text{blast}$

\section{One-Third Rule Solves Consensus}

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

\textbf{Theorem} \textit{OTR-consensus}:  
\begin{itemize}
\item assumes \run: \textit{HORun} \textit{OTR-M} \rho \textit{HOs} \textit{HOs} \textit{commG}: \textit{HOcommGlobal} \textit{OTR-M} \textit{HOs}
\item shows \textit{consensus} ($x \circ (\text{rho} \ 0)$) \textit{decide} \rho
\item using \textit{OTR-integrity}[\textit{OF run}] \textit{OTR-agreement}[\textit{OF run}] \textit{OTR-termination}[\textit{OF run} \ \textit{commG}]
\item by (auto simp: \textit{consensus-def} \textit{image-def})
\end{itemize}

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.

\textbf{Theorem} \textit{OTR-consensus-fg}:
\begin{itemize}
\item assumes \run: \textit{fg-run} \textit{OTR-M} \rho \textit{HOs} \textit{HOs} \textit{commG}: \textit{HOcommGlobal} \textit{OTR-M} \textit{HOs}
\item shows \textit{consensus} ($\lambda p. x (\text{state} (\text{rho} \ 0) \ p)) \textit{decide} (\text{state} \circ \rho)$
\item (is \textit{consensus} ?\textit{inits} - -)
\item proof (rule \textit{local-property-reduction}[\textit{OF run} \ \textit{consensus-is-local}])
\item fix \textit{crun}
\end{itemize}
assume crun: CSHORun OTR-M crun HOs HOs (λr q. undefined)
    and init: crun 0 = state (rho 0)
from crun have HOrun OTR-M crun HOs by (unfold HOrun-def SHORun-def)
from this commG have consensus (x ◦ (crun 0)) decide crun by (rule OTR-consensus)
with init show consensus ?inits decide crun by (simp add: o-def)
qed

end

theory UeDefs
imports ../HOModel

begin

6 Verification of the UniformVoting Consensus Algorithm

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

6.1 Model of the Algorithm

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

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

abbreviation
    N ≡ card (UNIV::Proc set) — number of processes

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

abbreviation nSteps ≡ 2

definition phase where phase (r::nat) ≡ r div nSteps

definition step where step (r::nat) ≡ r mod nSteps

The following record models the local state of a process.

record ’val pstate =
    x :: ’val — current value held by process
    vote :: ’val option — value the process voted for, if any
    decide :: ’val option — value the process has decided on, if any
Possible messages sent during the execution of the algorithm, and characteristic predicates to distinguish types of messages.

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

```ml
definition isValVote where isValVote m = \exists z v. m = ValVote z v
```

```ml
definition isVal where isVal m = \exists 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.

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

fun getval where
  getval (ValVote z v) = z
  | getval (Val z) = z
```

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

```ml
definition UV-initState where
  UV-initState p st =
  (vote st = None) \land (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.

```ml
definition msgRcvd where — processes from which some message was received
  msgRcvd (msgs :: Proc \rightarrow 'val msg) =
  \{ q . msgs q = None \}
```

```ml
definition smallestValRcvd where
  smallestValRcvd (msgs :: Proc \rightarrow ('val :: linorder) msg) =
  Min \{ v. \exists 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 $v$.

```ml
definition send0 where
  send0 r p q st = Val (x st)
```

```ml
definition next0 where
  next0 r p st (msgs :: Proc \rightarrow ('val :: linorder) msg) st' =
  \exists v.
  (\forall q \in msgRcvd msgs. msgs q = Some (Val v))
  \land st' = st \land
  (\forall (v \in \text{msgs}) 
  (vote := Some v, x := smallestValRcvd msgs))
  \lor
  (\forall (v \in \text{msgs}) 
  (vote := None, x := smallestValRcvd msgs))
  \land st' = st \land
  (x := smallestValRcvd msgs)
```

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

**definition send1 where**

\[
\text{send1 } r \ p \ q \ st \equiv \text{ValVote} \ (x \ st) \ (\text{vote} \ st)
\]

**definition valVoteRcvd where**

— processes from which values and votes were received

\[
\text{valVoteRcvd} \ (\text{msgs} : \text{Proc} \rightarrow \text{'val msg}) \equiv \{ q . \exists z \ v. \text{msgs} q = \text{Some} \ (\text{ValVote} \ v \ z) \}
\]

**definition smallestValNoVoteRcvd where**

\[
\text{smallestValNoVoteRcvd} \ (\text{msgs} : \text{Proc} \rightarrow \text{'val msg}) \equiv \text{Min} \ \{ v. \exists q. \text{msgs} q = \text{Some} \ (\text{ValVote} \ v \ q) \}
\]

**definition someVoteRcvd where**

— set of processes from which some vote was received

\[
\text{someVoteRcvd} \ (\text{msgs} : \text{Proc} \rightarrow \text{'val msg}) \equiv \{ q . q \in \text{msgRcvd} \text{msgs} \land \text{isValVote} \ (\text{the} \ (\text{msgs} q)) \land \text{getvote} \ (\text{the} \ (\text{msgs} q)) \neq \text{None} \}
\]

**definition identicalVoteRcvd where**

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

**definition x-update where**

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

**definition dec-update where**

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

**definition next1 where**

\[
\text{next1} \ r \ p \ q \ st \ \text{msgs} \ st' \equiv \text{x-update} \ st \ \text{msgs} \ st' \land \text{dec-update} \ st \ \text{msgs} \ st'
\]

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**

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

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

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**

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

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 Uniform Voting is not a coordinated algorithm.

**definition UV-HOMachine where**

\[ UV\text{-HOMachine} = () \]

\[ C\text{initState} = (\lambda p st crd. UV\text{-initState} p st), \]

\[ \text{sendMsg} = UV\text{-sendMsg}, \]

\[ C\text{nextState} = (\lambda r p st msg\text{s} crd st'. UV\text{-nextState} r p st msg\text{s} st'), \]

\[ HO\text{commPerRd} = UV\text{-commPerRd}, \]

\[ HO\text{commGlobal} = UV\text{-commGlobal} \]

\]

**abbreviation**

\[ UV\text{-M} \equiv (UV\text{-HOMachine}::\text{(Proc, 'val::linorder pstate, 'val msg) HOMachine}) \]

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 \( \exists pq. \ pq \in \text{msgRcvd} (\text{HOrcvdMsgs UV-M r p (HOs r p) (rho r)}) \)
\( \wedge \ pq \in \text{msgRcvd} (\text{HOrcvdMsgs UV-M r q (HOs r q) (rho r)}) \)
\( \wedge (\text{HOrcvdMsgs UV-M r p (HOs r p) (rho r)}) \ \text{pq} \)
\( = (\text{HOrcvdMsgs UV-M r q (HOs r q) (rho r)}) \ \text{pq} \)
using assms
by (auto simp: UV-HOMachine-def UV-commPerRd-def HOrcvdMsgs-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: \( \text{step r} = 0 \)
shows smallestValRcvd (HOrcvdMsgs UV-M r (HOs r q) (rho r)) \( \in \{ v. \exists p. (\text{HOrcvdMsgs UV-M r q (HOs r q) (rho r)}) \ p = \text{Some} (\text{Val v}) \} \)
(is smallestValRcvd ?msgs \( \in ?vals \)
unfolding smallestValRcvd-def proof (rule Min-in)
have \( \text{?vals} \subseteq \text{getval' ((the o ?msgs)'} (\text{HOs r q}) \)
by (auto simp: HOrcvdMsgs-def image-def)
thus finite ?vals by (auto simp: finite-subset)
next
from some-common-msg[of HOs, OF com]
obtain p where p \( \in \text{msgRcvd} ?msgs \) by blast
with s0 show \( \text{?vals} \neq \{ \} \)
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: \( \text{step r} \neq 0 \)
and nov: someVoteRcvd (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) = \( \{ \} \)
shows smallestValNoVoteRcvd (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) \( \in \{ v. \exists p. (\text{HOrcvdMsgs UV-M r q (HOs r q) (rho r)}) \ p = \text{Some} (\text{ValVote v None}) \} \)
(is smallestValNoVoteRcvd ?msgs \( \in ?vals \)
unfolding smallestValNoVoteRcvd-def proof (rule Min-in)
have \( \text{?vals} \subseteq \text{getval' ((the o ?msgs)'} (\text{HOs r q}) \)
by (auto simp: HOrcvdMsgs-def image-def)
thus finite ?vals by (auto simp: finite-subset)
next
from some-common-msg[of HOs, OF com]
obtain p where p \( \in \text{msgRcvd} ?msgs \) by blast
with s1 nov show \( \text{?vals} \neq \{ \} \)
by (auto simp: msgRcvd-def HOrcvdMsgs-def someVoteRcvd-def isValVote-def UV-HOMachine-def UV-sendMsg-def send1-def)

qed
The \textit{vote} field is reset every time a new phase begins.

\textbf{Lemma \resetvote:}
\begin{itemize}
  \item \textbf{Assumes}: \textit{run}: \textit{HORun UV-M rho HOs} \textbf{and} \textit{s0}: \textit{step} \textit{r'} = 0
  \item \textbf{Shows}: \textit{vote} (\textit{rho} \textit{r'} \textit{p}) = \textit{None}
\end{itemize}
\begin{proof}
\begin{itemize}
  \item \textbf{Cases} \textit{r'}
  \item \textbf{Assume} \textit{r'} = 0
  \item \textbf{With} \textit{run} \textbf{show} \textit{?thesis}
    \begin{itemize}
      \item \textbf{By} (\textit{auto simp: UV-HOMachine-def HORun-eq HOinitConfig-def initState-def UV-initState-def})
    \end{itemize}
  \end{itemize}
\end{proof}
\textbf{Next}
\begin{itemize}
  \item \textbf{Fix} \textit{r}
  \item \textbf{Assume} \textit{Sucr}: \textit{r'} = \textit{Suc} \textit{r}
  \item \textbf{From} \textit{run} \textbf{have} \textit{nxt}: nextState \textit{UV-M} \textit{r} \textit{p} (\textit{rho} \textit{r} \textit{p})
    \begin{itemize}
      \item \textit{HOrcvdMsgs UV-M} \textit{r} \textit{p} (\textit{HOs} \textit{r} \textit{p}) (\textit{rho} \textit{r})
      \item \textit{rho} (\textit{Suc} \textit{r} \textit{p})
    \end{itemize}
  \item \textbf{By} (\textit{auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def})
  \item \textbf{From} \textit{s0} \textbf{sucr} \textbf{have} \textit{step} \textit{r'} = 1 \textbf{by} (\textit{auto simp: step-def mod-Suc})
  \item \textbf{With} \textit{nxt} \textbf{sucr} \textbf{show} \textit{?thesis}
    \begin{itemize}
      \item \textbf{By} (\textit{auto simp: UV-HOMachine-def nextState-def UV-nextState-def next1-def})
    \end{itemize}
\end{itemize}
\textbf{Qed}

Processes only vote for the value they hold in their \textit{x} field.

\textbf{Lemma \xvoteeq:}
\begin{itemize}
  \item \textbf{Assumes}: \textit{run}: \textit{HORun UV-M rho HOs}
    \item \textbf{And} \textit{com}: \textit{\forall} \textit{r}. \textit{HOcommPerRd} \textit{UV-M} (\textit{HOs} \textit{r})
    \item \textbf{And} \textit{vote}: \textit{vote} (\textit{rho} \textit{r} \textit{p}) = \textit{Some} \textit{v}
  \item \textbf{Shows}: \textit{v} = \textit{x} (\textit{rho} \textit{r} \textit{p})
\end{itemize}
\begin{proof}
\begin{itemize}
  \item \textbf{Cases} \textit{r}
    \item \textbf{Case} \textit{0}
      \item \textbf{With} \textit{run} \textbf{vote} \textbf{show} \textit{?thesis} — no vote in initial state
        \begin{itemize}
          \item \textbf{By} (\textit{auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq initState-def UV-initState-def})
        \end{itemize}
    \item \textbf{Next}
      \item \textbf{Fix} \textit{r'}
        \item \textbf{Assume} \textit{r}: \textit{r} = \textit{Suc} \textit{r'}
        \item \textbf{Let} \textit{msgs} = \textit{HOrcvdMsgs UV-M} \textit{r'} \textit{p} (\textit{HOs} \textit{r'} \textit{p}) (\textit{rho} \textit{r'})
        \item \textbf{From} \textit{run} \textbf{have} \textit{nxt}: nextState \textit{UV-M} \textit{r'} \textit{p} (\textit{rho} \textit{r'} \textit{p}) \textit{msgs} (\textit{rho} (\textit{Suc} \textit{r'}) \textit{p})
          \begin{itemize}
            \item \textbf{By} (\textit{auto simp: HORun-eq HOnextConfig-eq nextState-def})
          \end{itemize}
        \item \textbf{With} \textit{vote} \textit{r}
          \item \textbf{Have} \textit{nxt0}: next0 \textit{r'} \textit{p} (\textit{rho} \textit{r'} \textit{p}) \textit{msgs} (\textit{rho} \textit{r} \textit{p}) \textbf{and} \textit{s0}: \textit{step} \textit{r'} = 0
            \begin{itemize}
              \item \textbf{By} (\textit{auto simp: nextState-def UV-HOMachine-def UV-nextState-def next1-def})
            \end{itemize}
        \item \textbf{From} \textit{run} \textit{s0} \textbf{have} \textit{vote} (\textit{rho} \textit{r'} \textit{p}) = \textit{None} \textbf{by} (\textit{rule reset-vote})
        \item \textbf{With} \textit{vote} \textit{nxt0}
          \item \textbf{Have} \textit{idv}: \textit{\forall} \textit{q} \in \textit{msgRcvd} \textit{msgs}. \textit{msgs} \textit{q} = \textit{Some} (\textit{Val} \textit{v})
            \begin{itemize}
              \item \textbf{And} \textit{x}: \textit{x} (\textit{rho} \textit{r} \textit{p}) = \textit{smallestValRcvd} \textit{msgs}
                \begin{itemize}
                  \item \textbf{By} (\textit{auto simp: next0-def})
                \end{itemize}
            \end{itemize}
          \item \textbf{Moreover}
            \item \textbf{From} \textit{com} \textbf{obtain} \textit{q} \textbf{where} \textit{q} \in \textit{msgRcvd} \textit{msgs}
by \[(force\ dest: some-common-msg)\]

\[\text{with idv have } \{x . \exists qq. \text{msgs qq = Some } (Val x)\} = \{v\}\]

by \[(auto\ simp: msgRcvd-def)\]

**hence** smallestValRcvd \?msgs = v

by \[(auto\ simp: smallestValRcvd-def)\]

ultimately

**show** \?thesis **by simp**

qed

**6.5 Proof of Irrevocability, Agreement and Integrity**

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

**lemma** decide-step:

assumes \(\text{run}: \text{HORun UV-M rho HOs}\)

and \(\text{decide}: \text{decide } (\rho (\text{Suc } r) p) \neq \text{decide } (\rho r p)\)

shows \(\text{step } r = 1\)

**proof** –

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

- from \(\text{run have nextState UV-M} r p (\rho r p) \text{msgs } (\rho (\text{Suc } r) p)\)

  by \[(auto\ simp: \text{HORun-eq} \ \text{HOnextConfig-eq} \ \text{nextState-def})\]

with \(\text{decide}\) **show** \?thesis

by \[(auto\ simp: \text{nextState-def} \ \text{UV-HOMachine-def} \ \text{UV-nextState-def} \ \text{next0-def} \ \text{step-def})\]

qed

**lemma** decide-nonnull:

assumes \(\text{run}: \text{HORun UV-M rho HOs}\)

and \(\text{decide}: \text{decide } (\rho (\text{Suc } r) p) \neq \text{decide } (\rho r p)\)

shows \(\text{decide } (\rho (\text{Suc } r) p) \neq \text{None}\)

**proof** –

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

- from \(\text{assms have s1: step } r = 1\) **by** \[(\text{rule \text{decide-step})}\]

with \(\text{decide}\) **show** \?thesis

by \[(auto\ simp: \text{UV-HOMachine-def} \ \text{HORun-eq} \ \text{HO\nextConfig-eq}\ \text{nextState-def} \ \text{UV-nextState-def})\]

with \(\text{decide}\) **show** \?thesis

by \[(auto\ simp: \text{next1-def} \ \text{dec-update-def})\]

**qed**

No process ever decides None.

**lemma** msgs-unanimity:

assumes \(\text{run}: \text{HORun UV-M rho HOs}\)

and \(\text{vote}: \text{vote } (\rho (\text{Suc } r) p) = \text{Some } v\)

and \(q: q \in \text{msgRcvd } (\text{HOrcvdMsgs UV-M} r p (\text{HOs r p}) (\rho r))\)

(is - \(\in \text{msgRcvd } \text{msgs}\))

shows \(\text{getval } (\text{the } (\text{msgs } q)) = v\)
proof
  have s0: step r = 0
proof (rule ccontr)
  assume step r ≠ 0
  hence step (Suc r) = 0 by (simp add: step-def mod-Suc)
  with run vote show False by (auto simp: reset-vote)
qed

with run have novote: vote (rho r p) = None by (auto simp: reset-vote)
from run have nextState UV-M r p (rho r p) ?msgs (rho (Suc r) p)
  by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
with s0 have nxt: next0 r p (rho r p) ?msgs (rho (Suc r) p)
  by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
with novote vote q show ?thesis by (auto simp: next0-def)
qed

Any two processes can only vote for the same value.

lemma vote-agreement:
  assumes run: HORun UV-M rho HOs
  and com: ∀ r. HOcommPerRd UV-M (HOs 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 HORun-eq HOinitConfig-eq
      initState-def UV-initState-def)
next
  fix r'
  assume r: r = Suc r'
  let ?msgs p = HORcdMsgs UV-M r' p (HOs r' p) (rho r')
  from com obtain pq
    where ?msgs p pq = ?msgs q pq
      and smp: pq ∈ msgRcvd (?msgs p) and smq: pq ∈ 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 smp r have getval (the (?msgs q pq)) = w
    by (simp add: msgs-unanimity)
  ultimately
  show ?thesis by simp
qed

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

lemma decide-equals-x:
  assumes run: HORun UV-M rho HOs
  and com: ∀ r. HOcommPerRd UV-M (HOs 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 nextState UV-M r p (\(\rho r p\)) (\(?msgs p\)) (\(\rho (\text{Suc} \ r) \ p\))
    by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} nextState-def)

with \(s_1\) have nxtp: nxt1 r p (\(\rho r p\)) (\(?msgs p\)) (\(\rho (\text{Suc} \ r) \ p\))
    by (auto simp: \text{UV-HOMachine-def} nextState-def \text{UV-nextState-def})

from run have nextState UV-M r q (\(\rho r q\)) (\(?msgs q\)) (\(\rho (\text{Suc} \ r) \ q\))
    by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} nextState-def)

with \(s_1\) have nxtq: nxt1 r q (\(\rho r q\)) (\(?msgs q\)) (\(\rho (\text{Suc} \ r) \ q\))
    by (auto simp: \text{UV-HOMachine-def} nextState-def \text{UV-nextState-def})

from \text{com} obtain \(pq\) where
    \(pq\): \(pq \in \text{msgRcvd} (\text{?msgs p})\)
    \(?msgs p\) \(pq) = (\text{?msgs q} \ pq)
    by (force dest: some-common-msg)
with decide decval nxtp
have vote: isValVote (the (\text{?msgs p pq}))
    getvote (the (\text{?msgs p pq})) = Some \(v\)
    by (auto simp: next1-def dec-update-def identicalVoteRcvd-def)

with nxtq pq obtain \(q'\) where
    \(q'\): \(q' \in \text{someVoteRcvd} (\text{?msgs q})\)
    \(x (\rho (\text{Suc} \ r) \ q) = \text{the} (\text{getvote} (\text{the} (\text{?msgs q q'})))\)
    by (auto simp: next1-def x-update-def someVoteRcvd-def)

with \(s_1\) \text{pq} vote show \(?thesis\)
    by (auto simp: \text{HORcvdMsgs-def} \text{UV-HOMachine-def} \text{UV-sendMsg-def} send1-def
        someVoteRcvd-def msgRcvd-def vote-agreement[OF \text{run} \text{com}])
qed

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

lemma same-x-stable:
    assumes \(\text{run}: \text{HORun} UV-M \rho \text{HOs}\)
        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 \(p: p \in \text{msgRcvd} \ ?msgs\)
        by (force dest: some-common-msg)
    from \text{run} have nextState UV-M r q (\(\rho r q\)) (\text{?msgs q}) (\(\rho (\text{Suc} \ r) \ q\))
        by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} nextState-def)
    hence \(\text{nxt0 q r} (\rho r q) (\text{?msgs q}) (\rho (\text{Suc} \ r) \ q) \land \text{step} r = 0\)
        \(\lor \text{nxt1 r q (\rho r q) (\text{?msgs q}) (\rho (\text{Suc} \ r) \ q) \land step} r \neq 0\)
        (is \(\text{nxt0} \lor \text{nxt1}\))
        by (auto simp: \text{UV-HOMachine-def} nextState-def \text{UV-nextState-def})

52
thus \(?thesis\)

proof

assume \(\text{nxt0}: \?\text{nxt0}\)
	hence \(x\ (\rho\ (\text{Suc}\ r)\ q) = \text{smallestValRcvd}\ \?\text{msgs}\)
	by (auto simp: \text{next0-def})

moreover

from \(\text{nxt0}\ \?\text{x}\) 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 \text{someVoteRcvd}\ \?\text{msgs} = \{\})

case True

with \(\text{nxt1}\ \?\text{x}\ True\)

have \(\forall\ p\ \in\ \text{msgRcvd}\ \?\text{msgs}.\ \?\text{msgs}\ p = \text{Some}\ (\text{ValVote}\ v\ None)\)
	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 \(\{x . \ \exists\ p.\ \?\text{msgs}\ p = \text{Some}\ (\text{ValVote}\ x\ None)\} = \{v\}\)
	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 \(\text{nxt1}\) obtain \(p'\ v'\ where\)

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

getvote \text{the} \ ((\?\text{msgs}\ p')) = \text{Some}\ v'x (\rho\ (\text{Suc}\ r)\ q) = v'
	by (auto simp: \text{someVoteRcvd-def} \text{next1-def} \text{x-update-def})

with \(\text{nxt1}\ \?\text{x}\)

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 \text{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: ∀ r. HOcommPerRd UV-M (HOs r)
and decide: decide (rho (Suc r) p) ≠ 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 \( \forall q. (rho (Suc r+k) q) = v \)
  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 some-
time in the past.

lemma decided-then-past-decision:
assumes run: HORun UV-M rho HOs
and dec: decide (rho n p) = Some v
shows \( \exists m n. (rho (Suc m) p) ≠ decide (rho m p) \)
  ∧ decide (rho (Suc m) p) = Some v
proof (induct n)
  from run show \( ?P 0 \) by (auto simp: HORun-eq UV-HOMachine-def HOinitConfig-eq
    initState-def UV-initState-def)
next
  fix n
  assume ih: \( ?P n \) thus \( ?P (Suc n) \) by force
qed
with dec show \( ?thesis \) by auto
qed

We can now prove the safety properties of the algorithm, and start with
proving Integrity.

lemma x-values-initial:
assumes run: HORun UV-M rho HOs
and com: ∀ 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
fix \( r p \)
assume \( \text{ih}: \exists q. x (\rho r p) = x (\rho 0 q) \)
let \( \text{run} \ \text{have} \ \text{nextState} \ UV-M r p (\rho r p) \ ?msgs \ (\rho (\text{Suc} r) p) \)
by (auto simp: \text{HORun-eq} \text{HOnextConfig-eq} \text{nextState-def})

hence \( \text{next0} r p (\rho r p) \ ?msgs \ (\rho (\text{Suc} r) p) \land \text{step} r = 0 \)
\lor \( \text{next1} r p (\rho r p) \ ?msgs \ (\rho (\text{Suc} r) p) \land \text{step} r \neq 0 \)
(is \( ?\text{next0} \lor ?\text{next1} \))
by (auto simp: \text{UV-HOMachine-def} \text{nextState-def} \text{UV-nextState-def})

thus \( \exists q. x (\rho (\text{Suc} r) p) = x (\rho 0 q) \)
proof
  assume \( \text{next0} \): \( ?\text{next0} \)
  hence \( x (\rho (\text{Suc} r) p) = \text{smallestValRcvd} \ ?msgs \)
  by (auto simp: next0-def)
  also with \( \text{com} \ \text{next0} \)
  have \( \ldots \in \{ v . \exists q. ?msgs q = \text{Some} (\text{Val} v) \} \)
  by (intro \text{minval-step0}) auto
  also with \( \text{next0} \)
  have \( \ldots = \{ x (\rho r q) \mid q . q \in \text{msgRcvd} \ ?msgs \} \)
  by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def}
    \text{msgRcvd-def} \text{send0-def})
  finally obtain \( q \) where \( x (\rho (\text{Suc} r) p) = x (\rho r q) \) by auto
  with \( \text{ih} \) show \( \text{thesis} \) by auto

next
assume \( \text{next1} : ?\text{next1} \)
show \( \text{thesis} \)
proof (cases someVoteRcvd \( ?msgs = \{\} \))
  case True
  with \( \text{next1} \)
  have \( x (\rho (\text{Suc} r) p) = \text{smallestValNoVoteRcvd} \ ?msgs \)
  by (auto simp: next1-def \text{x-update-def})
  also with \( \text{com} \ \text{next1} \ True \)
  have \( \ldots \in \{ v . \exists q. ?msgs q = \text{Some} (\text{ValVote} v \text{ None}) \} \)
  by (intro \text{minval-step1}) auto
  also with \( \text{next1} \ True \)
  have \( \ldots = \{ x (\rho r q) \mid q . q \in \text{msgRcvd} \ ?msgs \} \)
  by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def}
    \text{someVoteRcvd-def} \text{isValVote-def} \text{msgRcvd-def} \text{send1-def})
  finally obtain \( q \) where \( x (\rho (\text{Suc} r) p) = x (\rho r q) \) by auto
  with \( \text{ih} \) show \( \text{thesis} \) by auto

next
  case False
  with \( \text{next1} \) obtain \( q \) where
  \( q \in \text{someVoteRcvd} \ ?msgs \)
  \( x (\rho (\text{Suc} r) p) = \text{the} (\text{getvote} (\text{the} (\ ?msgs q))) \)
  by (auto simp: next1-def \text{x-update-def})
  with \( \text{next1} \)
  have \( \text{vote} (\rho r q) = \text{Some} (x (\rho (\text{Suc} r) p)) \)
  by (auto simp: \text{UV-HOMachine-def} \text{HOrcvdMsgs-def} \text{UV-sendMsg-def}
    \text{someVoteRcvd-def} \text{isValVote-def} \text{msgRcvd-def} \text{send1-def})
  with \( \text{run} \ \text{com} \ \text{have} x (\rho (\text{Suc} r) p) = x (\rho r q) \)
  by (rule \text{x-vote-eq})
  with \( \text{ih} \) show \( \text{thesis} \) by auto

55
 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-post-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 decidedp: decide (\rho (Suc r) p) \neq decide (\rho r p)
  and decvalp: decide (\rho (Suc r) p) = Some v
  and decidedq: 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 decidedp decvalp have x (\rho (Suc r+k) q) = v
  by (rule safety-argument)
  moreover
  from run com decidedq 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

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!: decide-nonnul)
    with p have w = v by (auto simp: uv-agreement[OF run com])
    with w show ?thesis by simp
  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 \(?msgs p = \text{HOrcvdMsgs} UV-M r p (HOs r p) (\rho r)\) from **hoeq** have msgeq: \(?msgs p = ?msgs q\) by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def send0-def send1-def)

show \(?thesis\)

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

  case True with run 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: UV-HOMachine-def HOrun-eq HOnextConfig-eq nextState-def UV-nextState-def)

hence \( \text{?nxt0 } p \text{ ?nxt0 } q \) by auto with msgeq show \(?thesis\) by (auto simp: next0-def)

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

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: UV-HOMachine-def HOrun-eq HOnextConfig-eq nextState-def 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: next1-def)

with msgeq have

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

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

by auto

show \(?thesis\)

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

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

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

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

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

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

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

qed

We now prove that UniformVoting terminates.

**theorem** uv-termination:
assumes run: HORun UV-M rho HOs
and commR: \( \forall r. \text{HOcommPerRd} \text{ UV-M } (\text{HOs } r) \)
and commG: \( \text{HOcommGlobal} \text{ UV-M } \text{HOs} \)

shows \( \exists r v. \text{decide } (\text{rho } r p) = \text{Some } v \)

proof –

First obtain a round where all \( x \) values agree.

from \( \text{commG} \) obtain \( r0 \) where \( r0: \forall q. \text{HOs } r0 \text{ q } = \text{HOs } r0 \text{ p} \)
by \( \text{force simp: UV-HOMachine-def UV-commGlobal-def} \)
let \( ?v = x \ (\text{rho } (\text{Suc } r0) \text{ p}) \)
from \( \text{run commR } r0 \) have \( xs: \forall q. x \ (\text{rho } (\text{Suc } r0) \text{ q}) = ?v \)
by \( \text{auto dest: hoeq-xeq} \)

Now obtain a round where all votes agree.

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

have \( \text{vote'}: \forall q. \text{vote } (\text{rho } (\text{Suc } r') \text{ q}) = \text{Some } ?v \)
proof
fix \( q \)
let \( ?msgs = \text{HOrcvdMsgs } \text{UV-M } r' \text{ q } (\text{HOs } r' \text{ q}) \ (\text{rho } r') \)
from \( \text{run stp'} \) have \( \text{next0 } r' \text{ q } (\text{rho } r' \text{ q}) \ ?msgs \ (\text{rho } (\text{Suc } r') \text{ q}) \)
by \( \text{force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def UV-nextState-def} \)
moreover
from \( \text{stp'} \ x' \) have \( \forall q' \in \text{msgRcvd } ?msgs. \ ?msgs \ q' = \text{Some } (\text{Val } ?v) \)
by \( \text{auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def send0-def msgRcvd-def} \)
moreover
from \( \text{commR} \) have \( \text{msgRcvd } ?msgs \neq {} \)
by \( \text{force dest: some-common-msg} \)
ultimately
show \( \text{vote } (\text{rho } (\text{Suc } r') \text{ q}) = \text{Some } ?v \)
by \( \text{auto simp: next0-def} \)
qed

At the subsequent round, process \( p \) will decide.

let \( ?r'' = \text{Suc } r' \)
let \( ?msgs' = \text{HOrcvdMsgs } \text{UV-M } ?r'' \text{ p } (\text{HOs } ?r'' \text{ p}) \ (\text{rho } ?r'') \)
from \( \text{stp'} \) have \( \text{stp'': step } ?r'' = 1 \)
by (simp add: step-def mod-Suc)
with run have next1 ?r'' p (rho ?r'' p) ?msgs' (rho (Suc ?r'') p)
  by (auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq
       nextState-def UV-nextState-def)
moreover
from stp'' vote' have identicalVoteRcvd ?msgs' ?v
  by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def
       send1-def identicalVoteRcvd-def isValVote-def)
moreover
from commR have msgRcvd ?msgs' ⊄ {}
  by (force dest: some-common-msg)
ultimately
have decide (rho (Suc ?r'') p) = Some ?v
  by (force simp: next1-def dec-update-def identicalVoteRcvd-def
       msgRcvd-def isValVote-def)
thus ?thesis by blast
qed

6.7 UniformVoting Solves Consensus

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

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

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

theorem uv-consensus-fg:
  assumes run: fg-run UV-M rho HOs HOs (\lambda q. undefined)
  and commR: \forall r. HOcommPerRd UV-M (HOs r)
  and commG: HOcommGlobal UV-M HOs
  shows consensus (\chi p. \chi (state (rho 0) p)) decide (state o rho)
  (is consensus ?inits - -)
proof (rule local-property-reduction[OF run consensus-is-local])
fix crun
assume crun: CSHORun UV-M crun HOs HOs (\lambda q. undefined)
  and init: crun 0 = state (rho 0)
from crun have HORun UV-M crun HOs
  by (unfold HORun-def SHORun-def)
from this commR commG have consensus (\chi \circ (crun 0)) decide crun
  by (rule uv-consensus)
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.

typedef 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.

```haskell
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
coord Φ :: Proc  — coordinator for current phase
```

Possible messages sent during the execution of the algorithm.

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

Characteristic predicates on messages.

```haskell
definition isValStamp where isValStamp m ≡ ∃v ts. m = ValStamp v ts
definition isVote where isVote m ≡ ∃v. m = Vote v
definition 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.

```haskell
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.

```haskell
definition LV-initState where
  LV-initState p st crd ≡
  vote st = None
  ∧ ¬(commit st)
  ∧ ¬(ready st)
  ∧ timestamp st = 0
  ∧ decide st = None
  ∧ coord Φ 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 -> 'val msg) ≡
{ q . 3 v ts. msgs q = Some (ValStamp v ts)}

definition highestStampRcvd where
highestStampRcvd msgs ≡
Max { ts . 3 q v. (msgs::Proc -> '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 ≡
if q = coordΦ st then ValStamp (x st) (timestamp st) else Null

definition next0 where
next0 r p st msgs crd st' ≡
if p = coordΦ st ∧ card (valStampsRcvd msgs) > N div 2
then (3 p v. msgs p = Some (ValStamp v (highestStampRcvd msgs))
∧ st' = st (\| vote := Some v, commit := True \))
else st' = st

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 ≡
if p = coordΦ st ∧ commit st then Vote (the (vote st)) else Null

definition next1 where
next1 r p st msgs crd st' ≡
if msgs (coordΦ st) ≠ None ∧ isVote (the (msgs (coordΦ st)))
then st' = st (\| x := val (the (msgs (coordΦ st))), timestamp := Suc(phase r) \)
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 ≡
if timestamp st = Suc(phase r) ∧ q = coordΦ st then Ack else Null

— processes from which an acknowledgement was received

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

**definition** next2 where

\[
\text{next2} \ r \ p \ st \ msgs \ crd \ st' \equiv \\
\text{if} \ p = \text{coord}\Phi \ st \land \text{card} (\text{acksRcvd} \ msgs) > N \text{ div } 2 \\
\text{then} \ st' = st \ (| \ ready := \text{True} |) \\
\text{else} \ st' = st
\]

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

**definition** send3 where

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

**definition** next3 where

\[
\text{next3} \ r \ p \ st \ msgs \ crd \ st' \equiv \\
(\if \msgs (\text{coord}\Phi \ st) \neq \text{None} \land \text{isVote} (\text{the} (\msgs (\text{coord}\Phi \ st))) \\
\text{then decide} \ st' = \text{decide} \ st \\
\text{else decide} \ st' = \text{decide} \ st) \\
\land (\if \ p = \text{coord}\Phi \ st \\
\text{then} \neg (\text{ready} \ st') \land \neg (\text{commt} \ st') \\
\text{else ready} \ st' = \text{ready} \ st \land \text{commt} \ st' = \text{commt} \ st) \\
\land \ x \ st' = x \ st \\
\land \text{vote} \ st' = \text{vote} \ st \\
\land \text{timestamp} \ st' = \text{timestamp} \ st \\
\land \text{coord}\Phi \ st' = \text{crd}
\]

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

**definition** LV-sendMsg :: nat ⇒ Proc ⇒ Proc ⇒ ‘val pstate ⇒ ‘val msg where

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

**definition**

\[
\text{LV-nextState} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{‘val pstate} \Rightarrow \text{‘val msg} \Rightarrow \text{bool} \\
\text{where} \\
\text{LV-nextState} \ r \equiv \\
\text{if} \ \text{step} \ r = 0 \ \text{then next0} \ r \\
\text{else if} \ \text{step} \ r = 1 \ \text{then next1} \ r \\
\text{else if} \ \text{step} \ r = 2 \ \text{then next2} \ r \\
\text{else next3} \ r
\]

64
7.2 Communication Predicate for \textit{LastVoting}

We now define the communication predicate that will be assumed for the correctness proof of the \textit{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.)

\textbf{definition}

$LV$-commPerRd where

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

\textbf{definition}

$LV$-commGlobal where

$LV$-commGlobal $HOs$ coords $\equiv$

$\exists ph::\mathbb{N}. \exists c::Proc.$

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

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

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

$\land \ (\forall p. \ c \in HOs \ (4*ph+2) \ p \cap HOs \ (4*ph+3) \ p)$

7.3 The \textit{LastVoting} Heard-Of Machine

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

\textbf{definition} $LV$-CHO\textit{Machine} where

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

$\langle$ CinitState $= LV$-\textit{initState}$,$

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

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

CHOcommPerRd $= LV$-\textit{commPerRd}$,$

CHOcommGlobal $= LV$-\textit{commGlobal}$\rangle$

\textbf{abbreviation}

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

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

7.4 Preliminary Lemmas

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

lemma timeStampsRcvdFinite:
finite \{ts . \exists q v. (msgs :: Proc \rightarrow '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 \neq {}
obtains p v where msgs p = Some (ValStamp v (highestStampRcvd msgs))
proof -
  let ?ts = \{ts . \exists q v. msgs q = Some (ValStamp v ts)\}
  from nempty have ?ts \neq {} by (auto simp add: valStampsRcvd-def)
  with timeStampsRcvdFinite
  have highestStampRcvd msgs \in ?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 \leq 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: \forall p. CinitState LV-M p (rho 0 p) (coords 0 p) \rightarrow P 0
and step0: \( \forall r. \)
\[
\begin{align*}
\text{step } r &= 0; \ P r; \ \text{phase} \ (\text{Suc} \ r) = \text{phase } r; \ \text{step} \ (\text{Suc} \ r) = 1; \\
\forall p. \ \text{next0 } r \ p \ (\rho \ r \ p) & \Rightarrow P \ (\text{Suc} \ r)
\end{align*}
\]

and step1: \( \forall r. \)
\[
\begin{align*}
\text{step } r &= 1; \ P r; \ \text{phase} \ (\text{Suc} \ r) = \text{phase } r; \ \text{step} \ (\text{Suc} \ r) = 2; \\
\forall p. \ \text{next1 } r \ p \ (\rho \ r \ p) & \Rightarrow P \ (\text{Suc} \ r)
\end{align*}
\]

and step2: \( \forall r. \)
\[
\begin{align*}
\text{step } r &= 2; \ P r; \ \text{phase} \ (\text{Suc} \ r) = \text{phase } r; \ \text{step} \ (\text{Suc} \ r) = 3; \\
\forall p. \ \text{next2 } r \ p \ (\rho \ r \ p) & \Rightarrow P \ (\text{Suc} \ r)
\end{align*}
\]

and step3: \( \forall r. \)
\[
\begin{align*}
\text{step } r &= 3; \ P r; \ \text{phase} \ (\text{Suc} \ r) = \text{Suc} \ (\text{phase } r); \ \text{step} \ (\text{Suc} \ r) = 0; \\
\forall p. \ \text{next3 } r \ p \ (\rho \ r \ p) & \Rightarrow P \ (\text{Suc} \ r)
\end{align*}
\]

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 nxt: CHOnextConfig LV-M r (\( \rho \ r \)) \( (\text{HOs } r) \)
(\( \text{coords } (\text{Suc} \ r) \)) \( (\rho \ (\text{Suc} \ r) ) \)
have \( \text{step } r \in \{0,1,2,3\} \) by (auto simp add: step-def)
thus \( P \ (\text{Suc} \ r) \) by (intro step0)

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 nxt stp show \( \text{thesis} \)
by (intro step0)
(\( \text{auto simp: LV-CHOMachine-def CHOnextConfig-eq} \)
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: \text{step-def mod-Suc})
with \( \text{ih nxt stp show} \) \text{thesis}
by (intro \text{step1})
  (auto simp: \text{LV-CHOMachine-def CHOnextConfig-eq}
    \text{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: \text{step-def mod-Suc})
with \( \text{ih nxt stp show} \) \text{thesis}
by (intro \text{step2})
  (auto simp: \text{LV-CHOMachine-def CHOnextConfig-eq}
    \text{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: \text{step-def mod-Suc})
with \( \text{ih nxt stp show} \) \text{thesis}
by (intro \text{step3})
  (auto simp: \text{LV-CHOMachine-def CHOnextConfig-eq}
    \text{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 \( \text{run} : \text{CHORun LV-M rho HOs coords} \)
and \text{step0}: \[ \begin{array}{l}
\text{step} \, r = 0; \text{step} (\text{Suc} \, r) = 1; \text{phase} (\text{Suc} \, r) = \text{phase} \, r; \\
\forall \, p. \text{next0} \, r \, p \, (\text{rho} \, r \, p) \\
\quad (\text{HOrcvdMsgs LV-M} \, r \, p \, (\text{HOs} \, r \, p) \, (\text{rho} \, r)) \\
\quad (\text{coords} (\text{Suc} \, r) \, p \, (\text{rho} \, (\text{Suc} \, r) \, p)) \\
\end{array} \]
\implies P \, r
and \text{step1}: \[ \begin{array}{l}
\text{step} \, r = 1; \text{step} (\text{Suc} \, r) = 2; \text{phase} (\text{Suc} \, r) = \text{phase} \, r; \\
\forall \, p. \text{next1} \, r \, p \, (\text{rho} \, r \, p) \\
\quad (\text{HOrcvdMsgs LV-M} \, r \, p \, (\text{HOs} \, r \, p) \, (\text{rho} \, r)) \\
\quad (\text{coords} (\text{Suc} \, r) \, p \, (\text{rho} \, (\text{Suc} \, r) \, p)) \\
\end{array} \]
\implies P \, r
and \text{step2}: \[ \begin{array}{l}
\text{step} \, r = 2; \text{step} (\text{Suc} \, r) = 3; \text{phase} (\text{Suc} \, r) = \text{phase} \, r; \\
\forall \, p. \text{next2} \, r \, p \, (\text{rho} \, r \, p) \\
\quad (\text{HOrcvdMsgs LV-M} \, r \, p \, (\text{HOs} \, r \, p) \, (\text{rho} \, r)) \\
\quad (\text{coords} (\text{Suc} \, r) \, p \, (\text{rho} \, (\text{Suc} \, r) \, p)) \\
\end{array} \]
\implies P \, r
and \text{step3}: \[ \begin{array}{l}
\text{step} \, r = 3; \text{step} (\text{Suc} \, r) = 0; \text{phase} (\text{Suc} \, r) = \text{Suc} \, (\text{phase} \, r); \\
\forall \, p. \text{next3} \, r \, p \, (\text{rho} \, r \, p) \\
\quad (\text{HOrcvdMsgs LV-M} \, r \, p \, (\text{HOs} \, r \, p) \, (\text{rho} \, r)) \\
\quad (\text{coords} (\text{Suc} \, r) \, p \, (\text{rho} \, (\text{Suc} \, r) \, p)) \\
\end{array} \]
\implies P \, r

68
shows \( P \) \( r \)

proof –

from run

have \( \text{nxt}: \text{CHOnextConfig} \ \text{LV-M} \ \text{r} \ \text{(rho} \ \text{r}) \ \text{(HOs} \ \text{r}) \)

\[
(coords \ (Suc \ r)) \ (rho \ (Suc \ r))
\]

by (auto simp add: CHORun-eq)

have \( \text{step} \ r \in \{0,1,2,3\} \) by (auto simp add: step-def)

thus \( P \) \( r \)

proof (auto)

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

hence \( \text{step} \ (Suc \ r) = 1 \)

by (auto simp add: step-def mod-Suc)

with \( \text{nxt} \ \text{stp} \ \text{show} \ ?\text{thesis} \)

by (intro step0)

(auto simp: LV-CHOMachine-def CHOnextConfig-eq
LV-nextState-def LV-sendMsg-def phase-Suc)

next

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

hence \( \text{step} \ (\text{Suc} \ r) = 2 \)

by (auto simp add: step-def mod-Suc)

with \( \text{nxt} \ \text{stp} \ \text{show} \ ?\text{thesis} \)

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 \( \text{nxt} \ \text{stp} \ \text{show} \ ?\text{thesis} \)

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 \( \text{nxt} \ \text{stp} \ \text{show} \ ?\text{thesis} \)

by (intro step3)

(auto simp: LV-CHOMachine-def CHOnextConfig-eq
LV-nextState-def LV-sendMsg-def phase-Suc)

qed

qed

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

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

assumes run: \( \text{CHORun} \ \text{LV-M} \ \text{rho} \ \text{HOs} \ \text{coords} \)
and \textit{init}: \textit{CinitState} \(LV-M\) \(p\) \((\text{rho} \ 0 \ p)\) \((\text{coords} \ 0 \ p)\) \(\Rightarrow\) \(P\ \ p\ \ 0\)
and \textit{step0}: \(\bigwedge r.\ [\text{step} \ r = 0; \ P \ p \ r; \ phase (\text{Suc} \ r) = \text{phase} \ r; \ step (\text{Suc} \ r) = 1; \ \) next0 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ (\text{Suc} \ r)\)
and \textit{step1}: \(\bigwedge r.\ [\text{step} \ r = 1; \ P \ p \ r; \ phase (\text{Suc} \ r) = \text{phase} \ r; \ step (\text{Suc} \ r) = 2; \) next1 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ (\text{Suc} \ r)\)
and \textit{step2}: \(\bigwedge r.\ [\text{step} \ r = 2; \ P \ p \ r; \ phase (\text{Suc} \ r) = \text{phase} \ r; \ step (\text{Suc} \ r) = 3; \) next2 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ (\text{Suc} \ r)\)
and \textit{step3}: \(\bigwedge r.\ [\text{step} \ r = 3; \ P \ p \ r; \ phase (\text{Suc} \ r) = \text{Suc} \ (\text{phase} \ r); \ step (\text{Suc} \ r) = 0; \)

next3 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ (\text{Suc} \ r)\)

shows \(P \ p \ n\)
by \((\text{rule} \ \textit{LV-induct}\{\text{OF} \ \text{run}\})\)

(auto intro: \textit{init} \textit{step0} \textit{step1} \textit{step2} \textit{step3})

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

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ r\)
and \textit{step1}: \(\bigwedge \[\text{step} \ r = 1; \ \text{step} \ (\text{Suc} \ r) = \text{phase} \ r; \) next1 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ r\)
and \textit{step2}: \(\bigwedge \[\text{step} \ r = 2; \ \text{step} \ (\text{Suc} \ r) = \text{phase} \ r; \) next2 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ r\)
and \textit{step3}: \(\bigwedge \[\text{step} \ r = 3; \ \text{step} \ (\text{Suc} \ r) = \text{Suc} \ (\text{phase} \ r); \) next3 \(r \ p\) \((\text{rho} \ r \ p)\)

\((\text{HOrcvdMsgs} \ LV-M \ r \ p \ (\text{HOs} \ r \ p) \ (\text{rho} \ r))\)

\((\text{coords} \ (\text{Suc} \ r) \ p) \ (\text{rho} \ (\text{Suc} \ r) \ p)\) \[]

\(\Rightarrow P \ p \ r\)

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:** \( run: \text{CHORun LV-M \rho \text{HOs coords}} \)
- **Shows:** \( \text{timestamp} (\rho n p) \leq (\text{if step } n < 2 \text{ then phase } n \text{ else Suc (phase } n)) \)
- **By:** (rule \( \text{LV-induct'} \)[OF \( \text{run} \), \( \text{where } P=\text{?P} \)])

Moreover, timestamps can only grow over time.

**Lemma LV-timestamp-increasing:**
- **Assumes:** \( run: \text{CHORun LV-M \rho \text{HOs coords}} \)
- **Shows:** \( \text{timestamp} (\rho n p) \leq \text{timestamp} (\rho (\text{Suc } n) p) \)
- **Proof:** (rule \( \text{LV-Suc'} \)[OF \( \text{run} \), \( \text{where } P=\text{?P} \)])

The case of \( \text{next1} \) 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.

**Lemma LV-timestamp-monotonic:**
- **Assumes:** \( run: \text{CHORun LV-M \rho \text{HOs coords and le: } m \leq n} \)
- **Shows:** \( \text{timestamp} (\rho m p) \leq \text{timestamp} (\rho n p) \)
- **Proof:**

Finally, timestamps have the property of being bounded and monotonic, ensuring a consistent and predictable flow of events in the system.
The following definition collects the set of processes whose timestamp is beyond a given bound at a system state.

**Definition** \( \text{procsBeyondTS} \) where

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

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

**Lemma** \( \text{procsBeyondTS-monotonic} \):

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

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

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

proof –

from \( p \) have \( ts \leq \text{timestamp} \ (\rho \ m \ p) \) (is \( \leq \) ?t\( m \))

by (simp add: \text{procsBeyondTS-def})

moreover

from \( \text{run le} \) have \( ?t \ m \leq ?t \ n \) by (rule \text{LV-timestamp-monotonic})

ultimately show \( \text{thesis} \)

by (simp add: \text{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** \( \text{notStep3EqualCoord} \):

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

shows \( \text{coord} \ (\rho \ (\text{Suc} \ r) \ p) = \text{coord} \ (\rho \ r \ p) \) (is \( ?P \ p \ r \))

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

(auto simp: \text{stp next0-def next1-def next2-def})

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

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

shows \( \text{coord} \ (\rho \ r \ p) = \text{coords} \ (4 \ast (\text{phase} \ r)) \ p \)

proof –

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

let \( ?r1 = (4 \ast (\text{phase} \ r)) \)

have \( \text{coord} \ (\rho \ ?r1 \ p) = \text{coords} \ ?r1 \ p \)

proof (cases \text{phase} \ r \ > \ 0)

  case False

  with \( \text{ih show} \ ?P \ (\text{Suc} \ k) \) by simp

  qed

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

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

next
  case True
  hence step (Suc ?r0) = 0 by (auto simp: step-def)
  hence step ?r0 = 3 by (auto simp: mod-Suc step-def)
moreover
  from run
  have LV-nextState ?r0 p (rho ?r0 p)
      (HOrcvdMsgs LV-M ?r0 p (HOs ?r0 p) (rho ?r0))
      (coords (Suc ?r0) p) (rho (Suc ?r0) p)
    by (auto simp: LV-CHOMachine-def CHORun-eq CHOnextConfig-eq)
ultimately
  have nxt: next3 ?r0 p (rho ?r0 p)
      (HOrcvdMsgs LV-M ?r0 p (HOs ?r0 p) (rho ?r0))
      (coords (Suc ?r0) p) (rho (Suc ?r0) p)
    by (auto simp: LV-nextState-def)
  hence coordΦ (rho (Suc ?r0) p) = coords (Suc ?r0) p
    by (auto simp: next3-def)
  with True show ?thesis by auto
qed
moreover
  from run
  have coordΦ (rho (Suc (Suc (Suc ?r1)))) p = coordΦ (rho ?r1 p)
    ∧ coordΦ (rho (Suc (Suc ?r1))) p = coordΦ (rho ?r1 p)
    ∧ coordΦ (rho (Suc ?r1)) p = coordΦ (rho ?r1 p)
    by (auto simp: notStep3EqualCoord step-def phase-def mod-Suc)
moreover
  have r ∈ {?r1, Suc ?r1, Suc (Suc ?r1), Suc (Suc (Suc ?r1))}
    by (auto simp: step-def phase-def mod-Suc)
ultimately
  show ?thesis by auto
qed

Votes only change at step 0.

lemma notStep0EqualVote [rule-format]:
  assumes run: CHORun LV-M rho HOs coords
  shows step r ≠ 0 → vote (rho (Suc r) p) = vote (rho r p) (is ?P p r)
  by (rule LV-Suc["OF run, where P=?P"]
       (auto simp: next0-def next1-def next2-def next3-def))

Commit status only changes at steps 0 and 3.

lemma notStep03EqualCommit [rule-format]:
  assumes run: CHORun LV-M rho HOs coords
  shows step r ≠ 0 ∧ step r ≠ 3 → commit (rho (Suc r) p) = commit (rho r p)
    (is ?P p r)
  by (rule LV-Suc["OF run, where P=?P"])

73
(auto simp: next0-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 ≠ 1 ⟹ timestamp (rho (Suc r) p) = timestamp (rho r p) (is ?P p r)

**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 ≠ 1 ⟹ x (rho (Suc r) p) = x (rho r p) (is ?P p r)

**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 ≤ step r; coordΦ (rho r p) = p; vote (rho r p) ≠ None; card {q . coordΦ (rho r q) = p} > N div 2

shows A

**proof** –

**have** commit (rho r p) ⟹

1 ≤ step r
∧ coordΦ (rho r p) = p
∧ vote (rho r p) ≠ None
∧ card {q . coordΦ (rho r q) = p} > N div 2
(is ?P p r is - ⟹ ?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) (HOrcedMsgs 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
\[
\begin{align*}
& \text{assume } cm': \text{commt } (\rho \text{ Suc n } p) \\
& \text{from stp ih have } cm: \neg \text{commt } (\rho \text{ n } p) \text{ by simp} \\
& \quad \text{with nxt cm'} \\
& \quad \text{have } \text{coord}(\rho \text{ n } p) = p \\
& \quad \quad \land \text{vote } (\rho \text{ Suc n } p) \neq \text{None} \\
& \quad \quad \land \text{card } (\text{valStampsRcvd } (\text{HOrcvdMsgs } \text{LV-M n } p \text{ HO } n \rho) (\rho n)) \\
& \quad \quad > N \text{ div } 2 \\
& \quad \quad \text{by } (\text{auto simp add: next0-def}) \\
& \quad \text{hence } \text{coord}(\rho \text{ n } q) = p \\
& \quad \quad \supseteq \{ q . \text{coord}(\rho n q) = p \} \\
& \quad \quad \text{by } (\text{auto simp: valStampsRcvd-def LV-CHOMachine-def} \\
& \quad \quad \text{HOrcvdMsgs-def LV-sendMsg-def send0-def}) \\
& \quad \text{hence } \text{card } (\text{valStampsRcvd } (\text{HOrcvdMsgs } \text{LV-M n } p \text{ HO } n \rho) (\rho n)) \\
& \quad \quad \leq \text{card } \{ q . \text{coord}(\rho n q) = p \} \\
& \quad \quad \text{by } (\text{auto intro: card-mono}) \\
& \text{moreover} \\
& \text{note } stp stp' \text{ run} \\
& \text{ultimately} \\
& \text{show } ?R (\text{Suc n}) \text{ by } (\text{auto simp: notStep3EqualCoord}) \\
& \quad \text{with } cmt \text{ show } ?\text{thesis} \text{ by } (\text{intro conds, auto}) \\
& \quad \text{qed} \\
& \text{— the remaining cases are all solved by expanding the definitions} \\
& \quad \text{qed } (\text{auto simp: LV-CHOMachine-def LV-initState-def next1-def next2-def} \\
& \quad \quad \text{next3-def notStep3EqualCoord[OF run]}) \\
& \text{with } cmt \text{ show } ?\text{thesis} \text{ by } (\text{intro conds, auto}) \\
& \quad \text{qed} \\
\end{align*}
\]

A process has a current timestamp only if:

- it is at step 2 or beyond,
- its coordinator has committed,
- its \text{x} value is the \text{vote} of its coordinator.

\textbf{lemma} \text{currentTimestampE};
\textbf{assumes} run: \text{CHORun \text{LV-M rho HO } coords} \\
\textbf{and} ts: \text{timestamp } (\rho \text{ r } p) = \text{Suc } (\text{phase } r) \\
\textbf{and} conds: 2 \leq \text{step } r; \\
\quad \text{commit } (\rho \text{ r } (\text{coord}(\rho \text{ r } p))); \\
\quad x (\rho \text{ r } p) = \text{the } (\text{vote } (\rho \text{ r } (\text{coord}(\rho \text{ r } p)))) \\
\textbf{shows} A \\
\textbf{proof} – \\
\text{let } ?\text{ts n } = \text{timestamp } (\rho \text{ n } p) \\
\text{let } ?\text{crd n } = \text{coord}(\rho \text{ n } p) \\
\text{have } ?\text{ts r } = \text{Suc } (\text{phase } r) \\
\quad 2 \leq \text{step } r 
\]

75
\( \land \text{commit}(\rho r (\text{?crd } r)) \) 
\( \land x(\rho r p) = \text{the}(\rho r (\text{?crd } r)) \) 
(\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) (\text{coords } 0 p) \text{ thus } ?Q p 0

by (\text{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** \text{n}

**assume** \text{stp'}: \text{step}(\text{Suc } n) = 1

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

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

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

**next**

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

**fix** \text{n}

**assume** \text{stp}: \text{step } n = 1 \text{ and } \text{stp'}: \text{step}(\text{Suc } n) = 2

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

\text{and nxt: next1 } n p (\text{HOrcvdMsgs LV-M } n p (\text{HOs } n p) (\rho n))

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

**show** ?Q p (\text{Suc } n)

**proof**

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

from \text{run stp LV-timestamp-bounded}[\text{where } n=n]

**have** ?ts n \leq \text{phase } n \text{ by } \text{auto}

**moreover**

from \text{run stp}

**have** \text{vote } (\rho (\text{Suc } n) (\text{?crd } (\text{Suc } n))) = \text{vote } (\rho n (\text{?crd } n))

by (\text{auto simp: notStep3EqualCoord notStep0EqualVote})

**moreover**

from \text{run stp}

**have** \text{commit } (\rho (\text{Suc } n) (\text{?crd } (\text{Suc } n))) = \text{commit } (\rho n (\text{?crd } n))

by (\text{auto simp: notStep3EqualCoord notStep03EqualCommit})

**moreover**

**note** \text{ts nxt stp stp'} \text{ ph}

**ultimately**

**show** ?R (\text{Suc } n)

by (\text{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** \text{n}
assume stp: \( \text{step } n = 2 \) \text{ and } stp': \( \text{step } (\text{Suc } n) = 3 \)
and ph: \( \text{phase } (\text{Suc } n) = \text{phase } n \)
and ih: \( ?Q \ p \ n \)
and nxt: \( \text{next}_2 n \ p \ (\rho n) \ (\text{HOrcvdMsgs } \text{LV-M } n \ p \ (\text{HOs } n \ p) \ (\rho n)) \)
(coords \( (\text{Suc } n) \ p \) \( (\rho n) \))
show \( ?Q \ p \ (\text{Suc } n) \)
proof
assume ts: \( ?ts \ (\text{Suc } n) = \text{Suc } (\text{phase } (\text{Suc } n)) \)
from run stp
have vt: \( \text{vote } (\rho n) \ (\text{(?crd } (\text{Suc } n))) = \text{vote } (\rho n) \ (\text{(?crd } n)) \)
by (auto simp add: notStep3EqualCoord notStep0EqualVote)
from run stp
have cmt: \( \text{commit } (\rho n) \ (\text{(?crd } (\text{Suc } n))) = \text{commit } (\rho n) \ (\text{(?crd } n)) \)
by (auto simp add: notStep3EqualCoord notStep03EqualCommit)
with vt ts ph stp stp' ih nxt
show \( ?R \ (\text{Suc } n) \)
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 stp': \( \text{step } (\text{Suc } n) = 0 \)
with run \text{LV-timestamp-bounded}[\text{where } n=\text{Suc } n]
have \( ?ts \ (\text{Suc } n) \leq \text{phase } (\text{Suc } n) \) \text{ by auto}
thus \( ?Q \ p \ (\text{Suc } n) \) \text{ by simp}
qed
with ts show \( ?\text{thesis} \) \text{ by (intro conds) auto}
qed

If a process \( p \) has its \text{ready} bit set then:

\begin{itemize}
    \item it is at step 3,
    \item it considers itself to be the coordinator of that phase and
    \item a majority of processes considers \( p \) to be the coordinator and has a current timestamp.
\end{itemize}

\textbf{lemma} \textit{readyE}:
\begin{itemize}
    \item \textbf{assumes} \text{run}: \text{CHORun } \text{LV-M } \rho \text{HOs coords and } \text{rdy}: \text{ready } (\rho r \ p)
    \item \text{and } conds: \[ \text{step } r = 3; \ \text{coord}\Phi \ (\rho r \ p) = p; \]
    \text{card} \{ q . \text{coord}\Phi \ (\rho r \ q) = p \}
    \land \ \text{timestamp } (\rho r \ q) = \text{Suc } (\text{phase } r) \} > N \div 2
    \item \text{shows } P \]
    \item \text{proof – }
    \text{let } ?qs n = \{ q . \text{coord}\Phi \ (\rho n \ q) = p \}
    \land \ \text{timestamp } (\rho n \ q) = \text{Suc } (\text{phase } n) \}
\end{itemize}
have \( \text{ready} \ (\rho \ r \ p) \rightarrow \)

\[
\begin{align*}
& \text{step } r = 3 \\
& \land \ \text{coord} \Phi \ (\rho \ r \ p) = p \\
& \land \ \text{card} \ (\text{?qs } r) > N \ \text{div} \ 2
\end{align*}
\]

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

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

— the interesting case is step 2

fix \( n \)

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

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

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

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

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

proof

assume \( \text{rdy} \): \text{ready } (\rho \ (\text{Suc } n) \ p)

from \( \text{stp} \ \text{ih} \) have \( \text{nrdy} \): \( \neg \ \text{ready } (\rho \ n \ p) \) by simp

with \( \text{rdy} \ \text{nxt} \) have \( \text{coord} \Phi \ (\rho \ n \ p) = p \\
\) by (auto simp: next2-def)

with \( \text{run} \ \text{stp} \) have \( \text{coord} : \text{coord} \Phi \ (\rho \ (\text{Suc } n) \ p) = p \\
\) by (simp add: notStep3EqualCoord)

let \( \text{?acks} = \text{acksRcvd} \ (\text{HOrcvdMsgs } \text{LV-M } n \ p \ (\text{HOs } n \ p) \ (\rho \ n))

from \( \text{nrdy} \ \text{rdy} \ \text{nxt} \) have \( \text{aRcvd} \): \( \text{card } \text{?acks} > N \ \text{div} \ 2 \\
\) by (auto simp: next2-def)

have \( \text{?acks} \subseteq \text{?qs } (\text{Suc } n) \)

proof (clarify)

fix \( q \)

assume \( q : q \in \text{?acks} \)

with \( \text{stp} \)

have \( n: \text{coord} \Phi \ (\rho \ n \ q) = p \land \text{timestamp } (\rho \ n \ q) = \text{Suc } (\text{phase } n) \\
\) by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def

\quad \text{acksRcvd-def send2-def isAck-def})

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

show \( \text{coord} : \text{coord} \Phi \ (\rho \ (\text{Suc } n) \ q) = p \\
\quad \land \text{timestamp } (\rho \ (\text{Suc } n) \ q) = \text{Suc } (\text{phase } (\text{Suc } n)) \\
\) by (simp add: notStep3EqualCoord notStep1EqualTimestamp)

qed

hence \( \text{?acks} \leq \text{?qs } (\text{Suc } n) \)

by (intro card-mono) auto

with \( \text{stp'} \ \text{coord} \ \text{aRcvd} \) show \( \text{?R } p \ (\text{Suc } n) \)

by auto

qed

— the remaining steps are all solved trivially

qed (auto simp: LV-CHOMachine-def LV-initState-def

\quad \text{next0-def next1-def next3-def})

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

A process decides only if the following conditions hold:

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

**lemma** decisionE:

**assumes** run: CHORun LV-M rho HOs coords
and dec: decide (rho (Suc r) p) ≠ decide (rho r p)
and conds: []
  step r = 3;
  decide (rho (Suc r) p) = Some (the (vote (rho r (coordΦ (rho r p)))));
  ready (rho r (coordΦ (rho r p))); commit (rho r (coordΦ (rho r p)))

[] ⇨ P

**shows** P

**proof** —

let ?cfg = rho r
let ?cfg' = rho (Suc r)
let ?crd p = coordΦ (?cfg p)
let ?dec' = decide (?cfg' p)

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

**have 1:** step r = 3
  ∧ ?dec' = Some (the (vote (?cfg (?crd p))))
  ∧ ready (?cfg (?crd p))
(is ?Q p r)

**proof** (rule LV-Suc[of run, where P=??])
— for step 3, we prove the thesis by expanding the relevant definitions

**assume** next3 r p (?cfg p) (HOrcvdMsgs LV-M r p (HOr r p) ?cfg)
  (coords (Suc r) p) (?cfg' p)

  and step r = 3

  with dec show ?thesis
      by (auto simp: next3-def send3-def isVote-def LV-CHOMachine-def
       HOrcvdMsgs-def LV-sendMsg-def)

**next**
— the other steps don’t change the decision

**assume** next0 r p (?cfg p) (HOrcvdMsgs LV-M r p (HOr r p) ?cfg)
  (coords (Suc r) p) (?cfg' p)

  with dec show ?thesis by (auto simp: next0-def)

**next**

**assume** next1 r p (?cfg p) (HOrcvdMsgs LV-M r p (HOr r p) ?cfg)
  (coords (Suc r) p) (?cfg' p)

  with dec show ?thesis by (auto simp: next1-def)

**next**

**assume** next2 r p (?cfg p) (HOrcvdMsgs LV-M r p (HOr r p) ?cfg)
  (coords (Suc r) p) (?cfg' p)

  with dec show ?thesis by (auto simp: next2-def)

**qed**

**hence** ready (?cfg (?crd p)) by blast
Because the coordinator is ready, there is a majority of processes that consider it to be the coordinator and that have a current timestamp.

with run
have card \{ q . \$\text{crd} q = \$\text{crd} p \land \text{timestamp} (\$\text{cfg} q) = \text{Suc} (\text{phase} r)\} > N \div 2 \text{ by (rule readyE)}
— Hence there is at least one such process . . .

hence card \{ q . \$\text{crd} q = \$\text{crd} p \land \text{timestamp} (\$\text{cfg} q) = \text{Suc} (\text{phase} r)\} \neq 0
by arith
then obtain q where \$\text{crd} q = \$\text{crd} p \text{ and timestamp} (\$\text{cfg} q) = \text{Suc} (\text{phase} r)
by auto
— . . . and by a previous lemma the coordinator must have committed.

with run have \text{commit} (\$\text{cfg} (\$\text{crd} p))
by (auto elim: currentTimestampE)
with I show ?thesis by (blast intro: conds)
qed

7.7 Proof of Integrity

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

**lemma** lv-integrityInvariant:

assumes run: \$\text{CHORun} LV-M \rho HOs coords

and inv: \[
\begin{align*}
\text{range} (x \circ (\rho n)) \subseteq \text{range} (x \circ (\rho 0)); \\
\text{range} (\text{vote} \circ (\rho n)) \subseteq \{\text{None}\} \cup \text{Some} \cdot \text{range} (x \circ (\rho 0)); \\
\text{range} (\text{decide} \circ (\rho n)) \subseteq \{\text{None}\} \cup \text{Some} \cdot \text{range} (x \circ (\rho 0))
\end{align*}
\] \implies A

shows A

proof —

let \$x0 = \text{range} (x \circ (\rho 0))
let \$x0opt = \{\text{None}\} \cup \text{Some} \cdot \$x0

have range \(x \circ \rho n) \subseteq \$x0
∧ range (\text{vote} \circ (\rho n)) \subseteq \$x0opt
∧ range (\text{decide} \circ (\rho n)) \subseteq \$x0opt
(is \$\text{Inv n} is \$X n ∧ \$\text{Vote n} ∧ \$\text{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
from run
show \( x \ (\rho \ (\text{Suc} \ n) \ p) \in \text{range} \ (\lambda q. \ x \ (\rho \ 0 \ q)) \) (is \ ?P \ p \ n)

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

Proof of second conjunct

have \( \texttt{vt'}: \ ?\text{Vote} \ (\text{Suc} \ n) \)
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 \to 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 vt 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 vt 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 vt have vote (rho n p) \in ?x0opt

ultimately show ?thesis by auto

qed

with v show \( \exists q. v = x ( \rho_0 q ) \) by auto

qed

Proof of third conjunct

have \texttt{dec'}: \texttt{Decide \( Suc n \)}

proof (clarsimp simp: image-def)

fix \( p v \)

assume \texttt{v}: \( \texttt{decide} ( \rho_\( Suc n \) p) = \texttt{Some} v \)

show \( \exists q. v = x ( \rho_0 q ) \)

proof (cases \texttt{decide} ( \rho_\( Suc n \) p) = \texttt{decide} ( \rho_\( n \) p))

\begin{itemize}
  \item case \texttt{True}
    
    with \texttt{dec True v} show ?thesis by (auto simp: image-def)
  \item next

  case \texttt{False}
    
    let \( \texttt{?crd} = \texttt{coord} \_ \Phi ( \rho_\( n \) p) \)

    from \texttt{False} run have \( \texttt{decide} ( \rho_\( Suc n \) p) = \texttt{Some} ( \texttt{the} ( \texttt{vote} ( \rho_\( n \) ?crd))) \)

    \hspace{1em} \texttt{and} \( \texttt{commit} ( \rho_\( n \) ?crd) \)

    by (auto elim: decisionE)

    from \texttt{vt} have \( \texttt{vote} ( \rho_\( n \) ?crd) \in \texttt{x0\_opt} \)

    \hspace{1em} \texttt{by} (auto simp: image-def)

    from run \texttt{commit} have \( \texttt{vote} ( \rho_\( n \) ?crd) \neq \texttt{None} \)

    \hspace{1em} \texttt{by} (rule commitE)

    with \texttt{d' \_ vtc show ?thesis by auto}

  qed

  qed

  from \texttt{x' \_ vtc'} \texttt{dec' show ?thesis by simp}

  qed

  qed

  with \texttt{inv} show ?thesis by simp

  qed

Integrity now follows immediately.

theorem \texttt{lv-integrity}:

assumes \( \texttt{run} : \texttt{CHORun \_ LV\_M \_ rho \_ HOs \_ coords} \)

\hspace{1em} and \( \texttt{dec} : \texttt{decide} ( \rho_\( n \) p) = \texttt{Some} v \)

shows \( \exists q. v = x ( \rho_0 q ) \)

proof –

\begin{itemize}
  \item from \texttt{run} have \( \texttt{decide} ( \rho_\( n \) p) \in \{ \texttt{None} \} \cup \{ \texttt{Some} \_ \_ ( \texttt{range} ( x \circ ( \rho_0 ))) \} \)

    \hspace{1em} \texttt{by} (rule \texttt{lv-integrity\_Invariant}) (auto simp: image-def)

  \item with \texttt{dec} show ?thesis by (auto simp: image-def)

  qed

83
7.8 Proof of Agreement and Irrevocability

The following lemmas closely follow a hand proof provided by Bernadette Charron-Bost.

If a process decides, then a majority of processes have a current timestamp.

**Lemma** decisionThenMajorityBeyondTS:

- **assumes** run: CHORun LV-M rho HOs coords
- and dec: decide (rho (Suc r) p) ≠ decide (rho r p)
- **shows** card (procsBeyondTS (Suc (phase r)) (rho r)) > N div 2

**using** run dec **proof** (rule decisionE)

Lemma decisionE tells us that we are at step 3 and that the coordinator is ready.

Let \( \?crd = \text{coord} \Phi (\rho r p) \)

Let \( \?qs = \{ q . \text{coord} \Phi (\rho r q) = \?crd \land \text{timestamp} (\rho r q) = \text{Suc} (\text{phase } r) \} \)

**assume** stp: step r = 3 and rdty: ready (rho r ?crd)

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

**from** run rdty **have** card \( \?qs > N \text{ div } 2 \) **by** (rule readyE)

**moreover**

**from** run (OF run, where \( n=r \)) **have** \( \forall q. \text{timestamp} (\rho r q) \leq \text{Suc} (\text{phase } r) \) **by** auto

**hence** \( \?qs \subseteq \text{procsBeyondTS} (\text{Suc} (\text{phase } r)) (\rho r) \)

**by** (auto simp: procsBeyondTS-def)

**hence** card \( \?qs \leq \text{card} \ (\text{procsBeyondTS} (\text{Suc} (\text{phase } r)) (\rho r)) \)

**by** (intro card-mono) auto

ultimately **show** \(?thesis \) **by** simp

qed

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

**Lemma** committedProcsEqual:

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

**proof** –

**from** run cmt have card \( \{ q . \text{coord} \Phi (\rho r q) = p \} > N \text{ div } 2 \)

**by** (blast elim: commitE)

**moreover**

**from** 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

qed

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

**Lemma** readyProcsEqual:
assumes run: CHORun LV-M rho HOs coords
and rdy: ready (rho r p) and rdy’: ready (rho r p’)
shows p = p’

proof —
let \( ?C p = \{ q . \text{coord}(\rho r q) = p \land \text{timestamp}(\rho r q) = \text{Suc}(\text{phase} r) \} \)
from run rdy have card (?C p) > N div 2
  by (blast elim: readyE)
moreover
from run rdy’ have card (?C p’) > N div 2
  by (blast elim: readyE)
ultimately
obtain q where \( \text{coord}(\rho r q) = p \) and \( p’ = \text{coord}(\rho r q) \)
  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: commit (rho r p)
shows \( \exists q \in \text{procsBeyondTS ts} (\rho r) . \text{vote}(\rho r p) = \text{Some}(x (\rho r q)) \)
(is ?Q r)

proof —
let \( ?bynd n = \text{procsBeyondTS ts} (\rho n) \)
have card (?bynd r) > N div 2 \land commit (\rho r p) \implies ?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: \( \forall q . \text{next0} n q (\rho n q) \)
  (HOrcvdMsgs LV-M n q (HOs n q) (\rho n))
  (coords (Suc n) q)
  (\rho (Suc n) q)
  (is \( \forall q . \text{?nxt} q) \)
from next have nexp: ?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)
show ?Q (Suc n)
proof —
let \( ?msgs = \text{HOrcvdMsgs} LV-M n p (HOs n p) (\rho n) \)
from stp run have \( \neg \text{commit} (\rho n p) \) by (auto elim: commitE)
with nexp ct obtain q v where
  v: \( ?msgs q = \text{Some} (\text{ValStamp} v (\text{highestStampRcvd} ?msgs)) \) and
\text{vote: } \text{vote } (\rho (\text{Suc } n) p) = \text{Some } v \text{ and } \\
\text{rcvd: } \text{card } (\text{valStampsRecvd } ? \text{msgs}) > N \text{ div } 2 \\
\text{by } (\text{auto simp: next0-def}) \\
\text{from } mj \text{ rcvd obtain } q' \text{ where } \\
q_1': q' \in ?bynd (\text{Suc } n) \text{ and } q_2': q' \in \text{valStampsRecvd } ? \text{msgs} \\
\text{by } (\text{rule majoritiesE'}) \\
\text{have timestamp } (\rho n q') \leq \text{timestamp } (\rho n q) \\
\text{proof —} \\
\text{from } q_2' \text{ obtain } v' ts' \\
\text{where } ts': ? \text{msgs } q' = \text{Some } (\text{ValStamp } v' ts') \\
\text{by } (\text{auto simp: valStampsRecvd-def}) \\
\text{hence } ts' \leq \text{highestStampRecvd } ? \text{msgs} \\
\text{by } (\text{rule highestStampRecvd-max}) \\
\text{moreover} \\
\text{from } ts' \text{ stp have timestamp } (\rho n q') = ts' \\
\text{by } (\text{auto simp: LV-CHOMachine-def HOrcvdMsgs-def} \\
\text{LV-sendMsg-def send0-def}) \\
\text{moreover} \\
\text{from } v \text{ stp have timestamp } (\rho n q) = \text{highestStampRecvd } ? \text{msgs} \\
\text{by } (\text{auto simp: LV-CHOMachine-def HOrcvdMsgs-def} \\
\text{LV-sendMsg-def send0-def}) \\
\text{ultimately} \\
\text{show } ? \text{thesis by simp} \\
\text{qed} \\
\text{moreover} \\
\text{from } \text{run stp} \\
\text{have timestamp } (\rho (\text{Suc } n) q') = \text{timestamp } (\rho n q') \\
\text{by } (\text{simp add: notStep1EqualTimestamp}) \\
\text{moreover} \\
\text{from } \text{run stp} \\
\text{have timestamp } (\rho (\text{Suc } n) q) = \text{timestamp } (\rho n q) \\
\text{by } (\text{simp add: notStep1EqualTimestamp}) \\
\text{moreover} \\
\text{note } q_1' \\
\text{ultimately} \\
\text{have } q \in ?bynd (\text{Suc } n) \\
\text{by } (\text{simp add: procsBeyondTS-def}) \\
\text{moreover} \\
\text{from } v \text{ vote stp} \\
\text{have } \text{vote } (\rho (\text{Suc } n) p) = \text{Some } (x (\rho n q)) \\
\text{by } (\text{auto simp: LV-CHOMachine-def HOrcvdMsgs-def} \\
\text{LV-sendMsg-def send0-def}) \\
\text{moreover} \\
\text{from } \text{run stp have } x (\rho (\text{Suc } n) q) = x (\rho n q) \\
\text{by } (\text{simp add: notStep1EqualX}) \\
\text{ultimately} \\
\text{show } ? \text{thesis by force} \\
\text{qed} \\
\text{qed}
We now prove that \( \text{next1} \) preserves the property. Observe that \( \text{next1} \) 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.

\[
\text{fix } n \\
\text{assume } \text{stp} : \text{step } n = 1 \\
\text{and } \text{nxt} : \forall q. \text{next1 } n \ q \quad (\text{HOrcvdMsgs} \ \text{LV-M } n \ q \ (\text{HOs} \ n \ q) \ (\rhoo n)) \\
\text{coords} \ (\text{Suc} \ n) \ q \\
\rhoo (\text{Suc} \ n) \ q)
\]

\( (\text{is } \forall q. \text{?nxt } q) \)

\text{and } \text{ih} : ?P \ p \ n

\text{from } \text{nxt } \text{have } \text{nxp} : ?\text{nxt } p \ ..

\text{show } ?P \ p \ (\text{Suc} \ n)

\text{proof} (\text{clarify})

\text{assume } \text{mj} : \text{card} (\text{?bynd} (\text{Suc} \ n)) > N \text { div 2}

\text{and } \text{ct} : \text{commit} \ (\rhoo (\text{Suc} \ n) \ p)

\text{from } \text{run } \text{stp } \text{ct'} \text{ have } \text{ct} : \text{commit} \ (\rhoo (\text{Suc} \ n) \ p)

\text{by} (\text{simp } \text{add} : \text{notStep03EqualCommit})

\text{from } \text{run } \text{stp} \text{ have } \text{vote} : \text{vote} (\rhoo (\text{Suc} \ n) \ p) = \text{vote} (\rhoo (\text{Suc} \ n) \ p)

\text{by} (\text{simp } \text{add} : \text{notStep0EqualVote})

\text{show } ?Q \ (\text{Suc} \ n)

\text{proof} (\text{cases } \exists q \in \text{?bynd} (\text{Suc} \ n). \rhoo (\text{Suc} \ n) \ q \neq \rhoo (\text{Suc} \ n) q)

\text{case } True

in this case the property holds because \( q \) updates its \( x \) field to the vote

\text{then obtain } q \text{ where}

\( q1 : q \in \text{?bynd} (\text{Suc} \ n) \text{ and } q2 : \rhoo (\text{Suc} \ n) q \neq \rhoo (\text{Suc} \ n) q \ ..

\text{from } \text{nxt } \text{have } ?\text{nxt } q \ ..

\text{with } \text{q2 } \text{stp}

\text{have } x : x (\rhoo (\text{Suc} \ n) q) = \text{the} (\text{vote} (\rhoo (\text{Suc} \ n) (\text{coord} \Phi (\rhoo (\text{Suc} \ n) q))))

\text{and } \text{coord} : \text{commit} (\rhoo (\text{Suc} \ n) (\text{coord} \Phi (\rhoo (\text{Suc} \ n) q)))

\text{by} (\text{auto } \text{simp} : \text{next1-def send1-def LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def isVote-def})

\text{from } \text{run } \text{ct} \text{ have } \text{vote} : \text{vote} (\rhoo (\text{Suc} \ n) \ p) \neq \text{None}

\text{by} (\text{rule } \text{commitE})

\text{from } \text{run } \text{coord } \text{ct} \text{ have } \text{coord} \Phi (\rhoo (\text{Suc} \ n) q) = p

\text{by} (\text{rule } \text{committedProcEqual})

\text{with } \text{q1 } x' \text{ vote } \text{vote} \text{ show } ?\text{thesis by auto}

\text{next}

\text{case } False

if no relevant process moves then \( \text{procsBeyondTS} \) doesn’t change and we invoke the induction hypothesis

\text{hence } \text{bynd} : \text{?bynd} (\text{Suc} \ n) = \text{?bynd} \ n
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
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: notStep0EqualVote)
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 commt 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 XOfTimestampBeyondDecision:**

** Assumes** run: CHORun LV-M rho HOs coords  
and dec: decide (rho (Suc r) p) \( \neq \) decide (rho r p)  
** Shows** \( \forall q \in \text{procsBeyondTS} (\text{Suc} (\text{phase} r)) (\text{rho} (r+k)). \)  
\( x (\text{rho} (r+k) q) = \text{the} (\text{decide} (\text{rho} (\text{Suc} r) p)) \)  
(is \( \forall q \in \text{?bynd} k. \) = ?v is ?P p k)  

**Proof** (induct \( k \))  
— base step  
show ?P p 0  
proof (clarify)  
fix q  
assume q: q \( \in \) ?bynd 0  
use preceding lemmas about the decision value and the \( x \) field of processes with fresh timestamps  
from run dec  
  have stp: step r = 3  
  and v: decide (rho (Suc r) p) \( = \) Some (the (vote (rho r (coord\( \Phi (\text{rho} r p)))))  
  and cmt: commt (rho r (coord\( \Phi (\text{rho} r p)))))  
  by (auto elim: decisionE)  
from stp LV-timestamp-bounded[OF run, where n=r]  
  have timestamp (rho r q) \( \leq \) Suc (phase r) by simp  
  with q have timestamp (rho r q) = Suc (phase r)  
  by (simp add: procsBeyondTS-def)  
with run  
  have x: x (rho (r+0) q) = the (vote (rho r (coord\( \Phi (\text{rho} r q)))))  
  and cmt': commt (rho r (coord\( \Phi (\text{rho} r q))))  
  by (auto elim: currentTimestampE)  
from run cmt cmt' have coord\( \Phi (\text{rho} r p) = \text{coord\( \Phi (\text{rho} r q))})  
  by (rule committedProcsEqual)  
with x v show x (rho (r+0) q) = ?v by simp  

ded  

next  
— induction step  
fix k  
assume ih: ?P p k  
show ?P p (Suc k)  
proof (clarify)  
fix q  
assume q: q \( \in \) ?bynd (Suc k)  
— distinguish the kind of transition—only step1 is interesting  
have x (rho (Suc (r + k)) q) = ?v (is ?X q (r+k))  
proof (rule LV-Suc[OF run, where P=?X])  
  assume stp: step (r + k) = 1
and nxt: next1 (r+k) q (rho (r+k) q)
   (HOrcvdMsgs LV-M (r+k) q (HOs (r+k) q) (rho (r+k)))
   (coords (Suc (r+k)) q)
   (rho (Suc (r+k)) q)

show ?thesis
proof (cases rho (Suc (r+k)) q = rho (r+k) q)
case True
  with q ih show ?thesis by (auto simp: procsBeyondTS-def)
next

  case False
  from run dec have card (?bynd 0) > N div 2
    by (simp add: decisionThenMajorityBeyondTS)
  moreover
  have ?bynd 0 ⊆ ?bynd k
    by (auto elim: procsBeyondTS-monotonic[OF run])
  hence card (?bynd 0) ≤ card (?bynd k)
    by (auto intro: card-mono)
  ultimately
  have maj: card (?bynd k) > N div 2 by simp
  let ?crd = coordΦ (rho (r+k) q)
  from False stp nxt have
cmnt: commit (rho (r+k) ?crd) and
  x: x (rho (Suc (r+k)) q) = the (vote (rho (r+k) ?crd))
  by (auto simp: next1-def LV-CHOMachine-def HOrcvdMsgs-def
    LV-sendMsg-def send1-def isVote-def)

  from run maj cmnt stp obtain q'
    where q1': q' ∈ ?bynd k
    and q2': vote (rho (r+k) ?crd) = Some (x (rho (r+k) q'))
  by (blast dest: commitThenVoteRecent)
  with x ih show ?thesis by auto
qed
next
— all other steps hold by induction hypothesis
assume step (r+k) = 0
with run have x: x (rho (Suc (r+k)) q) = x (rho (r+k) q)
  and ts: timestamp (rho (Suc (r+k)) q) = timestamp (rho (r+k) q)
  by (auto simp: notStep1EqualX notStep1EqualTimestamp)
from ts q have q ∈ ?bynd k
  by (auto simp: procsBeyondTS-def)
with x ih show ?thesis by auto
next
assume step (r+k) = 2
with run have x: x (rho (Suc (r+k)) q) = x (rho (r+k) q)
  and ts: timestamp (rho (Suc (r+k)) q) = timestamp (rho (r+k) q)
  by (auto simp: notStep1EqualX notStep1EqualTimestamp)
from ts q have q ∈ ?bynd k
  by (auto simp: procsBeyondTS-def)
with x ih show ?thesis by auto
next
assume step \(r+k = 3\)

with run have: \(x : (\rho (Suc (r+k)) q) = x (\rho (r+k) q)\)
    and ts: timestamp \((\rho (Suc (r+k)) q) = timestamp (\rho (r+k) q)\)
by (auto simp: notStep1EqualX notStep1EqualTimestamp)
from ts q have \(q \in ?bynd k\)
by (auto simp: procsBeyondTS-def)
with \(x\) \(ih\) show \(?\)thesis by auto
qed

thus \(x (\rho (r + Suc k) q) = ?v\) by simp
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** \(\text{laterProcessDecidesSameValue} :\)

assumes run: \(\text{CHO\text{-}Run LV\text{-}M \rho HOs coords}\)
and p: decide \((\rho (Suc r) p) \neq \text{decide} (\rho r p)\)
and q: decide \((\rho (Suc (r+k)) q) \neq \text{decide} (\rho (r+k) q)\)
shows decide \((\rho (Suc (r+k)) q) = \text{decide} (\rho (Suc r) p)\)

**Proof**

let \(?bynd k = \text{procsBeyondTS} (Suc (\text{phase r})) (\rho (r+k))\)
let \(?qcrd = \text{coord}\Phi (\rho (r+k) q)\)
from run p have notNone: \(\text{decide} (\rho (Suc r) p) \neq \text{None}\)
  by (auto elim: decisionE)
  — process \(q\) decides on the vote of its coordinator
from run q have dec: \(\text{decide} (\rho (Suc (r+k)) q) = \text{Some} (\text{the} (\text{vote} (\rho (r+k) ?qcrd)))\)
  and cmt: \(\text{commt} (\rho (r+k) ?qcrd)\)
  by (auto elim: decisionE)
  — that vote is the \(x\) field of some process \(q'\) with a recent timestamp
from run p have card \((?bynd 0) > N div 2\)
  by (simp add: decisionThenMajorityBeyondTS)
moreover
from run have \(?bynd 0 \subseteq ?bynd k\)
  by (auto elim: procsBeyondTS-monotonic)
hence card \((?bynd 0) \leq \text{card} (?bynd k)\)
  by (auto intro: card-mono)
ultimately
have maj: \(\text{card} (?bynd k) > N \text{ div 2}\) by simp
from 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 run p q'1 have \(x (\rho (r+k) q') = \text{the} (\text{decide} (\rho (Suc r) p))\)
  by (auto dest: XOfTimestampBeyondDecision)
  — which proves the assertion
A process that holds some decision \( v \) has decided \( v \) sometime in the past.

**lemma** decisionNonNullThenDecided:

**assumes** \( \text{run: CHORun LV-M rho HOs coords} \)
and \( \text{dec: decide (rho n p) = Some v} \)

**shows** \( \exists m < n. \) decide (rho (Suc m) p) \( \neq \) decide (rho m p)
\( \land \) decide (rho (Suc m) p) = Some v

**proof**

- let \( \?\text{dec k = decide (rho k p)} \)
- have \((\forall m < n. \?\text{dec (Suc m) \neq \?\text{dec m} \rightarrow \?\text{dec (Suc m) \neq Some v}}) \)
- \( \rightarrow \?\text{dec n} \neq \?\text{Some v} \)
- \( \text{(is \?P n is \?A n \rightarrow -)} \)

**proof** (induct \( n \))

- from \( \text{run show \?P 0} \)
  - by (auto simp: CHORun-eq LV-CHOMachine-def
  \text{CHOinitConfig-def LV-initState-def})

next

- fix \( n \)
- assume \( \text{ih: \?P n} \)
- show \( \?P (\text{Suc n}) \)

**proof** (clarify)

- assume \( p: \?A (\text{Suc n}) \) and \( v: \?\text{dec (Suc n) = Some v} \)
- from \( p \) have \( \?A n \) by simp
- with \( \text{ih} \) have \( \?\text{dec n} \neq \?\text{Some v} \) by simp
- moreover
- from \( p \)
- have \( \?\text{dec (Suc n) \neq \?\text{dec n} \rightarrow \?\text{dec (Suc n) \neq Some v} \) by simp
- ultimately
- have \( \?\text{dec (Suc n) \neq Some v} \) by auto
- with \( v \) show False by simp

**qed**

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

**qed**

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

**theorem** lv-irrevocability:

**assumes** \( \text{run: CHORun LV-M rho HOs coords} \)
and \( \text{p: decide (rho m p) = Some v} \)

**shows** decide (rho (m+k) p) = Some v

**proof**

- from \( \text{run p obtain n where} \)
  - \( n1: n < m \) 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)

92
have $\forall i. \text{decide} (\rho (\text{Suc} (n+i)) p) = \text{Some } v$ (is $\forall i. \text{decide} i$)

proof
  fix $i$
  show $\text{decide} i$
  proof (induct $i$)
      from $n3$ show $\text{decide } 0$ by simp
  next
  fix $j$
  assume $ih: \text{decide } j$
  show $\text{decide } (\text{Suc } j)$
  proof (rule ccontr)
      assume $ctr: \neg (\text{decide } (\text{Suc } j))$
      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 $\text{run } n2$
      have $\text{decide} (\rho (\text{Suc} (n + \text{Suc } j)) p) = \text{decide} (\rho (\text{Suc } n)) p)$
        by (rule laterProcessDecidesSameValue)
      with $ctr \text{ n3}$ show False by simp
  qed
  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 $\text{run}\colon \text{CHORun LV-M } \rho \text{HOs coords}$
    and $p: \text{decide} (\rho \text{ m p}) = \text{Some } v$
    and $q: \text{decide} (\rho \text{ n q}) = \text{Some } w$
  shows $v = w$
  proof --
    from $\text{run } p$ obtain $k$
      where $k1: \text{decide} (\rho (\text{Suc } k) p) \neq \text{decide} (\rho \text{ k p})$
        and $k2: \text{decide} (\rho (\text{Suc } k) p) = \text{Some } v$
        by (auto dest: decisionNonNullThenDecided)
    from $\text{run } q$ obtain $l$
      where $l1: \text{decide} (\rho (\text{Suc } l) q) \neq \text{decide} (\rho \text{ l q})$
        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 $\text{run } k1 \text{ l1 } m$
      have $\text{decide} (\rho (\text{Suc } l) q) = \text{decide} (\rho (\text{Suc } k) p)$
      qed
by (auto elim: laterProcessDecidesSameValue)
with k2 \text{ l2} show \text{ thesis} by simp
next
case False
hence l \leq k by simp
then obtain m where m: k = l + m by (auto simp: le_iff_add)
from run l1 k1 m have decide (\rho \text{ Suc k}) p = decide (\rho \text{ Suc l}) q
by (auto elim: laterProcessDecidesSameValue)
with l2 k2 show \text{ thesis} by simp
qed

7.9 Proof of Termination

The proof of termination relies on the communication predicate, which specifies 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.

\textbf{theorem} \texttt{lv-termination}: 
\begin{itemize}
  \item \textbf{assumes} \texttt{run: CHORun LV-M \rho HOs coords}
  \item \textbf{and} \texttt{commG: CHOcommGlobal LV-M HOs coords}
\end{itemize}
\textbf{shows} \exists r. \forall p. \texttt{decide (\rho r p) \neq None}

\textbf{proof} –

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

from \texttt{commG obtain} \texttt{ph c}
\begin{itemize}
  \item \texttt{where c: }\forall p. \texttt{coords (4*ph) p = c}
  \item \texttt{and maj0: card (HOs (4*ph) c) > N div 2}
  \item \texttt{and maj2: card (HOs (4*ph+1) c) > N div 2}
  \item \texttt{and rcv1: }\forall p. c \in \texttt{HOs (4*ph+1) p}
  \item \texttt{and rcv3: }\forall p. c \in \texttt{HOs (4*ph+3) p}
\end{itemize}
by (auto simp: LV-CHOMachine-def LV-commGlobal-def)

let ?r0 = 4*ph
let ?r1 = Suc ?r0
let ?r2 = Suc ?r1
let ?r3 = Suc ?r2
let ?r4 = Suc ?r3

Process $c$ is the coordinator of all steps of phase $ph$.

from \texttt{run c have} $c': \forall p. \texttt{coord} (\rho \texttt{ ?r p}) = c$
by (auto simp add: phase-def coordinators)
with \texttt{run have} $c1: \forall p. \texttt{coord} (\rho \texttt{ ?r1 p}) = c$
by (auto simp add: step-def mod-Suc notStep3EqualCoord)
with \texttt{run have} $c2: \forall p. \texttt{coord} (\rho \texttt{ ?r2 p}) = c$
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 c) (HOrcvdMsgs LV-M ?r c (HOs ?r 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)
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 = coordΦ (rho ?r3 p)
  from run 3 c3 crv3
  have cnd: ?msgs ?crd ≠ None ∧ 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) ≠ None (is ?P p (Suc (Suc (Suc (4*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 ∃ 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 ◦ (rho 0)) decide rho
proof –
  — the above statement of termination is stronger than what we need
from lv-termination[OF assms]
  obtain r where ∀ p. decide (rho r p) ≠ None ..
  hence ∀ p. ∃ r. decide (rho r p) ≠ 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 (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
  assume crun: CSHORun LV-M crun HOs HOs coords
and \( \text{init: crun 0 = state (rho 0)} \)

from \( \text{crun have CHORun LV-M crun HOs coords} \)

by \( \text{(unfold CHORun-def SHORun-def)} \)

from \( \text{this commG have consensus (x o (crun 0)) decide crun} \)

by \( \text{(rule lv-consensus)} \)

with \( \text{init show consensus ?inits decide crun} \)

by \( \text{(simp add: o-def)} \)

qed

end

theory UteDefs

imports ..../HOModel

begin

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

Algorithm \( \mathcal{U}_{T,E,\alpha} \) is presented in \cite{3}. It is an uncoordinated algorithm that tolerates value (a.k.a. Byzantine) faults, and can be understood as a variant of UniformVoting. 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 \cite{3}, using the framework of theory \( \text{HOModel} \).

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.

\textbf{typedecl Proc} — the set of processes

\textbf{axiomatization where} Proc-finite: OFCLASS(Proc, finite-class)

\textbf{instance} Proc :: finite \textbf{by} (rule Proc-finite)

\textbf{abbreviation}

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

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

\textbf{abbreviation}

\( nSteps \equiv 2 \)

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

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

\textbf{lemma} phase-zero [simp]: phase 0 = 0

\textbf{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: $2E \geq N + 2\alpha$
  and majT: $2T \geq N + 2\alpha$
  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**

\[
\text{send0} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow '\text{val msg}
\]

**where**

\[
\text{send0} \; r \; p \; q \; \text{st} \equiv \text{Val} \; (x \; \text{st})
\]

**definition**

\[
\text{next0} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow (\text{Proc} \Rightarrow '\text{val msg option}) \\
\Rightarrow '\text{val pstate} \Rightarrow \text{bool}
\]

**where**

\[
\text{next0} \; r \; p \; \text{st} \; \text{msgs} \; \text{st}' \equiv
\]

\[
(\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Val} \; v)\} > \alpha \land x \; \text{st}' = v) \\
\lor \neg(\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Val} \; v)\} > \alpha) \\
\land x \; \text{st}' = \text{undefined}
\]

\[
\land \neg(\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Val} \; v)\} > \alpha) \\
\land \text{decide} \; \text{st}' = \text{decide} \; \text{st}
\]

\[
\land \text{vote} \; \text{st}' = \text{None}
\]

In step 1, each process sends its current vote.

If it receives more than \(\alpha\) votes for a given value \(v\), it sets its \(x\) field to \(v\), else it sets \(x\) to a default value.

If the process receives more than \(E\) votes for \(v\), it decides \(v\), otherwise it leaves its decision unchanged.

**definition**

\[
\text{send1} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow '\text{val msg}
\]

**where**

\[
\text{send1} \; r \; p \; q \; \text{st} \equiv \text{Vote} \; (\text{vote} \; \text{st})
\]

**definition**

\[
\text{next1} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow (\text{Proc} \Rightarrow '\text{val msg option}) \\
\Rightarrow '\text{val pstate} \Rightarrow \text{bool}
\]

**where**

\[
\text{next1} \; r \; p \; \text{st} \; \text{msgs} \; \text{st}' \equiv
\]

\[
(\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Vote} \; (\text{Some} \; v))\} > \alpha \land x \; \text{st}' = v) \\
\lor \neg(\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Vote} \; (\text{Some} \; v))\} > \alpha) \\
\land x \; \text{st}' = \text{undefined}
\]

\[
\land (\exists\, v. \; \text{card} \; \{q. \; \text{msgs} \; q = \text{Some} \; (\text{Vote} \; (\text{Some} \; v))\} > \alpha) \\
\land \text{decide} \; \text{st}' = \text{decide} \; \text{st}
\]

\[
\land \text{vote} \; \text{st}' = \text{None}
\]

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

**definition**

\[
\text{Ute-sendMsg} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow '\text{val msg}
\]

**where**

\[
\text{Ute-sendMsg} \; (r::\text{nat}) \equiv \text{if} \; \text{step} \; r = 0 \; \text{then} \; \text{send0} \; r \; \text{else} \; \text{send1} \; r
\]

**definition**

\[
\text{Ute-nextState} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow '\text{val pstate} \Rightarrow (\text{Proc} \Rightarrow '\text{val msg option}) \\
\Rightarrow '\text{val pstate} \Rightarrow \text{bool}
\]

**where**

\[
\text{Ute-nextState} \; r \equiv \text{if} \; \text{step} \; r = 0 \; \text{then} \; \text{next0} \; r \; \text{else} \; \text{next1} \; r
\]
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** $Ute$-$commPerRd$ where

$Ute$-$commPerRd$ Ho$\varsigma$ SHo$\varsigma$ =

$\forall p. \text{card}(Ho$\varsigma$ p - SHo$\varsigma$ p) \leq \alpha$

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

$\land \text{card}(SHo$\varsigma$ p \cap Ho$\varsigma$ p) > T$

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

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

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

**definition** $Ute$-$commGlobal$ where

$Ute$-$commGlobal$ Ho$\varsigma$ SHo$\varsigma$ =

$\exists \Phi. (let r = Suc(nSteps*\Phi)$

$in (\exists \pi. \forall p. \pi = Ho$\varsigma$ r p \land \pi = SHo$\varsigma$ r p)$

$\land (\forall p. \text{card}(SHo$\varsigma$ (Suc r) p \cap Ho$\varsigma$ (Suc r) p) > T)$

$\land (\forall p. \text{card}(SHo$\varsigma$ (Suc (Suc r)) p \cap Ho$\varsigma$ (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** $Ute$-$SHOMachine$ where

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

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.

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

Processes can make a new decision only at second round of each phase.

\textbf{lemma decide-step:}
\textbf{assumes run: SHORun Ute-M rho HOs SHOs}
\textbf{and d1: decide (rho r p) \neq Some v}
\textbf{and d2: decide (rho (Suc r) p) = Some v}
\textbf{shows} \text{step r \neq 0}
\textbf{proof}
\textbf{assume} \text{sr:step r = 0}
\textbf{from} \text{run obtain} \text{\mu where} \text{Ute-nextState r p (rho r p) \mu (rho (Suc r) p)}
\textbf{unfolding} \text{Ute-SHOMachine-def nextState-def SHORun-eq SHOnextConfig-eq}
\textbf{by force}
\textbf{with} \text{sr have} \text{next0 r p (rho r p) \mu (rho (Suc r) p)}
\textbf{unfolding} \text{Ute-nextState-def by auto}
\textbf{hence} \text{decide (rho r p) = decide (rho (Suc r) p)}
\textbf{by (auto simp:next0-def)}

101
\end{verbatim}
with d1 d2 show False by auto

qed

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

qed

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

qed

lemma unique-majority-T:
assumes majv: card {qq::Proc. F qq = Some m} > T
and majw: card {qq::Proc. F qq = Some m'} > T
shows m = m'
proof -
from majT majv majw
have card {qq::Proc. F qq = Some m} > N div 2
  and card {qq::Proc. F qq = Some m'} > N div 2
  by auto
then obtain qq
  where qq ∈ {qq::Proc. F qq = Some m}
  and qq ∈ {qq::Proc. F qq = Some m'}
  by (rule majoritiesE')
thus !thesis by auto
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 nxtpq: 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 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} > T - α ∧ card {qq. sendMsg Ute-M r qq q (rho r qq) = Val vq} > T - α

(is card ?vrcvdp - ?ahop ⊆ ?usentp)

by (auto simp: SHOmsgVectors-def)

hence card ?usentp ≥ card ?vrcvdp - card ?ahop by auto

moreover

from nxtp stp have next0 r p = 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 ?vrcvdq - ?ahoq ⊆ ?usentq)

by (auto simp: SHOmsgVectors-def)

hence card ?usentq ≥ card ?vrcvdq - card ?ahoq by auto

moreover

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

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

with vp have card ?vrcvdp > 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 ?vrcvdq - ?ahoq ⊆ ?usentq)

by (auto simp: SHOmsgVectors-def)

hence card ?usentq ≥ card ?vrcvdq - card ?ahoq by auto

moreover

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

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

with vq have card {qq. mu q qq = Some (Val vq)} > T

103
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

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 $\text{Ute-SHOMachine-def}$ $\text{SHORun-eq}$ $\text{SHOnextConfig-eq}$ $\text{SHOmsgVectors-def}$ by blast

hence $\{qq. \mu p qq = \text{Some} (\text{Vote} (\text{Some} v))\} - (\text{HOs} r p - \text{SHOs} r p)$
    $\subseteq \{qq. \text{sendMsg Ute-M} r qq p (\rho r qq) = \text{Vote} (\text{Some} v)\}$
    (is $?vrcdp - {?ahop} \subseteq {?}sentp) by auto

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

hence $\text{card} {?}sentp \geq \text{card} {?vrcdp} - \text{card} {?ahop}$ by auto

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

moreover from run $d1 \ d2$ have $\text{step} r \neq 0$ by (rule $\text{decide-step}$)
   with $\text{next1}$ have $\text{card} \{qq. \mu p qq = \text{Some} (\text{Vote} (\text{Some} v))\} > E$
   unfolding $\text{next1-def}$ by auto

ultimately show $?thesis$ using $\text{alpha.lt-E}$ 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: $\text{SHOcommPerRd Ute-M} (\text{HOs} (\text{Suc} r)) (\text{SHOs} (\text{Suc} r))$
  and threshold: $\text{card} \{q. \text{sendMsg Ute-M} (\text{Suc} r) q p (\rho (\text{Suc} r) q) = \text{Vote} (\text{Some} v)\} > E - \alpha$
  (is $\text{card} ( {?msgs} p v) > -$)
  shows $\text{card} ( {?msgs} pp v \cap (\text{SHOs} (\text{Suc} r) pp \cap \text{HOs} (\text{Suc} r) pp)) > \alpha$
proof

  have $\forall q. \text{sendMsg Ute-M} (\text{Suc} r) q p (\rho (\text{Suc} r) q) = \text{sendMsg Ute-M} (\text{Suc} r) q pp (\rho (\text{Suc} r) q)$
    by (auto simp: $\text{Ute-SHOMachine-def}$ $\text{Ute-sendMsg-def}$ $\text{step-def}$ $\text{send0-def}$ $\text{send1-def}$)

  moreover from unsafe
  have $\text{card} (\text{SHOs} (\text{Suc} r) pp \cap \text{HOs} (\text{Suc} r) pp) > N + \alpha$
    by (auto simp: $\text{Ute-SHOMachine-def}$ $\text{step-def}$ $\text{send0-def}$ $\text{send1-def}$)

  ultimately show $?thesis$ using threshold by auto
  qed

moreover have $\text{card} ( {?msgs} pp v) + \text{card} (\text{SHOs} (\text{Suc} r) pp \cap \text{HOs} (\text{Suc} r) pp)$
  $= \text{card} ( {?msgs} pp v \cup (\text{SHOs} (\text{Suc} r) pp \cap \text{HOs} (\text{Suc} r) pp))$
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 2:**

**Assumptions:**
- \( \text{run} : \text{SHORun Ute-M rho HOs SHOs} \)
- \( \text{safe} : \forall r. \text{SHOcommPerRd Ute-M (HOs r) (SHOs r)} \)
- \( \text{nxtpp} : \text{nextState Ute-M (Suc r) pp (rho (Suc r) pp)} \)
- \( \mu pp : \mu pp \in \text{SHOmsgVectors Ute-M (Suc r) pp (rho (Suc r) pp)} \)
- \( \text{mupp} : \mu pp \in \text{SHOmsgVectors Ute-M (Suc r) pp (rho (Suc r) pp)} \)
- \( \text{threshold} : \text{card} \{ q. \text{sendMsg Ute-M (Suc r) q p (rho (Suc r) q)} \}
  = \text{Vote (Some v)} \} > E - \alpha \)

**Shows:**
- \( x (\text{rho (Suc r) pp}) = v \)

**Proof:**
- Assume \( \text{sr} : \text{step (Suc r)} = 0 \)
- 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 \( \text{threshold} \) obtain \( qq \) where
  \( \text{sendMsg Ute-M (Suc r) qq p (rho (Suc r) qq)} = \text{Vote (Some v)} \)
  by \( \text{force} \)
- Ultimately
  show \( \text{False} \) by \( \text{simp} \)

**QED**

**Lemma 2.1:**

**Assumptions:**
- \( \text{va} : \text{card} \{ qq. \mu pp qq = \text{Some (Vote (Some v))} \} > \alpha \)
- \( \text{is card} \{ \text{msgs v} \} > \alpha \)

**Proof:**
- From \( \text{mupp} \)
  have \( \text{SHOs (Suc r) pp \cap HOs (Suc r) pp} \)
  \( \subseteq \{ qq. \mu pp qq = \text{Some (sendMsg Ute-M (Suc r) qq pp (rho (Suc r) qq))} \} \)
  by \( \text{auto} \)
- Unfolding \( \text{SHOmsgVectors-def} \)
- Moreover
- Hence \( \text{card} \{ \text{msgs v} \} \geq \text{card} \{ \text{msgs v} \} \cap \text{SHOs (Suc r) pp \cap HOs (Suc r) pp} \)
  by \( \text{auto} \)
- Hence \( \text{card} \{ \text{msgs v} \} \)
  \( \geq \text{card} \{ \text{msgs v} \} \cap \text{SHOs (Suc r) pp \cap HOs (Suc r) pp} \)
by (auto intro: card-mono)
moreover
from usafe threshold
have alph: card (\{\text{sent} pp v\} \cap (SHOs (Suc r) pp \cap HOs (Suc r) pp)) > \alpha 
  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) \(\mu pp \ (rho (Suc (Suc r)) pp)\)
  by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
ultimately
obtain w where wa: card (\{msgs w\}) > \alpha and xw: x (rho (Suc (Suc r)) pp) = w
  unfolding next1-def by auto

have v = w
proof –
  note usafe
moreover
obtain qv where qv \in SHOs (Suc r) pp and \(\mu pp \ qv = \text{Some} \ (Vote \ (\text{Some} \ v))\)
proof –
  have ¬ (\{msgs v\} \subseteq HOs (Suc r) pp \ - \ SHOs (Suc r) pp)
    proof
      assume ?msgs v \subseteq HOs (Suc r) pp \ - \ SHOs (Suc r) pp
    hence card (\{msgs v\}) \leq\ card ((HOs (Suc r) pp) \ - \ (SHOs (Suc r) pp))
      by (auto simp: card-mono)
    moreover
    from usafe
    have \(\text{card} \ (HOs (Suc r) pp \ - \ SHOs (Suc r) pp) \leq \alpha\)
      by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
    moreover
    note va
    ultimately
    show False by auto
  qed
  then obtain qv
    where qv \notin HOs (Suc r) pp \ - \ SHOs (Suc r) pp
    and qv: \(\mu pp \ qv = \text{Some} \ (Vote \ (\text{Some} \ v))\)
    by auto
  with mupp have qv \in 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
\(qw \in \text{SHOs}(\text{Suc } r)\) \textbf{and} \(\mu pp \ qw = \text{Some (Vote (Some w))}\)

**Proof**

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

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

- **with stp mupp have** \(\text{vote (rho (Suc } r) \ qw) = \text{Some w}\)
  - **by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send1-def)**
  - **moreover**
  - **from run obtain** \(\mu qv \ mu qw\)
    - **where** \(\text{nextState Ute-M } r \ qv (((\rho \ rho) \ \ qv) \ \ muqv (\rho (\text{Suc } r) \ \ qv)\)
      - **and** \(\mu qv \in \text{SHOmsgVectors Ute-M } r \ qv (\rho (\text{HOs } r) \ \ qv) \ \ \text{(SHOs } r \ \ qv)\)
    - **and** \(\text{nextState Ute-M } r \ qw (((\rho \ rho) \ \ qw) \ \ muqw (\rho (\text{Suc } r) \ \ qw)\)
      - **and** \(\mu qw \in \text{SHOmsgVectors Ute-M } r \ qw (\rho (\text{HOs } r) \ \ qw) \ \ \text{(SHOs } r \ \ qw)\)
      - **by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq) blast**
      - **ultimately**
      - **show ?thesis using usafe by (auto dest: common-vote)**
  - **qed**

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

**Inductive argument for the agreement and validity theorems.**

**Lemma** safety-inductive-argument:

- **assumes** \(\text{run: } \text{SHORun Ute-M } \rho \ \text{HOs SHOs}\)
- **and** \(\text{comm: } \forall r. \ \text{SHOcommPerRd Ute-M (HOs } r) \ \ (\text{SHOs } r)\)
- **and** \(\text{sh: } E - \alpha < \text{card } \{q. \ \text{sendMsg Ute-M } r^' q p (\rho (r^' q) = \text{Vote (Some v))}\}\)
- **and** \(\text{stp1: } \text{step } r^' = \text{Suc } 0\)
- **shows** \(E - \alpha < \text{card } \{q. \ \text{sendMsg Ute-M (Suc } (Suc } r^')) q p (\rho (\text{Suc } (Suc } r^')) q) = \text{Vote (Some v)}\)
proof –
  from stp1 have \( r' > 0 \) by (auto simp: step-def)
  with stp1 obtain \( r \) where \( \rho : r' = \text{Suc} \ r \) and stpr:step \( (\text{Suc} \ r) = \text{Suc} \ 0 \)
  by (auto dest: gr0-implies-Suc)

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

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

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

  hence \( \forall \ p. \ \mu \ pp', \ \mu \ pp' \in \text{SHOmsgVectors} \ \text{Ute-M} \ (\text{Suc} \ r) \ p \ (\rho \ (\text{Suc} \ r')) \)
    \( (\text{HOs} \ (\text{Suc} \ r') \ p) \ (\text{SHOs} \ (\text{Suc} \ r') \ p) \)
    \( \Longrightarrow \ \text{card} \ (\text{SHOs} \ (\text{Suc} \ r') \ p \cap \text{HOs} \ (\text{Suc} \ r') \ p) \)
    \( \leq \ \text{card} \ \{ q. \ \mu \ pp' q = \text{Some} \ (\text{Val} \ v) \} \)
  by (auto simp: card-mono)

  moreover
  from 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 \( \forall \ p. \ \mu \ pp', \ \mu \ pp' \in \text{SHOmsgVectors} \ \text{Ute-M} \ (\text{Suc} \ r) \ p \ (\rho \ (\text{Suc} \ r')) \)
    \( (\text{HOs} \ (\text{Suc} \ r') \ p) \ (\text{SHOs} \ (\text{Suc} \ r') \ p) \)
    \( \Longrightarrow \ T < \text{card} \ \{ q. \ \mu \ pp' q = \text{Some} \ (\text{Val} \ v) \} \)
  by (auto dest: less-le-trans)

  show \( ?\text{thesis} \)
  proof
    have \( \forall \ pp. \ \text{vote} \ ((\rho \ (\text{Suc} \ r')) \ pp) = \text{Some} \ v \)
    proof
      fix \( pp \)
      from run obtain \( \mu \pp \)
        where \( \text{nextState} \ \text{Ute-M} \ (\text{Suc} \ r') \ pp \ (\rho \ (\text{Suc} \ r') \ pp) \ \mu \pp \)
          \( (\rho \ (\text{Suc} \ r') \ pp) \)
        and \( \mu \pp \in \text{SHOmsgVectors} \ \text{Ute-M} \ (\text{Suc} \ r') \ pp \ (\rho \ (\text{Suc} \ r')) \)
    
  109
A process that holds some decision \( v \) has decided \( v \) sometime in the past.

**Lemma decisionNonNullThenDecided:**

assumes run:SHORun \( Ute-M \) \( rho \) HOs SHOs and \( dec: decide (rho \ n \ p) = Some \ v \)

shows \( \exists m<n.\ decide (rho (Suc m) \ p) \neq decide (rho m \ p) \)
\& decide (rho (Suc m) \ p) = Some \ v

proof –

let \( ?dec k = decide ((rho k) \ p) \)
have \((\forall m<n.\ ?dec (Suc m) \neq ?dec (Suc m) \neq Some \ v) \)
\rightarrow ?dec n \neq Some \ v
(is \( ?P \ n \) is \( ?A n \rightarrow - \)

proof (induct \( n \))

from run show \( ?P 0 \)
by (auto simp: Ute-SHOMachine-def SHORun-eq HOinitConfig-eq
initState-def Ute-initState-def)

next
fix \( n \)
assume \( ih: ?P \ n \) thus \( ?P (Suc \ n) \) by force
qed

with \( dec \) show \( ?thesis \) by auto

qed
If process \( p1 \) has decided value \( v1 \) and process \( p2 \) later decides, then \( p2 \) must decide \( v1 \).

**lemma** laterProcessDecidesSameValue:

**assumes** run:SHORun Ute-M rho HOs SHOs
and comm:\( \forall \) r. SHOcommPerRd Ute-M (HOs r) (SHOs r)
and dv1:decide (rho (Suc r) p1) = Some v1
and dn2:decide (rho (r + k) p2) \( \neq \) Some v2
and dv2:decide (rho (Suc (r + k)) p2) = Some v2

**shows** \( v2 = v1 \)

**proof** –

from run dv1 obtain r1
where r1'r1 < Suc r
and dn1:decide (rho r1 p1) \( \neq \) Some v1
and dv1':decide (rho (Suc r1) p1) = Some v1
by (auto dest: decisionNonNullThenDecided)

from r1r obtain s where r1:Suc r = Suc (r1 + s)
by (auto dest: less-imp-Suc-add)
then obtain k' where kk':r + k = r1 + k'
by auto
with dn2 dv2
have dn2':decide (rho (r1 + k') p2) \( \neq \) Some v2
and dv2':decide (rho (Suc (r1 + k')) p2) = Some v2
by auto

from run dn1 dv1' dn2' dv2'
have rs0:step r1 = Suc 0 and rks0:step (r1 + k') = Suc 0
by (auto simp: mod-Suc step-def dest: decide-step)

have step (r1 + k') = step (step r1 + k')
unfolding step-def by (simp add: mod-add-left-eq)
with rs0 rks0 have step k' = 0 by (auto simp: step-def mod-Suc)
then obtain k'' where k'' = k''*nSteps by (auto simp: step-def)
with dn2' dv2'
have dn2'':decide (rho (r1 + k''*nSteps) p2) \( \neq \) Some v2
and dv2'':decide (rho (Suc (r1 + k''*nSteps)) p2) = Some v2
by auto

from rs0 have stp:step (r1 + k''*nSteps) = Suc 0
unfolding step-def by auto

have inv:card \{q. sendMsg Ute-M (r1 + k''*nSteps) q p1 (rho (r1 + k''*nSteps) q)\}
= Vote (Some v1) > E - \alpha

**proof** (induct k'')

from stp have step (r1 + 0*nSteps) = Suc 0
by (auto simp: step-def)
from run comm dn1 dv1'
show card \{q. sendMsg Ute-M (r1 + 0*nSteps) q p1 (rho (r1 + 0*nSteps) q)\}
\begin{verbatim}

\textbf{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: \( \forall r. \) 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} –
\textbf{from} run p obtain k
\textbf{where} k1: decide (rho (Suc k) p) \( \neq \) decide (rho k p)

\end{verbatim}
and \( k_2 \): decide \((\rho \cdot (Suc \ k) \ p) = \text{Some} \ v\)

by (auto dest: decisionNonNullThenDecided)

from run \ q \ obtain \ l

where \( l_1 \): decide \((\rho \cdot (Suc \ l) \ q) \neq \text{decide} \ (\rho \ l \ q)\)

and \( l_2 \): decide \((\rho \cdot (Suc \ l) \ q) = \text{Some} \ w\)

by (auto dest: decisionNonNullThenDecided)

show \(?\text{thesis}\)

proof (cases \( k \leq \ l \))

\begin{itemize}
  \item case \( \text{True} \)
    then obtain \( m \) where \( m: l = k + m \)

    by (auto simp add: le_iff_add)

    from run \ comm \ k_2 \ l_1 \ l_2 \ m \ obtain \ w

    by (auto elim!: laterProcessDecidesSameValue)

  \end{itemize}

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

next

\begin{itemize}
  \item case \( \text{False} \)
    hence \( l \leq k \)

    by simp

    then obtain \( m \) where \( m: k = l + m \)

    by (auto simp add: le_iff_add)

    from run \ comm \ l_2 \ k_1 \ k_2 \ m \ show \(?\text{thesis}\)

    by (auto elim!: laterProcessDecidesSameValue)

\end{itemize}

qed

Main lemma for the proof of the Validity property.

\textbf{lemma} \ validity-argument:

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

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

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

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

\textbf{and} \ \text{stp}: \ \text{step} \ r' = \text{Suc} \ 0

\textbf{shows} \ \text{card} \ \{q. \ \text{sendMsg} \ Ute-M \ r' \ q \ p \ (\rho \ r' \ q) = \text{Vote} \ (\text{Some} \ v)\} > E - \alpha

\textbf{proof} –

\textbf{define} \( k \) where \( k = r' \div nSteps\)

\textbf{with} \( \text{stp} \) have \( \text{stp}: \ r' = \text{Suc} \ 0 + k \cdot nSteps\)

\textbf{using} \ div-mul-and-mod-eq \ [of \ r' \ nSteps]

\textbf{by} (simp add: step-def)

\textbf{moreover}

have \( E - \alpha < \)

\text{card} \ \{q. \ \text{sendMsg} \ Ute-M \ (\text{Suc} \ 0 + k \cdot nSteps) \ q \ p \ ((\rho \ (\text{Suc} \ 0 + k \cdot nSteps)) \ q)\}

= \text{Vote} \ (\text{Some} \ v)\}

\textbf{proof} (induct \( k \))

\textbf{have} \( \forall pp. \ \text{vote} \ ((\rho \ (\text{Suc} \ 0)) \ pp) = \text{Some} \ v\)

\textbf{proof}

\textbf{fix} \( pp\)

\textbf{from run obtain} \( pp \pmb{\mu}\)

\textbf{where} \( \text{nextState} \ Ute-M \ 0 \ pp \ (\rho \ 0 \ pp) \ \pmb{\mu} pp \ (\rho \ (\text{Suc} \ 0) \ pp)\)

\textbf{and} \( \text{msgVectors} \ Ute-M \ 0 \ pp \ (\rho \ 0) \ (\text{HOs} \ 0 \ pp) \ (\text{SHOs} \ 0 \ pp)\)

\textbf{by} (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)

\[113\]
have majv: card \{ q. \mu pp q = Some (Val v)\} > T

proof –

from run init have \forall q. \sendMsg Ute-M 0 q pp (\rho 0 q) = Val v 
  by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
    Ute-sendMsg-def send0-def step-def)

moreover

from comm have shoT:card (SHOs 0 pp \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 = Some (\sendMsg Ute-M 0 q pp (\rho 0 q))\}
  by (auto simp: SHOmsgVectors-def)

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

ultimately

show \?thesis by (auto simp: less-le-trans)

qed

moreover

from nxtpp have next0 0 pp (\rho 0) pp \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)

from majv majw have v = w by (auto dest: unique-majority-T)

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

qed

with run

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

thus E - \alpha < 
  card \{ q. \sendMsg Ute-M (Suc 0 + 0 * nSteps) q p (\rho (Suc 0 + 0 * nSteps) q) 
    = Vote (Some v)\}
  using majE EltN by auto

next

fix k

assume ih: E - \alpha < 
  card \{ q. \sendMsg Ute-M (Suc 0 + k * nSteps) q p (\rho (Suc 0 + k * nSteps) q) 
    = Vote (Some v)\}

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

from run comm ih this
have \( E - \alpha < \) 
\[ \text{card} \{ q. \text{sendMsg} Ute-M (\text{Suc (Suc 0 + k * nSteps)}) q p \} \]
\[= \text{Vote (Some v) \} by (rule safety-inductive-argument)} \]
thus \( E - \alpha < \) 
\[ \text{card} \{ q. \text{sendMsg} Ute-M (\text{Suc 0 + Suc k * nSteps}) q p \}
\[= \text{Vote (Some v) \} by simp \}
qed
ultimately
show \(?thesis by simp \)
qed

The following theorem shows the Validity property of algorithm \( \mathcal{U}_{T,E,\alpha} \).

**Theorem** \( \text{ute-validity} \): 
assumes run: SHORun Ute-M rho HOs SHOs 
and comm: \( \forall r. \text{SHOcommPerRd Ute-M (HOs r) (SHOs r)} \) 
and init: \( \forall p, x (\text{rho 0 p}) = v \) 
and dw: decide (\( \text{rho r p} \)) = Some w 
shows \( v = w \)
proof –
from run dw obtain r1 
where \( \text{dwr1:decide ((rho r1) p) \neq Some w} \) 
and \( \text{dwr1:decide ((rho (Suc r1)) p) = Some w} \) 
by (force dest: decisionNonNullThenDecided) 
with run have step r1 \( \neq 0 \) by (rule decide-step) 
hence step r1 = Suc 0 by (simp add: step-def mod-Suc) 
with assms 
have \( E - \alpha < \) 
\[ \text{card} \{ q. \text{sendMsg} Ute-M r1 q p (\text{rho r1 q}) = \text{Vote (Some v) \} by (rule validity-argument)} \]
moreover 
from run comm dwr1 dwr1 
have \( \text{card} \{ q. \text{sendMsg} Ute-M r1 q p (\text{rho r1 q}) = \text{Vote (Some w) \} > E - \alpha} \) 
by (auto dest: decide-with-threshold-E) 
ultimately 
show \( v = w \) by (auto dest: unique-majority-E-\( \alpha \))
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** \( \text{set-x-from-vote} \): 
assumes run: SHORun Ute-M rho HOs SHOs 
and comm: \( \text{SHOcommPerRd Ute-M (HOs r) (SHOs r)} \)
and $stp$: step $(Suc r) = Suc 0$
and $π$: $∀ p. \text{HOs} (Suc r) p = \text{SHOs} (Suc r) p$
and $nxt$: nextState $Ute-M (Suc r) p (\rho (Suc r) p) μ (\rho (Suc (Suc r)) p)$
and $mu$: $μ ∈ \text{SHOmsgVectors} Ute-M (Suc r) p (\rho (Suc r))$
and $vp$: $α < \text{card} \{ qq. μ qq = \text{Some} (Vote (Some v)) \}$
shows $x ((\rho (Suc (Suc r)))) p) = v$

proof –
from $nxt stp vp$ obtain $wp$
  where $xwp: α < \text{card} \{ qq. μ qq = \text{Some} (Vote (Some wp)) \}$
  and $xwp: x (\rho (Suc (Suc r))) p) = wp$
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def)

have $wp = v$
proof –
from $xwp$ obtain $qq$ where $smw: μ qq = \text{Some} (Vote (Some wp))$
  by force
have $vote (\rho (Suc r) qq) = \text{Some} v$
proof –
from $smw$ $mu$ $π$
  have $μ qq = \text{Some} (sendMsg Ute-M (Suc r) qq p (\rho (Suc r) pp))$
    unfolding $\text{SHOmsgVectors-def}$ by force
  with $stp$ have $μ pp = \text{Some} (Vote (vote (\rho (Suc r) pp)))$
    by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def)
with $smw$ show $?thesis$ by auto
qed

moreover
from $wp$ obtain $qq$
  where $smw: μ qq = \text{Some} (Vote (Some v))$
  by force
have $vote (\rho (Suc r) qq) = \text{Some} v$
proof –
from $smw$ $mu$ $π$
  have $μ qq = \text{Some} (sendMsg Ute-M (Suc r) qq p (\rho (Suc r) qq))$
    unfolding $\text{SHOmsgVectors-def}$ by force
  with $stp$ have $μ qq = \text{Some} (Vote (vote (\rho (Suc r) qq)))$
    by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def)
with $smw$ show $?thesis$ by auto
qed

moreover
from $run$ obtain $μpp μqq$
  where nextState $Ute-M r pp (\rho r pp) μpp (\rho (Suc r) pp)$
  and $μpp$ $∈ \text{SHOmsgVectors} Ute-M r pp (\rho r) (\text{HOs} r pp) (\text{SHOs} r pp)$
  and $nextState Ute-M r qq ((\rho r) qq) μqq (\rho (Suc r) qq)$
  and $μqq$ $∈ \text{SHOmsgVectors} Ute-M r qq (\rho r) (\text{HOs} r qq) (\text{SHOs} r qq)$
  unfolding $\text{Ute-SHOMachine-def}$ $\text{SHORun-eq}$ $\text{SHOnextConfig-eq}$ by blast
ultimately
show $?thesis$ using $comm$ by (auto dest: common-vote)
qed

with $xp$ show $?thesis$ by simp
qed

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

**lemma** termination-argument-1:

- **assumes** run: SHORun Ute-M rho HOs SHOs
- **and** comm: SHOcommPerRd Ute-M (HOs $r$) (SHOs $r$)
- **and** $stp$: step (Suc $r$) = Suc 0
- **and** $\pi$: $\forall p. \pi 0 = HOs (Suc r) p \land \pi 0 = SHOs (Suc r) p$

**obtains** $v$ where

\[
\forall p \mu_p' q. \\
\begin{cases}
q \in SHOs (Suc (Suc r)) p \cap HOs (Suc (Suc r)) p; \\
\mu_p' \in SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r))) (HOs (Suc (Suc r)) p) (SHOs (Suc (Suc r)) p)
\end{cases}
\implies \mu_p' q = (Some (Val v))
\]

**proof**

from $\pi$ have hosho:$\forall p. SHOs (Suc r) p = SHOs (Suc r) p \cap HOs (Suc r) p$

by simp

have $\forall p q. x (rho (Suc (Suc r)) p) = x (rho (Suc (Suc r)) q)$

**proof**

fix $p q$

from run obtain $\mu_p$

where nxt: nextState Ute-M (Suc $r$) $p$ (rho (Suc $r$) $p$)

$\mu_p$ (rho (Suc (Suc $r$)) $p$)

and mu: $\mu_p \in SHOmsgVectors Ute-M (Suc (Suc r)) p (rho (Suc (Suc r))) (HOs (Suc (Suc r)) p) (SHOs (Suc (Suc r)) p)$

by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)

from run obtain $\mu_q$

where nxtq: nextState Ute-M (Suc $r$) $q$ (rho (Suc $r$) $q$)

$\mu_q$ (rho (Suc (Suc $r$)) $q$)

and muq: $\mu_q \in SHOmsgVectors Ute-M (Suc r) q (rho (Suc r)) (HOs (Suc r) q) (SHOs (Suc r) q)$

by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)

have $\forall qq. \mu_p qq = \mu_q qq$

**proof**

fix $qq$

show $\mu_p qq = \mu_q qq$

**proof** (cases $\mu_p qq = None$)

case False

with mu $\pi$ have 1:$qq \in SHOs (Suc r) p$ and 2:$qq \in SHOs (Suc r) q$

unfolding SHOmsgVectors-def by auto

from mu $\pi$ 1

have $\mu_p qq = Some (sendMsg Ute-M (Suc r) qq p (rho (Suc r) qq))$

unfolding SHOmsgVectors-def by auto

moreover
from \( \mu q \pi 2 \)

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

unfolding \( \text{SHOmsgVectors-def} \) by auto

ultimately

show \( \text{thesis} \)

by (auto simp: \( \text{Ute-SHOMachine-def} \) \( \text{Ute-sendMsg-def} \) \( \text{step-def} \) \( \text{send0-def} \) \( \text{send1-def} \))

next

case \( \text{True} \)

with \( \mu q \) have \( qq \notin \text{HOs (Suc r) p} \) unfolding \( \text{SHOmsgVectors-def} \) by auto

with \( \pi \mu q \) have \( \mu q q = \text{None} \) unfolding \( \text{SHOmsgVectors-def} \) by auto

with \( \text{True} \) show \( \text{thesis} \) by simp

qed

qed

hence \( \forall q. \{ qq. \mu q qq = \text{Some (Vote (Some v))} \} = \{ qq. \mu q qq = \text{Some (Vote (Some v))} \} \)

by auto

show \( x (\rho (\text{Suc (Suc r)}) p) = x (\rho (\text{Suc (Suc r)}) q) \)

proof (cases \( \exists v. \alpha < \text{card} \{ qq. \mu p qq = \text{Some (Vote (Some v))}\} \), clarify)

fix \( v \)

assume \( \forall p: \alpha < \text{card} \{ qq. \mu p qq = \text{Some (Vote (Some v))}\} \)

with \( \text{run comm stp \pi nxt \mu} \) have \( x (\rho (\text{Suc (Suc r)}) p) = v \)

by (auto dest: \( \text{set-x-from-vote} \))

moreover

from \( \forall vsets \forall p: \alpha < \text{card} \{ qq. \mu p qq = \text{Some (Vote (Some v))}\} \) by auto

with \( \text{run comm stp \pi nxtq \mu q} \) have \( x (\rho (\text{Suc (Suc r)}) q) = v \)

by (auto dest: \( \text{set-x-from-vote} \))

ultimately

show \( x (\rho (\text{Suc (Suc r)}) p) = x (\rho (\text{Suc (Suc r)}) q) \)

by auto

next

assume \( \forall nov: \neg (\exists v. \alpha < \text{card} \{ qq. \mu p qq = \text{Some (Vote (Some v))}\}) \)

with \( \text{nxtq stp} \) have \( x (\rho (\text{Suc (Suc r)}) q) = \text{undefined} \)

by (auto simp: \( \text{Ute-SHOMachine-def nextState-def} \)
\( \text{Ute-nextState-def nextState-def} \))

moreover

from \( \forall vsets \forall nov \)

have \( \neg (\exists v. \alpha < \text{card} \{ qq. \mu p qq = \text{Some (Vote (Some v))}\}) \) by auto

with \( \text{nxtq stp} \) have \( x (\rho (\text{Suc (Suc r)}) q) = \text{undefined} \)

by (auto simp: \( \text{Ute-SHOMachine-def nextState-def} \)
\( \text{Ute-nextState-def nextState-def} \))

ultimately

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

qed

qed

then obtain \( v \) where \( \forall q. x (\rho (\text{Suc (Suc r)}) q) = v \) by blast
moreover
from \( \text{stp} \) have \( \text{step} (\text{Suc} (\text{Suc} r)) = 0 \)
  by (auto simp: step-def mod-Suc)
hence \( \forall p \, \mu p' \, q \).
  \[
  \begin{align*}
  & q \in \text{SHOs} (\text{Suc} (\text{Suc} r)) \cap \text{HOs} (\text{Suc} (\text{Suc} r)) \, p; \\
  & \mu p' \in \text{SHOmsgVectors Ute-M} (\text{Suc} (\text{Suc} r)) \, p \, (\text{rho} (\text{Suc} (\text{Suc} r))) \\
  & \text{(HOs} (\text{Suc} (\text{Suc} r)) \, p) \, (\text{SHOs} (\text{Suc} (\text{Suc} r)) \, p) \\
  \end{align*}
  \]
  \( \Rightarrow \mu p' \, q = \text{Some} (\text{Val} (x \, (\text{rho} (\text{Suc} (\text{Suc} r)) \, q))) \)
  by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send0-def)
ultimately have \( \forall p \, \mu p' \, q \).
  \[
  \begin{align*}
  & q \in \text{SHOs} (\text{Suc} (\text{Suc} r)) \cap \text{HOs} (\text{Suc} (\text{Suc} r)) \, p; \\
  & \mu p' \in \text{SHOmsgVectors Ute-M} (\text{Suc} (\text{Suc} r)) \, p \, (\text{rho} (\text{Suc} (\text{Suc} r))) \\
  & \text{(HOs} (\text{Suc} (\text{Suc} r)) \, p) \, (\text{SHOs} (\text{Suc} (\text{Suc} r)) \, p) \\
  \end{align*}
  \]
  \( \Rightarrow \mu p' \, q = \text{Some} (\text{Val} v) \)
  by auto
with that show thesis by blast
qed

If a process \( p \) votes \( v \) at some round \( r \), then all messages received by \( p \) in \( r \)
that are not corrupted hold \( v \).

**lemma** termination-argument-2:
**assumes** \( \text{mup} : \mu p \in \text{SHOmsgVectors Ute-M} (\text{Suc} r) \, p \, (\text{rho} (\text{Suc} r)) \)
  \( \text{(HOs} (\text{Suc} r) \, p) \, (\text{SHOs} (\text{Suc} r) \, p) \)
and \( \text{nxtq} : \text{nextState Ute-M r q} \, (\text{rho r q}) \, \mu q \, (\text{rho} (\text{Suc} r) \, q) \)
and \( \text{vq} : \text{vote} (\text{rho} (\text{Suc} r) \, q) = \text{Some} v \)
and \( \text{qsho} : q \in \text{SHOs} (\text{Suc} r) \, p \cap \text{HOs} (\text{Suc} r) \, p \)
shows \( \mu p \, q = \text{Some} (\text{Vote} (\text{Some} v)) \)
**proof** –
from \( \text{nxtq} \) \( \text{vq} \) have \( \text{step} r = 0 \) by (auto simp: vote-step)
with \( \text{mup} \) \( \text{qsho} \) have \( \mu p \, q = \text{Some} (\text{Vote} (\text{vote} (\text{rho} (\text{Suc} r) \, q))) \)
  by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def
step-def send1-def mod-Suc)
with \( \text{vq} \) show \( \mu p \, q = \text{Some} (\text{Vote} (\text{Some} v)) \) by auto
qed

We now prove the Termination property.

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

obtain \( w \) where \( \text{vote}(\text{rho} (\text{Suc} (\text{Suc} r0))) p) = \text{Some} w \)
proof –

have \( \text{abc} : \forall p. \exists w. \text{vote} (\text{rho} (\text{Suc} (\text{Suc} r0))) p) = \text{Some} w \)
proof

fix \( p \)
from run stp obtain \( \mu p \)
where \( \text{nxt:nextState Ute-M} (\text{Suc} r0) p (\text{rho} (\text{Suc} r0)) p)\)
and \( \text{map:map} \in \text{SHOmsgVectors Ute-M} (\text{Suc} r0) p (\text{rho} (\text{Suc} r0))\)
(\( \text{HOs} (\text{Suc} r0) p) (\text{SHOs} (\text{Suc} r0) p) \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)

have \( \exists v. T < \text{card} \{ qq. \mu p qq = \text{Some} (\text{Val} v) \} \)
proof –

from \( t \) have \( \text{card} (\text{SHOs} (\text{Suc} r0) p) \cap \text{HOs} (\text{Suc} r0) p) > T .. \)
moreover
from run commR stp \( \pi r0 \) obtain \( v \) where
\[ \forall q. q \in \text{SHOs} (\text{Suc} r0) p \cap \text{HOs} (\text{Suc} r0) p; \]
\( \mu p' \in \text{SHOmsgVectors Ute-M} (\text{Suc} r0) p (\text{rho} (\text{Suc} r0))\)
(\( \text{HOs} (\text{Suc} r0) p) (\text{SHOs} (\text{Suc} r0) p) \)
[ \[ \implies \mu p' q = \text{Some} (\text{Val} v) \] ]

using termination-argument-1 by blast

with \( \text{map} \) obtain \( v \) where
\[ \forall qq. qq \in \text{SHOs} (\text{Suc} r0) p \cap \text{HOs} (\text{Suc} r0) p \implies \mu p qq = \text{Some} (\text{Val} v) \]
by auto

hence \( \text{SHOs} (\text{Suc} r0) p \cap \text{HOs} (\text{Suc} r0) p \subseteq \{ qq. \mu p qq = \text{Some} (\text{Val} v) \} \)
by auto

hence \( \text{card} (\text{SHOs} (\text{Suc} r0) p) \cap \text{HOs} (\text{Suc} r0) p) \)
\[ \leq \text{card} \{ qq. \mu p qq = \text{Some} (\text{Val} v) \} \]
by (auto intro: card-mono)
ultimately
have \( T < \text{card} \{ qq. \mu p qq = \text{Some} (\text{Val} v) \} \)
by auto
thus \( \text{thesis} \) by auto

qed
with \( \text{stp nxt show} \exists w. \text{vote} ((\text{rho} (\text{Suc} (\text{Suc} r0))) p) = \text{Some} w \)
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def
\( \text{step-def mod-Suc next0-def} \))

qed

then obtain \( qq \) \( w \) where \( \text{qq:vote} (\text{rho} (\text{Suc} (\text{Suc} r0))) qq) = \text{Some} w \)
by blast
have \( \forall pp. \text{vote} (\text{rho} (\text{Suc} (\text{Suc} r0))) pp) = \text{Some} w \)
proof
  fix pp
  from abc obtain wp where wpv:vote ((rho (Suc (Suc r0))) pp) = Some wp
    by blast
  from run obtain µpp µqq
    where nxtp: nextState Ute-M (Suc r0) pp (rho (Suc (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 nxtpQ: nextState Ute-M (Suc r0) qq (rho (Suc (Suc r0))) qq
      µqq (rho (Suc (Suc r0))) qq
    and muq: 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 wpv qqv have wp = w
    by (auto dest: common-vote)
  with wpv show vote ((rho (Suc (Suc r0))) pp) = Some w
    by auto
  qed
  with that show ?thesis by auto
  qed

from run obtain µp'
  where nxtp: nextState Ute-M (Suc (Suc r0)) p (rho (Suc (Suc r0))) p
    µp' (rho (Suc (Suc r0))) p
  and mupQ: 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
      nxtpq: 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 mupQ nxtpq 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 (HOs (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 0
unfolding step-def by auto
with nextp have next1 (Suc (Suc r0)) p (rho (Suc (Suc r0))) p \(\mu p' \) (rho (Suc (Suc r0))) p
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
moreover from e have \(E < \text{card} (\text{SHOs} (\text{Suc} (\text{Suc} r0))) p \cap \text{HOs} (\text{Suc} (\text{Suc} r0)) p) \) by auto
with wsho have majv: \(\text{card} \{qq. \mu p' \text{qq} = \text{Some} (\text{Vote} (\text{Some} w))\} > E\) by auto
ultimately show \(?thesis\) by (auto simp: next1-def)
qed

8.7 \(U_{T,E,\alpha}\) Solves Weak Consensus

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

theorem ute-weak-consensus:
assumes run: SHORun Ute-M rho HOs SHOs
and commR: \(\forall r. \text{SHOcommPerRd} \text{Ute-M} (\text{HOs} r) (\text{SHOs} r)\)
and commG: \(\text{SHOcommGlobal} \text{Ute-M} \text{HOs} \text{SHOs}\)
shows weak-consensus \((x \circ (\rho 0))\) 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 \((\lambda q. \text{undefined})\)
and commR: \(\forall r. \text{SHOcommPerRd} \text{Ute-M} (\text{HOs} r) (\text{SHOs} r)\)
and commG: \(\text{SHOcommGlobal} \text{Ute-M} \text{HOs} \text{SHOs}\)
shows weak-consensus \((\lambda p. x (\text{state} (\rho 0) p))\) decide \((\text{state} \circ \rho 0)\)
(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 \((\lambda q. \text{undefined})\)
and init: crun 0 = state (ho 0)
from crun have SHORun Ute-M crun HOs SHOs by (unfold SHORun-def)
from this commR commG
have weak-consensus \((x \circ (\text{crun} 0))\) decide crun
by (rule ute-weak-consensus)
with init show weak-consensus {?inits decide crun}
by (simp add: o-def)
qed
9 Verification of the $\mathcal{A}_{T,E,\alpha}$ Consensus algorithm

Algorithm $\mathcal{A}_{T,E,\alpha}$ is presented in [3]. Like $\mathcal{U}_{T,E,\alpha}$, it is an uncoordinated algorithm that tolerates value faults, and it is parameterized by values $T$, $E$, and $\alpha$ that serve a similar function as in $\mathcal{U}_{T,E,\alpha}$, allowing the algorithm to be adapted to the characteristics of different systems. $\mathcal{A}_{T,E,\alpha}$ can be understood as a variant of 

We formalize in Isabelle the correctness proof of the algorithm that appears in [3], using the framework of theory $\textit{HOModel}$. 

9.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable $\texttt{proc}$ of the generic HO model.

$$
typedecl \texttt{Proc} \quad \text{the set of processes}
$$

axiomatization where $\text{Proc-finite} : \text{OFCLASS}(\texttt{Proc}, \text{finite-class})$

instance $\texttt{Proc} :: \text{finite}$ by (rule Proc-finite)

abbreviation

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

The following record models the local state of a process.

$$
\text{record 'val pstate } =
\quad x :: '\text{val} \quad \text{— current value held by process}
\quad \text{decide :: 'val option} \quad \text{— value the process has decided on, if any}
$$

The $x$ field of the initial state is unconstrained, but no decision has yet been taken.

$$
definition \text{Ate-initState where}
\quad \text{Ate-initState} p st \equiv (\text{decide } st = \text{None})
$$

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

$$\textit{locale ate-parameters } =$$

fixes $\alpha :: \text{nat} \text{ and } T :: \text{nat} \text{ and } E :: \text{nat}$

assumes $TNaE : T \geq 2*(N + 2*\alpha - E)$
and $TltN : T < N$
and \( E \lt N \)

begin

The following are consequences of the assumptions on the parameters.

**lemma** \( \text{majE: } 2 \ast (E - \alpha) \geq N \)

**using** \( TNaE \ TltN \) by auto

**lemma** \( Egl\alpha: E > \alpha \)

**using** \( \text{majE} \ EltN \) by auto

**lemma** \( Tge2\alpha: T \geq 2 \ast \alpha \)

**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 \textit{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

\[
\text{mostOftenRcvd}(\text{msgs} :: \text{Proc} \Rightarrow \text{'val option}) \equiv \\
\{ v. \forall w. \text{card}\left(\text{qq. msgs qq = Some w}\right) \leq \text{card}\left(\text{qq. msgs qq = Some v}\right)\}
\]

**definition** \( \text{Ate-sendMsg :: nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{'val pstate} \Rightarrow \text{'val} \)

**where**

\( \text{Ate-sendMsg r p q st} \equiv x' st \)

**definition** \( \text{Ate-nextState :: nat} \Rightarrow \text{Proc} \Rightarrow (\text{'val:linorder}) \text{ pstate} \Rightarrow (\text{Proc} \Rightarrow \text{'val option}) \Rightarrow \text{'val pstate} \Rightarrow \text{bool} \)

**where**

\( \text{Ate-nextState r p st msgs st'} \equiv \\
(\text{if card}\left(\text{q. msgs q \neq None}\right) > T) \\
\text{then } x' st' = \text{Min}\left(\text{mostOftenRcvd} \text{ msgs}\right) \\
\text{else } x' st' = x st) \\
\wedge (\exists v. \text{card}\left(\text{q. msgs q = Some v}\right) > E \land \text{decide st'} = \text{Some v}) \\
\lor \lnot (\exists v. \text{card}\left(\text{q. msgs q = Some v}\right) > E) \\
\land \text{decide st'} = \text{decide st})
\]

9.2 Communication Predicate for \( A_{T,E,\alpha} \)

Following [3], we now define the communication predicate for the \( A_{T,E,\alpha} \) 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

\( \text{Ate-commPerRd HOrs SHOrs} \equiv \)
∀ p. card (\text{HOs} p - \text{SHOs} 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**

\textit{Ate-commGlobal} where

\textit{Ate-commGlobal} \text{HOs} \text{SHOs} \equiv

\begin{align*}
&\left( \forall p \exists r' > r. \text{card} (\text{HOs} r' p) > T \right) \\
&\land \left( \forall r \exists r' > r. \text{card} (\text{SHOs} r' p \cap \text{HOs} r' p) > E \right) \\
&\land \left( \forall r \exists r' > r. \exists \pi_1 \pi_2. \text{card} \pi_1 > E - \alpha \\
&\land \text{card} \pi_2 > T \\
&\land \left( \forall p \in \pi_1. \text{HOs} r' p = \pi_2 \land \text{SHOs} r' p \cap \text{HOs} r' p = \pi_2 \right) \\
\end{align*}

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

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

**definition** Ate-SHOMachine where

\textit{Ate-SHOMachine} = (\text{Proc}, \text{val:linorder pstate})

**abbreviation**

\textit{Ate-M} \equiv \text{SHOMachine}::(\text{Proc}, \text{val:linorder pstate}, \text{val})

end — locale ateparameters

end

theory AteProof

imports AteDefs ./Reduction

begin

context ateparameters

begin

125
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 np: decide (rho r p) ≠ Some v**
- **and vp: decide (rho (Suc r) p) = Some v**
- **shows** \( \{ qq. \text{sendMsg} Ate-M r qq p (rho r qq) = v \} > E - \alpha \)

**proof** –

- **from run obtain \( \mu p \)**
  - **where mw: \( \mu p \in SHOmsgVectors Ate-M r p (rho r) (HOs r p) (SHOs r p) \)**
  - **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**

- **have \( \{ qq. \mu p qq = Some v \} - (HOs r p - SHOs r p) \subseteq \{ qq. \text{sendMsg} Ate-M r qq p (rho r qq) = v \} \)**
  - **(is \( ?\text{vcvd}p - ?\text{ahop} \subseteq ?\text{vsent}p) \)**
    - **by (auto simp: SHOmsgVectors-def)**

**hence** \( \text{card} \ (?\text{vcvd}p - ?\text{ahop}) \leq \text{card} ?\text{vsent}p \)

- **and** \( \text{card} \ (?\text{vcvd}p - ?\text{ahop}) \geq \text{card} ?\text{vcvd}p - \text{card} ?\text{ahop} \)**
  - **by (auto simp: card-mono diff-card-le-card-Diff)**

**hence** \( \text{card} ?\text{vsent}p \geq \text{card} ?\text{vcvd}p - \text{card} ?\text{ahop} \) by auto

**moreover**

- **from nxt np vp have card ?\text{vcvd}p > 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 \( ?\text{thesis} \text{ using} \text{ Egta by auto} \)**

**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)**
- **and muq: \( \mu q \in SHOmsgVectors Ate-M r q (rho r) (HOs r q) (SHOs r q) \)**
- **and vsent: card \( \{ qq. \text{sendMsg} Ate-M r qq q (rho r qq) = v \} \geq E - \alpha \)**
  - **(is \( \text{card} ?\text{vsent} > - \))**
- **shows** \( \text{card} \ (\{ qq. \mu q qq \neq Some v \} \cap HOs r q) \leq N + 2*\alpha - E \)

**proof** –

- **from nxt muq**

**have** \( \{ qq. \mu q qq \neq Some v \} \cap HOs r q \subseteq (HOs r q - SHOs r q) \)

- **(is \( ?\text{notvcvd} - ?\text{aho} \subseteq ?\text{notsent} \))**
unfolding \textit{SHOmsgVectors-def} by \textit{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 \((\text{auto simp: card-mono diff-card-le-card-Diff})\)

moreover

from \text{\textit{comm}} have \(\text{card } ?\text{aho} \leq \alpha\)

by \((\text{auto simp: Ate-SHOMachine-def Ate-commPerRd-def})\)

moreover

have \(1: \text{card } ?\text{notvsent} + \text{card } ?\text{vsent} = \text{card } (?\text{notvrcvd} \cup ?\text{vsent})\)

by \((\text{subst card-Un-Int})\) \text{auto}

have \(?\text{vsent} = (\text{UNIV::Proc set})\) by \text{auto}

ultimately

show \(?\text{thesis using EltN Egta by auto}\)

qed

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

\textbf{lemma} \textit{mostOftenRcvd-v:}

\textbf{assumes} \textit{comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)}

\textbf{and} \textit{nxt: nextState Ate-M r q (rho r q) \(\mu q ((\text{rho (Suc r)}) q)\)}

\textbf{and} \textit{muq: \(\mu q \in \text{SHOmsgVectors Ate-M r q (rho r) (SHOs r q)}\)}

\textbf{and} \textit{threshold-T: \text{card } \{qq. \mu q qq \neq \text{None}\} > T}\)

\textbf{and} \textit{threshold-E: \text{card } \{qq. \text{sendMsg Ate-M r qq q (rho r qq) = v}\} > E - \alpha}\)

\textbf{shows} \textit{mostOftenRcvd \(\mu q = \{v\}\)}

\textbf{proof} –

from \textit{muq have hodef:HOs r q = \{qq. \mu q qq \neq \text{None}\}}

unfolding \textit{SHOmsgVectors-def by auto}

from \textit{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 \text{card } ?\text{heardnotv} \leq -\)

by \((\text{rule other-values-received})\)

moreover

have \(\text{card } ?\text{heardnotv} \geq T + 1 - \text{card } \{qq. \mu q qq = \text{Some } v\}\)

\textbf{proof} –

from \textit{muq}

have \(?\text{heardnotv} = (\text{HOs r q}) - \{qq. \mu q qq = \text{Some } v\}\)

and \(?\text{heardnotv} = \text{HOs r q} - \{qq. \mu q qq = \text{Some } v\}\)

unfolding \textit{SHOmsgVectors-def by auto}

hence \(\text{card } ?\text{heardnotv} = \text{card } (\text{HOs r q}) - \text{card } \{qq. \mu q qq = \text{Some } v\}\)

by \((\text{auto simp: card-Diff-subset})\)

with \textit{hodef threshold-T show }?\text{thesis by auto}\)

\textbf{qed}

ultimately

have \(\text{card } \{qq. \mu q qq = \text{Some } v\} > \text{card } ?\text{heardnotv}\)

using \textit{TNaE by auto}
moreover
{
  fix w
  assume w: w ≠ v
  with hodef have \{ qq, µq qq = Some w \} ⊆ ?heardnotv by auto
  hence card \{ qq, µq qq = Some w \} ≤ card ?heardnotv by (auto simp: card-mono)
}
ultimately
have \{ w. card \{ qq, µq qq = Some w \} ≥ card \{ qq, µq qq = Some v \} \} = \{ v \}
  by force
thus ?thesis unfolding mostOftenRcvd-def by auto
qed

If at some round more than E − α processes have their x variable set to v, then this is also true at next round.

**Lemma common-x-induct:**

assumes ran: SHORun Ate-M rho HOs SHOs
and comm: SHOcommPerRd Ate-M (HOs (r+k)) (SHOs (r+k))
and ih: card \{ qq. x (rho (r+k) qq) = v \} > E − α
shows card \{ qq. x (rho (r+Suc k) qq) = v \} > E − α

**Proof:**

from ih
have thrE∀ pp. card \{ qq. sendMsg Ate-M (r + k) qq pp (rho (r + k) qq) = v \} > E − α
  by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
{
  fix qq
  assume kv:x (rho (r + k) qq) = v
  from run obtain µqq
    where nxt: nextState Ate-M (r + k) qq (rho (r + k) qq) µqq ((rho (Suc (r + k))) qq)
    and muq: µqq ∈ SHOmsgVectors Ate-M (r + k) qq (rho (r + k)) (HOs (r + k) qq) (SHOs (r + k) qq)
    by (auto simp: SHORun-eq SHOnextConfig-eq)
  have x (rho (r + Suc k) qq) = v
  proof (cases card \{ pp. µqq pp ≠ None \} > T)
    case True
    with comm nxt muq thrE have mostOftenRcvd µqq = \{ 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 \( qupdate: x (rho (r + Suc k) q) \neq x (rho (r + k) q) \)
- 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: \( \mu q \in SHOmsgVectors Ate-M (r+k) q (rho (r+k)) \)
  \[(HOs (r+k) q) (SHOs (r+k) q)\]
  by (auto simp: SHORun-eq SHOnextConfig-eq)
- moreover
  from nxt qupdate
  have threshold-T: \( \text{card} \{qq. \mu q qq \neq \} > T \)
  - and \( xsmall: x (rho (r + Suc k) q) = \text{Min} (\text{mostOftenRcvd} \mu q) \)
  - by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
- moreover
  have \( E - \alpha < \text{card} \{qq. x (rho (r + k) qq) = v\} \)
  - proof (induct \( k \))
    from run comm \( d1 \) \( d2 \)
    have \( E - \alpha < \text{card} \{qq. sendMsg Ate-M r qq p (rho r qq) = v\} \)
    - by (auto dest: decide-sent-msgs-threshold)
    thus \( E - \alpha < \text{card} \{qq. x (rho (r + 0) qq) = v\} \)
    - by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
  next
  fix \( k \)
  assume \( E - \alpha < \text{card} \{qq. x (rho (r + k) qq) = v\} \)
  with run comm show \( E - \alpha < \text{card} \{qq. x (rho (r + Suc k) qq) = v\} \)
  - by (auto dest: common-x-induct)
  qed
with \( run \)

have \( E - \alpha < \text{card}\{qq, \text{sendMsg Ate-M} (r+k) qq \} \)
   \( \rho (r+k) qq = v \}
   by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def SHORun-eq SHOnextConfig-eq)

ultimately

have \( \text{mostOftenRcvd} \mu q = \{v\} \) by (auto dest: mostOftenRcvd-v)

A process that holds some decision \( v \) has decided \( v \) sometime in the past.

**lemma decisionNonNullThenDecided**:  
assumes \( \text{run} : \text{SHORun Ate-M} \rho \text{HOs} \text{SHOs} \)

and \( \text{dec} : \text{decide} (\rho n p) = \text{Some} v \)

obtains \( m \) where \( m < n \)

and \( \text{decide} (\rho m p) \neq \text{Some} v \)

and \( \text{decide} (\rho (\text{Suc} m) p) = \text{Some} v \)

**proof** –

let \( ?\text{dec} k = \text{decide} (\rho k p) \)

have \( (\forall m < n. \ ?\text{dec} (\text{Suc} m)) \neq \ ?\text{dec} (\text{Suc} m) \rightarrow \ ?\text{dec} n \neq \text{Some} v \)

(is \( ?P n \) is \( \bot A n \rightarrow \bot \))

**proof** (induct \( n \))

from \( \text{run} \) show \( ?P 0 \)
   by (auto simp: Ate-SHOMachine-def SHORun-eq HOinitConfig-eq
   initState-def Ate-initState-def)

**next**

fix \( n \)

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

**qed**

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

**qed**

9.5 Proof of Validity

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

**theorem ate-validity**:  
assumes \( \text{run} : \text{SHORun Ate-M} \rho \text{HOs} \text{SHOs} \)

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

and \( \text{initv} : \forall q. x (\rho 0 q) = v \)

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

shows \( w = v \)

**proof** –

{  
  \( \text{fix} \ r \)
  \( \text{have} (\forall qq. \text{sendMsg Ate-M} r qq \rho (\rho r qq) = v \)
  \( \text{proof} \) (induct \( r \))
  \( \text{from} \ \text{run initv} \) show \( (\forall qq. \text{sendMsg Ate-M} 0 qq \rho (\rho 0 qq) = v \)
  \( \text{by} \) (auto simp: SHORun-eq SHOnextConfig-eq Ate-SHOMachine-def Ate-sendMsg-def)  

130
next

fix r

assume ih: ∀ qq. sendMsg Ate-M r qq p (rho r qq) = v

have ∀ qq. x (rho (Suc r) qq) = v

proof

fix qq

from run obtain μ

where nxt: nextState Ate-M r qq (rho r qq) μqq (rho (Suc r) qq)

and mu: μqq ∈ SHOmsgVectors Ate-M r qq (rho r qq) (HOs r qq) (SHOs r qq)

by (auto simp: SHORun-eq SHOnextConfig-eq)

from nxt

have (card {pp. μqq pp ≠ None} > T ∧ x (rho (Suc r) qq) = Min (mostOftenRcvd μqq))

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

thus x (rho (Suc r) qq) = v

proof safe

assume x (rho (Suc r) qq) = x (rho r qq)

with ih show ?thesis

by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)

next

assume threshold-T: T < card {pp. μqq pp ≠ None}

and xsmall: x (rho (Suc r) qq) = Min (mostOftenRcvd μqq)

have card {pp. ∃ w. w ≠ v ∧ μqq pp = Some w} ≤ T div 2

proof −

from comm have 1:card (HOs r qq - SHOs r qq) ≤ α

by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)

moreover

from mu ih

have SHOs r qq ∩ HOs r qq ⊆ {pp. μqq pp = Some v}

and HOs r qq = {pp. μqq pp ≠ None}

by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def Ate-sendMsg-def)

hence {pp. μqq pp ≠ None} - {pp. μqq pp = Some v} ⊆ HOs r qq - SHOs r qq

by auto

hence card ({pp. μqq pp ≠ None} - {pp. μqq pp = Some v}) ≤ card (HOs r qq - SHOs r qq)

by (auto simp:card-mono)

ultimately

have card ({pp. μqq pp ≠ None} - {pp. μqq pp = Some v}) ≤ T div 2

using Tge2a by auto

moreover

have {pp. μqq pp ≠ None} - {pp. μqq pp = Some v} = {pp. ∃ w. w ≠ v ∧ μqq pp = Some w} by auto

ultimately

show ?thesis by simp
moreover
have \{ pp. \mu qq \neq None \}
= \{ pp. \mu qq = Some v \} \cup \{ pp. \exists w. w \neq v \land \mu qq = Some w \}
and \{ pp. \mu qq = Some v \} \cap \{ pp. \exists w. w \neq v \land \mu qq = Some w \} = 
\{
\}
by auto
hence card \{ pp. \mu qq \neq None \}
= card \{ pp. \mu qq = Some v \} + card \{ pp. \exists w. w \neq v \land \mu qq = Some w \}

by auto
moreover
note threshold-T
ultimately
have card \{ pp. \mu qq pp = Some v \} > card \{ pp. \exists w. w \neq v \land \mu qq pp = Some w \}
by auto
moreover
\{ fix w 
assume w \neq v 
hence \{ pp. \mu qq pp = Some w \} \subseteq \{ pp. \exists w. w \neq v \land \mu qq pp = Some w \}
by auto 

hence card \{ pp. \mu qq pp = Some w \} \leq card \{ pp. \exists w. w \neq v \land \mu qq pp = Some w \} 
by (auto simp: card-mono)
}
ultimately
have \( \forall w. w \neq v \implies card \{ pp. \mu qq pp = Some w \} < card \{ pp. \mu qq pp = Some v \} \)
by force
hence \( \forall w. card \{ pp. \mu qq pp = Some v \} \leq card \{ pp. \mu qq pp = Some w \} \implies w = v \)
by force
with \( \exists z \) have mostOftenRcvd \( \mu qq = \{ v \} \)
by (force simp: mostOftenRcvd-def)
with xsmall show \( x (\rho (Suc r) qq) = v \) by auto
qed
qed
thus \( \forall qq. sendMsg Ate-M (Suc r) qq p (\rho (Suc r) qq) = v \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
qed

note \( P = this \)

from \( run dp \) obtain \( rp \)
where \( rp: rp < r \) decide \( (\rho \rho rp p) \neq Some w \)
decide \( (\rho (Suc rp) p) = Some w \)
by (rule decisionNonNullThenDecided)

132
from run obtain \( \mu p \)

where \( \text{nxt: nextState Ate-M \( \rho p \) \( \mu p \) (\( \rho (\text{Suc} \( \rho p \)) \) \( p \))} \)

and \( \text{mu: } \mu p \in \text{SHOmsgVectors} \ Ate-M \( \rho p \) \( \text{HOs} \( \rho p \)) \ (\text{SHOs} \( \rho p \)) \)

by (auto simp: SHORun-eq SHOneExtConfig-eq)

\[
\begin{align*}
\{ & \text{fix } w \\
\text{assume } w: w \neq v \\
\text{from } \text{comm have } \text{card} (\text{HOs} \( \rho p \) - \text{SHOs} \( \rho p \)) \leq \alpha \\
& \text{by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)} \\
\text{moreover} \\
\text{from } \text{mu P} \text{ have } \text{SHOs} \( \rho p \) \cap \text{HOs} \( \rho p \) \subseteq \{pp. \mu p pp = \text{Some v}\} \\
& \text{and } \text{HOs} \( \rho p \) = \{pp. \mu p pp \neq \text{None}\} \\
& \text{by (auto simp: SHOmsgVectors-def)} \\
\text{hence } \{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some v}\} \\
& \subseteq \text{HOs} \( \rho p \) - \text{SHOs} \( \rho p \) \\
& \text{by auto} \\
\text{hence } \text{card} (\{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some v}\}) \\
& \leq \text{card} (\text{HOs} \( \rho p \) - \text{SHOs} \( \rho p \)) \\
& \text{by (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 \\
& \text{by simp} \\
\text{moreover} \\
\text{from } \text{w have } \{pp. \mu p pp = \text{Some w}\} \\
& \subseteq \{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some v}\} \\
& \text{by auto} \\
\text{hence } \text{card} \{pp. \mu p pp = \text{Some w}\} \leq \text{card} (\{pp. \mu p pp \neq \text{None}\} - \{pp. \mu p pp = \text{Some v}\}) \\
& \text{by (auto simp: card-mono)} \\
\text{ultimately} \\
\text{have } \text{card} \{pp. \mu p pp = \text{Some w}\} < E \text{ by simp} \\
\text{hence } \text{PP: } \bigwedge w. \text{card} \{pp. \mu p pp = \text{Some w}\} \geq E \implies w = v \text{ by force} \\
\text{from } \text{rp nxt mu have } \text{card} \{q. \mu p q = \text{Some w}\} > E \\
& \text{by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def Ate-commPerRd Ate-nextState-def)} \\
\text{with } \text{PP show } ?\text{thesis by auto} \\
\text{qed} \\
\end{align*}
\]

### 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 \( \rho \) \( \text{HOs} \) \( \text{SHOs} \)

and comm: SHOcommPerRd Ate-M (\( \text{HOs} \) \( r \)) (\( \text{SHOs} \) \( r \))
and \( nvp \) decide \((\rho r p) \neq \text{Some } v\)  
and \( vp \) decide \((\rho (\text{Suc } r) p) = \text{Some } v\)  
and \( nwq \) decide \((\rho r q) \neq \text{Some } w\)  
and \( wq \) decide \((\rho (\text{Suc } r) q) = \text{Some } w\)

shows \( w = v\)

proof –

have \( gtn: \text{card} \{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\} \) 
+ \( \text{card} \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\} > N\)

proof –

from \( \text{run comm } nvp \ vp \)
have \( \text{card} \{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\} > E - \alpha\)
by \( \text{(rule decide-sent-msgs-threshold)}\)
moreover
from \( \text{run comm } nwq \ wq \)
have \( \text{card} \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\} > E - \alpha\)
by \( \text{(rule decide-sent-msgs-threshold)}\)
ultimately
show \( \textit{thesis using majE by auto}\)
qed

show \( \textit{thesis}\)

proof \( \text{(rule ccontr)}\)

assume \( \text{wv:w} \neq v\)
have \( \forall qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq)\)
by \( \text{(auto simp: Ate-SHOMachine-def Ate-sendMsg-def)}\)
with \( \text{wv}\)
have \( \{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\} \)
\( \cap \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\} = \{\}\)
by \( \text{auto}\)
with \( \text{gtn}\)
have \( \text{card} \{\{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\}\) 
\( \cup \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\}\} > N\)
by \( \text{(auto simp: card-Un-Int)}\)
moreover
have \( \text{card} \{\{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\}\) 
\( \cup \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\}\} \leq N\)
by \( \text{(auto simp: card-mono)}\)
ultimately
show \( \textit{False by auto}\)
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 \( \text{run: SHORun } \text{Ate-M } \rho \text{HOS SHOs}\)
and \( \text{comm: } \forall r. \text{SHOcommPerRd } \text{Ate-M } (\text{HOS } r) (\text{SHOs } r)\)
and \( \text{nd1: decide } (\rho r p) \neq \text{Some } v\)

show \( \text{?thesis}\)

proof

(\text{rule ccontr})

assume \( \text{wv:w} \neq v\)
have \( \forall qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq)\)
by \( \text{(auto simp: Ate-SHOMachine-def Ate-sendMsg-def)}\)
with \( \text{wv}\)
have \( \{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\} \)
\( \cap \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\} = \{\}\)
by \( \text{auto}\)
with \( \text{gtn}\)
have \( \text{card} \{\{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\}\) 
\( \cup \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\}\} > N\)
by \( \text{(auto simp: card-Un-Int)}\)
moreover
have \( \text{card} \{\{qq. \text{sendMsg } \text{Ate-M } r \ qq p (\rho r qq) = v\}\) 
\( \cup \{qq. \text{sendMsg } \text{Ate-M } r \ qq q (\rho r qq) = w\}\} \leq N\)
by \( \text{(auto simp: card-mono)}\)
ultimately
show \( \textit{False by auto}\)
qed

qed
and \(d1\): decide \((\rho \text{ (Suc} r) \ p) = \text{Some} \ v\)
and \(nd2\): decide \((\rho \text{ (r+k) } q) \neq \text{Some} \ w\)
and \(d2\): decide \((\rho \text{ (Suc (r+k)) } q) = \text{Some} \ w\)
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: \(\{qq. \text{sendMsg Ate-M} r \ qq \ p (\rho r \ qq) = v\}\)
\(\cap \{qq. \text{sendMsg Ate-M} (r+k) \ qq \ (\rho (r+k) \ qq) = w\} = \{\}\)
(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-SHOMachine-def Ate-sendMsg-def)
have \(\exists k' < k. x (\rho (r + k') \ qq) \neq w \land x (\rho (r + \text{Suc} k') \ qq) = w\)
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 \(ksk': \text{Suc} k' < k\)
from \(ksk'\) have \(k' < k\) by simp
with \(ih\) show \(x (\rho (r + \text{Suc} k') \ qq) \neq w\) by auto
qed
\}
with \(f\) have \(\forall k' < k. x (\rho (r + \text{Suc} k') \ qq) \neq w\) by auto
moreover
from \(kgt0\) have \(k - 1 < k\) and \(kk: \text{Suc} (k - 1) = k\) by auto
ultimately
have \(x (\rho (r + \text{Suc} (k - 1)) \ qq) \neq w\) by blast
with \(rkw \ \text{kk}\) show \(\text{False}\) by simp
qed
then obtain \(k'\)
where \(k' < k\)
and \( w: x (\rho (r + \text{Suc } k') qq) = w \)
and \( qq \text{update}: x (\rho (r + \text{Suc } k') qq) \neq x (\rho (r + k') qq) \)

by \textbf{auto}

from \textbf{run comm nd1 d1} \textbf{qqupdate}
have \( x (\rho (r + \text{Suc } k') qq) = v \) by \textbf{rule common-x}
with \( w \ \textbf{vdifw} \ \textbf{show} \ \textbf{False} \) by \textbf{simp}

\textbf{qed}

from \textbf{run comm nd1 d1} \textbf{have} \( \text{sentv} \): \( \text{card } \text{sentv} > E - \alpha \)
by \textbf{(auto dest: decide-sent-msgs-threshold)}
from \textbf{run comm nd2 d2} \textbf{have} \( \text{card } \text{sentw} > E - \alpha \)
by \textbf{(auto dest: decide-sent-msgs-threshold)}
with \( \text{sentv} \ \text{majE} \ \textbf{have} \ (\text{card } \text{sentv}) + (\text{card } \text{sentw}) > N \)
by \textbf{simp}
with \( 1 \ \textbf{vdifw} \ \textbf{have} \ 2: \ (\text{card } \text{sentv} \cup \text{sentw}) > N \)
by \textbf{(auto simp: card-Un-Int)}
have \( \text{card } (\text{sentv} \cup \text{sentw}) \leq N \)
by \textbf{(auto simp: card-mono)}
with \( 2 \ \textbf{show} \ \textbf{False} \) by \textbf{simp}

\textbf{qed}

The Agreement property is now an immediate consequence.

\textbf{theorem ate-agreement:}

\textbf{assumes} \textbf{run} \( \textbf{SHORun} \ Ate-M \rho HOs SHOs \)
\textbf{and} \textbf{comm}: \( \forall \ r. \ \textbf{SHOcommPerRd} \ Ate-M (HOs r) (SHOs r) \)
\textbf{and} \( \textbf{p: decide} \ (\rho m p) = \text{Some } v \)
\textbf{and} \( \textbf{q: decide} \ (\rho n q) = \text{Some } w \)
\textbf{shows} \( w = v \)
\textbf{proof} –

from \textbf{run} \( \textbf{p obtain} \ k \ \textbf{where} \)
\( k: k < m \ \text{decide} \ (\rho k p) \neq \text{Some } v \ \text{decide} \ (\rho (\text{Suc } k) p) = \text{Some } v \)
by \textbf{(rule decisionNonNullThenDecided)}
from \textbf{run} \( \textbf{q obtain } l \ \textbf{where} \)
\( l: l < n \ \text{decide} \ (\rho l q) \neq \text{Some } w \ \text{decide} \ (\rho (\text{Suc } l) q) = \text{Some } w \)
by \textbf{(rule decisionNonNullThenDecided)}
\textbf{show} \ ?thesis
\textbf{proof} (cases \( k \leq l \))
\textbf{case} \( \text{True} \)
then obtain \( i \ \textbf{where} \ l = k+i \) by \textbf{(auto simp add: le-iff-add)}
with \textbf{run comm k l} \textbf{show} \ ?thesis
by \textbf{(auto dest: laterProcessDecidesSameValue)}

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

\textbf{qed}

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

and \( \text{commR}: \forall r. \, (\text{SHOcommPerRd}:((\text{Proc}, 'val::linorder \, pstate, 'val) \, \text{SHOMachine}) \Rightarrow (\text{Proc} \, \text{HO} \, r) \Rightarrow (\text{Proc} \, \text{HO} \, r) \Rightarrow \text{bool}) \)

and \( \text{commG}: \text{SHOcommGlobal} Ate-M HOs SHOs \)

shows \( \exists r \, v. \, \text{decide} (\rho \, r \, p) = \text{Some} \, v \)

proof –

from \( \text{commG} \) obtain \( r' \, \pi_1 \, \pi_2 \)

where \( \pi_e: \text{card} \, \pi_1 > E - \alpha \)

and \( \pi_t: \text{card} \, \pi_2 > T \)

and \( \text{hosho}: \forall p \in \pi_1. \, (\text{HOs} \, r' \, p = \pi_2 \land \text{SHOs} \, r' \, p \cap \text{HOs} \, r' \, p = \pi_2) \)

by \((\text{auto simp: Ate-SHOMachine-def Ate-commGlobal-def})\)

obtain \( v \) where

\( P1: \forall pp. \, \text{card} \, \{ qq. \, \text{sendMsg} \, Ate-M \, (\text{Suc} \, r') \, qq \, pp \, (\rho \, (\text{Suc} \, r') \, qq) = v \} > E - \alpha \)

proof –

have \( \forall p \in \pi_1. \forall q \in \pi_1. \, x \, (\rho \, (\text{Suc} \, r') \, p) = x \, (\rho \, (\text{Suc} \, r') \, q) \)

proof (clarify)

fix \( p \, q \)

assume \( p: \, p \in \pi_1 \) and \( q: \, q \in \pi_1 \)

from \( \text{run} \) obtain \( \mu p \)

where \( \text{nxtp}: \, \text{nextState} \, Ate-M \, r' \, p \, (\rho \, r' \, p) \, \mu p \, (\rho \, (\text{Suc} \, r') \, p) \)

and \( \text{mup}: \, \mu p \in \text{SHOmsgVectors} \, Ate-M \, r' \, p \, (\rho \, r') \, (\text{HOs} \, r') \, p \) (\( \text{SHOs} \, r' \, p \))

by \((\text{auto simp: SHORun-eq SHOnextConfig-eq})\)

from \( \text{run} \) obtain \( \mu q \)

where \( \text{nxtq}: \, \text{nextState} \, Ate-M \, r' \, q \, (\rho \, r' \, q) \, \mu q \, (\rho \, (\text{Suc} \, r') \, q) \)

and \( \text{muq}: \, \mu q \in \text{SHOmsgVectors} \, Ate-M \, r' \, q \, (\rho \, r') \, (\text{HOs} \, r') \, q \) (\( \text{SHOs} \, r' \, q \))

by \((\text{auto simp: SHORun-eq SHOnextConfig-eq})\)

from \( \text{mup} \) \( \text{muq} \) \( p \) \( q \)

have \( \{ qq. \, \mu p \, qq \neq \text{None} \} = \text{HOs} \, r' \, q \)

and \( 2: \{ qq. \, \mu q \, qq = \text{Some} \, (\text{sendMsg} \, Ate-M \, r' \, qq \, q \, (\rho \, r' \, qq)) \} \supset \text{SHOs} \, r' \, q \cap \text{HOs} \, r' \, q \)

and \( \{ qq. \, \mu p \, qq \neq \text{None} \} = \text{HOs} \, r' \, p \)

and \( 4: \{ qq. \, \mu p \, qq = \text{Some} \, (\text{sendMsg} \, Ate-M \, r' \, qq \, p \, (\rho \, r' \, qq)) \} \supset \text{SHOs} \, r' \, p \cap \text{HOs} \, r' \, p \)

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

with \( p \, q \) \( \text{hosho} \)

137
have aa: \( \pi_2 = \{qq. \mu qq \neq \text{None}\} \)

and cc: \( \pi_2 = \{qq. \mu pq \neq \text{None}\} \) by auto

from p q hosho 2
have bb:{qq. \mu pq qq \neq \text{None}} \supseteq \pi_2
by auto

from p q hosho 4
have dd:{qq. \mu pq qq \neq \text{None}} \supseteq \pi_2
by auto

have Min (mostOftenRcvd \mu pq) = Min (mostOftenRcvd \mu qq)

proof −

have \( \forall qq. \text{sendMsg Ate-M r}' qq p (\rho r' qq) = \text{sendMsg Ate-M r}' qq q (\rho r' qq) \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)

with aa bb cc dd
have \( \forall qq. \mu pq qq \neq \text{None} \rightarrow \mu pq qq = \mu pq qq \)
by force

moreover
from aa bb cc dd
have \( \{qq. \mu pq qq \neq \text{None}\} = \{qq. \mu pq qq \neq \text{None}\} \) by auto

hence \( \forall qq. \mu pq qq = \text{None} \leftrightarrow \mu pq qq = \text{None} \) by blast

hence \( \forall qq. \mu pq qq = \text{None} \rightarrow \mu pq qq = \mu pq qq \) by auto

ultimately
have \( \forall qq. \mu pq qq = \mu pq qq \) by blast

thus \(?thesis by\) (auto simp: mostOftenRcvd-def)

qed

with \( \pi t aa nxtq \pi t cc nxtp \)

show \( x (\rho (\text{Suc r}') p) = x (\rho (\text{Suc r}') q) \)
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 \( \forall p \in \pi 1. \text{sendMsg Ate-M (Suc r') p pp (rho (Suc r') p) = v} \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)

hence \( \text{card } \pi 1 \leq \text{card } \{qq. \text{sendMsg Ate-M (Suc r') qq pp (rho (Suc r') qq) = v}\} \)
by (auto intro: card-mono)

with \( \pi ea \)

have \( E - \alpha < \text{card } \{qq. \text{sendMsg Ate-M (Suc r') qq pp (rho (Suc r') qq) = v}\} \)
by simp

} with that show \(?thesis by blast\)

qed

{ fix \( k pp \)

have \( E - \alpha < \text{card } \{qq. \text{sendMsg Ate-M (Suc r' + k) qq pp (rho (Suc r' + k) qq) = v}\} \)
(is \(?P k\)

138
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) ≠ 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 μpp 
        where nxt: nextState Ate-M (Suc r' + k) pp (rho (Suc r' + k) pp) μpp 
          (rho (Suc r' + Suc k) pp) 
        and mu: μpp ∈ 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. μpp qq ≠ None\} > T 
        and zsmall: x (rho (Suc r' + Suc k) pp) = Min (mostOftenRcved μpp) 
        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\} . 
        ultimately 
      have mostOftenRcved μpp = \{v\} by (auto dest!: mostOftenRcved-v) 
      with zsmall 
      have x (rho (Suc r' + Suc k) pp) = v by simp
  }

  139
\( P3 = \text{this} \)

have \( P4 \forall pp. \exists k. x (\rho (Suc r' + Suc k) pp) = v \)
proof
fix \( pp \)
from \( \text{commG} \) have \( \exists r'' > r'. \text{card} (HOs r'' pp) > T \)
by (auto simp: \( \text{Ate-SHOMachine-def \ Ate-commGlobal-def} \))
then obtain \( k \) where \( Suc r' + k > r' \) and \( t:\text{card} (HOs (Suc r' + k) pp) > T \)
by (auto dest: \( \text{less-imp-Suc-add} \))
moreover
from \( \text{run} \) obtain \( \mu pp \)
where \( \text{nxt: nextState Ate-M (Suc r' + k) pp (\rho (Suc r' + k) pp) \mu pp (Suc r' + Suc k) pp) } \)
and \( \text{mu: \mu pp \in SHOmsgVectors Ate-M (Suc r' + k) pp (\rho (Suc r' + k)) (HOs (Suc r' + k) pp) (SHOs (Suc r' + k) pp) } \)
by (auto simp: \( \text{SHORun-eq \ SHOnextConfig-eq} \))
moreover
have \( x (\rho (Suc r' + Suc k) pp) = v \)
proof
from \( \text{commR} \) have \( \text{SHOcommPerRd::((\text{Proc}, val::linorder pstate, val::linorder) SHOMachine) \Rightarrow (Proc HO) \Rightarrow (Proc HO) \Rightarrow bool} \)
\( \text{Ate-M (HOs (Suc r' + k)) (SHOs (Suc r' + k)) } \) ..
moreover
from \( \mu \) have \( \text{HOs (Suc r' + k) pp = \{q. \mu pp q \neq None\} } \)
by (auto simp: \( \text{SHOmsgVectors-def} \))
with \( \text{nxt t} \)
have \( \text{threshold-T: card \{q. \mu pp q \neq None\} > T} \)
and \( \text{xsmall: x (\rho (Suc r' + Suc k) pp) = Min (mostOftenRcvd \mu pp)} \)
by (auto simp: \( \text{Ate-SHOMachine-def \ nextState-def \ Ate-nextState-def} \))
moreover
from \( P2 \)
have \( E - \alpha < \text{card} \{qq. \text{sendMsg Ate-M (Suc r' + k) qq pp (\rho (Suc r' + k)) qq\} = v \} \).
ultimately
have \( \text{mostOftenRcvd \mu pp = \{v\}} \)
using \( \text{nxt \mu \text{ by \ (auto dest!: mostOftenRcvd-v)}} \)
with \( \text{xsmall show \ ?thesis by \ auto} \)
qed
thus \( \exists k. x (\rho (Suc r' + Suc k) pp) = v \) ..
qed

have \( P5a: \forall pp. \exists rr. \forall k. x (\rho (rr + Suc k) pp) = v \)
proof
fix \( pp \)
from \( P4 \) obtain \( \text{rk} \) where
\( \text{xsmall: x (\rho (Suc r' + Suc rk) pp) = v (is x (\rho \text{?rr pp) = v)}} \)
\begin{align*} \text{} \end{align*}
by blast
have \( \forall k. \ x (\rho (\?rr + k) \ pp) = v \)
proof
  fix \( k \)
  show \( x (\rho (\?rr + k) \ pp) = v \)
  proof (induct \( k \))
    from \( x rrv \)
    show \( x (\rho (\?rr + 0) \ pp) = v \) by simp
  next
    fix \( k \)
    assume \( \text{ih} \): \( x (\rho (\?rr + k) \ pp) = v \)
    obtain \( k' \) where \( rrk: \Suc r' + k' = \?rr + k \) by auto
    show \( x (\rho (\?rr + \Suc k) \ pp) = v \)
    proof (rule ccontr)
      assume \( \text{nv} \): \( x (\rho (\Suc r + \Suc k') \ pp) \neq v \)
      with \( rrk \ \text{ih} \)
      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 \( rrk \ \text{nv} \)
      show \( \text{False} \) by (simp add: ac-simps)
    qed
  qed
  qed
  thus \( \exists rr. \ \forall k. \ x (\rho (rr + k) \ pp) = v \) 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 \( \text{imgR}: R' (\text{UNIV::Proc set}) \neq {} \)
  and \( R: \ \forall pp k. \ x (\rho (R pp + k) \ pp) = v \)
  by blast
define \( rr \) where \( rr = \text{Max} (R' \text{ UNIV}) \)

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: Ate-SHOMachine-def Ate-commGlobal-def)
with \( P5 \) obtain \( r' \)
  where \( r' > rr \)
    and \( \text{card} \ (\text{SHOs} r' p \cap \text{HOs} r' p) > E \)
    and \( \forall pp. \ \text{sendMsg} Ate-M r' pp p (\rho r' pp) = v \)
  by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
moreover
from run obtain $\mu p$
where \textit{nxt}: nextState $Ate-M r' p$ (rho $r'$ p) $\mu p$ (rho (Suc $r'$) p)
and \textit{mu}: $\mu p \in SHO\text{msg}\text{Vectors} Ate-M r' p$ (rho $r'$) (HOs $r'$ p) (SHOs $r'$ p)
by (auto simp: SHORun-eq SHOnextConfig-eq)

from \textit{mu}
have $\text{card} (\text{HOs} r' p \cap \text{HOs} r' p) \leq \text{card} \{ q. \mu p q = \text{Some } (\text{sendMsg} Ate-M r' q p (rho r' q))\}$
by (auto simp: intro: card-mono)
ultimately
have $\text{threshold-E}: \text{card} \{ q. \mu p q = \text{Some } v \} > E$ by auto
with \textit{nxt} show \textit{thesis}
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)

\texttt{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.

\textbf{theorem \textit{ate-weak-consensus}:}
\textbf{assumes} run: SHORun $Ate-M$ rho HOs SHOs
\hspace{1em}and commR: $\forall r. \text{SHO}\text{comm}\text{PerRd} Ate-M (\text{HOs} r) (\text{SHOs} r)$
\hspace{1em}and commG: $\text{SHO}\text{comm}\text{Global} Ate-M \text{HOs} \text{SHOs}$
\textbf{shows} weak-consensus $(x \circ (\text{rho } 0))$ decide $\text{rho}$
\textbf{unfolding} weak-consensus-def \textbf{using} assms
\textbf{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.

\textbf{theorem \textit{ate-weak-consensus-fg}:}
\textbf{assumes} run: fg-run $Ate-M$ rho HOs SHOs ($\lambda r q. \text{undefined}$)
\hspace{1em}and commR: $\forall r. \text{SHO}\text{comm}\text{PerRd} Ate-M (\text{HOs} r) (\text{SHOs} r)$
\hspace{1em}and commG: $\text{SHO}\text{comm}\text{Global} Ate-M \text{HOs} \text{SHOs}$
\textbf{shows} weak-consensus $(\lambda p. x (\text{state } (\text{rho } 0) p))$ decide (state $\circ$ rho)
\textbf{is weak-consensus \textit{?inits - -}}
\textbf{proof} \texttt{(rule local-property-reduction[OF run weak-consensus-is-local])}
\textbf{fix} crun
\textbf{assume} crun: CSHORun $Ate-M$ crun HOs SHOs ($\lambda r q. \text{undefined}$)
\hspace{1em}and \textit{init}: crun $0 = \text{state } (\text{rho } 0)$
\textbf{from} crun \textbf{have} SHORun $Ate-M$ crun HOs SHOs \textbf{by} (unfold SHORun-def)
\textbf{from} this \textbf{commR} \textbf{commG}
\textbf{have} weak-consensus $(x \circ (\text{crun } 0))$ decide crun
\hspace{1em}by (rule ate-weak-consensus)
\textbf{with} \textit{init} \textbf{show} weak-consensus \textit{?inits decide crun}
\hspace{1em}by (simp add: o-def)

\texttt{qed}
10 Verification of the \textit{EIGByz}_f Consensus Algorithm

Lynch \cite{lynch1980} presents \textit{EIGByz}_f, a version of the \textit{exponential information gathering} algorithm tolerating Byzantine faults, that works in \textit{f} rounds, and that was originally introduced in \cite{lamport1982}.

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

\textbf{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 \([p,q] \mapsto \textsf{Some} \ v\) 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 \textit{labels}, which encapsulate lists of distinct process identifiers and whose length is at most \textit{f+1}.

\textbf{definition} \textit{Label} = \{\text{\texttt{xs::Proc list. length \texttt{xs} \leq Suc f \land distinct \texttt{xs}}}\} 

\textbf{typedef} \textit{Label} = \textit{Label} 

\textbf{by} \ (\text{auto simp: Label-def intro: exI|where \texttt{x= [[]]}}) — the empty list is a label

There is a finite number of different labels.

\textbf{lemma} \textit{finite-Label}: \textit{finite Label}
proof  
  have \( \{xs.\ \text{set}\ \{xs\} \subseteq (\text{UNIV}::\text{Proc\ set}) \land \text{length}\ \{xs\} \leq \text{Suc}\ f\} \)  
  by (auto simp: Label-def)  
moreover  
  have finite \( \{xs.\ \text{set}\ \{xs\} \subseteq (\text{UNIV}::\text{Proc\ set}) \land \text{length}\ \{xs\} \leq \text{Suc}\ f\} \)  
  by (rule finite-lists-length-le) auto  
ultimately  
  show \( \text{thesis} \) by (auto elim: finite-subset)  
qed

lemma finite-UNIV-Label: finite (UNIV::Label set)  
proof  
  from finite-Label have finite (Abs-Label :: Label) by simp  
moreover  
  \{  
    fix l::Label  
    have l \in Abs-Label :: Label  
      by (rule Abs-Label-cases) auto  
  \}  
  hence (UNIV::Label set) = (Abs-Label :: Label) by auto  
ultimately show \( \text{thesis} \) by simp  
qed

lemma finite-Label-set [iff]: finite (S :: Label set)  
using finite-UNIV-Label by (auto intro: finite-subset)  

Utility functions on labels.  
definition root-node where  
  root-node \equiv Abs-Label []  
definition length-lbl where  
  length-lbl l \equiv length (Rep-Label l)  
lemma length-lbl [intro]: length-lbl l \leq \text{Suc}\ f  
unfolding length-lbl-def using Label-def Rep-Label by auto  
definition is-leaf where  
  is-leaf l \equiv length-lbl l = \text{Suc}\ f  
definition last-lbl where  
  last-lbl l \equiv last (Rep-Label l)  
definition butlast-lbl where  
  butlast-lbl l \equiv Abs-Label (butlast (Rep-Label l))  
definition set-lbl where  
  set-lbl l = set (Rep-Label l)  

The children of a non-leaf label are all possible extensions of that label.
definition children where
children l ≡
  if is-leaf l
  then {}  
  else { Abs-Label (Rep-Label l @ [p]) | p . p \notin set-lbl l }

10.2 Model of the Algorithm

The following record models the local state of a process.

record 'val pstate = 
  vals :: Label \Rightarrow 'val option 
  newvals :: Label \Rightarrow '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 uncon-
strained.

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

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

At every round, every process sends its current vals tree to all processes. In
fact, only the level of the tree corresponding to the round number is used
(cf. definition of extend-vals below).

definition EIG-sendMsg where
EIG-sendMsg r p q st ≡ 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' = (\lambda l.  
    if length-lbl l = Suc r \land msgs (last-lbl l) \neq None  
    then (the (msgs (last-lbl l))) (butlast-lbl l)  
    else if length-lbl l = Suc r \land 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' \land 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.

**fun** 

fixupval :: 'val option ⇒ 'val where

  | fixupval None = undefined
  | fixupval (Some v) = v

**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

**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))

**definition**

next-end where

  next-end r p 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.

**definition**

EIG-nextState where

  EIG-nextState r ≡
   if r < f then next-main r
   else if r = f then next-end r
   else (λ p st msgs st'. st' = st)

### 10.3 Communication Predicate for EIGByz

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

**definition**

SKr :: Proc HO ⇒ Proc HO ⇒ Proc set where

  SKr HO SHO ≡ { 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.

**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

\[
EIG\text{-commPerRd} \text{ HO SHO} \equiv \text{card} (SKr \text{ HO SHO}) > (N + f) \div 2
\]

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

\[
\left( \bigcap_{p \in \Pi, r' \in \mathbb{N}} \text{SHO}(p, r') \right) \subseteq \left( \bigcap_{p \in \Pi} \text{SHO}(p, r) \right),
\]

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

\[
EIG\text{-SHOMachine} = \emptyset
\]

\[
\text{CinitState} = (\lambda p \text{ st crd. } EIG\text{-initState} p \text{ st}),
\]

147
\[ \text{sendMsg} = \text{EIG-sendMsg}, \]
\[ \text{CnextState} = (\lambda r \ p \ s t \ m s s \ s t'. \ \text{EIG-nextState} \ r \ p \ s t \ m s s \ s t'), \]
\[ \text{SHOcommPerRd} = \text{EIG-commPerRd}, \]
\[ \text{SHOcommGlobal} = \text{EIG-commGlobal} \]
\]

\begin{enumerate}
\item \textbf{abbreviation} \text{EIG-M} \equiv (\text{EIG-SHOMachine}::(\text{Proc}, 'val \ pstate, 'val \ Msg) \ \text{SHOMachine})
\end{enumerate}

end
theory \ EigbyzProof
imports \ EigbyzDefs \ ../Majorities \ ../Reduction
begin

\section{10.5 Preliminary Lemmas}

Some technical lemmas about labels and trees.

\textbf{lemma} \text{not-leaf-length}:
\begin{enumerate}
\item \textbf{assumes} \ l: \neg (\text{is-leaf} \ l)
\item \textbf{shows} \ \text{length-lbl} \ l \leq \ f
\item \textbf{using} \ \text{length-lbl[of l]} \ by \ (simp \ add: \ \text{is-leaf-def})
\end{enumerate}

\textbf{lemma} \text{nil-is-Label}:
\begin{enumerate}
\item \textbf{by} (auto \ simp: \ \text{Label-def})
\end{enumerate}

\textbf{lemma} \text{card-set-lbl}:
\begin{enumerate}
\item \textbf{by} (auto \ simp: \ \text{set-lbl-def} \ \text{length-lbl-def})
\item \textbf{using} \ \text{Rep-Label[of l, unfolded \ Label-def]}
\item \textbf{by} (auto \ elim: \ \text{distinct-card})
\end{enumerate}

\textbf{lemma} \text{Rep-Label-root-node} \ [simp]: \text{Rep-Label root-node} = []
\begin{enumerate}
\item \textbf{using} \ \text{nil-is-Label} \ by \ (simp \ add: \ \text{Abs-Label-inverse})
\end{enumerate}

\textbf{lemma} \text{root-node-length} \ [simp]: \text{length-lbl} \ \text{root-node} = 0
\begin{enumerate}
\item \textbf{by} (simp \ add: \ \text{length-lbl-def})
\end{enumerate}

\textbf{lemma} \text{root-node-not-leaf}:
\begin{enumerate}
\item \textbf{by} (simp \ add: \ \text{is-leaf-def})
\end{enumerate}

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

\textbf{lemma} \text{butlast-rep-in-label}:
\begin{enumerate}
\item \textbf{assumes} \ l:l \neq \ \text{root-node}
\item \textbf{shows} \ \text{butlast} \ (\text{Rep-Label} \ l) \in \ \text{Label}
\end{enumerate}

\textbf{proof} –
\begin{enumerate}
\item \textbf{have} \ \text{Rep-Label} \ l \neq []
\item \textbf{proof}
\item \textbf{assume} \ \text{Rep-Label} \ l = []
\item \textbf{hence} \ \text{Rep-Label} \ l = \text{Rep-Label} \ \text{root-node} \ by \ \text{simp}
\item \textbf{with} \ l \ \textbf{show} \ False \ by \ (simp \ only: \ \text{Rep-Label-inject})
\end{enumerate}

148
The label of a child is well-formed.

**lemma Rep-Label-append:**

**assumes** \( l : \neg(is\text{-}leaf\ l) \)

**shows** \((Rep\text{-}Label\ l \ @ \ [p] \in\ Label) = (p \notin set\text{-}lbl\ l)\)

\( (\text{is} \ ?lhs = ?rhs \text{ is} (\?l' \in -) = -) \)

**proof**

**assume** \( lhs : ?lhs \text{ thus} ?rhs \)

**by** \((auto\ simp:\ Label\text{-}def\ set\text{-}lbl\text{-}def)\)

**next**

**assume** \( p : ?rhs \)

**from** \( l[THEN\ not\text{-}leaf\text{-}length] \text{ have} length\ ?l' \leq Suc\ f \)

**by** \((simp\ add: length\text{-}lbl\text{-}def)\)

**moreover**

**from** \( Rep\text{-}Label[of\ l] \text{ have}\ \text{distinct} (Rep\text{-}Label\ l)\)

**by** \((simp\ add: Label\text{-}def)\)

**with** \( p \text{ have} \ \text{distinct} \ ?l' \text{ by} (simp\ add: set\text{-}lbl\text{-}def)\)

**ultimately**

**show** \( ?lhs \text{ by} (simp\ add: Label\text{-}def)\)

**qed**

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

**lemma children-length:**

**assumes** \( l \in\ children\ h \)

**shows** \( length\text{-}lbl\ l = Suc\ (length\text{-}lbl\ h) \)

**using** \( label\text{-}children[OF\ assms] \text{ by} (auto\ simp: length\text{-}lbl\text{-}def)\)

The root node is never a child.

**lemma children-not-root:**

**assumes** \( root\text{-}node \in\ children\ l \)

**shows** \( P \)
The label of a child with the last element removed is the label of the parent.

**Lemma children-butlast-lbl**:
- **Assumes** \( c \in \text{children } l \)
- **Shows** \( \text{butlast-lbl } c = l \)
- **Using** \( \text{label-children} \) of \( \text{assms} \)
- **By** \((\text{auto simp: root-node-def})\)

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

**Lemma root-iff-no-child**:
- \( l = \text{root-node} \) if and only if \( \forall l'. l \notin \text{children } l' \)

**Proof**
- Assume \( l = \text{root-node} \)
- Thus \( \forall l'. l \notin \text{children } l' \) by (auto elim: children-not-root)

- Next
  - Assume \( \text{rhs}: \forall l'. l \notin \text{children } l' \)
  - Show \( l = \text{root-node} \)
    - By (rule rev-exhaust[of \( l \)])
    - Assume \( \text{Rep-Label } l = [] \)
    - Hence \( \text{Rep-Label } l = \text{Rep-Label root-node} \) by simp
    - Thus ?thesis by \((\text{simp only: Rep-Label-inject})\)

- Next
  - Fix \( l' q \)
  - Assume \( l': \text{Rep-Label } l = l' @ [q] \)
  - Let \( ?l' = \text{Abs-Label } l' \)
    - From \( \text{Rep-Label[of } l' \text{ ] have } l' \in \text{Label} \) by \((\text{simp add: Label-def})\)
    - Hence \( ?\text{repl': Rep-Label } ?l' = l' \) by \((\text{rule Abs-Label-inverse})\)

  - From \( \text{Rep-Label[of } l \text{ ] have } l' \in \text{Label} \) by \((\text{simp add: Label-def})\)
    - With \( l' \) have \( \text{Rep-Label } l = \text{Rep-Label } (\text{Abs-Label } (l' @ [q])) \)
      - By \((\text{simp add: Abs-Label-inverse})\)
      - Hence \( l = \text{Abs-Label } (l' @ [q]) \) by \((\text{simp add: Rep-Label-inject})\)
    - Moreover
      - From \( \text{Rep-Label[of } l \text{ ] have length } l' < \text{Suc } f q \notin \text{set } l' \)
        - By \((\text{auto simp: Label-def})\)
      - Moreover
        - Note \( \text{repl'} \)
        - Ultimately have \( l \in \text{children } ?l' \)
          - By \((\text{auto simp: children-def is-leaf-def length-lbl-def set-lbl-def})\)
          - With \( \text{rhs} \) show ?thesis by \text{blast}

**QED**

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

**Lemma children-last-set**:
- **Assumes** \( l: \neg(\text{is-leaf } l) \)
- **Shows** \( \text{last-lbl } (\text{children } l) = \text{UNIV} - \text{set-lbl } l \)
proof
  show \( \text{last-lbl} \ '(\text{children } l) \subseteq \text{UNIV} - \text{set-lbl } l \)
  by (auto dest: label-children simp: last-lbl-def)
next
  show \( \text{UNIV} - \text{set-lbl } l \subseteq \text{last-lbl} ' (\text{children } l) \)
  proof (auto simp: image-def)
    fix \( p \)
    assume \( p: p \notin \text{set-lbl } l \)
    with \( l \) have \( c: \text{Abs-Label} (\text{Rep-Label } l \oplus [p]) \in \text{children } l \)
    by (auto simp: children-def)
    with \( \text{Rep-Label-append} [OF } l] p \)
    show \( \exists c \in \text{children } l. p = \text{last-lbl } c \)
    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.

\textbf{lemma} \( \text{last-lbl-inj-on-children}: \text{inj-on last-lbl} (\text{children } l) \)
proof (auto simp: inj-on-def)
  fix \( c \ c' \)
  assume \( c: c \in \text{children } l \) and \( c': c' \in \text{children } l \)
  and eq: \( \text{last-lbl } c = \text{last-lbl } c' \)
  from \( c \ c' \) obtain \( p \ p' \)
    where \( p: \text{Rep-Label } c = \text{Rep-Label } l \oplus [p] \)
    and \( p': \text{Rep-Label } c' = \text{Rep-Label } l \oplus [p'] \)
    by (auto dest!: label-children)
  from \( p \ p' \) eq have \( p = p' \) by (simp add: last-lbl-def)
  with \( p \ p' \) have \( \text{Rep-Label } c = \text{Rep-Label } c' \) 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 \( l \).

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

Suppose a non-root label $l'$ of length $r+1$ ending in $q$, and suppose that $q$ is well heard by process $p$ in round $r$. Then the value with which $p$ decorates $l$ is the one that $q$ associates to the parent of $l$.

**lemma sho-correct-vals:**

**assumes** run: SHORun EIG-M rho HOs SHOs
and $l' : l' \in \text{children } l$
and shop: last-lbl $l' \in \text{SHOs (length-lbl } l) \cap \text{HOs (length-lbl } l) p$
(is $\tilde{q} \in \text{SHOs (len } l) \cap -$)

**shows** vals (rho (len $l' ) \ p) $l' = \text{vals (rho (len } l \ q) \ l}$

**proof** —
- let $\tilde{r} = \text{len } l$
from run obtain $\mu p$
  where nxt: nextState EIG-M $\tilde{r} \ p$ (rho $\tilde{r} \ p$) $\mu p$ (rho (Suc $\tilde{r}) \ p$)
  and mnu: $\mu p \in \text{SHOmsgVectors EIG-M } \tilde{r} \ p$ (rho $\tilde{r}$) (HOs $\tilde{r} \ p$) (SHOs $\tilde{r} \ p$)
  by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with shop
have msl:$\mu p$ $\tilde{q} = \text{Some (vals (rho } \tilde{r} \ q))$
  by (auto simp: EIG-SHOMachine-def EIG-sendMsg-def SHOmsgVectors-def)
from nxt length-lbl[of $l'$] children-length[OF l']
have extend-vals $\tilde{r} \ p$ (rho $\tilde{r} \ p$) $\mu p$ (rho (Suc $\tilde{r}$) \ p)
  by (auto simp: EIG-SHOMachine-def nextState-def EIG-nextState-def
next-main-def next-end-def extend-vals-def)
with msl $l'$ show ?thesis
  by (auto simp: extend-vals-def children-length children-butlast-lbl)
qed

A process fixes the value vals $l$ of a label at state length-lbl $l$, and then never modifies the value.

**lemma keep-vals:**

**assumes** run: SHORun EIG-M rho HOs SHOs

**shows** vals (rho (length-lbl $l + n$) \ p) $l = \text{vals (rho (length-lbl } l \ p) \ l}$
(is $?v n = {?vl}$)

**proof** (induct $n$)

show $?v 0 = {?vl}$ by simp
next
fix $n$
assume ih: $?v n = {?vl}$
let $?r = \text{length-lbl } l + n$
from run obtain $\mu p$
  where nxt: nextState EIG-M $?r \ p$ (rho $?r \ p$) $\mu p$ (rho (Suc $?r \ p$))
  by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with ih show $?v (\text{Suc } n) = {?vl}$
  by (auto simp: EIG-SHOMachine-def nextState-def EIG-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** run: SHORun EIG-M rho HOs SHOs
- and \( l' \): \( l' \in \text{children } l \)
- and skr: last-lbl \( l' \in SKr (HOs (length-lbl l)) (SHOs (length-lbl l)) \)
- shows vals (rho (length-lbl \( l' \) \( p \) \( l' \)) \( = \) vals (rho (length-lbl \( l' \)) \( q \) \( l' \)) \)
  - **using** assms unfolding SKr-def by (auto simp: sho-correct vals)

Suppose that \( l \) is a non-root label whose last element was well heard by all processes at round \( r \), and that \( l' \) 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** run: SHORun EIG-M rho HOs SHOs
- and \( l \): \( l \in \text{children } t \)
- and skrl: last-lbl \( l \in SKr (HOs (length-lbl t)) (SHOs (length-lbl t)) \)
- and \( l' \): \( l' \in \text{children } l \)
- and skrl': last-lbl \( l' \in SKr (HOs (length-lbl l)) (SHOs (length-lbl l)) \)
- shows vals (rho (length-lbl \( l' \) \( p \) \( l' \)) \( = \) vals (rho (length-lbl \( l \)) \( p \) \( l \)) \)
  - **using** assms by (auto simp: SKr-def sho-correct vals)

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

**lemma Lynch-6-16-c:**
- **assumes** commR: EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))
  - (is EIG-commPerRd (HOs ?r) -)
- and \( l \): ¬(is-leaf \( l \))
- shows card \{ \( l' \in \text{children } l \). last-lbl \( l' \in SKr (HOs ?r) (SHOs ?r) \}
  - > card (children \( l \)) div 2
  - (is card ?lhs > -)
  - **proof**
  - let \( ?skr = SKr (HOs ?r) (SHOs ?r) \)
  - have last-lbl \( \cdot ?lhs = ?skr - set-lbl l \)
  - **proof**
    - from children-last-set[OF \( l \)]
    - show last-lbl \( \cdot ?lhs \subseteq ?skr - set-lbl l \)
      - by (auto simp: children-length)
  - **next**
    - fix \( p \)
    - assume \( p: p \in ?skr \ p \notin \text{set-lbl } l \)
    - with children-last-set[OF \( l \)]
have \( p \in \text{last-lbl \ ' children } l \) by auto
with \( p \) have \( p \in \text{last-lbl \ ' ?lhs} \)
by (auto simp: image-def children-length)

\}
thus \( ?skr - \text{set-lbl } l \subseteq \text{last-lbl \ ' ?lhs} \) by auto
qed

moreover
from \( \text{last-lbl-inj-on-children[of } l \) \]
have inj-on last-lbl \( ?lhs \) by (auto simp: inj-on-def)
ultimately
have card \( ?lhs \) \( = \) card \( \langle ?skr - \text{set-lbl } l \rangle \) by (auto dest: card-image)

finally have card \( ?lhs \) \( \geq \) (card \( ?skr \)) \( - ?r \)
using card-set-lbl[of \( l \) \]
by simp

moreover
from commR have card \( ?skr \) \( > \) \( (N + f) \) div 2
by (auto simp: EIG-commPerRd-def)
with not-leaf-length[of \( l \) \]
have (card \( ?skr \) - \( ?r \) \( > \) \( (N - ?r) \) div 2 by auto
with card-children[of \( l \) \]
have (card \( ?skr \) - \( ?r \) \( > \) \( \text{card (children } l \) \) 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 run: \( \langle \text{SHORun EIG-M rho HOs SHOs} \) \]
and commR: \( \langle \text{EIG-commPerRd (HOs (length-lbl } l \) (SHOs (length-lbl } l \) \) \]
and notleaf: \( \langle \text{is EIG-commPerRd (HOs } ?r \) \- \)
and unif: \( \langle \land \langle l' \in \text{children } l; \)
\langle 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 unif
have card \( \{ l' \in \text{children } l; \text{last-lbl } l' \in \text{SKr (HOs } ?r \) \) \( \langle SHOs } ?r \) \}
\( \leq \) card \( \langle \text{newvals (rho (Suc } f \) p \) \text{ (children } l \) \)
by (auto intro: card-mono)
with lynch-6-16-c[of \( l \) \( \text{SHOs, OF } \text{commR notleaf} \)
have maj: \( \text{has-majority } v \) \( \text{newvals (rho (Suc } f \) p \) \text{ (children } l \) \)
by (simp add: has-majority-def)

from run have check-newvals \( \text{ (rho (Suc } f \) p \) \)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq
    nextState-def EIG-nextState-def next-end-def)

with maj notleaf obtain w
  where wmaj: has-majority w (newvals (rho (Suc f) p)) (children l)
    and wupd: newvals (rho (Suc f) p) l = w
  by (auto simp: check-newvals-def)

from maj wmaj have w = v by (auto simp: has-majority-def elim: abs-majoritiesE')

with wupd show ?thesis by simp
qed

A node whose label l ends with a process which is well heard at round length-lbl l will have its newvals field set (at round f + 1) to the “fixed-up” value given by vals.

lemma lynch-6-16-d:
  assumes run: SHORun EIG-M rho HOs SHOs
    and commR: ∀ r. EIG-commPerRd (HOs r) (SHOs r)
    and notroot: l ∈ children t
    and skr: last-lbl l ∈ SKr (HOs (length-lbl t)) (SHOs (length-lbl t))
        (is ∈ SKr (HOs (?len t)) -)
  shows newvals (rho (Suc f) p) l = fixupval (vals (rho (?len) p) l)
    (is ?P l)
  using notroot skr proof (induct Suc f - (?len l) arbitrary: l t)

fix l t
  assume 0 = Suc f - (?len l)
with length-lbl[of l] have leaf: is-leaf l by (simp add: is-leaf-def)

from run have check-newvals (rho (Suc f) p)
  by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq
    nextState-def EIG-nextState-def next-end-def)

with leaf show ?P l
  by (auto simp: check-newvals-def is-leaf-def)

next
fix k l t
  assume ih: \ l' t'.
    [k = Suc f - length-lbl l'; l' ∈ children t';
     last-lbl l' ∈ SKr (HOs (?len t')) (SHOs (?len t'))] \[ \Rightarrow \] ?P l'
  and flk: Suc k = Suc f - ?len l
  and notroot: l ∈ children t
  and skr: last-lbl l ∈ SKr (HOs (?len t)) (SHOs (?len t))

let ?v = fixupval (vals (rho (?len) p) l)
from flk have notlf: ¬(is-leaf l) by (simp add: is-leaf-def)

{ fix l'
  assume l': l' ∈ children l
    and skr': last-lbl l' ∈ SKr (HOs (?len l)) (SHOs (?len l))

155
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:**

\[
\text{from } \text{run notroot skr } l' \ \text{skr'}
\]

**have** \( \text{vals} (\rho (\text{len } l) p) l' = \text{vals} (\rho (\text{len } l) p) \ l \)

**by** (rule lynch-6-16-a)

**moreover**

**from** \( \text{flk } l' \ \text{have} \ k = \text{Suc } f - \text{len } l' \)

**by** (simp add: children-length)

**from** this \( l' \ \text{skr'} \ \text{have} \ ?P l' \)

**by** (rule ih)

**ultimately**

**have** \( \text{newvals} (\rho (\text{Suc } f) p) l' = ?v \)

**using** notroot \( l' \)

\[
}\}

**with** \( \text{run commR notlf} \) **show** \( ?P \ l \)

**by** (auto intro: newvals-skr-uniform)

qed
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 \text{assms} \ \text{proof} \ (\text{auto simp: subtrees-def append-eq-append-conv2})
fix \( xs \)
assume \( \text{Rep-Label} \ l' \ @ \ xs = \text{Rep-Label} \ l' \)
hence \( \text{Rep-Label} \ l' = \text{Rep-Label} \ l'' \ @ \ xs \) by (rule sym)
thus \( \exists \ ys. \ \text{Rep-Label} \ l' = \text{Rep-Label} \ l'' @ \ ys .. \)
qed

\text{lemma subtrees-cases:}
assumes \( l' \); \( l' \in \text{subtrees} \ l \)
and \( \text{self} \); \( l' = l \implies P \)
and \( \text{child} \); \( \forall c. \ [ c \in \text{children} \ l \ ]; \ l' \in \text{subtrees} \ c \implies P \)
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: \text{Rep-Label-inject})
next
fix \( p t' \)
assume \( \text{cons} \); \( t = p \# t' \)
from \( \text{Rep-Label[of} \ l'] \ t \) have \( \text{length} \ (\text{Rep-Label} \ l @ t) \leq \text{Suc} \ f \)
by (auto simp: \text{Label-def})
with \( \text{cons} \) have \( \text{notleaf} \); \( \neg (\text{is-leaf} \ l) \)
by (auto simp: \text{is-leaf-def \text{length-lbl-def}})

let \( \text{?c} = \text{Abs-Label} \ (\text{Rep-Label} \ l @ [p]) \)
from \( \text{t cons} \ \text{Rep-Label[of} \ l'] \) have \( p: p \notin \text{set-lbl} \ l \)
by (auto simp: \text{Label-def \text{set-lbl-def}})
with \( \text{notleaf} \) have \( \text{c: ?c \in children} \ l \)
by (auto simp: \text{children-def})
moreover
from \( \text{notleaf} \) \( p \) have \( \text{Rep-Label} \ l @ [p] \in \text{Label} \)
by (auto simp: \text{Rep-Label-append})
hence \( \text{Rep-Label} \ ?c = (\text{Rep-Label} \ l @ [p]) \)
by (auto simp: \text{Abs-Label-inverse})
with \( \text{cons} \) \( t \) have \( l' \in \text{subtrees} \ ?c \)
by (auto simp: \text{subtrees-def})
ultimately show \( \text{thesis} \) by blast
qed
thus \( \text{thesis} \) by (auto elim!: \text{self \ child})
qed

\text{lemma subtrees-leaf:}
assumes \( l \); \( \text{is-leaf} \ l \) 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 ?thesis by (simp add: children-def)

qed

lemma 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 ?thesis by (simp add: Rep-Label-inject)

qed

A set $C$ of labels is a subcovering w.r.t. label $l$ if for all leaf subtrees $s$ of $l$
there exists some label $h \in C$ such that $s$ is a subtree of $h$ and $h$ is a subtree
of $l$.

definition subcovering where

subcovering $C \ l \equiv \forall s \in \text{subtrees } l. \ is-leaf \ s \longrightarrow (\exists h \in C. \ h \in \text{subtrees } l \land s \in \text{subtrees } h)$

A covering is a subcovering w.r.t. the root node.

abbreviation covering where

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 lynch-6-18-a:

assumes SHORun EIG-M rho HOs SHOs

and $\forall r. \ EIG\text{-commPerRd } (\text{HOs } r) (\text{SHOs } r)$

and $l \in \text{children } t$

and $\text{last-lbl } l \in \text{SKr } (\text{HOs } (\text{length-lbl } t)) (\text{SHOs } (\text{length-lbl } t))$

shows common rho $l$

using assms

by (auto simp: common-def lynch-6-16-d lynch-6-15
 intro: arg-cong[where $f=\text{fixupval}$])

lemma lynch-6-18-b:

assumes run: SHORun EIG-M rho HOs SHOs

and $\text{commG: } EIG\text{-commGlobal } HOs \ SHOs$

and $\text{commR: } \forall r. \ EIG\text{-commPerRd } (\text{HOs } r) (\text{SHOs } r)$

shows covering $\{l. \exists t. \ l \in \text{children } t \land \text{last-lbl } l \in (\text{SK } HOs \ SHOs)\}$

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 (simp add: is-leaf-def)

with commG have \( N < \text{card} (\text{SK HOs SHOs}) + \text{card} (\text{set-lbl } l) \)
  by (simp add: EIG-commGlobal-def)

hence \( \exists q \in \text{set-lbl } l \cdot q \in \text{SK HOs SHOs} \)
  by (auto dest: majorities-intersect)

then obtain \( l_1 \ q \ l_2 \) where
  \( l : \text{Rep-Label } l = (l_1 \ @ [q]) @ l_2 \) and \( q \in \text{SK HOs SHOs} \)

unfolding set-lbl-def by (auto intro: split-list-propE)

let \( ?h = \text{Abs-Label } (l_1 @ [q]) \)
from Rep-Label[of \( l \)] \( l @ [q] \in \text{Label} \)
  by (simp add: Label-def)

hence \( \text{length-lbl } ?h \neq 0 \)
  by (simp add: length-lbl-def)

hence \( ?h \neq \text{root-node} \)
  by auto

then obtain \( t \) where \( t : ?h \in \text{children } t \)
  by (auto simp: root-iff-no-child)

moreover from reph q have \( \text{last-lbl } ?h \in \text{SK HOs SHOs} \)
  by (simp add: last-lbl-def)

moreover from reph l have \( l \in \text{subtrees } ?h \)
  by (simp add: subtrees-def)

ultimately show \( \exists h. (\exists t. h \in \text{children } t) \land \text{last-lbl } h \in \text{SK HOs SHOs} \land l \in \text{subtrees } h \)
  by blast

qed

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

**Lemma lynch-6-19-a:**

**Assumes** \( \text{cov} : \text{subcovering } C \ l \)

and \( l : l \notin C \)

and \( e : e \in \text{children } l \)

**Shows** \( \text{subcovering } C \ e \)

**Proof** (clarsimp simp: subcovering-def)

fix \( s \)

assume \( s : s \in \text{subtrees } e \)

and leaf: is-leaf \( s \)

from \( s \text{ children-in-subtree}(\text{OF } e) \) have \( s \in \text{subtrees } l \)
  by (rule subtrees-trans)

with leaf \( \text{cov} \) obtain \( h \) where \( h : h \in C \ h \in \text{subtrees } l \ s \in \text{subtrees } h \)
  by (auto simp: subcovering-def)

with \( l \) obtain \( e' \) where \( e' : e' \in \text{children } l \ h \in \text{subtrees } e' \)
  by (auto elim: subtrees-cases)

from \( (s \in \text{subtrees } h) \land h \in \text{subtrees } e') \) have \( s \in \text{subtrees } e' \)
  by (rule subtrees-trans)

with \( s \) have \( e \in \text{subtrees } e' \lor e' \in \text{subtrees } e \)
  by (rule subtrees-tree)

with \( e \ e' \) have \( e' = e \)
  by (auto dest: children-subtrees-equal)

with \( e' \) show \( \exists h \in C. \ h \in \text{subtrees } e \land s \in \text{subtrees } h \)
  by blast

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

**Lemma Lynch-6-19-b:**

**Assumes**
- $\text{run}$: $\text{SHORun} \ EIG-M \ \rho \ \text{HOs} \ \text{SHOs}$
- and $\text{cov}$: subcovering $C \ l$
- and $\text{com}$: $\forall l' \in C. \ \text{common} \ \rho \ l'$

**Shows** common $\rho \ l$

**Using** $\text{cov}$ **Proof** (induct $\text{Suc} \ f - \text{length-lbl} \ l$ arbitrary: $l$)

**Fix** $l$

**Assume** $0$: $0 = \text{Suc} \ f - \text{length-lbl} \ l$
- and $C$: subcovering $C \ l$
  **From** $0 \ \text{length-lbl}[\text{of} \ l]$ **Have** $\text{is-leaf} \ l$
  - **By** ($\text{simp add: is-leaf-def}$)
  **With** $C$ **Obtain** $h$ **Where** $h$: $h \in C \ h \in \text{subtrees} \ l \ l \in \text{subtrees} \ h$
  - **By** ($\text{auto simp: subcovering-def}$)
  **Hence** $l \in C$ **By** ($\text{auto dest: subtrees-antisym}$)
  **With** $\text{com}$ **Show** common $\rho \ l$ ..

**Next**

**Fix** $k \ l$

**Assume** $k$: $\text{Suc} \ k = \text{Suc} \ f - \text{length-lbl} \ l$
- and $C$: subcovering $C \ l$
  **And** $\text{ih}$: $\forall l' \ [ k = \text{Suc} \ f - \text{length-lbl} \ l' ; \ \text{subcovering} \ C \ l' ] \Rightarrow \ \text{common} \ \rho \ l'$

**Show** common $\rho \ l$

**Proof** (cases $l \in C$)
  - **Case** $\text{True}$
    **With** $\text{com}$ **Show** ?thesis ..
  - **Next**
    **Case** $\text{False}$
    **With** $C$ **Have** $\forall e \in \text{children} \ l. \ \text{subcovering} \ C \ e$
    - **By** ($\text{blast intro: Lynch-6-19-a}$)
  **Moreover**
  **From** $k$ **Have** $\forall e \in \text{children} \ l. \ k = \text{Suc} \ f - \text{length-lbl} \ e$
  - **By** ($\text{auto simp: children-length}$)
  **Ultimately**
  **Have** $\text{com-ch}$: $\forall e \in \text{children} \ l. \ \text{common} \ \rho \ e$
  - **By** ($\text{blast intro: ih}$)

**Show** ?thesis

**Proof** ($\text{clarsimp simp: common-def}$)

**Fix** $p \ q$
  **From** $k$ **Have** $\text{notleaf}$: $\neg(\text{is-leaf} \ l)$ **By** ($\text{simp add: is-leaf-def}$)
  **Let** $?r = \text{Suc} \ f$
  **From** $\text{com-ch}$
  **Have** $\forall e \in \text{children} \ l. \ \text{newvals} \ (\rho \ ?r \ p) \ e = \text{newvals} \ (\rho \ ?r \ q) \ e$
  - **By** ($\text{auto simp: common-def}$)
  **Hence** $\forall w. \ \{ e \in \text{children} \ l. \ \text{newvals} \ (\rho \ ?r \ p) \ e = w \}$
    $= \{ e \in \text{children} \ l. \ \text{newvals} \ (\rho \ ?r \ q) \ e = w \}$
  - **By** $\text{auto}$
  **Moreover**

160
The root of the tree is a common node.

**Lemma lynch-6-20:**

**Assumes** run: SHORun EIG-M rho HOs SHOs

and commG: EIG-commGlobal HOs SHOs

and commR: \( \forall r. \ EIG\text{-}commPerRd (HOs r) (SHOs r) \)

**Shows** common rho root-node

**Using** run lynch-6-18-b[OF assms]

**Proof** (rule lynch-6-19-b, clarify)

fix \( l \) \( t \)

assume \( l \in \text{children} \ t \) \( \text{last-lbl} \ l \in \text{SK} \ HOs \ SHOs \)

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 \( r < \text{Suc} \ f \) then None

else Some (newvals (rho (Suc \( f \)) \( p \)) root-node))

(is \( ?P \) \( r \))

**Proof** (induct \( r \))

from run show \( ?P \ 0 \)

by (auto simp: EIG-SHOMachine-def SHORun-eq HOinitConfig-eq

initState-def EIG-initState-def)

**Next**

fix \( r \)

assume \( ih: \ ?P \ r \)

from run obtain \( \mu p \)

where EIG-nextState \( r \) \( p \) \( (rho \ r \ p) \mu p \ (rho \ (Suc \ r) \ p) \)
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
qede

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 \( ?r = \text{Suc} \ f \)

\{
  \text{fix} \ l' \\
  \text{assume} \ l': \ l' \in \text{children root-node} \\
  \text{and} \ skr: \ \text{last-lbl} \ l' \in SKr (\text{HOs} \ 0) (\text{SHOs} \ 0) \\
  \text{with} \ \text{run} \ v \ \text{have} \ \text{vals} (\rho \ (?\text{len} \ l') \ p) \ l' = \text{Some} \ v \\
  \text{by} (\text{auto dest: sho-correct-vals simp: SKr-def})
\}

\text{moreover} \\
\text{from} \ \text{run commR} \ l' \ skr \\
\text{have} \ \text{newvals} (\rho \ ?r \ p) \ l' = \text{fixupval} (\text{vals} (\rho \ (?\text{len} \ l') \ p) \ l') \\
\text{by} (\text{auto intro: lynch-6-16-d})

\text{ultimately} \\
\text{have} \ \text{newvals} (\rho \ ?r \ p) \ l' = v \text{ by simp}
\}

\text{with} \ \text{run commR root-node-not-leaf} \\
\text{have} \ \text{newvals} (\rho \ ?r \ p) \ \text{root-node} = v \\
\text{by} (\text{auto intro: newvals-skr-uniform})

\text{with} \ \text{dp} \ \text{show} \ \text{?thesis} \text{ by} (\text{simp add: decide[OF run]})

qed

Termination is trivial for \( EIGByz_f \).

\textbf{theorem} \ Termination:\n\text{assumes} \ \text{SHORun} \ EIG-M \ \rho \ \text{HOs} \ \text{SHOs} \\
\text{shows} \ \exists \ r \ v. \ \text{decide} (\rho \ r \ p) = \text{Some} \ v \\
\text{using} \ \text{assms} \ \text{by} (\text{auto simp: decide})

10.8 \ \textit{EIGByz}_f \text{ Solves Weak Consensus}

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

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

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

**Theorem** eig-weak-consensus-fg:

assumes run: fg-run EIG-M rho HOs SHOs (λr q. undefined)

and commR: ∀ r. EIG-commPerRd (HOs r) (SHOs r)

and commG: EIG-commGlobal HOs SHOs

shows weak-consensus (λp. the (vals (state (rho 0) p) root-node))

decide (state o rho)

(is weak-consensus ?inits - -)

**Proof** (rule local-property-reduction[OF run weak-consensus-is-local])

fix crun

assume crun: CSHORun EIG-M crun HOs SHOs (λr q. undefined)

and init: crun 0 = state (rho 0)

from crun have SHORun EIG-M crun HOs SHOs by (unfold SHORun-def)

from this commR commG

have weak-consensus (λp. the (vals (crun 0 p) root-node)) decide crun

by (rule eig-weak-consensus)

with init show weak-consensus ?inits decide crun

by (simp add: o-def)

qed

end

11 Conclusion

In this contribution we have formalized the Heard-Of model in the proof assistant Isabelle/HOL. We have established a formal framework, in which fault-tolerant distributed algorithms can be represented, and that caters for different variants (benign or malicious faults, coordinated and uncoordinated algorithms). We have formally proved a reduction theorem that 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 [10], but our proof here is an order of magnitude shorter, although we prove safety and liveness properties, whereas only safety was considered in [10].

We also emphasize that the uniform characterization of fault assumptions via communication predicates in the HO model lets us consider the effects of transient failures, contrary to standard models that consider only permanent failures. For example, our correctness proof for the EIGByz 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


