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**  

2 **Heard-Of Algorithms**  
   2.1 The Consensus Problem  
   2.2 A Generic Representation of Heard-Of Algorithms  

3 **Reduction Theorem**  
   3.1 Fine-Grained Semantics  
   3.2 Properties of the Fine-Grained Semantics  
   3.3 From Fine-Grained to Coarse-Grained Runs  
   3.4 Locally Similar Runs and Local Properties  
   3.5 Consensus as a Local Property  

4 **Utility Lemmas About Majorities**  

5 **Verification of the One-Third Rule Consensus Algorithm**  
   5.1 Model of the Algorithm  
   5.2 Communication Predicate for One-Third Rule  
   5.3 The One-Third Rule Heard-Of Machine  
   5.4 Proof of Integrity  
   5.5 Proof of Agreement  
   5.6 Proof of Termination  
   5.7 One-Third Rule Solves Consensus  

6 **Verification of the UniformVoting Consensus Algorithm**  
   6.1 Model of the Algorithm  
   6.2 Communication Predicate for UniformVoting  
   6.3 The UniformVoting Heard-Of Machine  
   6.4 Preliminary Lemmas  
   6.5 Proof of Irrevocability, Agreement and Integrity  
   6.6 Proof of Termination  
   6.7 UniformVoting Solves Consensus  

7 **Verification of the LastVoting Consensus Algorithm**  
   7.1 Model of the Algorithm  
   7.2 Communication Predicate for LastVoting  
   7.3 The LastVoting Heard-Of Machine  

2
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 Consen-
sus 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, Ut,E,α, AT,E,α, and EIG-
Byzf, are designed to tolerate value failures, and solve a weaker variant of
the Consensus problem.

A preliminary report on the formalization of the LastVoting algorithm in the
of the reduction theorem relating coarse-grained and fine-grained executions,
and [5] reports on the formal verification of the Ut,E,α, AT,E,α, and EIGByzf
algorithms.

theory HOModel
imports Main
begin

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

2 Heard-Of Algorithms

2.1 The Consensus Problem

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

More formally, we represent runs of algorithms as ω-sequences of configu-
rations (vectors of process states). Hence, a run is modeled as a function
of type nat ⇒ 'proc ⇒ 'pst where type variables 'proc and 'pst represent
types of processes and process states, respectively. The Consensus property
is expressed with respect to a collection vals of initially proposed values (one
per process) and an observer function dec::'pst ⇒ val option that retrieves
the decision (if any) from a process state. The Consensus problem is stated
as the conjunction of the following properties:

Integrity. Processes can only decide initially proposed values.

Agreement. Whenever processes p and q decide, their decision values must
be the same. (In particular, process p may never change the value it
decides, which is referred to as Irrevocability.)

**Termination.** Every process decides eventually.

The above properties are sometimes only required of non-faulty processes, since nothing can be required of a faulty process. The Heard-Of model does not attribute faults to processes, and therefore the above formulation is appropriate in this framework.

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

definition consensus :: ('proc ⇒ 'val) ⇒ ('pst ⇒ 'val option) ⇒ ('proc,'pst) run ⇒ bool

where
consensus vals dec rho ≡
(∀ n p v. dec (rho n p) = Some v ⇒ v ∈ range vals)
∧ (∀ m n p q v w. dec (rho m p) = Some v ∧ dec (rho n q) = Some w
              ⇒ v = w)
∧ (∀ p. ∃ n. dec (rho n p) ≠ None)
```

A variant of the Consensus problem replaces the Integrity requirement by

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

```plaintext
definition weak-consensus where
weak-consensus vals dec rho ≡
(∀ v. (∀ p. vals p = v) ⇒ (∀ n p w. dec (rho n p) = Some w ⇒ w = v))
∧ (∀ m n p q v w. dec (rho m p) = Some v ∧ dec (rho n q) = Some w
             ⇒ v = w)
∧ (∀ p. ∃ n. dec (rho n p) ≠ None)
```

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

**lemma consensus-then-weak-consensus:**
- **assumes** consensus vals dec rho
- **shows** weak-consensus vals dec rho
  - **using** assms by (auto simp: consensus-def weak-consensus-def image-def)

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

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

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

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

  case True
  then obtain \( w \) where \( w: \forall \ p. \ \text{vals} \ p = w .. \)
  with \( \text{dec} \ w \) 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 bc show \( ?\text{thesis} \)

unfolding \( \text{consensus-def} \) \( \text{weak-consensus-def} \) by (auto elim!: \( \text{integrity} \))

qed

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

2.2 A Generic Representation of Heard-Of Algorithms

Charron-Bost and Schiper [7] introduce the Heard-Of (HO) model for representing fault-tolerant distributed algorithms. In this model, algorithms execute in communication-closed rounds: at any round \( r \), processes only receive messages that were sent for that round. For every process \( p \) and round \( r \), the “heard-of set” \( \text{HO}(p, r) \) 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 [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 $HO(p, r)$ and $SHO(p, r)$. However, the code of a process does not have access to these sets. In particular, process $p$ has no way of determining if a message it received from another process $q$ corresponds to what $q$ should have sent or if it has been corrupted.

Certain algorithms rely on the assignment of “coordinator” processes for each round. Just as the collections $HO(p, r)$, the definitions assume an external coordinator assignment such that $coord(p, r)$ denotes the coordinator of process $p$ and round $r$. Again, the correctness of algorithms may depend on hypotheses about coordinator assignments – e.g., it may be assumed that processes agree sufficiently often on who the current coordinator is.

The following definitions provide a generic representation of HO and SHO algorithms in Isabelle/HOL. A (coordinated) HO algorithm is described by the following parameters:

- a finite type ‘proc of processes,
- a type ‘pst of local process states,
- a type ‘msg of messages sent in the course of the algorithm,
- a predicate $CinitState$ such that $CinitState p st crd$ is true precisely of the initial states $st$ of process $p$, assuming that $crd$ is the initial coordinator of $p$,
- a function $sendMsg$ where $sendMsg r p q st$ yields the message that process $p$ sends to process $q$ at round $r$, given its local state $st$, and
- a predicate $CnextState$ where $CnextState r p st msgs crd st'$ characterizes the successor states $st'$ of process $p$ at round $r$, given current state $st$, the vector $msgs :: \text{‘proc} \Rightarrow \text{‘msg option}$ of messages that $p$ received at round $r$ ($msgs q = None$ indicates that no message has been received from process $q$), and process $crd$ as the coordinator for the following round.

Note that every process can store the coordinator for the current round in its local state, and it is therefore not necessary to make the coordinator a parameter of the message sending function $sendMsg$.

We represent an algorithm by a record as follows.

```isabelle
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** \( \text{isNCAlgorithm} \) where
\[
isNCAlgorithm \ alg = \left( \forall p \ st \ crd \ crd'. \ C_{\text{initState}} \ alg \ p \ st \ crd = C_{\text{initState}} \ alg \ p \ st \ crd' \right) \land \left( \forall r \ p \ st \ msgs \ crd \ crd' \ st'. \ C_{\text{nextState}} \ alg \ r \ p \ st \ msgs \ crd \ st' = C_{\text{nextState}} \ alg \ r \ p \ st \ msgs \ crd' \ st' \right)
\]

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

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

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** \( \text{CSHOinitConfig} \) where
\[
\text{CSHOinitConfig} \ A \ cfg \ (\text{coord}::'\text{proc} \ \text{coord}) \equiv \forall p. \ C_{\text{initState}} \ A \ p \ (cfg \ p) \ (\text{coord} \ p)
\]
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

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

Predicate \( \text{CSHOnextConfig} \) uses the preceding function and the algorithm’s \( C\text{nextState} \) 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

\[
\text{CSHOnextConfig} \ A \ r \ \text{cfg} \ \text{HO} \ \text{SHO} \ \text{coord} \ \text{cfg}^{'}) \equiv \\
\forall p. \exists \mu \in \text{SHOmsgVectors} \ A \ r \ p \ \text{cfg} \ (\text{HO} \ p) \ (\text{SHO} \ p). \\
\text{CnextState} \ A \ r \ p \ (\text{cfg} \ p) \ \mu \ (\text{coord} \ p) \ (\text{cfg}^{'} \ p)
\]

**definition \( \text{CSHORun} \)** where

\[
\text{CSHORun} \ A \ \rho \ \text{HOs} \ \text{SHOs} \ \text{coords} \equiv \\
\text{CHOinitConfig} \ A \ \rho \ 0 \ \text{coords} \ 0 \ \\
\wedge (\forall r. \text{CSHOnextConfig} \ A \ r \ (\rho \ r) \ (\text{HOs} \ r) \ (\text{SHOs} \ r) \ (\text{coords} \ (\text{Suc} \ r))) \\
(\rho \ (\text{Suc} \ r)))
\]

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

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

**lemma \( \text{HOinitConfig-eq} \)**:

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

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

**definition \( \text{SHOnextConfig} \)** where

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

**lemma \( \text{SHOnextConfig-eq} \)**:

\[
\text{SHOnextConfig} \ A \ r \ \text{cfg} \ \text{HO} \ \text{SHO} \ \text{cfg}^{'} = \\
(\forall p. \exists \mu \in \text{SHOmsgVectors} \ A \ r \ p \ \text{cfg} \ (\text{HO} \ p) \ (\text{SHO} \ p). \\
\text{nextState} \ A \ r \ p \ (\text{cfg} \ p) \ \mu \ (\text{cfg}^{'} \ p))
\]

by (auto simp: \( \text{SHOnextConfig-def} \ \text{CSHOnextConfig-def} \ \text{SHOmsgVectors-def} \ \text{nextState-def} \))
definition **SHORun** where

\[ \text{SHORun } A \rho \text{ HOs SHOs } \equiv \]
\[ \text{CSHORun } A \rho \text{ HOs SHOs } (\lambda r. \text{undefined}) \]

**lemma** **SHORun-eq**:

\[ \text{SHORun } A \rho \text{ HOs SHOs } = \]
\[ (\text{HOinitConfig } A (\rho \ 0)) \]
\[ \land \ (\forall r. \ \text{SHOnextConfig } A \ r \ (\rho \ r) \ (\text{HOs } r) \ (\text{SHOs } r) \ (\rho \ (\text{Suc } r))) \]

by (auto simp: **SHORun-def** CSHORun-def HOinitConfig-def SHOnextConfig-def)

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

definition **HOrcvdMsgs** where

\[ \text{HOrcvdMsgs } A \ r \ p \ \text{HO } \text{cfg} \equiv \]
\[ \lambda q. \ \text{if } q \in \text{HO } \text{then Some } (\text{sendMsg } A \ r \ q \ p \ (\text{cfg } q)) \text{ else None} \]

**lemma** **SHOmsgVectors-HO**:

\[ \text{SHOmsgVectors } A \ r \ p \ \text{cfg } \text{HO} \text{ HO} = \{\text{HOrcvdMsgs } A \ r \ p \ \text{HO} \ \text{cfg}\} \]

unfolding **SHOmsgVectors-def** HOrcvdMsgs-def by auto

With coordinators

definition **CHOnextConfig** where

\[ \text{CHOnextConfig } A \ r \ \text{cfg } \text{HO } \text{coord } \text{cfg}' \equiv \]
\[ \text{CSHOnextConfig } A \ r \ \text{cfg } \text{HO } \text{coord } \text{cfg}' \]

**lemma** **CHOnextConfig-eq**:

\[ \text{CHOnextConfig } A \ r \ \text{cfg } \text{HO } \text{coord } \text{cfg}' = \]
\[ (\forall p. \ \text{CnextState } A \ r \ p \ (\text{cfg } p) \ (\text{HOrcvdMsgs } A \ r \ p \ (\text{HO} \ p) \ \text{cfg}) \]
\[ (\text{coord } p) \ (\text{cfg}' \ p)) \]

by (auto simp: **CHOnextConfig-def** CSHOnextConfig-def SHOmsgVectors-HO)

definition **CHORun** where

\[ \text{CHORun } A \rho \text{ HOs coords } \equiv \text{CSHORun } A \rho \text{ HOs HOs coords} \]

**lemma** **CHORun-eq**:

\[ \text{CHORun } A \rho \text{ HOs coords } = \]
\[ (\text{CHOinitConfig } A (\rho \ 0) \ (\text{coords } 0)) \]
\[ \land \ (\forall r. \ \text{CHOnextConfig } A \ r \ (\rho \ r) \ (\text{HOs } r) \ (\text{coords } (\text{Suc } r)) \ (\rho \ (\text{Suc } r)))) \]

by (auto simp: **CHORun-def** CSHORun-def CHOinitConfig-def CHOnextConfig-def)

Without coordinators

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

definition \text{HORun} \ where \\
\text{HORun (A \ rho \ \text{HOs})} \\
\equiv \text{SHORun (A \ rho \ \text{HOs} \ \text{HOs})}

lemma \text{HORun-eq}: 
\text{HORun (A \ rho \ \text{HOs})} \\
= \left( \text{HOinitConfig (A \ (\rho \ 0))} \wedge \left( \forall \ r. \text{HOnextConfig (A \ r \ (\rho \ r) \ (\text{HOs} \ r)) (\rho \ (\text{Suc} \ r))} \right) \right) \\
\text{by (auto simp: HORun-def SHORun-eq HOnextConfig-def)}

The following derived proof rules are immediate consequences of the definition of \text{CHORun}; they simplify automatic reasoning.

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

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

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

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

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

record ('proc, 'pst, 'msg) HOMachine = ('proc, 'pst, 'msg) CHOAlgorithm +
\(HO\text{commPerRd}::\text{proc HO} \Rightarrow \text{bool}\)
\(HO\text{commGlobal}::(\text{nat} \Rightarrow \text{proc HO}) \Rightarrow \text{bool}\)

\text{record} (\text{'proc, 'pst, 'msg}) CPU\text{Machine} = (\text{'proc, 'pst, 'msg}) CPU\text{Algorithm} +
\text{CPUcommPerRd}::(\text{nat} \Rightarrow \text{proc HO}) \Rightarrow (\text{proc HO}) \Rightarrow \text{bool}
\text{CPUcommGlobal}::(\text{nat} \Rightarrow \text{proc HO}) \Rightarrow (\text{nat} \Rightarrow \text{proc coord}) \Rightarrow \text{bool}

\text{record} (\text{'proc, 'pst, 'msg}) SHOM\text{Machine} = (\text{'proc, 'pst, 'msg}) SHO\text{Algorithm} +
\text{SHOcommPerRd}::(\text{'proc HO}) \Rightarrow (\text{'proc HO}) \Rightarrow \text{bool}
\text{SHOcommGlobal}::(\text{nat} \Rightarrow \text{proc HO}) \Rightarrow (\text{nat} \Rightarrow \text{proc HO}) \Rightarrow \text{bool}

\text{record} (\text{'proc, 'pst, 'msg}) CS\text{HOM\text{Machine} = (\text{'proc, 'pst, 'msg}) CS\text{HAlgorithm} +}
\text{C\text{SHOcommPerRd}::(\text{'proc HO}) \Rightarrow (\text{'proc HO}) \Rightarrow \text{proc coord} \Rightarrow \text{bool}
\text{C\text{SHOcommGlobal}::(\text{nat} \Rightarrow \text{proc HO}) \Rightarrow (\text{nat} \Rightarrow \text{proc HO})}
\Rightarrow (\text{nat} \Rightarrow \text{'proc coord}) \Rightarrow \text{bool}

end — theory HOModel

theory Reduction
imports HOModel ../Stuttering-Equivalence/StutterEquivalence
begin

3 Reduction Theorem

We have defined the semantics of HO algorithms such that rounds are executed atomically, by all processes. This definition is surprising for a model of asynchronous distributed algorithms since it models a synchronous execution of rounds. However, it simplifies representing and reasoning about the algorithms. For example, the communication network does not have to be modeled explicitly, since the possible sets of messages received by processes can be computed from the global configuration and the collections of HO and SHO sets.

We will now define a more conventional “fine-grained” semantics where communication is modeled explicitly and rounds of processes can be arbitrarily interleaved (subject to the constraints of the communication predicates). We will then establish a \textit{reduction theorem} that shows that for every fine-grained run there exists an equivalent round-based (“coarse-grained”) run in the sense that the two runs exhibit the same sequences of local states of all processes, modulo stuttering. We prove the reduction theorem for the most general class of coordinated SHO algorithms. It is easy to see that the theorem equally holds for the special cases of uncoordinated or HO algorithms, and since we have in fact defined these classes of algorithms from the more general ones, we can directly apply the general theorem.

As a corollary, interesting properties remain valid in the fine-grained semantics if they hold in the coarse-grained semantics. It is therefore enough to verify such properties in the coarse-grained semantics, which is much eas-
ier to reason about. The essential restriction is that properties may not depend on states of different processes occurring simultaneously. (For example, the coarse-grained semantics ensures by definition that all processes execute the same round at any instant, which is obviously not true of the fine-grained semantics.) We claim that all “reasonable” properties of fault-tolerant distributed algorithms are preserved by our reduction. For example, the Consensus (and Weak Consensus) problems fall into this class. The proofs follow Chaouch-Saad et al. [4], where the reduction theorem was proved for uncoordinated HO algorithms.

3.1 Fine-Grained Semantics

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

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

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

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

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

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

14
round config = (\(\lambda p. 0\))
\(\land (\forall p. \text{CinitState}\ A p (\text{state config}\ p) (\text{coord}\ p))\)
\(\land \text{sent config} = (\lambda p. \{\})\)
\(\land \text{rcvd config} = (\lambda p q. \text{None})\)
\(\land \text{network config} = \{\}\)

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

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

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

**definition fg-send-msg where**

\(\text{fg-send-msg}\ A p q\ \text{config}\ \text{config}'\equiv\)
\(q \notin (\text{sent}\ \text{config}\ p)\)
\(\land \text{config}' = \text{config} \{\}\)
\(\land\text{sent} := (\text{sent}\ \text{config})(p := (\text{sent}\ \text{config}\ p) \cup \{q\})\),
\(\text{network} := \text{network}\ \text{config} \cup\)
\(\{(p, \text{round config}\ p, q,\)
\(\text{sendMsg}\ A (\text{round config}\ p) p q (\text{state config}\ p))\}\ \}\)

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 fg-rcv-msg where**

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

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 $C_{nextState}$ 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**

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

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

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

**definition fg-next-config where**

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

- $(\exists q. fg-send-msg A p q config config')$
- $(\exists q. fg-rcv-msg p q HO SHO config config')$
- $(fg-local A p HO crd config config')$

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**

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

- $fg-init-config A (rho 0) (coords 0)$
- $(\forall i. \exists p. fg-next-config A p$
  - $(HOs (round (rho i) p) p)$
  - $(SHOs (round (rho i) p) p)$
  - $(coords (round (rho (Suc i)) p) p)$
  - $(rho i) (rho (Suc i)))$

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

**definition fg-start-round where**

$fg-start-round rho p r \equiv LEAST (n::nat). round (rho n) p = r$
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** \( \rho: \text{fg-run } A \rho \text{ HOs SHOs coords} \)

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

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

**Proof** –

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

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

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

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

by (auto simp: fg-run-def)

with \( \text{rd} \) show \( ?\text{thesis} \)

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

**Qed**

Round numbers never decrease.

**Lemma fg-round-numbers-increase:**

**Assumes** \( \rho: \text{fg-run } A \rho \text{ HOs SHOs coords and } n: n \leq m \)

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

**Proof** –

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

\{ 
  fix \( i \)
  have \( \text{round } (\rho n) \text{ } p \leq \text{round } (\rho (n+i)) \text{ } p \) (is \( ?P i \))
  proof (induct \( i \))
    show \( ?P 0 \) by simp
  next
    fix \( j \)
    assume \( \text{ih: } ?P j \)
    from \( \rho \) have \( \exists p'. \text{fg-next-config } A p' (\text{HOs } (\text{round } (\rho (n+j)) \text{ } p') \text{ } p') \)
    (\text{SHOs } (\text{round } (\rho (n+j)) \text{ } p') \text{ } p')
    (\text{coords } (\text{round } (\rho (\text{Suc } (n+j))) \text{ } p') \text{ } p')
    (\rho (n+j)) (\rho (\text{Suc } (n+j)))
    by (auto simp: fg-run-def)
    hence \( \text{round } (\rho (n+j)) \text{ } p \leq \text{round } (\rho (n + \text{Suc } j)) \text{ } p \)
    by (auto simp: fg-next-config-def fg-send-msg-def fg-rcv-msg-def fg-local-def)
    with \( \text{ih} \) show \( ?P (\text{Suc } j) \) by auto
    qed
  \}
  with \( k \) show \( ?\text{thesis} \) by simp
  qed

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

**Lemma** $fg$-same-round-same-state:
assumes $\rho$: $fg$-run A $\rho$ HOs SHO coords
and rd: round ($\rho$ m) $p = \text{round} (\rho$ n) $p$
shows $\text{state} (\rho$ m) $p = \text{state} (\rho$ n) $p$

**Proof**

{ 
fix $k$ $i$

have round $(\rho (k+i)) p = \text{round} (\rho k) p$
\implies state $(\rho (k+i)) p = \text{state} (\rho k) p$
(is $\forall R \ i \implies \forall S \ i$)

**Proof (induct $i$)**
show $\forall S \ 0 \ \text{by simp}$

next

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

qed

note aux = this
show $\forall$thesis

**Proof (cases $n \leq m$)**
case $\text{True}$
then obtain $k$ where $m = n + k$ by (auto simp: le-iff-add)
with rd show $\forall$thesis by (auto simp: aux)

next
case $\text{False}$
hence $m \leq n \ \text{by simp}$
then obtain $k$ where $n = m + k$ by (auto simp: le-iff-add)
with rd show $\forall$thesis by (auto simp: aux)

qed

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

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

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

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

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

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

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

proof (rule antisym)

from 1 have \( \text{round} (\rho \text{(fg-start-round} \rho p r)) p \leq \?rd \)
  by (rule fg-round NUMBERS increases[OF \rho])
thus \( r \leq \?rd \) by (simp add: fg-start-round[OF \rho])

next

show \(?rd \leq r\)
proof (rule ccontr)
  assume \( \neg \)thesis
  hence Suc \( r \leq \?rd \) by simp
  hence \( \text{fg-start-round} \rho p \text{(Suc} r) = \text{Suc} n \) by (rule fg-start-round)
  hence \( \text{fg-local} \ A p \text{(HOs} r p) \text{(coords} \text{(Suc} r) p) \rho n \text{(rho} \text{(Suc} n)) \)

next

from \( n \) show \( \text{Suc} r \leq \?rd \) by (simp add: fg-start-round[OF \rho])

finally show False by simp

qed

qed

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

lemma fg-local-transition-from-round:
assumes \( \rho: \text{fg-run} \ A \rho \text{ HOs SHOs coords} \)
obtains \( n \) where \( \text{round} (\rho n) p = r \)
  and \( \text{fg-start-round} \rho p \text{(Suc} r) = \text{Suc} n \)
  and \( \text{fg-local} \ A p \text{(HOs} r p) \text{(coords} \text{(Suc} r) p) \rho n \text{(rho} \text{(Suc} n)) \)

proof

have \( \text{fg-start-round} \rho p \text{(Suc} r) \neq 0 \) (is \(?start \neq 0\))
proof
  assume contr: \(?start = 0\)
  from \( \rho \) have \( \text{round} (\rho \text{(?start}) p = \text{Suc} r \) by (rule fg-start-round)
  with contr \( \rho \) show False by (auto simp: fg-run-def fg-init-config-def)
qed

then obtain \( n \) where \( n: \?start = \text{Suc} n \) by (auto simp: gr0-conv-Suc)
with \( \text{fg-start-round}[OF \rho, of} p \text{Suc} r \]

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

have 1: \( \text{round} (\rho n) p = r \)
proof (rule fg-round-between-start-rounds[OF \rho])
  have \( \text{fg-start-round} \rho p r < \text{fg-start-round} \rho p \text{(Suc} r) \)
    by (rule fg-start-round-strictmono[of \rho, THEN strict monoD]) simp
  with \( n \) show \( \text{fg-start-round} \rho p r \leq n \) by simp
next

from \( n \) show \( n < \?start \) by simp
qed

from \( \rho \) obtain \( p' \) where
  \( \text{fg-next-config} \ A p' \text{(HOs} \text{(round} \rho n) p') \rho p' \)
  \( \text{(SHOs} \text{(round} \rho n) p') \rho p' \)
  \( \text{(coords} \text{(round} \rho \text{(Suc} n)) p') p' \)
  \( \rho n \) \( \rho \text{(Suc} n)) \)


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

**Lemma fg-invariant1:**

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

**And** \( m::(p,r,q,m) \in \text{network}(\rho n) \) \( \Rightarrow \) \( \text{msg} n \)

**Shows** \( m = \text{sendMsg} A r p q (\text{state}(\rho (\text{fg-start-round} \rho p r)) p) \)

**Using** \( m \)

**Proof** (induct \( n \))

— the base case is trivial because the network is empty

**Assume** \( \text{msg} 0 \) with \( \rho \) show \( \text{thesis} \)

by (auto simp: \( \text{fg-run-def} \) \( \text{fg-init-config-def} \))

**Next**

**Fix** \( n \)

**Assume** \( m::\text{msg} (\text{Suc} n) \) and \( \text{ih}::\text{msg} n \Rightarrow \text{thesis} \)

**From** \( \rho \) obtain \( p' \) where

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

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

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

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

\( \Rightarrow \text{fg-next-config - - ?HO ?SHO ?crd ?cfg ?cfg'} \)

by (force simp: \( \text{fg-run-def} \))

**Thus** \( \text{thesis} \)

**Proof** (auto simp: \( \text{fg-next-config-def} \))

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

**Fix** \( q' \)

**Assume** \( \text{send}: \text{fg-send-msg} A p' q' ?cfg ?cfg' \)

**Show** \( \text{thesis} \)
proof (cases ?msg n)
  case True
  with ih show ?thesis .
next
  case False
  with send m' have 1: p' = p round ?cfg p = r
    and 2: m = sendMsg A r p q (state ?cfg p)
    by (auto simp: fg-send-msg-def)
  from rho 1 have state ?cfg p = state (rho (fg-start-round rho p r)) p
    by (auto simp: fg-start-round fg-same-round-same-state)
  with 1 2 show ?thesis by simp
qed
next
fix q'
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-rev-msg} steps of process \(q\), the proof is immediately reduced to the induction hypothesis.

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

next

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

next

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

qed

\end{itemize}

lemma \texttt{fg-invariant2b}:

\begin{itemize}
  \item assumes \(\texttt{rho}: \texttt{fg-run} A \texttt{rho} \texttt{HOs} \texttt{SHOs} \texttt{coords}\)
    \item and \(m: \texttt{rcvd} (\texttt{rho} n) q p = \texttt{Some} m\) (is \(\texttt{rcvd} n\))
    \item and \(\texttt{sho}: p \in \texttt{SHOs} (\texttt{round} (\texttt{rho} n) q) q\) (is \(p \in \texttt{SHOs}\) (?rd n) q)
  \item shows \(m = \texttt{sendMsg} A (\texttt{?rd} n) p q\)
    \item (state (\texttt{rho} (\texttt{fg-start-round rho} (\texttt{?rd} n)))) p)
    \item (is \(\texttt{?P} n\))
  \item using \(\texttt{m sho}\) proof (induct n)
    \item — The base case is trivial because \(q\) has not received any message initially
    \item assume \(\texttt{rcvd} 0\) with \(\texttt{rho show} \ ?P 0\)
      \item by \(\texttt{auto simp: fg-run-def fg-init-config-def}\)
  \item next
    \item fix \(n\)
    \item assume \(\texttt{rcvd}: \texttt{rcvd} (\texttt{Suc} n)\) and \(p: p \in \texttt{SHOs} (\texttt{?rd} (\texttt{Suc} n)) q\)
      \item and \(\texttt{ih}: \texttt{rcvd} n \implies p \in \texttt{SHOs} (\texttt{?rd} n) q \implies \texttt{?P} n\)
    \item — For the inductive step we again distinguish the possible transitions
    \item from \(\texttt{rho}\) obtain \(p'\) where
      \item \(\texttt{fg-next-config} A p' (\texttt{HOs} (\texttt{round} (\texttt{rho} n) p') p')\)
      \item \((\texttt{SHOs} (\texttt{round} (\texttt{rho} n) p') p')\)
\end{itemize}
(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 \( \text{ih p rd show \( ?\)thesis by simp} \)
next
case False
from \( rcvmsg \) obtain \( m' m'' \) where
  \( (q', \text{round } ?cfg p', p', m') \in \text{network } ?cfg \)
  \( \text{rcvd } ?cfg' = (\text{rcvd } ?cfg)(p' := (\text{rcvd } ?cfg p')(q' :=}
    \text{ if } q' \in ?SHO \text{ then Some } m' \text{ else Some } m'')) \)
by (auto simp: fg-rcv-msg-def split del: split-if-asm)
with \( \text{False rcvd p rd have } (p, ?rd n, q, m) \in \text{network } ?cfg \) by auto
with \( rho rd show ?\text{thesis} \) 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 \( p \text{ ih show } ?P (Suc n) \) 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 \( p \text{ ih show } ?P (Suc n) \) 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 \( fg\text{-}start\text{-}round \) instants), the sequences of process states in \( \rho \) and \( \sigma \) are easily seen to be stuttering equivalent, and this will be formally stated below.

**Definition** coarse-run where

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

**Theorem** reduction:

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

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

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

**From** \( \rho \) **Show** \( CHOinitConfig A \ (\text{Suc } r) (\text{Suc } r) \)

**Proof** (auto simp add: CHOinitConfig-def)

**Fix** \( r \)**

**Show** \( CSHOnextConfig A r \ (\text{Suc } r) \ (\text{Suc } r) \)

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

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

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

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

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

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

**Proof** (rule fg-same-round-same-state [OF \( \rho \)])

**Have** \( \text{cfg} : \text{Suc } r \)

**Unfolding** coarse-run-def **Proof** (rule fg-same-round-same-state [OF \( \rho \)])

**From** \( n \) **Show** \( \text{round } (\rho (fg\text{-}start\text{-}round \rho p r)) p = \text{round } \text{cfg} p \)

**Proof** (simp add: fg-start-round [OF \( \rho \)])

**Qed**

**From** \( \text{start} \) **Have** \( \text{cfg}' : \text{Suc } r \)

**By** (simp add: coarse-run-def)

**Have** \( \text{rcvd} : \text{rcvd } \text{cfg} p \in SHOmsgVectors A p (\text{Suc } r) \) \( HOs \)

**Proof** (auto simp: SHOmsgVectors-def)

**Fix** \( q \)

**Assume** \( q \in HOs \)

**With** \( n \) **Show** \( \exists m. \text{rcvd } \text{cfg} p q = \text{Some } m \) **By** (auto simp: fg-local-def)

**Next**

**Fix** \( q m \)

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

**With** \( \rho n \) **Show** \( q \in HOs \) **By** (auto simp: fg-invariant2a)
3.4 Locally Similar Runs and Local Properties

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

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

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

Local similarity is an equivalence relation.

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

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

\textbf{lemma} \texttt{locally-similar-trans} [trans]: \\
locally-similar g σ → locally-similar σ τ → locally-similar g τ \\
\texttt{by} (force simp add: locally-similar-def elim: stutter-equiv-trans)

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

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

**Lemma** \( \text{coarse-run-locally-similar} \):

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

**Shows** locally-similar (state \( \circ \rho \)) (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 \( ?\text{fg} \approx ?\text{cgr} \))

**Proof** (rule stutter-equivI)

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

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

from \( \rho \) show fg-start-round \( \rho \) \( 0 \) \( = \) \( 0 \)

by (rule fg-start-round-0)

next

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

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

next

fix \( r \) \( n \)

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

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

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

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

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

qed

next

show stutter-sampler id \( ?\text{cgr} \)

by (rule id-stutter-sampler)

next

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

by (auto simp: coarse-run-def)

qed

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

**Assumes** \( \rho \colon \text{fg-run A } \rho \text{ HO} \text{s SHO} \text{s coords} \) and \( P \colon \text{local-property } P \)

**And** coarse-correct:

\[ \wedge \text{crho. } [ \text{CSHORun A crho HO} \text{s SHO} \text{s coords; crho } 0 = \text{state } (\rhoho 0) ] \]

\[ \Rightarrow P \text{ crho} \]

**Shows** \( P \) (state \( \circ \rho \))

**Proof** –

have coarse-run \( \rho \) \( 0 \) = state (rho \( 0 \))

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

3.5 Consensus as a Local Property

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

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

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

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

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

proof

from sim have \((\lambda r. \sigma \ r \ p) \approx (\lambda r. \tau \ r \ p)\) by (simp add: locally-similar-def)
then obtain \(n\) where \(\sigma \ m \ p = \tau \ n \ p\) by (rule stutter-equiv-element-right)
with \(\text{trm show } \text{thesis by auto}\)
qed

theorem consensus-is-local: local-property (consensus vals dec)
proof (auto simp: local-property-def consensus-def)
fix \(\sigma \ t \ n \ p \ v\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ n \ p \ v. \ dec \ (\sigma \ n \ p) = \text{Some } v \rightarrow v \in \text{range vals}\)
and \(\text{dec} \ (\tau \ n \ p) = \text{Some } v\)
thus \(v \in \text{range vals}\) by (blast intro: integrity-is-local)
next
fix \(\sigma \ t \ m \ n \ p \ q \ v \ w\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ m \ n \ p \ q \ v \ w. \ dec \ (\sigma \ m \ p) = \text{Some } v \wedge \text{dec} \ (\sigma \ n \ q) = \text{Some } w \rightarrow v = w\)
and \(\text{dec} \ (\tau \ m \ p) = \text{Some } v \wedge \text{dec} \ (\tau \ n \ q) = \text{Some } w\)
thus \(v = w\) by (blast intro: agreement-is-local)
next
fix \(\sigma \ t \ p\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ p. \exists \ m \ v. \ dec \ (\sigma \ m \ p) = \text{Some } v\)
thus \(\exists n \ w. \ dec \ (\tau \ n \ p) = \text{Some } w\) 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 \(\sigma \ t \ n \ p \ v \ w\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ n \ p \ v \ w. \ dec \ (\sigma \ n \ p) = \text{Some } v \rightarrow w = v\)
and \(\text{dec} \ (\tau \ n \ p) = \text{Some } w\)
thus \(w = v\) by (blast intro: validity-is-local)
next
fix \(\sigma \ t \ m \ n \ p \ q \ v \ w\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ m \ n \ p \ q \ v \ w. \ dec \ (\sigma \ m \ p) = \text{Some } v \wedge \text{dec} \ (\sigma \ n \ q) = \text{Some } w \rightarrow v = w\)
and \(\text{dec} \ (\tau \ m \ p) = \text{Some } v \wedge w: \text{dec} \ (\tau \ n \ q) = \text{Some } w\)
thus \(v = w\) by (blast intro: agreement-is-local)
next
fix \(\sigma \ t \ p\)
assume locally-similar \(\sigma \ t\)
and \(\forall \ p. \exists \ m \ v. \ dec \ (\sigma \ m \ p) = \text{Some } v\)
thus \(\exists n \ w. \ dec \ (\tau \ n \ p) = \text{Some } w\) by (blast dest: termination-is-local)
qed

end

theory Majorities

29
imports Main
begin

4 Utility Lemmas About Majorities

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

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

lemma abs-majorities-intersect:
  assumes crd: card $E < card S + card T$
  and s: $S \subseteq E$ and t: $T \subseteq E$ and e: finite $E$
  shows $S \cap T \neq \{\}$
proof (clarify)
  assume contra: $S \cap T = \{\}$
  from s t e have finite $S$ and finite $T$ by (auto simp: finite-subset)
  with crd contra have card $E < card (S \cup T)$ by (auto simp add: card-Un-Int)
  moreover from s t e have card $(S \cup T) \leq card E$ by (simp add: card-mono)
  ultimately show False by simp
qed

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

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

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

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

lemma majorities-intersect:
assumes \( \text{crd :: card (UNIV :: ('a :: finite) set) < card (S :: 'a set) + card T} \)
shows \( S \cap T \neq \{\}\) by (rule abs-majorities-intersect \{ OF crd \}) auto

lemma majoritiesE:
assumes \( \text{crd :: card (UNIV :: ('a :: finite) set) < card (S :: 'a set) + card (T :: 'a set)} \)
obtains \( p \) where \( p \in S \) and \( p \in T \) using crd majorities-intersect by blast

lemma majoritiesE':
assumes \( S :: card (S :: ('a :: finite) set) > (\text{card (UNIV :: 'a set)}) \text{ div 2} \)
and \( T :: card (T :: ('a set)) > (\text{card (UNIV :: 'a set)}) \text{ div 2} \)
obtains \( p \) where \( p \in S \) and \( p \in T \) by (rule abs-majoritiesE' \{ OF S T \}) auto

end

theory OneThirdRuleDefs
imports ../HOModel
begin

5 Verification of the One-Third Rule Consensus Algorithm

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

5.1 Model of the Algorithm

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

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

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

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

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

definition OTR-initState where
  OTR-initState p st ≡ decide st = None

Given a vector \( \text{msgs} \) of values (possibly null) received from each process, \( \text{HOV \ msgs \ v} \) 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 \}

\( \text{MFR \ msgs \ v} \) ("most frequently received") holds for vector \( \text{msgs} \) 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 \( \text{HOV \ msgs \ v} \).

definition MFR :: (Proc ⇒ 'val option) ⇒ 'val ⇒ bool where
  MFR msgs v ≡ \( \forall \ w. \ card (HOV msgs w) \leq \ card (HOV msgs v) \)

lemma MFR-exists: \( \exists v. \ MFR \ msgs \ v \)
proof
  let \?cards = \{ card (HOV msgs v) | v . True \}
  let \?mfr = Max \?cards
  have \( \forall v. \ card (HOV msgs v) \leq N \) by (auto intro: card_mono)
  hence \?cards \subseteq \{ 0 .. N \} by auto
  hence \text{fin}: \ finite \?cards by (metis atLeast0AtMost finite-atMost finite-subset)
  hence \?mfr \in \?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) \leq \?mfr by (rule Max-ge) auto
    thus card (HOV msgs w) \leq card (HOV msgs v) by (unfold v)
  qed
  thus \?thesis ..
qed

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

lemma MFR-in-msgs:
  assumes \( \text{HO:HOs m p \neq \{\}} \)
  and \( v: \text{MFR (HOrcvdMsgs OTR-M m p (HOs m p) (rho m)) v} \)
  (is MFR \?msgs v)
  shows \( \exists q \in \text{HOs m p} . v = \text{the} (\?msgs q) \)
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)
by auto
hence HOp: 0 < card (HOV ?msgs (the (?msgs q)))
  by auto
also from v have ... ≤ card (HO ?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**

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

**definition OTR-commGlobal where**

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

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

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

**definition OTR-HOMachine where**

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

**abbreviation OTR-M \equiv OTR-HOMachine::(Proc, 'val::linorder pstate, 'val) HOMachine**

end

theory OneThirdRuleProof
imports OneThirdRuleDefs ..://Reduction ..://Majorities
begin

We prove that One-Third Rule solves the Consensus problem under the communication predicate defined above. The proof is split into proofs of the Integrity, Agreement, and Termination properties.

### 5.4 Proof of Integrity

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

**definition VInv where**

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

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 HOnextConfig-eq OTR-HOMachine-def initState-def
                OTR-initState-def VInv-def image-def)

next
fix m
assume ih: \( VInv \, rho \, m \)
let \( ?xinit = range (x \circ (rho \, 0)) \)
have range \((x \circ (rho \, (Suc \, m))) \) \( \subseteq \ ?xinit \)
proof (clarsimp)
fix p
from run have nxt: OTR-nextState m p (rho m p)
  \((HOrcvdMsgs \, OTR-M \, m \, p \, (HOs \, m \, p) \, (rho \, m))\)
  \((rho \, (Suc \, m) \, p)\)
  \(\text{(is OTR-nextState - - ?st \, msgs \, ?st')}\)
by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
show \( x \, ?st' \in \, ?xinit \)
proof (cases \((2*\, N) \, \text{div} \, 3 < \, \text{card} \, (HOs \, m \, p))\)
case True
  hence HO: HOs m p \( \neq \, \{\} \) by auto
  let \( ?MFRs = \{v. \, MFR \, ?msgs \, v\} \)
  have Min ?MFRs \( \in \) ?xinit
  proof (rule Min-in)
    from HO have ?MFRs \( \subseteq \) \((the \, \circ \, ?msgs)\,^{-1}(HOs \, m \, p)\)
    by (auto simp: image-def intro: MFR-in-msgs)
    thus finite ?MFRs by (auto elim: finite-subset)
  next
  from MFR-exists show ?MFRs \( \neq \, \{\} \) by auto
  qed
  with HO have \( \exists q \in \, HOs \, m \, p. \, \text{Min} \, \, ?MFRs \, = \, \text{the} \, (\, ?msgs \, q)\)
  by (intro MFR-in-msgs auto)
  hence \( \exists q \in \, HOs \, m \, p. \, \text{Min} \, \, ?MFRs \, = \, x \, (\, rho \, m \, q)\)
  by (auto simp: HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)
  moreover
  from True nxt have \( x \, ?st' = \, \text{Min} \, \, ?MFRs\)
  by (simp add: OTR-nextState-def HOrcvdMsgs-def)
  ultimately
  show \( \, \text{thesis} \, \text{using} \, \text{ih} \) by (auto simp: VInv-def image-def)
next
  case False
  with nxt \text{ih} \text{show} \( \, \text{thesis} \)
  by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def Let-def)
  qed
  qed

35
moreover
have ∀ p. decide ((ρ (Suc m)) p) ∈ {None} ∪ (Some ‘ ?xinit)
proof
  fix p
  from run
  have nxt: OTR-nextState m p (ρ m p)
    (HOrcvdMsgs OTR-M m p (HOs m p) (ρ m))
    (ρ (Suc m) p)
    (is OTR-nextState - - ?st ?msgs ?st')
  by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)
show decide ?st' ∈ {None} ∪ (Some ‘ ?xinit)
proof (cases (2∗N) div 3 < card {q. ?msgs q ≠ None})
  assume HO: (2∗N) div 3 < card {q. ?msgs q ≠ None}
  show ?thesis
proof (cases ∃ v. TwoThirds ?msgs v)
  case True
  let ?dec = v where TwoThirds ?msgs v
  from True have TwoThirds ?msgs ?dec by (rule someI-ex)
  hence HOV ?msgs ?dec ≠ {} by (auto simp add: TwoThirds-def)
  then obtain q where x (ρ m q) = ?dec
  by (auto simp: HOV-def HOrcvdMsgs-def OTR-HOMachine-def OTR-sendMsg-def)
  from sym[OF this] nxt ih show ?thesis
  by (auto simp: OTR-nextState-def VInv-def image-def)
next
  case False
  with HO nxt ih show ?thesis
  by (auto simp: OTR-nextState-def VInv-def HOrcvdMsgs-def image-def)
qed
next
  case False
  with nxt ih show ?thesis
  by (auto simp: OTR-nextState-def VInv-def image-def)
qed
hence range (decide o (ρ (Suc m))) ⊆ {None} ∪ (Some ‘ ?xinit) by auto
ultimately
show VInv ρ (Suc m) by (auto simp: VInv-def image-def)
qed

Integrity is an immediate consequence.

theorem OTR-integrity:
assumes run: HORun OTR-M ρ HOs and dec: decide (ρ n p) = Some v
shows ∃ q. v = x (ρ θ q)
proof –
  let ?xinit = range (x o (ρ 0))
  from run have VInv ρ no n by (rule vine-invariant)
  hence range (decide o (ρ n)) ⊆ {None} ∪ (Some ‘ ?xinit)
    by (auto simp: VInv-def Let-def)
hence  \( \text{decide}\ ((\rho \, n) \, p) \in \{\text{None}\} \cup (\text{Some} \, \tau) \)
by (auto simp: image-def)
with \( \text{dec} \) show \(?\text{thesis} \) by auto
qed

5.5 Proof of Agreement

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

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

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

have \{ q . ?msgs q = Some v \} ⊆ \{ q . x (\rho n q) = v \} 
by (auto simp: OTR-HOMachine-def OTR-sentMsg-def HOrecvMsgs-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 \( \text{thesis} \) by simp
qed

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

**lemma A2:**
assumes \( \text{run}: \text{HORun} \ OTR-M \ \rho \ \text{HOs} \) 
and \( \text{HO}: (2*\text{N}) \div 3 < \text{card} \{ q . \text{HOrecvMsgs} \ OTR-M \ n \ p \ (\text{HOs} n p) (\rho n) q \neq \text{None} \} \)
and \( \text{maj}: (2*\text{N}) \div 3 < \text{card} \{ q . x (\rho n q) = v \} \)
shows \( x (\rho (\text{Suc} n) p) = v \)

proof –
from \( \text{run} \)
have \( \text{nxt}: \text{OTR-nextState} \ n \ p \ (\rho n p) \)
\( (\text{HOrecvMsgs} \ OTR-M \ n \ p \ (\text{HOs} n p) (\rho n)) \)
\( (\rho (\text{Suc} n) p) \)
(is \( \text{OTR-nextState} - - ?st ?msgs ?st' \))
by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)

let \( ?\text{HOVothers} = \bigcup \{ \text{HOV} ?msgs w \mid w . w \neq v \} \)
— processes from which \( p \) received values different from \( v \)

have \( w: \text{card} \ ?\text{HOVothers} \leq \text{N} \div 3 \)
proof –
have \( \text{card} \ ?\text{HOVothers} \leq \text{card} \ (\text{UNIV} - \{ q . x (\rho n q) = v \}) \)
by (auto simp: HOV-def)

intro: card-mono)
also have \( \ldots = \text{N} - \text{card} \{ q . x (\rho n q) = v \} \)
by (auto simp: card-Diff-subset)
also from \( \text{maj} \) have \( \ldots \leq \text{N} \div 3 \) by auto
finally show \( \text{thesis} \).
qed

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

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

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

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

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

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

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

with \( \text{HOV} \) have \( \text{MFR ?msgs v} \) by (auto simp: MFR-def)

moreover
have \( \forall w. w \neq v \rightarrow \neg (\text{MFR ?msgs w}) \)
proof (auto simp: MFR-def not-le)
fix \( w \)
assume \( w \neq v \)
with \( \text{cardw: } \forall w. w \neq v \rightarrow \text{card} (\text{HOV ?msgs w}) \leq N \text{ div } 3 \) by auto
thus \( \exists v. \text{card} (\text{HOV ?msgs w}) < \text{card} (\text{HOV ?msgs v}) \).

qed

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

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

qed

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

lemma \( A3 \):
assumes \( \text{run: HORun OTR-M rho HOs} \)
and \( n\cdot (2+N) \text{ div } 3 < \text{card} \ \{ q . (\rho n q) = v \} \) is \( \text{twothird n} \)
shows \( \text{twothird (n+k)} \)
proof (induct \( k \))
from \( n \) show \( \text{twothird (n+0)} \) by simp
next
fix \( m \)
assume \( m\cdot \text{twothird (n+m)} \)
have \( \forall q. x (\rho (n+m) q) = v \rightarrow x (\rho (n + \text{Suc } m) q) = v \)
proof (rule+)
fix \( q \)
assume \( q\cdot x ((\rho (n+m)) q) = v \)
let \( ?msgs = \text{HORcvdMsgs OTR-M (n+m) q (H Os (n+m) q) (rho (n+m))} \)
show \( x (\rho (n + \text{Suc } m) q) = v \)
proof (cases \( (2+N) \text{ div } 3 < \text{card} \ \{ q . ?msgs q \neq \text{None} \} \))
case True
from m have \((2+N) \div 3 < \text{card } \{ q . x (\rho (n+m) q) = v \} \) by simp
with True run show \(\text{thesis} \) by (auto elim: A2)

next
  case False
  with run q show \(\text{thesis} \) by (auto dest: nextState-change)
qed

hence \(\text{card } \{ q . x (\rho (n+m) q) = v \} \leq \text{card } \{ q . x (\rho (n + \text{Suc \( m \)) q) = v \} \)
by (auto intro: card-mono)
with m show \(\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 \( \text{run: HORun OTR-M rho HOs} \)
and \( \text{dec: decide (\( \rho n p \)) = Some v (is \( dec n \))} \)
shows \( \forall k. (2+N) \div 3 < \text{card } \{ q . x (\rho (n+k) q) = v \} \)
(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: HORun-eq HOinitConfig-eq OTR-HOMachine-def initState-def OTR-initState-def)

next
— For the inductive step, we assume that process \( p \) has decided on \( v \).
fix \( m \)
assume \( \text{ih: ?dec m \( \implies \forall k. \text{twothird (m+k) \) and \( m: ?dec (\text{Suc } m) \)} \)
show \( \forall k. \text{twothird} ((\text{Suc } m) + k) \)
proof
  fix \( k \)
  have \( \text{twothird} (m + \text{Suc } k) \)

There are two cases to consider: if \( p \) had already decided on \( v \) before, the assertion follows from the induction hypothesis. Otherwise, the assertion follows from lemmas \( A1 \) and \( A3 \).

proof (cases ?dec \( m \))
  case True with \( \text{ih show } \text{thesis by blast} \)
next
  case False
  with run \( m \) have \( \text{twothird } m \) by (auto elim: A1)
  with run show \( \text{thesis by (blast dest: A3)} \)
qed
thus \( \text{twothird} ((\text{Suc } m) + k) \) by simp
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 \((2*N) \ div 3 < \text{card } \{q. \ x \ (\rho \ (n+k) \ q) = v\}\) (is - < card \(?V\))
  
  - By (blast dest: \(A4\))

  moreover from \(\text{run } p'\) have \((2*N) \ div 3 < \text{card } \{q. \ x \ (\rho \ ((n+k)+0) \ q) = w\}\) (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 \(k \text{ where } m = n+k\) by (auto simp add: le_iff_add)

  with \(\text{run } p \ p' \text{ show } ?\text{thesis by } (auto \ elim: \ A5)\)

- Next
  
  - Case \(False\)

  hence \(m \leq n\) by auto

  then obtain \(k \text{ where } n = m+k\) by (auto simp add: le_iff_add)

  with \(\text{run } p \ p' \text{ have } w = v\) 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** \( \text{run: HORun OTR-M rho HOs} \)

**And** \( \text{commG: H0commGlobal OTR-M HOs} \)

**Shows** \( \exists r \, v. \, \text{decide} (\rho \, r \, p) = \text{Some} \, v \)

**Proof**

- **From** \( \text{commG obtain } r_0 \Pi \text{ where} \)
  
  \( \pi: \forall q. \text{HOS} \, r_0 \, q = \Pi \text{ and} \, \pi: \text{card} \, \Pi > (2* \, N) \, \text{div} \, 3 \)
  
  by (auto simp: OTR-HOMachine-def OTR-commGlobal-def)

- **Let** \( ?\text{msgs} \, q \, r = \text{HOrcvdMsgs} \, \text{OTR-M} \, r \, q \, (\text{HOSs} \, r \, q) \, (\text{rho} \, r) \)

- **From** \( \text{run} \, \pi \text{ have } \forall \, p \, q. \, ?\text{msgs} \, q \, r_0 = ?\text{msgs} \, p \, r_0 \)
  
  by (auto simp: HORun-eq OTR-HOMachine-def HOrcvdMsgs-def OTR-sendMsg-def)

- **Then** **obtain** \( \mu \text{ where } \forall \, q. \, ?\text{msgs} \, q \, r_0 = \mu \) by auto

- **Moreover**

  - **From** \( \pi \, \pi: \text{have } \forall \, p. \, (2* \, N) \, \text{div} \, 3 < \text{card} \, \{ q. \, ?\text{msgs} \, p \, r_0 \, q = \text{None} \} \)
    
    by (auto simp: HORun-eq HOnextConfig-eq HOrcvdMsgs-def)

  - **With** \( \text{run have } \forall \, q. \, x \,(\text{rho} \, (\text{Suc} \, r_0) \, q) = \text{Min} \, \{ v. \, \text{MFR} \,(?\text{msgs} \, q \, r_0) \, v \} \)
    
    by (auto simp: HORun-eq HOnextConfig-eq OTR-HOMachine-def HOnextState-def OTR-nextState-def)

  - **Ultimately**

    - **Have** \( \forall \, q. \, x \,(\text{rho} \, (\text{Suc} \, r_0) \, q) = \text{Min} \, \{ \text{v. MFR} \, \mu \, \text{v} \} \) by auto

    - **Then** **obtain** \( v \) **where** \( \forall \, q. \, x \,(\text{rho} \, (\text{Suc} \, r_0) \, q) = v \) by auto

- **Have** \( P: \forall k. \forall q. \, x \,(\text{rho} \, (\text{Suc} \, r_0+k) \, q) = v \)

**Proof**

- **Fix** \( k \)

  - **Show** \( \forall \, q. \, x \,(\text{rho} \, (\text{Suc} \, r_0+k) \, q) = v \)

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

      - **From** \( v \text{ show } \forall \, q. \, x \,(\text{rho} \, (\text{Suc} \, r_0+0) \, q) = v \) by simp

    - **Next**

      - **Fix** \( q \)

        - **Show** \( x \,(\text{rho} \, (\text{Suc} \, r_0 + \text{Suc} \, k) \, q) = v \)

          - **Proof** (cases \( (2* \, N) \, \text{div} \, 3 < \text{card} \, \{ p. \, ?\text{msgs} \, q \,(\text{Suc} \, r_0 + k) \, p = \text{None} \} \})

            - **Case** \( \text{True} \)

              - **Have** \( N > \, 0 \) by (rule finite-UNIV-card-ge-0) simp

              - **With** \( \pih \)

                - **Have** \( (2* \, N) \, \text{div} \, 3 < \text{card} \, \{ p. \, x \,(\text{rho} \, (\text{Suc} \, r_0 + k) \, p) = \text{v} \} \) by auto

              - **With** \( \text{True run show } \, ?\text{thesis by (auto elim: A2)} \)

            - **Next**

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

                - **With** \( \text{run } \pih \text{ show } \, ?\text{thesis by (auto dest: nextState-change)} \)

        qed

        qed

42
qed

from commG obtain r0' Π'
  where r0': r0' ≥ Suc r0
  and pi' ∨ q. HOs r0' q = Π'
  and pic': card Π' > (2*N) div 3
  by (force simp: OTR-HOMachine-def OTR-commGlobal-def)

from r0' P have v' ∨ q. x (rho r0' q) = v by (auto simp: le-iff-add)

from run have OTR-nextState r0' p (rho r0' p) (?msgs p r0') (rho (Suc r0') p)
  by (simp add: HORun-eq HOnextConfig-eq OTR-HOMachine-def nextState-def)

moreover
from pi' pic' have (2*N) div 3 < card {q. (?msgs p r0') q ≠ None}
  by (auto simp: HORcvdMsgs-def OTR-sendMsg-def)

moreover
from pi' pic' v' have TwoThirds (?msgs p r0') v
  by (simp add: TwoThirds-def HORcvdMsgs-def OTR-HOMachine-def
            OTR-sendMsg-def HOV-def)

ultimately
have decide (rho (Suc r0') p) = Some (ε v. TwoThirds (?msgs p r0') v)
  by (auto simp: OTR-nextState-def)

thus ?thesis by blast

qed

5.7 One-Third Rule Solves Consensus

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

theorem OTR-consensus:
  assumes run: HORun OTR-M rho HOs and commG: HOcommGlobal OTR-M HOs
  shows consensus (x ◦ (rho 0)) decide rho
    using OTR-integrity[OF run] OTR-agreement[OF run] OTR-termination[OF run commG]
    by (auto simp: consensus-def image-def)

By the reduction theorem, the correctness of the algorithm also follows for
fine-grained runs of the algorithm. It would be much more tedious to estab-
lish this theorem directly.

theorem OTR-consensus-fg:
  assumes run: fg-run OTR-M rho HOs HOs (λq. undefined)
    and commG: HOcommGlobal OTR-M HOs
  shows consensus (λp. x (state (rho 0) p)) decide (state ◦ rho)
    (is consensus ?inits - -)
  proof (rule local-property-reduction[OF run consensus-is-local])
    fix crun

43
6 Verification of the \textit{UniformVoting} Consensus Algorithm

Algorithm \textit{UniformVoting} is presented in [7]. It can be considered as a deterministic version of Ben-Or's well-known probabilistic Consensus algorithm [2]. We formalize in Isabelle the correctness proof given in [7], using the framework of theory \textit{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 $\texttt{Proc}$ of the generic HO model.

\texttt{typedecl Proc -- the set of processes}
\texttt{axiomatization where Proc-finite: OFCLASS(Proc, finite-class)}
\texttt{instance Proc :: finite by (rule Proc-finite)}

\texttt{abbreviation}
\texttt{  N \equiv 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.

\texttt{abbreviation}
\texttt{  nSteps \equiv 2}

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

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

The following record models the local state of a process.

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

**datatype**

\[
\text{′val msg} = \text{Val ′val} \\
\mid \text{ValVote ′val ′val option} \\
\mid \text{Null} \quad \text{— dummy message in case nothing needs to be sent}
\]

**definition isValVote**

\[
\text{isValVote } m \equiv \exists z v. \, m = \text{ValVote } z v
\]

**definition isVal**

\[
\text{isVal } m \equiv \exists v. \, m = \text{Val } v
\]

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

**fun getvote**

\[
\text{getvote } (\text{ValVote } z v) = v
\]

**fun getval**

\[
\text{getval } (\text{ValVote } z v) = z \\
\mid \text{getval } (\text{Val } z) = z
\]

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

**definition UV-initState**

\[
\text{UV-initState } p \, s t \equiv (\text{vote } s t = \text{None}) \wedge (\text{decide } s t = \text{None})
\]

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

**definition msgRcvd**

\[
\text{msgRcvd } (\text{msgs} :: \text{Proc} \mapsto (\text{′val} :: \text{linorder}) \text{msg}) \equiv \\
\{ q . \, \text{msgs } q = \text{Some } (\text{Val } v) \}
\]

**definition smallestValRcvd**

\[
\text{smallestValRcvd } (\text{msgs} :: \text{Proc} \mapsto (\text{′val} :: \text{linorder}) \text{msg}) \equiv \\
\text{Min } \{ v . \, \exists q. \, \text{msgs } q = \text{Some } (\text{Val } v) \}
\]

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

**definition send0**

\[
\text{send0 } r \, p \, q \, s t \equiv \text{Val } (x \, s t)
\]

**definition next0**

\[
\text{next0 } r \, p \, s \, t \, (\text{msgs} :: \text{Proc} \mapsto (\text{′val} :: \text{linorder}) \text{msg}) \, s t' \equiv \\
(\exists v. \, (\forall q \in \text{msgRcvd } \text{msgs}. \, \text{msgs } q = \text{Some } (\text{Val } v)) \\
\wedge s t' = s t \wedge (\operatorname{vote} := \text{Some } v, \, x := \text{smallestValRcvd } \text{msgs})) \\
\vee \neg (\exists v. \, (\forall q \in \text{msgRcvd } \text{msgs}. \, \text{msgs } q = \text{Some } (\text{Val } v)) \\
\wedge s t' = s t \wedge (x := \text{smallestValRcvd } \text{msgs}))
\]
In step 1, each process sends its current $x$ and vote values.

**definition** `send1` **where**

$send1 \ r \ p \ q \ st \ \equiv \ ValVote \ (x \ st) \ (vote \ st)$

**definition** `valVoteRcvd` **where**

— processes from which values and votes were received

$valVoteRcvd \ (msgs :: Proc \rightarrow \ ('val:\linorder) \ msg) \ \equiv$

$\{ q \cdot \exists v. \ msgs q = Some (ValVote v None) \}$

**definition** `smallestValNoVoteRcvd` **where**

$smallestValNoVoteRcvd \ (msgs :: Proc \rightarrow \ ('val:\linorder) \ msg) \ \equiv$

$Min \ \{ v. \ \exists q. \ msgs q = Some (ValVote v None) \}$

**definition** `someVoteRcvd` **where**

— set of processes from which some vote was received

$someVoteRcvd \ (msgs :: Proc \rightarrow \ ('val:\linorder) \ msg) \ \equiv$

$\{ q. \ q \in msgRcvd msgs \land isValVote (the (msgs q)) \land getvote (the (msgs q)) \neq None \}$

**definition** `identicalVoteRcvd` **where**

$identicalVoteRcvd \ (msgs :: Proc \rightarrow \ ('val:\linorder) \ msg) \ v \ \equiv$

$\forall q \in msgRcvd msgs. \ isValVote (the (msgs q)) \land getvote (the (msgs q)) = Some v$

**definition** `x-update` **where**

$x-update \ st \ msgs \ st' \ \equiv$

$(\exists q \in someVoteRcvd msgs. \ x \ st' = the (getvote (the (msgs q))))$

$\lor \ someVoteRcvd msgs = \{ \} \land x \ st' = smallestValNoVoteRcvd msgs$

**definition** `dec-update` **where**

$dec-update \ st \ msgs \ st' \ \equiv$

$(\exists v. \ identicalVoteRcvd msgs v \land decide \ st' = Some v)$

$\lor \ \neg(\exists v. \ identicalVoteRcvd msgs v \land decide \ st' = decide \ st)$

**definition** `next1` **where**

$next1 \ r \ p \ st \ msgs \ st' \ \equiv$

$x-update \ st \ msgs \ st'$

$\land \ dec-update \ st \ msgs \ st'$

$\land \ vote \ st' = None$

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

**definition** `UV-sendMsg` **where**

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

**definition** `UV-nextState` **where**

$UV-nextState \ r \ \equiv \ if \ step \ r = 0 \ then \ next0 \ r \ else \ next1 \ r$
6.2 Communication Predicate for UniformVoting

We now define the communication predicate for the UniformVoting algorithm to be correct.

The round-by-round predicate requires that for any two processes there is always one process heard by both of them. In other words, no “split rounds” occur during the execution of the algorithm [7]. Note that in particular, heard-of sets are never empty.

**definition UV-commPerRd**

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

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

**definition UV-commGlobal**

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

6.3 The UniformVoting Heard-Of Machine

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

**definition UV-HOMachine**

\[ UV-HOMachine = ( \) \]

- `CinitState = (\lambda p \ st \ crd. \ UV-initState \ p \ st),`
- `sendMsg = UV-sendMsg,`
- `CnextState = (\lambda r \ p \ st \ msgs \ crd \ st'. \ UV-nextState \ r \ p \ st \ msgs \ st'),`
- `HOcommPerRd = UV-commPerRd,`
- `HOcommGlobal = UV-commGlobal`

**abbreviation UV-M ≡ (UV-HOMachine::(Proc, val::linorder pstate, val msg) HOMachine)**

end

theory UvProof
imports UvDefs ../Reduction
begin

6.4 Preliminary Lemmas

At any round, given two processes \( p \) and \( q \), there is always some process which is heard by both of them, and from which \( p \) and \( q \) have received the same message.
lemma some-common-msg:
assumes HOcommPerRd UV-M (HOs r)
shows ∃pq. pq ∈ msgRcvd (HOrcvdMsgs UV-M r p (HOs r p) (rho r)) 
∧ pq ∈ msgRcvd (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) 
∧ (HOrcvdMsgs UV-M r p (HOs r p) (rho r)) pq 
= (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) pq
using assms
by (auto simp: UV-HOMachine-def UV-commPerRd-def HOrcvdMsgs-def UV-sendMsg-def send0-def send1-def msgRcvd-def)

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

lemma minval-step0:
assumes com: HOcommPerRd UV-M (HOs r) and s0: step r = 0
shows smallestValRcvd (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) 
∈ {v. ∃p. (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) p = Some (Val v)}
(is smallestValRcvd ?msgs ∈ {?vals})
unfolding smallestValRcvd-def proof (rule Min-in)
have ?vals ⊆ getval' ((the o ?msgs) · (HOs r q)) 
by (auto simp: HOrcvdMsgs-def image-def)
thus finite ?vals by (auto simp: finite-subset)
next
from some-common-msg[of HOs, OF com]
obtain p where p ∈ msgRcvd ?msgs by blast
with s0 show ?vals ≠ {} 
by (auto simp: msgRcvd-def HOrcvdMsgs-def UV-HOMachine-def UV-sendMsg-def send0-def)
qed

When executing step 1 and no vote has been received, the minimum among values received in messages carrying no vote is well defined.

lemma minval-step1:
assumes com: HOcommPerRd UV-M (HOs r) and s1: step r ≠ 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)) 
∈ {v. ∃p. (HOrcvdMsgs UV-M r q (HOs r q) (rho r)) p = Some (ValVote v None)}
(is smallestValNoVoteRcvd ?msgs ∈ {?vals})
unfolding smallestValNoVoteRcvd-def proof (rule Min-in)
have ?vals ⊆ getval' ((the o ?msgs) · (HOs r q)) 
by (auto simp: HOrcvdMsgs-def image-def)
thus finite ?vals by (auto simp: finite-subset)
next
from some-common-msg[of HOs, OF com]
obtain p where p ∈ msgRcvd ?msgs by blast
with s1 nov show ?vals ≠ {} 
by (auto simp: msgRcvd-def HOrcvdMsgs-def someVoteRcvd-def isValVote-def UV-HOMachine-def UV-sendMsg-def send1-def)
qed
The vote field is reset every time a new phase begins.

**Lemma reset-vote:**
- **Assumes** run: HORun UV-M rho HOs and s0: step r’ = 0
- **Shows** vote (rho r’ p) = None

**Proof**
- **Cases** r’
  - Assume r’ = 0
    - With run show ?thesis
      - by (auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq initState-def UV-initState-def)

**Next**
- Fix r
  - Assume suc: r’ = Suc r
  - From run have nxt: nextState UV-M r p (rho r p)
    (HOrcvdMsgs UV-M r p (HOs r p) (rho r))
    (rho (Suc r) p)
    - by (auto simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def)
  - From s0 suc have step r = 1 by (auto simp: step-def mod-Suc)
  - With nxt suc show ?thesis
    - by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def next1-def)

**Qed**

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

**Lemma x-vote-eq:**
- **Assumes** run: HORun UV-M rho HOs
  - and com: \( \forall r. \) HOcommPerRd UV-M (HOs r)
  - and vote: vote (rho r p) = Some v
- **Shows** v = x (rho r p)

**Proof**
- **Cases** r
  - Case 0
    - With run vote show ?thesis — no vote in initial state
      - by (auto simp: UV-HOMachine-def HORun-eq HOinitConfig-eq initState-def UV-initState-def)
  - Next
    - Fix r’
      - Assume r: r = Suc r’
        - Let msgs = HOrcvdMsgs UV-M r’ p (HOs r’ p) (rho r’)
        - From run have nxtState UV-M r’ p (rho r’ p) msgs (rho (Suc r’) p)
          - by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
        - With vote r have nxt0: nxt0 r’ p (rho r’ p) msgs (rho r p) and s0: step r’ = 0
          - by (auto simp: nextState-def UV-HOMachine-def UV-nextState-def next1-def)
        - From run s0 have vote (rho r’ p) = None by (rule reset-vote)
          - With vote nxt0 have idv: \( \forall q \in \text{msgRcvd} \) msgs, \( \forall \text{msgs} q = \text{Some} \) (Val v)
            - and x: x (rho r p) = smallestValRcvd ?msgs
              - by (auto simp: nxt0-def)
          - Moreover
            - From com obtain q where q \in \text{msgRcvd} \( \forall \text{msgs} \)
by (force dest: some-common-msg)
with idv have \{ x . \exists qq. \{msgs qq = Some (Val x)\} = \{v\}
  by (auto simp: msgRcvd-def)
  hence smallestValRcvd \{msgs\} = v
  by (auto simp: smallestValRcvd-def)
ultimately
  show \{thesis\} by simp
qed

6.5 Proof of Irrevocability, Agreement and Integrity

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

lemma decide-step:
  assumes run: HORun UV-M rho HOs
  and decide: decide (rho (Suc r) p) \neq decide (rho r p)
  shows step r = 1
proof –
  let \{msgs\} = HOrcvdMsgs UV-M r p (HOs r p) (rho r)
  from run have nextState UV-M r p (rho r p) \{msgs\} (rho (Suc r) p)
    by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
  with decide show \{thesis\}
    by (auto simp: nextState-def UV-HOMachine-def UV-nextState-def
         next0-def step-def)
qed

No process ever decides None.

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

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

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

have s0: \text{step } r = 0 
proof (rule ccontr) 
assume \text{step } r \neq 0 
hence \text{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 \text{com}: \forall 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 = HOrcvdMsgs UV-M r' p (HOs r' p) (rho r') 
from \text{com} obtain pq 
  where ?msgs p pq = ?msgs q pq 
  and smp: pq \in msgRcvd (?msgs p) \and smq: pq \in msgRcvd (?msgs q) 
  by (force dest: some-common-msg) 
moreover 
from run p smp r have getval (the (?msgs p pq)) = v 
  by (simp add: msgs-unanimity) 
moreover 
from run q 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 \text{com}: \forall r. HocommPerRd UV-M (HOs r)
\[
\text{and decide: } \text{decide} \left( \rho \left( \text{Suc } r \right) p \right) \neq \text{decide} \left( \rho \ r \ p \right) \\
\text{and decval: } \text{decide} \left( \rho \left( \text{Suc } r \right) p \right) = \text{Some } v \\
\text{shows } x \left( \rho \left( \text{Suc } r \right) q \right) = v
\]

**proof**

1. **let** \( ?\text{msgs } p' = \text{HORcvdMsgs } UV-M \ r \ p' \left( \text{HOs } r \ p' \right) \left( \rho \ r \right) \)
2. **from** run **have** \( s1: \text{step } r = 1 \) **by** (rule decide-step)
3. **from** run **have** \( \text{nextState } UV-M \ r \ p \left( \rho \ r \ p \right) \left( ?\text{msgs } p \right) \left( \rho \ \text{Suc } r \right) p \)
   **by** (auto simp: HORun-eq HOnextConfig-eq nextState-def)
4. **with** \( s1 \) **have** \( \text{next } q: \text{next1 } r \ q \left( \rho \ q \right) \left( ?\text{msgs } q \right) \left( \rho \ \text{Suc } r \right) q \)
   **by** (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
5. **from** run **have** \( \text{nextState } UV-M \ r \ q \left( \rho \ q \right) \left( ?\text{msgs } q \right) \left( \rho \ \text{Suc } r \right) q \)
   **by** (auto simp: HORun-eq HOnextConfig-eq nextState-def)
6. **with** \( s1 \) **have** \( \text{next } q: \text{next1 } r \ q \left( \rho \ q \right) \left( ?\text{msgs } q \right) \left( \rho \ \text{Suc } r \right) q \)
   **by** (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)

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

**lemma** \( \text{same-x-stable:} \)

1. **assumes** \( \text{run: } \text{HORun } UV-M \ rho \ \text{HOs} \)
2. **and** \( \text{comm: } \forall r. \ \text{HOCommPerRd } UV-M \ \left( \text{HOs } r \right) \)
3. **and** \( \text{x: } \forall p. \ x \left( \rho \ r \ p \right) = v \)
4. **shows** \( x \left( \rho \ \text{Suc } r \right) q = v \)

**proof**

1. **let** \( ?\text{msgs } p = \text{HORcvdMsgs } UV-M \ r \ q \left( \text{HOs } r \ q \right) \left( \rho \ r \ q \right) \)
2. **from** comm **obtain** \( p \) **where** \( p: p \in \text{msgRcvd } ?\text{msgs} \)
   **by** (force dest: some-common-msg)
3. **from** run **have** \( \text{nextState } UV-M \ r \ q \left( \rho \ r \ q \right) ?\text{msgs } \left( \rho \ \text{Suc } r \right) q \)
   **by** (auto simp: HORun-eq HOnextConfig-eq nextState-def)
4. **hence** \( \text{next0 } r \ q \left( \rho \ r \ q \right) ?\text{msgs } \left( \rho \ \text{Suc } r \right) q \land \text{step } r = 0 \\
   \lor \text{next1 } r \ q \left( \rho \ r \ q \right) ?\text{msgs } \left( \rho \ \text{Suc } r \right) q \land \text{step } r \neq 0 \\
   \left( \text{is } ?\text{next } 0 \lor ?\text{next } 1 \right) \)
   **by** (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)

52
thus \( ?\text{thesis} \)
proof
  assume \( \text{nxt0} : ?\text{nxt0} \)
  hence \( x \ (\rho \ (\text{Suc} \ r) \ q) = \text{smallestValRcvd} \ ?\text{msgs} \)
    by (auto simp: \text{nxt0-def})
moreover
  from \( \text{nxt0} \ x \) have \( \forall \ p \in \text{msgRcvd} \ ?\text{msgs}. \ ?\text{msgs} \ p = \text{Some} \ (\text{Val} \ v) \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def msgRcvd-def send0-def)
  from this \( p \) have \( \{ x . \ \exists \ p. \ ?\text{msgs} \ p = \text{Some} \ (\text{Val} \ x) \} = \{ v \} \)
    by (auto simp: msgRcvd-def)
  hence \( \text{smallestValRcvd} \ ?\text{msgs} = v \)
    by (auto simp: smallestValRcvd-def)
ultimately
  show \( ?\text{thesis} \) by simp
next
  assume \( \text{nxt1} : ?\text{nxt1} \)
  show \( ?\text{thesis} \)
proof
  (cases someVoteRcvd \( ?\text{msgs} = \{ \} \))
  case True
  with \( \text{nxt1} \ x \) have \( x \ (\rho \ (\text{Suc} \ r) \ q) = \text{smallestValNoVoteRcvd} \ ?\text{msgs} \)
    by (auto simp: next1-def x-update-def)
moreover
  from \( \text{nxt1} \ x \) \( \) True
  have \( \forall \ p \in \text{msgRcvd} \ ?\text{msgs}. \ ?\text{msgs} \ p = \text{Some} \ (\text{ValVote} \ v \ \text{None}) \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def msgRcvd-def send1-def someVoteRcvd-def isValVote-def)
  from this \( p \) have \( \{ x . \ \exists \ p. \ ?\text{msgs} \ p = \text{Some} \ (\text{ValVote} \ x \ \text{None}) \} = \{ v \} \)
    by (auto simp: msgRcvd-def)
  hence \( \text{smallestValNoVoteRcvd} \ ?\text{msgs} = v \)
    by (auto simp: smallestValNoVoteRcvd-def)
ultimately
  show \( ?\text{thesis} \) by simp
next
  case False
  with \( \text{nxt1} \) obtain \( p' \ v' \) where
    \( p' : p' \in \text{msgRcvd} \ ?\text{msgs} \) isValVote \( (\text{the} \ (\?\text{msgs} \ p')) \)
    getvote \( (\text{the} \ (\?\text{msgs} \ p')) = \text{Some} \ v' \ (\rho \ (\text{Suc} \ r) \ q) = v' \)
    by (auto simp: someVoteRcvd-def next1-def x-update-def)
  with \( \text{nxt1} \) have \( x \ (\rho \ (\text{Suc} \ r) \ q) = x \ (\rho \ r \ p') \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def msgRcvd-def send1-def isValVote-def x-vote-eq[OF run comm])
  with \( x \) 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 p) = v by simp
next
  fix k q
  assume q: (rho (Suc r+k) q) = v
  with run com show x (rho Suc r Suc k p) = 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 ∃m<n. decide (rho Suc m p) ≠ decide (rho m p)
  ∧ decide (rho Suc m p) = Some v
proof
  let ?dec k = decide (rho k p)
  have (∀ m<n. ?dec (Suc m) ≠ ?dec m ⋱ ?dec (Suc m) ≠ Some v)
  → ?dec n ≠ Some v
  (is ?P n is ?A n → -)
  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 ∃ q. x (rho r p) = x (rho 0 q)
proof (induct r arbitrary: p)
  fix p
  show ∃ q. x (rho 0 p) = x (rho 0 q) by auto
next
fix r p
assume ih: \( p \). \( \exists q. x (\rho r p') = x (\rho 0 q) \)
let \( \text{run} \) have nextState UV-M r p (rho r p) (HOs r p) (rho r)
from run have nextState UV-M r p (rho r p) \( ?\text{msgs} (\rho (\text{Suc} r) p) \)
  by (auto simp: HORun-eq HOnextConfig-eq nextState-def)
hence nextState UV-M r p (rho r p) \( ?\text{msgs} (\rho (\text{Suc} r) p) \wedge \text{step} r = 0 \)
  \( \lor \) nextState UV-M r p (rho r p) \( ?\text{msgs} (\rho (\text{Suc} r) p) \wedge \text{step} r \neq 0 \)
(is nextStateUV-M r p \( ?\text{msgs} (\rho (\text{Suc} r) p) \))
  by (auto simp: UV-HOMachine-def nextState-def UV-nextState-def)
thus \( \exists q. x (\rho (\text{Suc} r) p) = x (\rho 0 q) \)
proof
  assume \( ?\text{nxt0} \)
  hence \( x (\rho (\text{Suc} r) p) = \text{smallestValRcvd} \ ?\text{msgs} \)
  by (auto simp: next0-def)
  also with \( \text{com} \) \( ?\text{nxt0} \)
  have \( \ldots \in \{ v . \exists q. \ ?\text{msgs} q = \text{Some} (\text{Val} v) \} \)
  by (intro minval-step0) auto
  also with \( \text{nxt0} \)
  have \( \ldots = \{ x (\rho r q) \mid q . q \in \text{msgRcvd} \ ?\text{msgs} \} \)
  by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def msgRcvd-def 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
assumes \( ?\text{nxt1} \)
show \( ?\text{thesis} \)
proof
  (cases someVoteRcvd ?msgs \( = \{ \} \))
  case True
  with \( ?\text{nxt1} \)
  have \( x (\rho (\text{Suc} r) p) = \text{smallestValNoVoteRcvd} \ ?\text{msgs} \)
  by (auto simp: next1-def x-update-def)
  also with \( \text{com} \) \( ?\text{nxt1} \) True
  have \( \ldots \in \{ v . \exists q. \ ?\text{msgs} q = \text{Some} (\text{ValVote} v \text{ None}) \} \)
  by (intro minval-step1) auto
  also with \( \text{nxt1} \) True
  have \( \ldots = \{ x (\rho r q) \mid q . q \in \text{msgRcvd} \ ?\text{msgs} \} \)
  by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def someVoteRcvd-def isValVote-def msgRcvd-def send1-def)
  finally obtain q where \( x (\rho (\text{Suc} r) p) = x (\rho r q) \) by auto
with \( \text{ih} \)
  show \( ?\text{thesis} \) by auto
next
  case False
  with \( ?\text{nxt1} \)
  obtain q where
    \( q \in \text{someVoteRcvd} \ ?\text{msgs} \)
    \( x (\rho (\text{Suc} r) p) = \text{the} (\text{getvote} (\text{the} \ ?\text{msgs} q)) \)
    by (auto simp: next1-def x-update-def)
  with \( ?\text{nxt1} \)
  have \( \text{vote} (\rho r q) = \text{Some} (x (\rho (\text{Suc} r) p)) \)
    by (auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def someVoteRcvd-def isValVote-def msgRcvd-def send1-def)
  with \( \text{run} \)
  have \( x (\rho (\text{Suc} r) p) = x (\rho r q) \)
    by (rule x-vote-eq)
  with \( \text{ih} \)
  show \( ?\text{thesis} \) by auto
next

qed

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

proof –
  from run dec obtain k where
    decide (rho (Suc k) p) ≠ 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: ∀ r. HOcommPerRd UV-M (HOs r)
  and decidep: decide (rho (Suc r) p) ≠ decide (rho r p)
  and decvalp: decide (rho (Suc r) p) = Some v
  and decideq: decide (rho (Suc (r+k)) q) ≠ decide (rho (r+k) q)
  and decvalq: decide (rho (Suc (r+k)) q) = Some w
shows v = w

proof –
  from run com decidep decvalp have x (rho (Suc r+k) q) = v
    by (rule safety-argument)
moreover
  from run com decideq decvalq have x (rho (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: ∀ 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) ≠ decide (rho k p)
    decide (rho (Suc k) p) = Some v
by (auto dest: decided-then-past-decision)
from \texttt{run q obtain l where}
\texttt{l: decide (\rho\ (\text{Suc}\ l)\ q) \neq decide (\rho\ l\ q)}
\texttt{decide (\rho\ (\text{Suc}\ l)\ q) = Some \ w}
by (auto dest: decided-then-past-decision)
show \ ?thesis
proof (cases \texttt{k \leq l})
case \texttt{True}
then obtain \texttt{m where m: l = k+m} by (auto simp: le-iff-add)
from \texttt{run com k l m show \ ?thesis} by (blast dest: two-decisions-agree)
next
case \texttt{False}
hence \texttt{l \leq k} by simp
then obtain \texttt{m where m: k = l+m} by (auto simp: le-iff-add)
from \texttt{run com k l m show \ ?thesis} by (blast dest: two-decisions-agree)
qed
qed

Irrevocability is a consequence of Agreement and the fact that no process can decide \texttt{None}.

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

6.6 \textbf{Proof of Termination}

Two processes having the same \textit{Heard-Of} set at some round will hold the same value in their \texttt{x} variable at the next round.

\textbf{lemma hoeq-xeq:}
\texttt{assumes run: HORun UV-M \rho HOs}
\texttt{and com: \forall \ r. HOcommPerRd UV-M (HOs r)}
\texttt{and hoeq: HOs r p = HOs r q}
shows $x \cdot (\rho \cdot (\text{Suc} \ r) \cdot p) = x \cdot (\rho \cdot (\text{Suc} \ r) \cdot q)$

**proof** –

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

from $\text{hoveq}$ have $\text{msgeq} : \text{msgs} \ p = \text{msgs} \ q$

by (auto simp: $\text{UV-HOMachine-def} \ \text{HOrcvdMsgs-def} \ \text{UV-sendMsg-def} \ \text{send0-def} \ \text{send1-def}$)

**show** $\text {?thesis}$

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

**case** True

with run have $\forall \ p. \ \text{next0} \ r \ p \ (\rho \ r \ p) \ (\text{msgs} \ p) \ (\rho \cdot (\text{Suc} \ r) \cdot p)$ (is $\forall \ p. \ ?\text{nxt0} \ p$)

by (force simp: $\text{UV-HOMachine-def} \ \text{HORun-eq} \ \text{HOnextConfig-eq} \ \text{nextState-def} \ \text{UV-nextState-def}$)

**hence** $?\text{nxt0} \ p \ ?\text{nxt0} \ q \ \text{by auto}$

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

**next**

assume $\text{stp: step} \ r \neq 0$

with run have $\forall \ p. \ \text{next1} \ r \ p \ (\rho \ r \ p) \ (\text{msgs} \ p) \ (\rho \cdot (\text{Suc} \ r) \cdot p)$ (is $\forall \ p. \ ?\text{nxt1} \ p$)

by (force simp: $\text{UV-HOMachine-def} \ \text{HORun-eq} \ \text{HOnextConfig-eq} \ \text{nextState-def} \ \text{UV-nextState-def}$)

**hence** $\text{x-update} \ (\rho \ r \ p) \ (\text{msgs} \ p) \ (\rho \cdot (\text{Suc} \ r) \cdot p)$

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

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

with msgeq have $\text{x'}: \text{x-update} \ (\rho \ r \ p) \ (\text{msgs} \ p) \ (\rho \cdot (\text{Suc} \ r) \cdot p)$

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

by auto

**show** $\text{thesis}$

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

**case** True

with $\text{x'}$ show $?\text{thesis}$

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

**next**

case False

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

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

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

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

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

qed

We now prove that UniformVoting terminates.

**theorem** $\text{uv-termination}$:
assumes \( \text{run}: \text{HORun} \quad \text{UV-M} \quad \text{rho} \quad \text{HOs} \)
and \( \text{commR}: \forall r. \text{HOcommPerRd} \quad \text{UV-M} \quad (\text{HOs} \quad r) \)
and \( \text{commG}: \text{HOcommGlobal} \quad \text{UV-M} \quad \text{HOs} \)
shows \( \exists r \quad v. \quad \text{decide} \quad (\text{rho} \quad r \quad p) = \text{Some} \quad v \)
\[ \text{proof} - \]
First obtain a round where all \( x \) values agree.

\[
\text{from } \text{commG} \text{ obtain } r_0 \text{ where } r_0: \forall q. \text{HOs} \quad r_0 \quad q = \text{HOs} \quad r_0 \quad p \\
\quad \text{by } (\text{force simp: UV-HOMachine-def UV-commGlobal-def})
\]
\[ \text{let } ?v = x \quad (\text{rho} \quad (\text{Suc} \quad r_0) \quad p) \]
\[ \text{from } \text{run commR} \quad r_0 \text{ have } xs: \forall q. \quad x \quad (\text{rho} \quad (\text{Suc} \quad r_0) \quad q) = ?v \\
\quad \text{by } (\text{auto dest: hoeq-xeq}) \]

Now obtain a round where all votes agree.

\[
\text{def } r' \equiv \text{if step} \quad (\text{Suc} \quad r_0) = 0 \text{ then } \text{Suc} \quad r_0 \text{ else } \text{Suc} \quad (\text{Suc} \quad r_0) \\
\text{have } \text{stp}'_\cdot \quad \text{step} \quad r' = 0 \\
\quad \text{by } (\text{simp add: } r' \text{-def step-def mod-Suc})
\]
\[ \text{have } x': \forall q. \quad x \quad (\text{rho} \quad r' \quad q) = ?v \\
\quad \text{proof } (\text{auto simp: r' \text{-def}}) \\
\quad \text{fix } q \\
\quad \text{from } xs \text{ show } x \quad (\text{rho} \quad (\text{Suc} \quad r_0) \quad q) = ?v .. \\
\text{next} \\
\quad \text{fix } q \\
\quad \text{from } \text{run commR} \quad xs \text{ show } x \quad (\text{rho} \quad (\text{Suc} \quad (\text{Suc} \quad r_0)) \quad q) = ?v \\
\quad \text{by } (\text{rule same-x-stable})
\]
\[ \text{qed} \]
\[ \text{have } \text{vote}'_\cdot : \forall q. \quad \text{vote} \quad (\text{rho} \quad (\text{Suc} \quad r') \quad q) = \text{Some} \quad ?v \\
\quad \text{proof} \\
\quad \text{fix } q \\
\quad \text{let } ?msgs = \text{HOrcvdMsgs} \quad \text{UV-M} \quad \text{r' \quad q} \quad (\text{HOs} \quad r' \quad q) \quad (\text{rho} \quad r')
\]
\[ \text{from } \text{run stp}' \quad \text{have } \text{next0} \quad r' \quad q \quad (\text{rho} \quad r' \quad q) \quad ?msgs \quad (\text{rho} \quad (\text{Suc} \quad r') \quad q) \\
\quad \text{by } (\text{force simp: UV-HOMachine-def HORun-eq HOnextConfig-eq nextState-def UV-nextState-def})
\]
\[ \text{moreover} \\
\text{from } \text{stp}' \quad x' \quad \text{have } \forall q' \in \text{msgRcvd} \quad ?msgs. \quad ?msgs \quad q' = \text{Some} \quad \text{(Val} \quad ?v) \\
\quad \text{by } (\text{auto simp: UV-HOMachine-def HOrcvdMsgs-def UV-sendMsg-def send0-def msgRcvd-def})
\]
\[ \text{moreover} \\
\text{from } \text{commR} \quad \text{have } \text{msgRcvd} \quad ?msgs \neq \{\} \\
\quad \text{by } (\text{force dest: some-common-msg})
\]
\[ \text{ultimately} \\
\text{show } \text{vote} \quad (\text{rho} \quad (\text{Suc} \quad r') \quad q) = \text{Some} \quad ?v \\
\quad \text{by } (\text{auto simp: next0-def})
\]
\[ \text{qed} \]

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

\[
\text{let } ?r'' = \text{Suc} \quad r' \\
\text{let } ?msgs'' = \text{HOrcvdMsgs} \quad \text{UV-M} \quad \text{r'' \quad p} \quad (\text{HOs} \quad \text{?r''} \quad p) \quad (\text{rho} \quad \text{?r''}) \\
\text{from } \text{stp}' \quad \text{have } \text{stp}''_\cdot : \quad \text{step} \quad \text{r''} = 1 \\
\]
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 msgRcvd-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.

\textbf{Theorem} \texttt{uv-consensus}:
\begin{itemize}
  \item assumes \texttt{run: HORun UV-M rho HOs}
  \item assumes \texttt{commR: \forall r. HOcommPerRd UV-M (HOs r)}
  \item assumes \texttt{commG: HOcommGlobal UV-M HOs}
\end{itemize}
shows consensus (\lambda r. x (\texttt{state (rho 0) p})) decide rho
using \texttt{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.

\textbf{Theorem} \texttt{uv-consensus-fg}:
\begin{itemize}
  \item assumes \texttt{run: fg-run UV-M rho HOs HOs (\lambda r q. undefined)}
  \item assumes \texttt{commR: \forall r. HOcommPerRd UV-M (HOs r)}
  \item assumes \texttt{commG: HOcommGlobal UV-M HOs}
\end{itemize}
shows consensus (\lambda r. x (\texttt{state (rho 0) p})) decide (\texttt{state o rho})
(is consensus ?inits - -)
\begin{itemize}
  \item proof (rule local-property-reduction[OF \texttt{run consensus-is-local}])
\end{itemize}
fix \texttt{crun}
\begin{itemize}
  \item assume \texttt{crun: CSHORun UV-M crun HOs HOs (\lambda r q. undefined)}
  \item assume \texttt{crun 0 = state (rho 0)}
\end{itemize}
from \texttt{crun} have HORun UV-M crun HOs
by (unfold HORun-def SHORun-def)
from this \texttt{commR commG have consensus (\texttt{x o (crun 0)}) decide crun}
by (rule uv-consensus)
with init show consensus ?init decide crun
  by (simp add: o-def)
qed

end

theory LastVotingDefs
imports ../HOModel
begin

7 Verification of the LastVoting Consensus Algorithm

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

7.1 Model of the Algorithm

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

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

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

The algorithm proceeds in phases of 4 rounds each (we call steps the individual rounds that constitute a phase). The following utility functions compute the phase and step of a round, given the round number.

definition phase where phase (r::nat) ≡ r div 4

definition step where step (r::nat) ≡ r mod 4

lemma phase-zero [simp]: phase 0 = 0
  by (simp add: phase-def)

lemma step-zero [simp]: step 0 = 0
  by (simp add: step-def)

lemma phase-step: (phase r * 4) + step r = r
  by (auto simp add: phase-def step-def)
The following record models the local state of a process.

```plaintext
record pstate =
  x :: 'val — current value held by process
  vote :: 'val option — value the process voted for, if any
  commit :: bool — did the process commit to the vote?
  ready :: bool — for coordinators: did the round finish successfully?
  timestamp :: nat — time stamp of current value
  decide :: 'val option — value the process has decided on, if any
  coordΦ :: Proc — coordinator for current phase
```

Possible messages sent during the execution of the algorithm.

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

Characteristic predicates on messages.

```plaintext
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.

```plaintext
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.

```plaintext
definition LV-initState where
  LV-initState p st crd ≡
  vote st = None
  ∧ ∼(commit st)
  ∧ ∼(ready st)
  ∧ timestamp st = 0
  ∧ decide st = None
  ∧ coordΦ st = crd
```

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

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

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

In step 0, each process sends its current \text{x} and \text{timestamp} values to its coordinator.

A process that considers itself to be a coordinator updates its \text{vote} field if it has received messages from a majority of processes. It then sets its \text{commit} field to true.

definition \text{send0} where
\text{send0} r p q st ≡ if q = \text{coord} st then \text{ValStamp} \langle x st, timestamp st \rangle else Null

definition \text{next0} where
\text{next0} r p st msgs crd st' ≡ if p = \text{coord} st ∧ \text{commit} st then \text{Vote} \langle \text{the} vote st \rangle else st

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

definition \text{send1} where
\text{send1} r p q st ≡ if p = \text{coord} st ∧ \text{commit} st then \text{Vote} \langle \text{the} (\text{vote} st) \rangle else Null

definition \text{next1} where
\text{next1} r p st msgs crd st' ≡ if msgs (\text{coord} st) \neq \text{None} ∧ 
\text{isVote} (\langle \text{msg} (\text{coord} st) \rangle) \text{then } st' = st \langle x := \text{val} \langle \text{msg} (\text{coord} st) \rangle, \text{timestamp} := \text{Suc} \langle \text{phase} r \rangle \rangle \text{else } st' = st

In step 2, processes that have current timestamps send an acknowledgement to their coordinator.

A coordinator sets its \text{ready} field to true if it receives a majority of acknowledgements.

definition \text{send2} where
\text{send2} r p q st ≡ if timestamp st = \text{Suc} \langle \text{phase} r \rangle ∧ q = \text{coord} st then \text{Ack} else Null

— processes from which an acknowledgement was received

definition \text{acksRcvd} where
acksRcvd \( (msgs \rightarrow \text{Proc}) \equiv \{ q . \; msgs \neq \text{None} \land \text{isAck} \left( \text{the} \left( \text{msgs} \; q \right) \right) \}\)

definition next2 where

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

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

definition send3 where

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

definition next3 where

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

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

definition LV-sendMsg :: nat \rightarrow \text{Proc} \rightarrow \text{Proc} \rightarrow 'val pstate \rightarrow 'val msg where

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

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

\[
\text{where} \\
\text{LV-nextState } r \equiv \\
\text{if step } r = 0 \text{ then next0 } r \text{ else if step } r = 1 \text{ then next1 } r \text{ else if step } r = 2 \text{ then next2 } r \text{ else next3 } r
\]
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} \\
\textit{LV-commPerRd} where \\
$LV$-$commPerRd$ $r$ ($HO$:$HO$) ($coord$:$Proc$ $coord$) $\equiv$ True

\textbf{definition} \\
\textit{LV-commGlobal} where \\
$LV$-$commGlobal$ $HOs$ $coords$ $\equiv$ \\
$\exists ph$::nat. $\exists c$::Proc. \\
$(\forall p. $coords$ (4*ph) p = c) \\
\land$ card ($HOs$ ($4*ph$) $c$) $>$ $N$ div 2 \\
\land$ card ($HOs$ ($4*ph+2$) $c$) $>$ $N$ div 2 \\
\land$ $(\forall p. c \in HOs$ ($4*ph+1$) $p \cap HOs$ ($4*ph+3$) $p)$

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} \textit{LV-CHOMachine} where \\
$LV$-$CHOMachine$ $\equiv$ \\
$\langle$ CinitState $=$ $LV$-$initState$, \\
sendMsg $=$ $LV$-$sendMsg$, \\
CnextState $=$ $LV$-$nextState$, \\
CHOcommPerRd $=$ $LV$-$commPerRd$, \\
CHOcommGlobal $=$ $LV$-$commGlobal$ $\rangle$

\textbf{abbreviation} \\
$LV$-$M$ $\equiv$ ($LV$-$CHOMachine$::($Proc$, 'val pstate, 'val msg$) $CHOMachine$)

end
7.4 Preliminary Lemmas

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

lemma timeStampsRcvdFinite:
finite {ts . ∃ q v. (msgs::Proc → 'val msg) q = Some (ValStamp v ts)}
is_finite ?ts

proof –
have ?ts = stamp ' the ' msgs ' (valStampsRcvd msgs)
  by (force simp add: valStampsRcvd-def image-def)
thus ?thesis by auto
qed

lemma highestStampRcvd-exists:
assumes nempty: valStampsRcvd msgs ≠ {}
obtains p v where msgs p = Some (ValStamp v (highestStampRcvd msgs))

proof –
let ?ts = {ts . ∃ q v. msgs q = Some (ValStamp v ts)}
from nempty have ?ts ≠ {} by (auto simp add: valStampsRcvd-def)
with timeStampsRcvdFinite
have highestStampRcvd msgs ∈ ?ts
  unfolding highestStampRcvd-def by (rule Max-in)
then obtain p v where msgs p = Some (ValStamp v (highestStampRcvd msgs))
  by (auto simp add: highestStampRcvd-def)
with that show thesis .
qed

lemma highestStampRcvd-max:
assumes msgs p = Some (ValStamp v ts)
shows ts ≤ highestStampRcvd msgs
using assms unfolding highestStampRcvd-def
by (blast intro: Max-ever timeStampsRcvdFinite)

lemma phase-Suc:
phase (Suc r) = (if step r = 3 then Suc (phase r) else phase r)

unfolding step-def phase-def by presburger

Many proofs are by induction on runs of the LastVoting algorithm, and we derive a specific induction rule to support these proofs.

lemma LV-induct:
assumes run: CHORun LV-M rho HOs coords
and init: ∀ p. CinitState LV-M p (rho 0 p) (coords 0 p) ⇒ P 0
\[\forall r.\]
\[
\begin{cases}
\text{step } r = 0; & P r; \text{phase } (\text{Suc } r) = \text{phase } r; \text{step } (\text{Suc } r) = 1; \\
\forall p. \text{next} 0 r p (\rho r p) & \\
(\text{HOrcvdMsgs LV-M } r p (\text{HOs } r p) (\rho r)) \\
(\text{coords } (\text{Suc } r) p) \\
(\rho (\text{Suc } r) p) \\
\end{cases}
\implies P (\text{Suc } r)
\]

\text{and step1: } \forall r.
\[\begin{cases}
\text{step } r = 1; & P r; \text{phase } (\text{Suc } r) = \text{phase } r; \text{step } (\text{Suc } r) = 2; \\
\forall p. \text{next} 1 r p (\rho r p) & \\
(\text{HOrcvdMsgs LV-M } r p (\text{HOs } r p) (\rho r)) \\
(\text{coords } (\text{Suc } r) p) \\
(\rho (\text{Suc } r) p) \\
\end{cases}
\implies P (\text{Suc } r)
\]

\text{and step2: } \forall r.
\[\begin{cases}
\text{step } r = 2; & P r; \text{phase } (\text{Suc } r) = \text{phase } r; \text{step } (\text{Suc } r) = 3; \\
\forall p. \text{next} 2 r p (\rho r p) & \\
(\text{HOrcvdMsgs LV-M } r p (\text{HOs } r p) (\rho r)) \\
(\text{coords } (\text{Suc } r) p) \\
(\rho (\text{Suc } r) p) \\
\end{cases}
\implies P (\text{Suc } r)
\]

\text{and step3: } \forall r.
\[\begin{cases}
\text{step } r = 3; & P r; \text{phase } (\text{Suc } r) = \text{Suc } (\text{phase } r); \text{step } (\text{Suc } r) = 0; \\
\forall p. \text{next} 3 r p (\rho r p) & \\
(\text{HOrcvdMsgs LV-M } r p (\text{HOs } r p) (\rho r)) \\
(\text{coords } (\text{Suc } r) p) \\
(\rho (\text{Suc } r) p) \\
\end{cases}
\implies P (\text{Suc } r)
\]

\text{shows } P n

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

\text{next}
\text{fix } r
\text{assume } \text{ih: } P r
\text{and } \text{nxt: } \text{CHOnextConfig LV-M } r (\rho r) (\text{HOs } r) (\text{coords } (\text{Suc } r) p) (\rho (\text{Suc } r) p)
\text{have } \text{step } r \in \{0, 1, 2, 3\} \text{ by (auto simp add: step-def)}
\text{thus } P (\text{Suc } r) \text{ by (auto simp add: step-def)}
\text{proof auto}
\text{assume } \text{stp: } \text{step } r = 0
\text{hence } \text{step } (\text{Suc } r) = 1
\text{by (auto simp add: step-def mod-Suc)}
\text{with } \text{ih nxt stp show } \text{?thesis}
\text{by (intro step0)}
\text{(auto simp: LV-CHOMachine-def CHOnextConfig-eq}
\text{LV-nextState-def LV-sendMsg-def phase-Suc)}

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

67
hence \( \text{step} (\text{Suc } r) = 2 \)
by (auto simp add: step-def mod-Suc)
with \( \text{ih} \) \( \text{nxt} \) \( \text{stp} \) show ?thesis
by (intro step1)
(auto simp: LV-CHOMachine-def CHOnextConfig-eq
LV-nextState-def LV-sendMsg-def phase-Suc)

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

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

qed

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

lemma LV-Suc:
assumes \( \text{run}: \text{CHORun} \text{ LV-M } \rho \text{ HO}_s \text{ coords} \)
and \( \text{step0}: [ \text{ step } r = 0; \text{ step} (\text{Suc } r) = 1; \text{ phase} (\text{Suc } r) = \text{phase } r; \)
\( \forall p. \text{next0 } r p (\rho r p) \)
\( (\text{HOrcvdMsgs} \text{ LV-M } r p (\text{HO}_s r p) (\rho r)) \)
\( (\text{coords} (\text{Suc } r) p (\rho (\text{Suc } r) p)) \]
\( \implies P r \)
and \( \text{step1}: [ \text{ step } r = 1; \text{ step} (\text{Suc } r) = 2; \text{ phase} (\text{Suc } r) = \text{phase } r; \)
\( \forall p. \text{next1 } r p (\rho r p) \)
\( (\text{HOrcvdMsgs} \text{ LV-M } r p (\text{HO}_s r p) (\rho r)) \)
\( (\text{coords} (\text{Suc } r) p (\rho (\text{Suc } r) p)) \]
\( \implies P r \)
and \( \text{step2}: [ \text{ step } r = 2; \text{ step} (\text{Suc } r) = 3; \text{ phase} (\text{Suc } r) = \text{phase } r; \)
\( \forall p. \text{next2 } r p (\rho r p) \)
\( (\text{HOrcvdMsgs} \text{ LV-M } r p (\text{HO}_s r p) (\rho r)) \)
\( (\text{coords} (\text{Suc } r) p (\rho (\text{Suc } r) p)) \]
\( \implies P r \)
and \( \text{step3}: [ \text{ step } r = 3; \text{ step} (\text{Suc } r) = 0; \text{ phase} (\text{Suc } r) = \text{Suc} (\text{phase } r); \)
\( \forall p. \text{next3 } r p (\rho r p) \)
\( (\text{HOrcvdMsgs} \text{ LV-M } r p (\text{HO}_s r p) (\rho r)) \)
\( (\text{coords} (\text{Suc } r) p (\rho (\text{Suc } r) p)) \]
\( \implies P r \)
shows \( P_r \)

proof –

from run

have \( \text{nxt: CHOnextConfig LV-M r (\rho_r)} \ (\text{HOs}_r) \ (\text{coords}(\text{Suc}_r)) \ (\rho(\text{Suc}_r)) \)
  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 \( \text{stp: step}_r = 0 \)
  hence \( \text{step}(\text{Suc}_r) = 1 \)
    by (auto simp add: step-def mod-Suc)

  with \( \text{nxt stp show } \) \( ? \text{thesis} \)
    by (intro step0)
    (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                 LV-nextState-def LV-sendMsg-def phase-Suc)

next

  assume \( \text{stp: step}_r = \text{Suc}_0 \)
  hence \( \text{step}(\text{Suc}_r) = 2 \)
    by (auto simp add: step-def mod-Suc)

  with \( \text{nxt stp show } \) \( ? \text{thesis} \)
    by (intro step1)
    (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                 LV-nextState-def LV-sendMsg-def phase-Suc)

next

  assume \( \text{stp: step}_r = 2 \)
  hence \( \text{step}(\text{Suc}_r) = 3 \)
    by (auto simp add: step-def mod-Suc)

  with \( \text{nxt stp show } \) \( ? \text{thesis} \)
    by (intro step2)
    (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                 LV-nextState-def LV-sendMsg-def phase-Suc)

next

  assume \( \text{stp: step}_r = 3 \)
  hence \( \text{step}(\text{Suc}_r) = 0 \)
    by (auto simp add: step-def mod-Suc)

  with \( \text{nxt stp show } \) \( ? \text{thesis} \)
    by (intro step3)
    (auto simp: LV-CHOMachine-def CHOnextConfig-eq
                 LV-nextState-def LV-sendMsg-def phase-Suc)

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 \( \text{run: CHORun LV-M} \ \rho \ \text{HOs} \ \text{coords} \)
and \textit{init}: \textit{CinitState}\ LV-M p (rho 0 p) (coords 0 p) \implies P p 0

and \textit{step0}: \forall r. [ step r = 0; P p r; phase (Suc r) = phase r; step (Suc r) = 1;
  next0 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p (Suc r)

and \textit{step1}: \forall r. [ step r = 1; P p r; phase (Suc r) = phase r; step (Suc r) = 2;
  next1 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p (Suc r)

and \textit{step2}: \forall r. [ step r = 2; P p r; phase (Suc r) = phase r; step (Suc r) = 3;
  next2 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p (Suc r)

and \textit{step3}: \forall r. [ step r = 3; P p r; phase (Suc r) = Suc (phase r); step (Suc r) = 0;
  next3 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p (Suc r)

\textit{shows} P p n

by (rule \textit{LV-induct}![OF run])

(auto intro: init step0 step1 step2 step3)

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

\textit{and} \textit{step1}: [ step r = 1; step (Suc r) = phase r;
  next1 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p r

\textit{and} \textit{step2}: [ step r = 2; step (Suc r) = phase r;
  next2 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p r

\textit{and} \textit{step3}: [ step r = 3; step (Suc r) = 0; phase (Suc r) = Suc (phase r);
  next3 r p (rho r p)
  (HOrcvdMsgs\ LV-M r p (HOs r p) (rho r))
  (coords (Suc r) p) (rho (Suc r) p) ]
  \implies P p r

\textit{shows} P p r
by (rule LV-Suc[OF run])
(auto intro: step0 step1 step2 step3)

7.5 Boundedness and Monotonicity of Timestamps

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

**Lemma LV-timestamp-bounded:**

- **Assumes** run :: CHORun LV-M rho HOs coords
- **Shows** timestamp (rho n p) ≤ (if step n < 2 then phase n else Suc (phase n))

**Proof** (rule LV-induct[OF run], where P = ?P)
(auto simp: LV-CHOMachine-def LV-initState-def next0-def next1-def next2-def next3-def)

Moreover, timestamps can only grow over time.

**Lemma LV-timestamp-increasing:**

- **Assumes** run :: CHORun LV-M rho HOs coords
- **Shows** timestamp (rho n p) ≤ timestamp (rho (Suc n) p)

**Proof** (rule LV-Suc[OF run, where P = ?P])
(auto simp: LV-CHOMachine-def LV-initState-def next0-def next1-def next2-def next3-def)

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

**Assume** stp :: step n = 1

- **And** next :: next1 n p (rho n p)
  (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
  (coords (Suc n) p) (rho (Suc n) p)

**From** stp **have** ?ts ≤ phase n

**Using** LV-timestamp-bounded[OF run, where n=n, where p=p] by auto

**With** nat show ?thesis by (auto simp add: next1-def)

**Qed** (auto simp add: next0-def next2-def next3-def)

**Lemma LV-timestamp-monotonic:**

- **Assumes** run :: CHORun LV-M rho HOs coords and le :: m ≤ n
- **Shows** timestamp (rho m p) ≤ timestamp (rho n p)

**Proof**

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

**Have** ?ts m ≤ ?ts (m + k)

**Proof** (induct k)
  - **Case** 0 **show** ?P 0 by simp

**Next**

- **Fix** k

**Assume** ih :: ?P k

**From** run **have** ?ts (m + k) ≤ ?ts (m + Suc k)

**By** (auto simp add: LV-timestamp-increasing)
with th show ?P (Suc k) by simp
qed

with k show ?thesis by simp
qed

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

definition procsBeyondTS where
procsBeyondTS ts cfg ≡ \{ p . ts ≤ timestamp cfg p \}

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

lemma procsBeyondTS-monotonic:
assumes run: CHORun LV-M rho HOs coords
and p: p ∈ procsBeyondTS ts (rho m) and le: m ≤ n
shows p ∈ procsBeyondTS ts (rho n)
proof –
from p have ts ≤ timestamp (rho m p) (is - ≤ ?ts m)
  by (simp add: procsBeyondTS-def)
moreover
from run le have ?ts m ≤ ?ts n by (rule LV-timestamp-monotonic)
ultimately show ?thesis
  by (simp add: procsBeyondTS-def)
qed

7.6 Obvious Facts About the Algorithm

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

Coordinators change only at step 3.

lemma notStep3EqualCoord:
assumes run: CHORun LV-M rho HOs coords and stp:step r ≠ 3
shows coordΦ (rho (Suc r) p) = coordΦ (rho r p) (is ?P p r)
by (rule LV-Suc\[OF run, where P=\{P\}]
  (auto simp: stp next0-def next1-def next2-def)

lemma coordinators:
assumes run: CHORun LV-M rho HOs coords
shows coordΦ (rho r p) = coords (4*(phase r)) p
proof –
let ?r0 = (4*(phase r) - 1)
let ?r1 = (4*(phase r))
have coordΦ (rho ?r1 p) = coords ?r1 p
proof (cases phase r > 0)
case False
hence phase $r = 0$ by auto

with run show ?thesis
  by (auto simp: LV-CHOMachine-def CHORun-eq CHOinitConfig-def LV-initState-def)

next
case True
hence step (Suc $?r0) = 0 by (auto simp: step-def)
hence step $?r0 = 3 by (auto simp: mod-Suc step-def)

moreover
from run
have LV-nextState $?r0 p (rho $?r0 p)
  (HOrcvdMsgs LV-M $?r0 p (HOs $?r0 p) (rho $?r0))
  (coords (Suc $?r0) p) (rho (Suc $?r0) p)
by (auto simp: LV-CHOMachine-def CHORun-eq CHOnextConfig-eq)

ultimately
have nxt: next3 $?r0 p (rho $?r0 p)
  (HOrcvdMsgs LV-M $?r0 p (HOs $?r0 p) (rho $?r0))
  (coords (Suc $?r0) p) (rho (Suc $?r0) p)
by (auto simp: LV-nextState-def)

hence 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 $?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 −→ commt (rho (Suc r) p) = commt (rho r p)
  (is ??P p r)
by (rule LV-Suc"[OF run, where P=??P]"

73
Timestamps only change at step 1.

**lemma notStep1EqualTimestamp** [rule-format]:
assumes run: CHORun LV-M rho HOs coords
shows step r \(\neq 1 \rightarrow \text{timestamp} (\rho (\text{Suc } r) \ p) = \text{timestamp} (\rho r \ p)
\) (is \(?P r\))
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 \(\neq 1 \rightarrow x (\rho (\text{Suc } r) \ p) = x (\rho r \ p)\) (is \(?P p\))
by (rule LV-Suc[OF run, where \(P=?P]\))
(auto simp: next0-def next1-def next2-def next3-def)

A process \(p\) has its \(commit\) flag set only if the following conditions hold:

- the step number is at least 1,
- \(p\) considers itself to be the coordinator,
- \(p\) has a non-null \(vote\),
- a majority of processes consider \(p\) as their coordinator.

**lemma commitE**:
assumes run: CHORun LV-M rho HOs coords and cmt: commit (rho r p)
and conds: \[ 1 \leq \text{step } r; \text{coordΦ} (\rho r p) = p; \text{vote} (\rho r p) \neq \text{None}; \text{card} \{q . \text{coordΦ} (\rho r q) = p\} > N \text{ div } 2 \]

shows \(A\)

proof —
have \(\text{commit} (\rho r p) \rightarrow\)
\[ 1 \leq \text{step } r \]
\(\land \text{coordΦ} (\rho r p) = p\)
\(\land \text{vote} (\rho r p) \neq \text{None}\)
\(\land \text{card} \{q . \text{coordΦ} (\rho r q) = p\} > N \text{ div } 2\)
(is \(?P p\) is - \(\rightarrow \?R r\))

proof (rule LV-induct[OF run, where \(P=?P]\))
— the only interesting step is step 0

fix \(n\)

assume next: next0 n p (rho n p) (HOrcvdMsgs LV-M n p (HOs n p) (rho n))
(coords (Suc n) p) (rho (Suc n) p)
and ph: phase (Suc n) = phase n
and stp: step n = 0 and stp': step (Suc n) = 1
and ih: \(?P p n\)

show \(?P p (Suc n)\)
proof
assume \( cm' : \text{commt}\ (\rho\ (\text{Suc\ } n)\ p) \)
from \( \text{stp\ ih} \) have \( \neg \text{commt}\ (\rho\ n\ p) \) by simp
with \( \text{nxt\ } cm' \)
have \( \text{coord}\ (\rho\ n\ p) = p \)
\[ \land \text{vote}\ (\rho\ (\text{Suc\ } n)\ p) \neq \text{None} \]
\[ \land \text{card}\ (\text{valStampsRcvd}\ (\text{HOrcvdMsgs}\ \text{LV-M}\ n\ p\ (\text{HOs}\ n\ p)\ (\rho\ n))) > N\ \text{div}\ 2 \]
by (auto simp add: next0-def)
moreover
from \( \text{stp} \)
have \( \text{valStampsRcvd}\ (\text{HOrcvdMsgs}\ \text{LV-M}\ n\ p\ (\text{HOs}\ n\ p)\ (\rho\ n)) \subseteq \{ q . \text{coord}\ (\rho\ n\ q) = p \} \)
by (auto simp: valStampsRcvd-def LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def send0-def)
hence \( \text{card}\ (\text{valStampsRcvd}\ (\text{HOrcvdMsgs}\ \text{LV-M}\ n\ p\ (\text{HOs}\ n\ p)\ (\rho\ n))) \leq \text{card}\ \{ q . \text{coord}\ (\rho\ n\ q) = p \} \)
by (auto intro: card-mono)
moreover
note \( \text{stp\ stp'}\ \text{run} \)
ultimately
show \(?R\ (\text{Suc}\ n)\) by (auto simp: notStep3EqualCoord)
qed
d
— the remaining cases are all solved by expanding the definitions
qed

A process has a current timestamp only if:

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

lemma \( \text{currentTimestampE} \):
assumes \( \text{run}: \text{CHORun}\ \text{LV-M}\ \rho\ \text{HOs\ coords} \)
and \( \text{ts: timestamp}\ (\rho\ r\ p) = \text{Suc}\ (\text{phase}\ r) \)
and \( \text{conds}: 2 \leq \text{step}\ r; \)
\[ \text{commt}\ (\rho\ r\ (\text{coord}\ (\rho\ r\ p))); \]
\[ x\ (\rho\ r\ p) = \text{the}\ (\text{vote}\ (\rho\ r\ (\text{coord}\ (\rho\ r\ p)))) \]
\[ \imp A \]
shows \( A \)
proof
let \( \text{ts\ n} = \text{timestamp}\ (\rho\ n\ p) \)
let \( \text{crd\ n} = \text{coord}\ (\rho\ n\ p) \)
have \( \text{ts\ r} = \text{Suc}\ (\text{phase}\ r) \imp 2 \leq \text{step}\ r \)
∧ commt (rho r (-uppercase crd r))
∧ x (rho r p) = the (vote (rho r (uppercase crd r)))
(is ?Q p r is - → ?R r)

proof (rule LV-induct[OF run, where P=Q])
— The assertion is trivially true initially because the timestamp is 0.

assume CinitState LV-M p (rho 0 p) (coords 0 p) thus ?Q p 0
by (auto simp: LV-CHOMachine-def LV-initState-def)

next

The assertion is trivially preserved by step 0 because the timestamp in the post-state cannot be current (cf. lemma LV-timestamp-bounded).

fix n

assume stp': step (Suc n) = 1
with run LV-timestamp-bounded[where n=Suc n] have ?ts (Suc n) ≤ phase (Suc n) by auto
thus ?Q p (Suc n) by simp

next

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

fix n

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

qed

next

For step 2, the assertion follows from the induction hypothesis, observing that none of the relevant state components change.

fix n
assume stp: step n = 2 and stp': step (Suc n) = 3
and ph: phase (Suc n) = phase n
and ih: "Q p n
and nxt: next2 n p ( rho n) (HOrcvdMsgs LV-M n p (HOs n p) (rho))
(show "Q p (Suc n)
proof
assume ts: "ts (Suc n) = Suc (phase (Suc n))
from run stp
have vt: vote (rho (Suc n) ("crd (Suc n))) = vote (rho n ("crd n))
by (auto simp add: notStep3EqualCoord notStep0EqualVote)
from run stp
have cmt: commt (rho (Suc n) ("crd (Suc n))) = commt (rho n ("crd n))
by (auto simp add: notStep3EqualCoord notStep03EqualCommit)
with vt ts ph stp stp' ih nxt
show "R (Suc n)
by (auto simp add: next2-def)
qed
next

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

fix n
assume stp': step (Suc n) = 0
with run LV-timestamp-bounded[where n=Suc n]
have "ts (Suc n) \leq phase (Suc n) by auto
thus "Q p (Suc n) by simp
qed
with ts show "thesis by (intro conds) auto
qed

If a process p has its ready bit set then:

• it is at step 3,
• it considers itself to be the coordinator of that phase and
• a majority of processes considers p to be the coordinator and has a current timestamp.

lemma readyE:
assumes run: CHORun LV-M rho HOs coords and rdy: ready (rho r p)
and conds: \[ step r = 3; coordΦ (rho r p) = p; \]
\[ \text{card} \{ q . \text{coordΦ} (rho r q) = p \land \text{timestamp} (rho r q) = Suc (phase r) \} > N \text{ div } 2 \]
shows P
proof
let \( qs n = \{ q . \text{coordΦ} (rho n q) = p \land \text{timestamp} (rho n q) = Suc (phase n) \} \)
have ready (\rho \, r \, p) \implies 
\begin{align*}
& \text{step } r = 3 \\
& \text{coord} \Phi (\rho \, r \, p) = p \\
& \text{card} (\mathbb{?}qs \, r) > N \div 2 \\
& (\text{is } \mathbb{?}Q \, p \, r \implies -\implies \mathbb{?}R \, p \, r)
\end{align*}

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

— the interesting case is step 2

fix \ n

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

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

\quad \text{and } \ 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 \mathbb{?}Q \ p \ (\text{Suc } n)

proof

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

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

with \ rdy \ nxt \ have \ 
\text{coord} \Phi (\rho \ n \ p) = p

by \ (\text{auto simp: next2-def})

with \ run \ stp \ have \ 
\text{coord} \Phi (\rho \ (\text{Suc } n) \ p) = p

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

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

from \ nrdy \ rdy \ nxt \ have \ aRcvd: \text{card } ?acks > N \div 2

by \ (\text{auto simp: next2-def})

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

proof \ (\text{clarify})

fix \ q

assume \ q: \ q \in \ ?acks

with \ stp

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

by \ (\text{auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def acksRcvd-def send2-def isAck-def})

with \ run \ stp \ ph

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

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

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

qed

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

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

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

by \ auto

qed

— the remaining steps are all solved trivially

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

with \ rdy \ show \ \mathbb{?}thesis \ \text{by} \ (\text{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 \textit{ready} and \textit{commit} bits set.

\textbf{lemma} \textit{decisionE}:
\textbf{assumes} \textit{run}: \textit{CHORun} \textit{LV-M} \textit{rho} \textit{HOs} \textit{coords}
\textbf{and} \textit{dec}: \textit{decide} (\textit{rho} (\textit{Suc} \textit{r}) \textit{p}) \neq \textit{decide} (\textit{rho} \textit{r} \textit{p})
\textbf{and} \textit{conds}:
\begin{enumerate}
  \item \textit{step} \textit{r} = 3;
  \item \textit{decide} (\textit{rho} (\textit{Suc} \textit{r}) \textit{p}) = \textit{Some} (\textit{vote} (\textit{rho} \textit{r} (\textit{coord} \textit{Phi} (\textit{rho} \textit{r} \textit{p}))));
  \item \textit{ready} (\textit{rho} \textit{r} (\textit{coord} \textit{Phi} (\textit{rho} \textit{r} \textit{p}))); \textit{commit} (\textit{rho} \textit{r} (\textit{coord} \textit{Phi} (\textit{rho} \textit{r} \textit{p})))
\end{enumerate}
\implies \textit{P}
\textbf{shows} \textit{P}
\textbf{proof} –
\begin{enumerate}
  \item let \textit{?cfg} = \textit{rho} \textit{r}
  \item let \textit{?cfg'} = \textit{rho} (\textit{Suc} \textit{r})
  \item let \textit{?crd} \textit{p} = \textit{coord} \textit{Phi} (\textit{?cfg} \textit{p})
  \item let \textit{?dec'} = \textit{decide} (\textit{?cfg'} \textit{p})
\end{enumerate}
Except for the assertion about the \textit{commit} field, the assertion can be proved directly from the next-state relation.
\begin{enumerate}
  \item \textbf{have 1:} \textit{step} \textit{r} = 3
\end{enumerate}
\begin{enumerate}
  \item \wedge \textit{?dec'} = \textit{Some} (\textit{vote} (\textit{?cfg} (\textit{?crd} \textit{p})))
  \item \wedge \textit{ready} (\textit{?cfg} (\textit{?crd} \textit{p}))
\end{enumerate}
\begin{enumerate}
  \item (is \textit{?Q} \textit{p} \textit{r})
\end{enumerate}
\begin{enumerate}
  \item \textbf{proof} (rule \textit{LV-Suc}[OF \textit{run}, \textbf{where} \textit{P}=?Q])
\end{enumerate}
— for step 3, we prove the thesis by expanding the relevant definitions
\begin{enumerate}
  \item \textbf{assume} \textit{next3} \textit{r} \textit{p} (\textit{?cfg} \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{r} \textit{p}) \textit{?cfg})
\end{enumerate}
\begin{enumerate}
  \item \text{(\textit{coords} (\textit{Suc} \textit{r}) \textit{p}) (\textit{?cfg} \textit{p})}
\end{enumerate}
\begin{enumerate}
  \item \textbf{and} \textit{step} \textit{r} = 3
\end{enumerate}
\begin{enumerate}
  \item \textbf{with} \textit{dec} \textbf{show} \textit{?thesis}
\end{enumerate}
\begin{enumerate}
  \item \textbf{by} (\textbf{auto simp:} \textit{next3-def} \textit{send3-def} \textit{isVote-def} \textit{LV-CHOMachine-def} \textit{HOrcvdMsgs-def} \textit{LV-sendMsg-def})
\end{enumerate}
\begin{enumerate}
  \item \textbf{next}
\end{enumerate}
\begin{enumerate}
  \item \text{— the other steps don’t change the decision}
\end{enumerate}
\begin{enumerate}
  \item \textbf{assume} \textit{next0} \textit{r} \textit{p} (\textit{?cfg} \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{r} \textit{p}) \textit{?cfg})
\end{enumerate}
\begin{enumerate}
  \item \text{(\textit{coords} (\textit{Suc} \textit{r}) \textit{p}) (\textit{?cfg} \textit{p})}
\end{enumerate}
\begin{enumerate}
  \item \textbf{with} \textit{dec} \textbf{show} \textit{?thesis} \textbf{by} (\textbf{auto simp:} \textit{next0-def})
\end{enumerate}
\begin{enumerate}
  \item \textbf{next}
\end{enumerate}
\begin{enumerate}
  \item \textbf{assume} \textit{next1} \textit{r} \textit{p} (\textit{?cfg} \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{r} \textit{p}) \textit{?cfg})
\end{enumerate}
\begin{enumerate}
  \item \text{(\textit{coords} (\textit{Suc} \textit{r}) \textit{p}) (\textit{?cfg} \textit{p})}
\end{enumerate}
\begin{enumerate}
  \item \textbf{with} \textit{dec} \textbf{show} \textit{?thesis} \textbf{by} (\textbf{auto simp:} \textit{next1-def})
\end{enumerate}
\begin{enumerate}
  \item \textbf{next}
\end{enumerate}
\begin{enumerate}
  \item \textbf{assume} \textit{next2} \textit{r} \textit{p} (\textit{?cfg} \textit{p}) (\textit{HOrcvdMsgs} \textit{LV-M} \textit{r} \textit{p} (\textit{HOs} \textit{r} \textit{p}) \textit{?cfg})
\end{enumerate}
\begin{enumerate}
  \item \text{(\textit{coords} (\textit{Suc} \textit{r}) \textit{p}) (\textit{?cfg} \textit{p})}
\end{enumerate}
\begin{enumerate}
  \item \textbf{with} \textit{dec} \textbf{show} \textit{?thesis} \textbf{by} (\textbf{auto simp:} \textit{next2-def})
\end{enumerate}
\textbf{qed}
\begin{enumerate}
  \item \textbf{hence} \textit{ready} (\textit{?cfg} (\textit{?crd} \textit{p})) \textbf{by} \textit{blast}
\end{enumerate}
Because the coordinator is ready, there is a majority of processes that consider it to be the coordinator and that have a current timestamp.

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

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

### 7.7 Proof of Integrity

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

**Lemma** lv-integrityInvariant:

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

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

\[ \implies A \]

**Shows** A

**Proof** –

let \( ?x0 = \text{range } (x \circ (\rho 0)) \)

let \( ?x0opt = \{\text{None}\} \cup \text{Some } ?x0 \)

have range \( (x \circ \rho n) \subseteq ?x0 \)

\[ \land \text{range } (\text{vote } \circ \rho n) \subseteq ?x0opt \]

\[ \land \text{range } (\text{decide } \circ \rho n) \subseteq ?x0opt \]

(is \( \text{?Inv } n \) is \( \text{?X } n \land \text{?Vote } n \land \text{?Decide } n \))

**Proof (induct n)**

from run show \( ?\text{Inv } 0 \)

by (auto simp: CHORun-eq CHOinitConfig-def LV-CHOMachine-def LV-initState-def)

**Next**

fix \( n \)

assume \( \text{ih: } ?\text{Inv } n \text{ thus } ?\text{Inv } (\text{Suc } n) \)

**Proof (clarify)**

assume \( x: ?\text{X } n \) and \( \text{vt: } ?\text{Vote } n \) and \( \text{dec: } ?\text{Decide } n \)

Proof of first conjunct

\[ \text{have } x': ?\text{X } (\text{Suc } n) \]

**Proof (clarsimp)**

\[ \text{fix } p \]
from run
show x (\rho (Suc n) p) \in \text{range}\ (\lambda q. x (\rho 0 q)) \ (\text{is } ?P p n)
proof (rule LV-Suc[\textbf{where } P=\ ?P])
— only step1 is of interest
assume stp: stp n = 1
and nxt: next1 n p (\rho n p)
\ (\text{HOrcvdMsgs } LV-M n p \ (\text{HOS} n p) \ (\rho n))
\ (\text{coords} \ (Suc n) p) \ (\rho \ (Suc n) p)
show \ ?thesis
proof (cases \rho (Suc n) p = \rho n p)
next
assume step n = 0
with run have x (\rho (Suc n) p) = x (\rho n p)
by (simp add: notStep1EqualX)
with x show \ ?thesis by auto
next
assume step n = 2
with run have x (\rho (Suc n) p) = x (\rho n p)
by (simp add: notStep1EqualX)
with x show \ ?thesis by auto
next
assume step n = 3
with run have x (\rho (Suc n) p) = x (\rho n p)
by (simp add: notStep1EqualX)
with x show \ ?thesis by auto
qed
qed

Proof of second conjunct

have vt': \ ?Vote \ (Suc n)
proof (clarsimp simp: image-def)
fix p v
assume v: vote (rho (Suc n) p) = Some v
from run have vote (rho (Suc n) p) = Some v \rightarrow v \in {?}x0 (is {?}P p n)
proof (rule LV-Suc'[where P=?P])
-- here only step0 is of interest
assume stp: step n = 0
and nxt: next0 n p (rho n p)
(HOrcvdMsgs LV-M n p (HOs n p) (rho n))
(coords (Suc n) p) (rho (Suc n) p)
show ?thesis
proof (cases rho (Suc n) p = rho n p)
case True
from vt have vote (rho n p) \in {?}x0opt
by (auto simp: image-def)
with True show ?thesis by auto
next
case False
from nxt stp False v obtain q where v = x (rho n q)
by (auto simp: next0-def send0-def LV-CHOMachine-def
HOrcvdMsgs-def LV-sendMsg-def)
with x show ?thesis by (auto simp: image-def)
qed
-- the other cases don’t change the vote
next
assume step n = 1
with run have vote (rho (Suc n) p) = vote (rho n p)
by (simp add: notStep0EqualVote)
moreover from 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
by (auto simp: image-def)
ultimately
show ?thesis by auto
qed
with v show \( \exists q. v = x (\rho 0 q) \) by auto
qed

Proof of third conjunct

have \( \text{dec'} : \text{Decide (Suc n)} \)
proof (clarsimp simp: image-def)
  fix p v
  assume v: decide (\rho (Suc n) p) = Some v
  show \( \exists q. v = x (\rho 0 q) \)
  proof (cases decide (\rho (Suc n) p) = decide (\rho n p))
    case True
    with dec True v show ?thesis by (auto simp: image-def)
  next
    case False
    let \( \text{?crd} = \text{coord}_\Phi (\rho n p) \)
    from False run have d': decide (\rho (Suc n) p) = Some (the (vote (\rho n \text{?crd})))
      and cmt: commit (\rho n \text{?crd})
      by (auto elim: decisionE)
    from vt have vtc: vote (\rho n \text{?crd}) \( \in ?x0opt \)
      by (auto simp: image-def)
    from run cmt have vote (\rho n \text{?crd}) \( \neq \) None
      by (rule commitE)
    with d' v vtc show ?thesis by auto
    qed
  qed
  from x' vt' dec' show ?thesis by simp
  qed
  with inv show ?thesis by simp
  qed

Integrity now follows immediately.

theorem lv-integrity:
  assumes run: CHORun LV-M rho HOs coords
  and dec: decide (\rho n p) = Some v
  shows \( \exists q. v = x (\rho 0 q) \)
proof
  from run have decide (\rho n p) \( \in \{\text{None}\} \cup \text{Some } \langle \text{range } (x \circ (\rho 0)) \rangle \)
    by (rule lv-integrityInvariant) (auto simp: image-def)
  with dec show ?thesis by (auto simp: image-def)
  qed
7.8 Proof of Agreement and Irrevocability

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

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

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

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

let ?crd = coord\(\Phi\) (rho r p)
let ?qs = \{ q . coord\(\Phi\) (rho r q) = ?crd ∧ timestamp (rho r q) = Suc (phase r) \}
assume stp: step r = 3 and rdy: ready (rho r ?crd)

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

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

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

**Lemma** committedProcsEqual:
- **assumes** run: CHORun LV-M rho HOs coords
- and cmt: commt (rho r p) and cmt': commt (rho r p')
- **shows** p = p'
- **proof** –
  from run cmt have card \{ q . coord\(\Phi\) (rho r q) = p\} > N div 2
  by (blast elim: commtE)
moreover
from run cmt' have card \{ q . coord\(\Phi\) (rho r q) = p'\} > N div 2
by (blast elim: commtE)
ultimately
obtain q where coord\(\Phi\) (rho r q) = p and p' = 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: \texttt{CHORun LV-M rho HOs coords}
and rdg: ready (rho r p) and rdg': ready (rho r p')
shows p = p'

proof —
let \( \forall C \ p = \{ q \ . \ \text{coord}(\rho r q) = p \land \text{timestamp}(\rho r q) = \text{Suc}(\text{phase r}) \} \)
from run rdg have card (\( \forall C \ p \)) > N div 2
by (blast elim: readyE)
moresover from run rdg' have card (\( \forall C \ p' \)) > 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 \( \forall \text{thesis by simp} \)

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.

\textbf{lemma commitThenVoteRecent:}
assumes run: \texttt{CHORun LV-M rho HOs coords}
and maj: \( \text{card}(\text{procsBeyondTS ts (rho r)}) > N \text{ div } 2 \)
and cmt: \( \text{commt}(\rho r p) \)
shows \( \exists q \in \text{procsBeyondTS ts (rho r)} \ . \ \text{vote}(\rho r p) = \text{Some}(x (\rho r q)) \)
(is \( \forall Q \ r \))

proof —
let \( \forall \text{bynd n} = \text{procsBeyondTS ts (rho n)} \)
have \( \text{card}(\forall \text{bynd r}) > N \text{ div } 2 \land \text{commt}(\rho r p) \rightarrow \forall Q \ r \ (\text{is \ ?P p r}) \)
proof (rule LV-induct[\textit{OF run}])

\( \text{next0} \) establishes the property

fix n
assume stp: step n = 0
and nxt: \( \forall q . \text{next0 n q (rho n q)} \)
(\( \text{HOrcvdMsgs LV-M n q (HOS n q) (rho n)} \))
(\( \text{coords (Suc n) q)} \))
(\( \text{rho (Suc n) q)} \))
(is \( \forall q . \ ?nxt q \))
from nxt have \text{nxp: \ ?nxtr p ..}
show \( \forall P p (\text{Suc n}) \)
proof (clarify)
assume mj: \( \text{card}(\forall \text{bynd (Suc n)}) > N \text{ div } 2 \)
and ct: \( \text{commt}(\rho (\text{Suc n}) p) \)
show \( \forall Q (\text{Suc n}) \)
proof —
let \( \forall \text{msgs} = \text{HOrcvdMsgs LV-M n p (HOS n p) (rho n)} \)
from stp run have \( \neg \text{commt}(\rho (\text{n x n p}) \) by (auto elim: commitE)
with \text{nxp ct obtain q v where}
\( v: \forall \text{msgs q = Some (ValStamp v (highestStampRcvd ?msgs)) and} \)
vote: vote (\rho \cdot \text{Suc } n \cdot p) = \text{Some } v \text{ and}
rcvd: \text{card } (\text{valStampsRcvd } ?msgs) > N \text{ div } 2
   \text{by (auto simp: next0-def)}

\text{from mj rcvd obtain } q' \text{ where}
   q1': q' \in ?bynd (Suc n) \text{ and } q2': q' \in \text{valStampsRcvd } msgs
   \text{by (rule majoritiesE')}
\text{have timestamp } (\rho \cdot \text{Suc } n \cdot q') \leq \text{timestamp } (\rho \cdot n \cdot q)

\text{proof --}
   \text{from q2' obtain v' ts'}
     \text{where ts': ?msgs } q' = \text{Some } (\text{ValStamp } v' \cdot ts')
     \text{by (auto simp: valStampsRcvd-def)}
   \text{hence ts'} \leq \text{highestStampRcvd } ?msgs
     \text{by (rule highestStampRcvd-max)}
   \text{moreover}
     \text{from ts' stp have timestamp } (\rho \cdot n \cdot q') = ts'
       \text{by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def send0-def)}
   \text{moreover}
     \text{from v stp have timestamp } (\rho \cdot n \cdot q) = \text{highestStampRcvd } ?msgs
       \text{by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def send0-def)}
   \text{ultimately}
   \text{show } ?\text{thesis by simp}

\text{qed}

\text{moreover}
\text{from run stp}
\text{have timestamp } (\rho \cdot \text{Suc } n \cdot q') = \text{timestamp } (\rho \cdot n \cdot q')
   \text{by (simp add: notStep1EqualTimestamp)}
\text{moreover}
\text{from run stp}
\text{have timestamp } (\rho \cdot \text{Suc } n \cdot q) = \text{timestamp } (\rho \cdot n \cdot q)
   \text{by (simp add: notStep1EqualTimestamp)}
\text{moreover}
\text{note q1'}
\text{ultimately}
\text{have } q \in ?bynd (Suc n)
   \text{by (simp add: procsBeyondTS-def)}
\text{moreover}
\text{from v vote stp}
\text{have vote } (\rho \cdot \text{Suc } n \cdot p) = \text{Some } (x (\rho \cdot n \cdot q))
   \text{by (auto simp: LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def send0-def)}
\text{moreover}
\text{from run stp have } x (\rho \cdot \text{Suc } n \cdot q) = x (\rho \cdot n \cdot q)
   \text{by (simp add: notStep1EqualX)}
\text{ultimately}
\text{show } ?\text{thesis by force}
\text{qed}
\text{qed}
next

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.

\[
\text{fix } n
\]
\[
\text{assume } \text{stp}: \text{step } n = 1
\]
\[
\text{and } \text{nxt}: \forall q. \text{next1 } n \ q \ (\rho n \ q)
\]
\[
(\text{HOrcvdMsgs} \ \text{LV-M } n \ q \ (\text{HOs } n \ q) \ (\rho n) )
\]
\[
(\text{coords } (\text{Suc } n) \ q)
\]
\[
(\rho (\text{Suc } n) \ q)
\]
\[
(\text{is } \forall q. \ ?\text{nxt } q)
\]
\[
\text{and } \text{ih}: \ ?P \ p \ n
\]
\[
\text{from } \text{nxt } \text{have } \ ?\text{nxt } p ..
\]
\[
\text{show } ?P \ p \ (\text{Suc } n)
\]
\[
\text{proof } (\text{clarify})
\]
\[
\text{assume } m:\text{j }': \text{card } (\?\text{bynd } (\text{Suc } n)) > N \text{ div } 2
\]
\[
\text{and } c:\text{l }': \text{commit } (\rho (\text{Suc } n) \ p)
\]
\[
\text{from } \text{run } \text{stp } c' \text{ have } c: \text{commit } (\rho n \ p)
\]
\[
\text{by } (\text{simp add: notStep03EqualCommit})
\]
\[
\text{from } \text{run } \text{stp } \text{have } \text{vote}' : \text{vote } (\rho (\text{Suc } n) \ p) = \text{vote } (\rho n \ p)
\]
\[
\text{by } (\text{simp add: notStep0EqualVote})
\]
\[
\text{show } ?Q \ (\text{Suc } n)
\]
\[
\text{proof } (\text{cases } \exists q \in ?\text{bynd } (\text{Suc } n). \rho (\text{Suc } n) \ q \neq \rho n \ q)
\]
\[
\text{case True}
\]
in this case the property holds because \( q \) updates its \( x \) field to the vote

\[
\text{then obtain q where}
\]
\[
q1 : q \in \?\text{bynd } (\text{Suc } n) \text{ and } q2 : \rho (\text{Suc } n) \ q \neq \rho n \ q ..
\]
\[
\text{from } \text{nxt } \text{have } ?\text{nxt } q ..
\]
\[
\text{with } q2 \text{ stp}
\]
\[
\text{have } x': x (\rho (\text{Suc } n) \ q) = \text{the } (\text{vote } (\rho n (\text{coord} \Phi (\rho n \ q))))
\]
\[
\text{and } \text{coord}: \text{commit } (\rho n (\text{coord} \Phi (\rho n \ q)))
\]
\[
\text{by } (\text{auto simp: next1-def send1-def LV-CHOMachine-def HOrcvdMsgs-def}
\]
\[
\text{LV-sendMsg-def isVote-def})
\]
\[
\text{from } \text{run } c \text{ have } \text{vote} : \text{vote } (\rho n \ p) \neq \text{None}
\]
\[
\text{by } (\text{rule commitE})
\]
\[
\text{from } \text{run coord } c \text{ have } \text{coord} \Phi (\rho n \ q) = p
\]
\[
\text{by } (\text{rule committedProcsEqual})
\]
\[
\text{with } q1 \ x' \text{ vote } \text{vote}' \text{ show } ?\text{thesis} \text{ by auto}
\]
\[
\text{next}
\]
\[
\text{case False}
\]

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: procBeyondTS-def)
fix r
assume ts: ts ≤ timestamp (rho n r)
from run have timestamp (rho n r) ≤ timestamp (rho (Suc n) r)
  by (simp add: LV-timestamp-monotonic)
with ts show ts ≤ timestamp (rho (Suc n) r) by simp
qed
with mj' have mj: card (?bynd n) > N div 2 by simp
with ct ih obtain q where
  q ∈ ?bynd n and vote (rho n p) = Some (x (rho n q))
  by blast
with vote' bynd False show ?thesis by auto
qed

next

step2 preserves the property, via the induction hypothesis.

fix n
assume stp: step n = 2
  and nxt: ∀ q. next2 n q (rho n q)
    (HorcvdMsgs LV-M n q (Hos n q) (rho n))
    (coords (Suc n) q)
    (rho (Suc n) q)
    (is ∀ q. ?nxt q)
  and ih: ?P p n
from nxt have nxp: ?nxt p ..
show ?P p (Suc n)
proof (clarify)
assume mj': card (?bynd (Suc n)) > N div 2
  and ct': commit (rho (Suc n) p)
from run stp ct' have ct: commit (rho n p)
  by (simp add: notStep03EqualCommit)
from run stp have vote': vote (rho (Suc n) p) = vote (rho n p)
  by (simp add: 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: procBeyondTS-def)
from run stp have ∀ q. x (rho (Suc n) q) = x (rho n q)
  by (simp add: notStep1EqualX)
with bynd' vote' ct mj' ih show ?Q (Suc n)
  by auto
qed

the initial state and the step3 transition are trivial because the commit flag cannot
be set.

qed (auto elim: commitE[OF run])
with maj cmt show ?thesis by simp
The following lemma gives the crucial argument for agreement: after some process \( p \) has decided, all processes whose timestamp is beyond the timestamp at the point of decision contain the decision value in their \( x \) field.

**lemma** \( \text{XOfTimestampBeyondDecision} \):

**assumes** \( \text{run}: \text{CHORun} \ (LV-M \ \rho ) \ HOs \ \text{coods} \)
and \( \text{dec}: \text{decide} \ (\rho \ (\text{Suc} \ r) \ p) \neq \text{decide} \ (\rho \ r \ p) \)
**shows** \( \forall q \in \text{procsBeyondTS} \ (\text{Suc} \ (\text{phase} \ r)) \ (\rho \ (r+k)) \).

\[ x \ (\rho \ (r+k) \ q) = \text{the} \ (\text{decide} \ (\rho \ (\text{Suc} \ r) \ p)) \]

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

**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 \( \text{run} \ \text{dec} \)
have \( \text{stp}: \text{step} \ r = 3 \)
and \( \text{v}: \text{decide} \ (\rho \ (\text{Suc} \ r) \ p) = \text{Some} \ (\text{the} \ (\text{vote} \ (\rho \ r \ (\text{coord} \Phi \ (\rho \ r \ p))))) \)
and \( \text{cmt}: \text{commt} \ (\rho \ r \ (\text{coord} \Phi \ (\rho \ r \ p))) \)
by (auto elim: \text{decisionE})
from \( \text{stp} \ \text{LV-timestamp-bounded}\ [\text{OF} \ \text{run}, \ \text{where} \ n=r] \)
have \( \text{timestamp} \ (\rho \ r \ q) \leq \text{Suc} \ (\text{phase} \ r) \) by simp
with \( q \)
have \( \text{timestamp} \ (\rho \ r \ q) = \text{Suc} \ (\text{phase} \ r) \)
by (simp add: \text{procsBeyondTS-def})
with \( \text{run} \)
have \( x: \ x \ (\rho \ r \ q) = \text{the} \ (\text{vote} \ (\rho \ r \ (\text{coord} \Phi \ (\rho \ r \ q)))) \)
and \( \text{cmt'}: \text{commt'} \ (\rho \ r \ (\text{coord} \Phi \ (\rho \ r \ q))) \)
by (auto elim: \text{currentTimestampE})
from \( \text{run} \ \text{cmt} \ \text{cmt'} \)
have \( \text{coord} \Phi \ (\rho \ r \ p) = \text{coord} \Phi \ (\rho \ r \ q) \)
by (rule \text{committedProcsEqual})
with \( x \ \text{v} \)
show \( x \ (\rho \ (r+0) \ q) = ?v \) by simp
qed
next
— induction step
fix \( k \)
assume \( \text{ih}: ?P \ p \ k \)
show \( ?P \ p \ (\text{Suc} \ k) \)
**proof** (clarify)
fix \( q \)
assume \( q: q \in \ ?bynd \ (\text{Suc} \ k) \)
— distinguish the kind of transition—only \text{step1} is interesting
have \( x \ (\rho \ (\text{Suc} \ (r+k)) \ q) = ?v \ (\text{is} \ ?X q \ (r+k)) \)
**proof** (rule \( \text{LV-Suc}^{\text{OF} \ \text{run}, \ \text{where} \ P=?X} \))
assume \( \text{stp}: \text{step} \ (r+k) = 1 \)

89
and \text{nxt: next1} (r+k) q (\rho (r+k) q) \\
\hspace{1cm} (\text{HOrcvdMsgs} \text{LV-M} (r+k) q (\text{HOs} (r+k) q) (\rho (r+k))) \\
\hspace{1cm} (\text{coords} (\text{Suc} (r+k)) q) \\
\hspace{1cm} (\rho (\text{Suc} (r+k)) q) \\
\text{show} \ ?\text{thesis} \\
\text{proof} \ (\text{cases} \ \rho (\text{Suc} (r+k)) q = \rho (r+k) q) \\
\hspace{1cm} \text{case} \ True \\
\hspace{1cm} \text{with} \ q \ \text{ih} \ ?\text{thesis} \ \text{by} \ (\text{auto simp: procsBeyondTS-def}) \\
\text{next} \\
\hspace{1cm} \text{case} \ False \\
\hspace{1cm} \text{from} \ \text{run dec} \ \text{have} \ \text{card} \ (\text{?bynd} 0) > N \div 2 \ \text{by simp} \\
\hspace{2cm} \text{by} \ (\text{simp add: decisionThenMajorityBeyondTS}) \\
\hspace{1cm} \text{moreover} \\
\hspace{2cm} \text{have} \ \text{?bynd} 0 \subseteq \text{?bynd} k \\
\hspace{3cm} \text{by} \ (\text{auto elim: procsBeyondTS-monotonic[OF run]}) \\
\hspace{2cm} \text{hence} \ \text{card} \ (\text{?bynd} 0) \leq \text{card} \ (\text{?bynd} k) \\
\hspace{3cm} \text{by} \ (\text{auto intro: card-mono}) \\
\hspace{1cm} \text{ultimately} \\
\hspace{2cm} \text{have} \ \text{maj: card} \ (\text{?bynd} k) > N \div 2 \ \text{by simp} \\
\hspace{2cm} \text{let} \ ?\text{crd} = \text{coordΦ} (\rho (r+k) q) \\
\hspace{1cm} \text{from} \ \text{False stp nxt have} \\
\hspace{2cm} \text{cmt: commit} (\rho (r+k) ?\text{crd}) \ \text{and} \\
\hspace{3cm} x: x (\rho (\text{Suc} (r+k)) q) = \text{the} (\text{vote} (\rho (r+k) ?\text{crd})) \\
\hspace{3cm} \text{by} \ (\text{auto simp: next1-def LV-CHOMachine-def HOrcvdMsgs-def} \ \\
\hspace{3cm} \ \text{LV-sendMsg-def send1-def isVote-def}) \\
\hspace{2cm} \text{from} \ \text{run maj cmt stp obtain} \ q' \\
\hspace{3cm} \text{where} \ q1': q' \in \text{?bynd} k \\
\hspace{4cm} \text{and} \ q2': \text{vote} (\rho (r+k) ?\text{crd}) = \text{Some} (x (\rho (r+k) q')) \\
\hspace{4cm} \text{by} \ (\text{blast dest: commitThenVoteRecent}) \\
\hspace{2cm} \text{with} \ x \ \text{ih} \ \text{show} \ ?\text{thesis} \ \text{by} \ \text{auto} \\
\text{qed} \\
\text{next} \\
\hspace{1cm} — \text{all other steps hold by induction hypothesis} \\
\text{assume} \ step \ (r+k) = 0 \\
\hspace{1cm} \text{with} \ \text{run have} \ x: x (\rho (\text{Suc} (r+k)) q) = x (\rho (r+k) q) \\
\hspace{2cm} \text{and} \ ts: \text{timestamp} (\rho (\text{Suc} (r+k)) q) = \text{timestamp} (\rho (r+k) q) \\
\hspace{2cm} \text{by} \ (\text{auto simp: notStep1EqualX notStep1EqualTimestamp}) \\
\hspace{2cm} \text{from} \ ts q \ \text{have} \ q \in \text{?bynd} k \\
\hspace{3cm} \text{by} \ (\text{auto simp: procsBeyondTS-def}) \\
\hspace{2cm} \text{with} \ x \ \text{ih} \ \text{show} \ ?\text{thesis} \ \text{by} \ \text{auto} \\
\text{next} \\
\hspace{1cm} \text{assume} \ step \ (r+k) = 2 \\
\hspace{1cm} \text{with} \ \text{run have} \ x: x (\rho (\text{Suc} (r+k)) q) = x (\rho (r+k) q) \\
\hspace{2cm} \text{and} \ ts: \text{timestamp} (\rho (\text{Suc} (r+k)) q) = \text{timestamp} (\rho (r+k) q) \\
\hspace{2cm} \text{by} \ (\text{auto simp: notStep1EqualX notStep1EqualTimestamp}) \\
\hspace{2cm} \text{from} \ ts q \ \text{have} \ q \in \text{?bynd} k \\
\hspace{3cm} \text{by} \ (\text{auto simp: procsBeyondTS-def}) \\
\hspace{2cm} \text{with} \ x \ \text{ih} \ \text{show} \ ?\text{thesis} \ \text{by} \ \text{auto} \\
\text{next}
assume \( \text{step} (r+k) = 3 \)

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

thus \( x (\rho (r + \text{Suc} k) q) = ?v \) 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 laterProcessDecidesSameValue:
assumes run: \( \text{CHO} \text{Run} \text{LV-M} \rho \text{Hos} \text{coords} \)
and p: \( \text{decide} (\rho (\text{Suc} r) p) \neq \text{decide} (\rho r p) \)
and q: \( \text{decide} (\rho (\text{Suc} (r+k)) q) \neq \text{decide} (\rho (r+k) q) \)
shows \( \text{decide} (\rho (\text{Suc} (r+k)) q) = \text{decide} (\rho (\text{Suc} r) p) \)
proof –
let \( ?bynd k = \text{procsBeyondTS} (\text{Suc} (\text{phase} r)) (\rho (r+k)) \)
let \( ?qcrd = \text{coord} \Phi (\rho (r+k) q) \)
from run p have notNone: \( \text{decide} (\rho (\text{Suc} r) p) \neq \text{None} \)
by (auto elim: decisionE)
— process \( q \) decides on the vote of its coordinator
from run q have dec: \( \text{decide} (\rho (\text{Suc} (r+k)) q) = \text{Some} (\text{the} (\text{vote} (\rho (r+k) ?qcrd))) \)
and cmt: \( \text{commit} (\rho (r+k) ?qcrd) \)
by (auto elim: decisionE)
— that vote is the \( x \) field of some process \( q’ \) with a recent timestamp
from 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 \( \text{card} (?bynd 0) \leq \text{card} (?bynd k) \)
by (auto intro: card-mono)
ultimately
have maj: \( \text{card} (?bynd k) > N \div 2 \) 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 \( x (\rho (r+k) q’) = \text{the} (\text{decide} (\rho (\text{Suc} r) p)) \)
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** run: CHORun LV-M rho HOs coords
and \( \text{dec} \) \text{decide} (rho n p) = Some v

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

**proof**

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

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

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

**next**

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

**proof** (clarify)

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

**qed**

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

**qed**

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

**theorem** lv-irrevocability:

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

**shows** \( \text{decide} (rho (m+k) p) = \text{Some v} \)

**proof**

- from \( \text{run} \ p \) obtain \( n \) where
  \( n1: n < m \text{ and} \)
  \( n2: \text{decide} (rho (Suc n) p) \neq \text{decide} (rho n p) \text{ and} \)
  \( n3: \text{decide} (rho (Suc n) p) = \text{Some v} \)
  by (auto dest: decisionNonNullThenDecided)
have \( \forall i. \text{decide} (\rho (\text{Suc} (n+i)) p) = \text{Some} v \) (is \( \forall i. \text{?dec} i \))

proof
fix \( i \)
show \( \text{?dec} i \)
proof (induct \( i \))
from \( n3 \) show \( \text{?dec} 0 \) by simp
next
fix \( j \)
assume \( \text{ih} : \text{?dec} j \)
show \( \text{?dec} (\text{Suc} j) \)
proof (rule ccontr)
assume \( \text{ctr} : \neg (\text{?dec} (\text{Suc} j)) \)
with \( \text{ih} \)
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 \( \text{ctr n3} \) show \( \text{False} \) 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 \( \text{run} : \text{CHO}Run \text{ LV-M} \rho \text{HOs coords} \)
and \( p : \text{decide} (\rho m p) = \text{Some} v \)
and \( q : \text{decide} (\rho n q) = \text{Some} w \)
shows \( v = w \)
proof –
from \( \text{run p obtain} k \)
where \( k1 : \text{decide} (\rho (\text{Suc} k) p) \neq \text{decide} (\rho k p) \)
and \( k2 : \text{decide} (\rho (\text{Suc} k) p) = \text{Some} v \)
by (auto dest: decisionNonNullThenDecided)
from \( \text{run q obtain} l \)
where \( l1 : \text{decide} (\rho (\text{Suc} l) q) \neq \text{decide} (\rho l q) \)
and \( l2 : \text{decide} (\rho (\text{Suc} l) q) = \text{Some} w \)
by (auto dest: decisionNonNullThenDecided)
show \( \text{?thesis} \)
proof (cases \( k \leq l \))
case \text{True}
then obtain \( m \) where \( m : l = k+m \) by (auto simp: le-iff-add)
from \( \text{run k1 l1 m} \)
have \( \text{decide} (\rho (\text{Suc} l) q) = \text{decide} (\rho (\text{Suc} k) p) \)
by (auto elim: laterProcessDecidesSameValue)

with k2 l2 show ?thesis by simp

next
  case False
  hence l ≤ k by simp
  then obtain m where m: k = l + m by (auto simp: le_iff_add)
  have decide (rho (Suc k) p) = decide (rho (Suc l) q)
    by (auto elim: laterProcessDecidesSameValue)
  with l2 k2 show ?thesis by simp
qed

7.9 Proof of Termination

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

theorem lv-termination:
  assumes run: CHORun LV-M rho HOs coords
  and commG: CHOcommGlobal LV-M HOs coords
  shows ∃ r. ∀ p. decide (rho r p) ≠ None
proof –

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

from commG obtain ph c
  where c: ∀ p. coords (4 * ph) p = c
  and maj0: card (HOS (4 * ph) c) > N div 2
  and maj2: card (HOS (4 * ph + 2) c) > N div 2
  and rcv1: ∀ p. c ∈ HOS (4 * ph + 1) p
  and rcv3: ∀ p. c ∈ HOS (4 * ph + 3) p
  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 run c have c: ∀ p. coordΦ (rho ?r p) = c
  by (auto simp add: phase-def coordinators)

with run have c1: ∀ p. coordΦ (rho ?r1 p) = c
  by (auto simp add: step-def mod-Suc notStep3EqualCoord)

with run have c2: ∀ p. coordΦ (rho ?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: commit (rho ?r1 c) (is ?P c (4∗ph))
proof (rule LV-Suc'[OF run, where P=?P], auto simp: step-def)
  assume next0 ?r c (rho ?r c) (HOrcvdMsgs LV-M ?r c (HOs ?r c) (rho ?r))
  (coords (Suc ?r) c) (rho (Suc ?r) c)
  with c' maj0 show commit (rho (Suc ?r) c)
  by (auto simp: step-def next0-def send0-def valStampsRcvd-def LV-CHOMachine-def HOrcvdMsgs-def LV-sendMsg-def)

qed

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

have 2: ∀ p. timestamp (rho ?r2 p) = Suc ph
proof
  fix p
  let ?msgs = HOrcvdMsgs LV-M ?r1 p (HOs ?r1 p) (rho ?r1)
  let ?crd = coordΦ (rho ?r1 p)
  from run 1 c1 rcv1
  have c: ?msgs ?crd ≠ None ∧ isVote (the (?msgs ?crd))
  show timestamp (rho ?r2 p) = Suc ph (is ?P p (Suc (4∗ph)))
  proof (rule LV-Suc'[OF run, where P=?P], auto simp: step-def mod-Suc)
    assume next1 ?r1 p (rho ?r1 p) ?msgs (coords (Suc ?r1) p) (rho ?r2 p)
    with cnd show ?thesis by (auto simp: next1-def phase-def)
  qed
  qed

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

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

qed

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

have 4: ∀ p. decide (rho ?r4 p) ≠ None
proof
fix p
let ?msgs = HOLMsgs LV-M ?r3 p (HO ?r3) (rho ?r3)
let ?crd = coord (rho ?r3)
from run 3 c3 rev3
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 (Suc p)))))
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 (λp. 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 \( \text{init: crun } 0 = \text{state (rho 0)} \)

from crun have CHORun LV-M crun HOs coords
  by (unfold CHORun-def SHORun-def)

from this commG have consensus \((x \circ \text{crun } 0)\) decide crun
  by (rule lv-consensus)

with init show consensus ?inits decide crun
  by (simp add: o-def)

qed

end

theory UteDefs

begin

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

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

We formalize in Isabelle the correctness proof of the algorithm that appears in [3], using the framework of theory \( \text{HOModel} \).

8.1 Model of the Algorithm

We begin by introducing an anonymous type of processes of finite cardinality that will instantiate the type variable \( '\text{proc} \) of the generic HO model.

typedecl Proc — the set of processes

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

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

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

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

abbreviation
  \( nSteps \equiv 2 \)

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

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

lemma phase-zero [simp]: phase 0 = 0

by (simp add: phase-def)
lemma step-zero [simp]: step 0 = 0
by (simp add: step-def)

lemma phase-step: (phase r * nSteps) + step r = r
by (auto simp add: phase-def step-def)

The following record models the local state of a process.

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

Possible messages sent during the execution of the algorithm.

datatype ′val msg =
  Val ′val | Vote ′val option

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

definition Ute-initState where
Ute-initState p st ≡ (vote st = None) ∧ (decide st = None)

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

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

begin

Simple consequences of the above parameter constraints.

lemma alpha-lt-N: α < N
using EltN majE by auto

lemma alpha-lt-T: α < T
using majT alpha-lt-N by auto

lemma alpha-lt-E: α < 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.
\text{definition} \quad \text{send}0 :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow \text{msg} \\
\text{where} \\
\text{send}0 \ r \ p \ q \ st \equiv \text{Val} (x \ st)

\text{definition} \quad \text{next}0 :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow (\text{Proc} \Rightarrow \text{msg} \Rightarrow \text{bool}) \\
\quad \Rightarrow \text{Pstate} \Rightarrow \text{bool} \\
\text{where} \\
\text{next}0 \ r \ p \ st \ msgs \ st' \equiv \\
(\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Val} v)\} > T \land st' = st (\| \text{vote} := \text{Some} v \|) \\
\lor \neg (\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Val} v)\} > T) \land st' = st (\| \text{vote} := \text{None} \|)

In step 1, each process sends its current \text{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.

\text{definition} \quad \text{send}1 :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow \text{msg} \\
\text{where} \\
\text{send}1 \ r \ p \ q \ st \equiv \text{Vote} (\text{vote} \ st)

\text{definition} \quad \text{next}1 :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow (\text{Proc} \Rightarrow \text{msg} \Rightarrow \text{bool}) \\
\quad \Rightarrow \text{Pstate} \Rightarrow \text{bool} \\
\text{where} \\
\text{next}1 \ r \ p \ st \msgs \ st' \equiv \\
(\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Vote} (\text{Some} v))\} > \alpha \land x \ st' = v \\
\lor \neg (\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Vote} (\text{Some} v))\} > \alpha) \\
\land x \ st' = \text{undefined} ) \\
\land (\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Vote} (\text{Some} v))\} > E \land \text{decide} \ st' = \text{Some} v \\
\lor \neg (\exists v. \text{card} \{q. \text{msgs} q = \text{Some} (\text{Vote} (\text{Some} v))\} > E) \\
\land \text{decide} \ st' = \text{decide} \ st \\
\land \text{vote} \ st' = \text{None})

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

\text{definition} \quad \text{Ute-sendMsg} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow \text{msg} \\
\text{where} \\
\text{Ute-sendMsg} (r::\text{nat}) \equiv \text{if} \ r = 0 \ \text{then} \ \text{send}0 \ r \ \text{else} \ \text{send}1 \ r

\text{definition} \quad \text{Ute-nextState} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Pstate} \Rightarrow (\text{Proc} \Rightarrow \text{msg} \Rightarrow \text{bool}) \\
\quad \Rightarrow \text{Pstate} \Rightarrow \text{bool} \\
\text{where} \\
\text{Ute-nextState} \ r \equiv \text{if} \ r = 0 \ \text{then} \ \text{next}0 \ r \ \text{else} \ \text{next}1 \ 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

$Ute-commPerRd$ $HOrs$ $SHOrs$ $\equiv$

\[
\forall p. \text{card } (HOrs p - SHOrs p) \leq \alpha \\
\land \text{card } (SHOrs p \cap HOrs p) > N + 2*\alpha - E - 1 \\
\land \text{card } (SHOrs p \cap HOrs p) > T
\]

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

- all $HO$ and $SHO$ sets of all processes are equal in the second step of phase $\Phi$, i.e. all processes receive messages from the same set of processes, and none of these messages is corrupted,

- every process receives more than $T$ correct messages in the first step of phase $\Phi + 1$, and

- every process receives more than $E$ correct messages in the second step of phase $\Phi + 1$.

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

**definition** $Ute-commGlobal$ where

$Ute-commGlobal$ $HOs$ $SHOs$ $\equiv$

\[
\exists \Phi. (\text{let } r = \text{Suc } (nSteps*\Phi) \\
in (\exists \pi. \forall p. \pi = HOs r p \land \pi = SHOs r p) \\
\land (\forall p. \text{card } (SHOs (\text{Suc r}) p \cap HOs (\text{Suc r}) p) > T) \\
\land (\forall p. \text{card } (SHOs (\text{Suc (Suc r)}) p \cap HOs (\text{Suc (Suc r)}) p) > E))
\]

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 msgs st'. Ute-nextState r p st msgs st'),
SHOcommPerRd = Ute-commPerRd,
SHOcommGlobal = Ute-commGlobal
\)

abbreviation
Ute-M ≡ (Ute-SHOMachine::(Proc, 'val pstate, 'val msg) SHOMachine)
end — locale ute-parameters
end

theory UteProof
imports UteDefs ../Majorities ../Reduction
begin

context ute-parameters
begin

8.4 Preliminary Lemmas

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

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

Processes can make a new decision only at second round of each phase.

lemma decide-step:
  assumes run: SHORun Ute-M rho HOs SHOs
  and d1: decide (rho r p) ≠ Some v
  and d2: decide (rho (Suc r) p) = Some v
  shows step r ≠ 0
proof
  assume sr: step r = 0
  from run obtain µ where Ute-nextState r p (rho r p) µ (rho (Suc r) p)
    unfolding Ute-SHOMachine-def nextState-def SHORun-eq SHOnextConfig-eq
    by force
  with sr have next0 r p (rho r p) µ (rho (Suc r) p)
    unfolding Ute-nextState-def by auto
  hence decide (rho r p) = decide (rho (Suc r) p)
    by (auto simp: next0-def)
  ...
with \( d1 \) \( d2 \) show False by auto

qed

lemma unique-majority-E:
assumes majv: \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m\} > E \)
and majw: \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m'\} > E \)
shows \( m = m' \)

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

qed

lemma unique-majority-E-\( \alpha \):
assumes majv: \( \text{card} \{qq::\text{Proc. } F \ qq = m\} > E - \alpha \)
and majw: \( \text{card} \{qq::\text{Proc. } F \ qq = m'\} > E - \alpha \)
shows \( m = m' \)

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

qed

lemma unique-majority-T:
assumes majv: \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m\} > T \)
and majw: \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m'\} > T \)
shows \( m = m' \)

proof –
  from majT majv majw
  have \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m\} > N \text{ div } 2 \)
  and \( \text{card} \{qq::\text{Proc. } F \ qq = \text{Some } m'\} > N \text{ div } 2 \)
  by auto
  then obtain qq
    where \( qq \in \{qq::\text{Proc. } F \ qq = \text{Some } m\} \)
    and \( qq \in \{qq::\text{Proc. } F \ qq = \text{Some } m'\} \)
    by (rule majoritiesE)
  thus \( ?\text{thesis} \) by auto

102
No two processes may vote for different values in the same round.

**lemma** common-vote:

**assumes** unsafe: SHOcommPerRd Ute-M HO SHO
and nxtp: nextState Ute-M r p (ρ p) μ p (ρ (Suc r) p)
and mup: μ p ∈ SHOmsgVectors Ute-M r p (ρ r) (HO p) (SHO p)
and nxtq: nextState Ute-M r q (ρ r q) μ q (ρ (Suc r) q)
and muq: μ q ∈ SHOmsgVectors Ute-M r q (ρ r) (HO q) (SHO q)
and vp: vote (ρ (Suc r) p) = Some vp
and vq: vote (ρ (Suc r) q) = Some vq

**shows** vp = vq using assms proof

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

**proof**

- **have** card {qq. sendMsg Ute-M r qq p (ρ r qq) = Val vp} > T − α
  ∧ card {qq. sendMsg Ute-M r qq q (ρ r qq) = Val vq} > T − α
  (is card ?vrcvdp − ?ahop ⊆ ?vrsentp)

**proof**

- **from** nxtp vp **have** stp:step r = 0 by (auto simp: vote-step)
  **from** mup
  **have** {qq. μ p qq = Some (Val vp)} − (HO p − SHO p)
  ⊆ {qq. sendMsg Ute-M r qq p (ρ r qq) = Val vp}
  (is ?vrcvdp − ?ahop ⊆ ?vrsentp)

**proof**

- **hence** card (?vrcvdp − ?ahop) ≤ card ?vrcvdp − card ?ahop
  by (auto simp: SHOmsgVectors-def)
  **hence** card ?vrsentp ≥ card ?vrcvdp − card ?ahop by auto

**moreover**

- **from** nxtp stp **have** next0 r p (ρ r p) μ p (ρ (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** muq
  **have** {qq. μ q qq = Some (Val vq)} − (HO q − SHO q)
  ⊆ {qq. sendMsg Ute-M r qq q (ρ r qq) = Val vq}
  (is ?vrcvdq − ?ahoq ⊆ ?vrsentq)

**proof**

- **hence** card (?vrcvdq − ?ahoq) ≤ card ?vrsentq
  and card (?vrcvdq − ?ahoq) ≥ card ?vrcvdq − card ?ahoq
  by (auto simp: card-mono diff-card-le-card-Diff)
  **hence** card ?vrsentq ≥ card ?vrcvdq − card ?ahoq by auto

**moreover**

- **from** nxtq stp **have** next0 r q (ρ r q) μ q (ρ (Suc r) q)
  by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)

**with** vq **have** card {qq. μ 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
qed
thus ?thesis using majT by auto
qed

show ?thesis
proof (rule ccontr)
assume vpq:vq ≠ 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
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))
unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq SHOmsgVectors-def by blast

hence {qq. µ p qq = Some (Vote (Some v))} − (HOs r p − SHOs r p)
  ⊆ {qq. sendMsg Ute-M r qq p (rho r qq) = Vote (Some v)}
  (is ?vrcvdp − ?ahop ⊆ ?vSENTp) by auto

hence card (?vrcvdp − ?ahop) ≤ card ?vSENTp
  and card (?vrcvdp − ?ahop) ≥ card ?vrcvdp − card ?ahop
  by (auto simp: card-mono diff-card-le-card-Diff)

hence card ?vSENTp ≥ card ?vrcvdp − card ?ahop by auto
moreover
from usafe have card (HOs r p − SHOs r p) ≤ α
  by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
moreover
from run d1 d2 have step r ≠ 0 by (rule decide-step)
with next1 have next1 r p (rho r p) µ p (rho (Suc r) p)
  by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
with run d1 d2 have card {qq. µ p qq = Some (Vote (Some v))} > E
  unfolding next1-def by auto
ultimately
show ?thesis using alpha-lt-E by auto
qed

8.5 Proof of Agreement and Validity

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

lemma common-x-argument-1:
  assumes usafe:SHOcommPerRd Ute-M (HOs (Suc r)) (SHOs (Suc r))
  and threshold: card {q. sendMsg Ute-M (Suc r) q p (rho (Suc r) q)}
  = Vote (Some v)} > E - α
  (is card (?msgs p v) > -)
  shows card (?msgs pp v ∩ (SHOs (Suc r) pp ∩ HOs (Suc r) pp)) > α
proof –
  have card (?msgs pp v) + card (SHOs (Suc r) pp ∩ HOs (Suc r) pp) > N + α
  proof –
    have ∀ q. sendMsg Ute-M (Suc r) q p (rho (Suc r) q) = sendMsg Ute-M (Suc r) q pp (rho (Suc r) q)
      by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def step-def send0-def send1-def)
  moreover
  from usafe
  have card (SHOs (Suc r) pp ∩ HOs (Suc r) pp) > N + 2*α - E - 1
    by (auto simp: Ute-SHOMachine-def step-def Ute-commPerRd-def)
  ultimately
  show ?thesis using threshold by auto
qed

moreover
have card (?msgs pp v) + card (SHOs (Suc r) pp ∩ HOs (Suc r) pp)
  = card (?msgs pp v ∪ (SHOs (Suc r) pp ∩ HOs (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 common-x-argument-2:**

**Assumes** run: SHORun Ute-M rho HOs SHOs

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

**And** nxtpp: nextState Ute-M (Suc r) pp (rho (Suc r) pp)

$$\mu_{pp} (\rho (\text{Suc (Suc r)}) pp)$$

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

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

**Shows** $x (\rho (\text{Suc (Suc r)}) pp) = v$

**Proof**

- **Have** stp: step (Suc r) $\neq 0$

**Proof**

- Assume $sr$: step (Suc r) = 0

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

**By** (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send0-def)

**Moreover**

**From** threshold obtain $qq$ where

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

**By** force

- **Ultimately**

- **Show** False by simp

**Qed**

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

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

**Proof**

- **From** mupp

- **Have** SHOs (Suc r) pp $\cap$ HOs (Suc r) pp

$$\subseteq \{qq. \mu_{pp} qq = \text{Some (sendMsg Ute-M (Suc r) qq pp (rho (Suc r) qq))}\}$$

**Unfolding** SHOmsgVectors-def **By** auto

**Moreover**

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

**By** auto

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

$$\geq \text{card ((?sent pp v) \cap (SHOs (Suc r) pp \cap HOs (Suc r) pp))}$$
by (auto intro: card-mono)
moreover
from usafe threshold
have alph: \( \text{card} (\text{?sent } \text{pp } v) \cap (\text{SHOs (Suc } r) \text{ pp } \cap \text{HOs (Suc } r) \text{ pp}) > \alpha \)
  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: \( \text{card } \text{(SHOs } \text{Suc } r \text{ pp}) \cap \text{HOs (Suc } r) \text{ pp}) > \alpha \)
  by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
have v = w
proof
  note usafe
moreover
obtain qv where qv \( \in \text{SHOs (Suc } r) \text{ pp and } \mu pp \) qv = Some \( \text{(Vote (Some } v) \) )
proof
  have \( \neg (\text{?msgs } v \subseteq \text{HOs (Suc } r) \text{ pp } \setminus \text{SHOs (Suc } r) \text{ pp}) \)
    proof
      assume \( \text{?msgs } v \subseteq \text{HOs (Suc } r) \text{ pp } \setminus \text{SHOs (Suc } r) \text{ pp}) \)
      hence \( \text{card } (\text{?msgs } v) \leq \text{card } ((\text{HOs (Suc } r) \text{ pp}) \setminus (\text{SHOs (Suc } r) \text{ pp}) \) )
      by (auto simp: card-mono)
    moreover
    from usafe
    have \( \text{card } (\text{HOs (Suc } r) \text{ pp } \setminus \text{SHOs (Suc } r) \text{ pp}) \leq \alpha \)
      by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)
    moreover
    note va
    ultimately
    show False by auto
qed
then obtain qv
  where qv \( \notin \text{HOs (Suc } r) \text{ pp } \setminus \text{SHOs (Suc } r) \text{ pp}) \)
  and qv: \( \mu pp \) qv = Some \( \text{(Vote (Some } v) \) )
  by auto
with mupp have qv \( \in \text{SHOs (Suc } r) \text{ pp}) \)
  unfolding SHOmsgVectors-def
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
\( qw \in SHOs \ (Suc \ r) \ pp \ \textbf{and} \ \mu pp \ qw = \text{Some} \ (Vote \ (Some \ w)) \)

**proof**

\( \neg (msgs \ w \subseteq HOs \ (Suc \ r) \ pp - SHOs \ (Suc \ r) \ pp) \)

**proof**

\( \text{assume} \ ?msgs \ w \subseteq HOs \ (Suc \ r) \ pp - SHOs \ (Suc \ r) \ pp \)

\( \text{hence} \ \text{card} \ ( ?msgs \ w ) \leq \text{card} \ ( (HOs \ (Suc \ r) \ pp) - (SHOs \ (Suc \ r) \ pp)) \)

\( \text{by} \ (\text{auto simp: card-mono}) \)

**moreover**

\( \text{have card (HOs (Suc r) pp - SHOs (Suc r) pp) \leq \alpha} \)

\( \text{by} \ (\text{auto simp: Ute-SHOMachine-def Ute-commPerRd-def}) \)

**moreover**

\( \text{note wa} \)

**ultimately**

\( \text{show False by auto} \)

**qed**

**then obtain** \( qw \)

\( \text{where} \ qw \notin HOs \ (Suc \ r) \ pp - SHOs \ (Suc \ r) \ pp \)

\( \text{and} \ qw: \ \mu pp \ qw = \text{Some} \ (Vote \ (Some \ w)) \)

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

**with** \( \text{mupp} \) **have** \( qw \in SHOs \ (Suc \ r) \ pp \)

**unfolding** \( SHOmsgVectors-def \) **by** \( \text{auto} \)

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

**qed**

**with** \( \text{stp mupp} \) **have** \( \text{vote (rho (Suc r) qw)} = \text{Some w} \)

**by** \( (\text{auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send1-def}) \)

**moreover**

**from** \( \text{run obtain} \ \mu qv \ \mu qw \)

\( \text{where} \ nextState \ Ute-M \ r \ qv \ ((\rho r) \ qv) \ \mu qv \ (\rho (Suc \ r) \ qv) \)

\( \text{and} \ \mu qv \in SHOmsgVectors \ Ute-M \ r \ qv \ ((\rho r) \ (HOs \ r \ qv)) \ (SHOs \ r \ qv) \)

\( \text{and} \ nextState \ Ute-M \ r \ qw \ ((\rho r) \ qw) \ \mu qw \ (\rho (Suc \ r) \ qw) \)

\( \text{and} \ \mu qw \in SHOmsgVectors \ Ute-M \ r \ qw \ ((\rho r) \ (HOs \ r \ qw)) \ (SHOs \ r \ qw) \)

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

**ultimately**

**show** \( \text{?thesis using unsafe (auto dest: common-vote)} \)

**qed**

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

**qed**

Inductive argument for the agreement and validity theorems.

**lemma** safety-inductive-argument:

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

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

\( \text{and} \ \text{sh: } E = \alpha < \text{card} \ \{ q. \ sendMsg \ Ute-M \ r\' q p \ (\rho (Suc r') q) = \text{Vote} \ (Some \ v) \}\)

\( \text{and} \ \text{stp1: } \text{step } r' = \text{Suc } 0 \)

**shows** \( E = \alpha < \text{card} \ \{ q. \ sendMsg \ Ute-M \ (Suc \ (Suc r')) q p \ (\rho (Suc (Suc r')) q) \)

\( = \text{Vote} \ (Some \ v) \}\)

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

have \( \forall pp. \; x \; (\rho \; \text{(Suc } \; \text{Suc } \; r)) \; pp = v \)
proof
  fix pp
from run obtain \( \mu pp \)
  where \( \mu pp \in \text{SHOmsgVectors } \text{Ute-M } r' \; pp \; (\rho \; r' ) \; (\text{SHOs } r' \; pp) \)
    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 \( rr' \) show \( x \; (\rho \; \text{(Suc } \; r)) \; pp = v \)
  by (auto dest: common-x-argument-2)
qed
with run stpr
have \( \forall pp \; p. \; \text{sendMsg } \text{Ute-M } (\text{Suc } \; \text{Suc } \; r) \; pp \; p \; (\rho \; \text{(Suc } \; \text{Suc } \; r)) \; pp = \text{Val } v \)
  by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
    Ute-sendMsg-def send0-def mod-Suc step-def)
with \( rr' \)
have \( \lambda p \; \mu pp. \; \mu pp' \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \; p \; (\rho \; (\text{Suc } r')) \)
  (\text{HOs } (\text{Suc } r') \; p \; (\text{SHOs } (\text{Suc } r') \; p)

\[ \subseteq \{ q. \; \mu pp' \; q = \text{Some } (\text{Val } v) \} \]
  by (auto simp: SHOmsgVectors-def)
hence \( \lambda p \; \mu pp. \; \mu pp' \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \; p \; (\rho \; (\text{Suc } r')) \)
  (\text{HOs } (\text{Suc } r') \; p \; (\text{SHOs } (\text{Suc } r') \; p)

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

show \?thesis
proof
have \( \forall pp. \; \text{vote } ((\rho \; (\text{Suc } \; \text{Suc } \; r')) \; pp) = \text{Some } v \)
proof
  fix pp
from run obtain \( \mu pp \)
  where \( \text{nxtpp: } \text{nextState } \text{Ute-M } (\text{Suc } r') \; pp \; (\rho \; (\text{Suc } \; r')) \; pp \; \mu pp \)
    (\rho \; (\text{Suc } \; \text{Suc } \; r')) \; pp \)
    and \( \text{mpp: } \mu pp \in \text{SHOmsgVectors } \text{Ute-M } (\text{Suc } r') \; pp \; (\rho \; (\text{Suc } r')) \)

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** \( \exists m < n. \, \text{decide} (\rho (Suc m) p) \neq \text{decide} (\rho m p) \)
\& \text{decide} (\rho (Suc m) p) = Some v

**Proof** –

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

**Proof** (induct n)

- **From run show ?P 0**
  - **By** (auto simp: Ute-SHOMachine-def SHORun-eq HOMinitConfig-eq
    initState-def Ute-initState-def)

**next**

- **Fix** n
- **Assume** ih: ?P n thus ?P (Suc n) by force

**qed**

- **With dec show ?thesis by auto**

**qed**
If process $p_1$ has decided value $v_1$ and process $p_2$ later decides, then $p_2$
must decide $v_1$.

**lemma laterProcessDecidesSameValue:**

assumes run: $	ext{SHORun Ute-M rho HOs SHOs}$
and comm: $\forall r. \text{SHOcommPerRd Ute-M (HOs r) (SHOs r)}$
and $\text{dv1:decide} (\text{rho} (\text{Suc} r) p_1) = \text{Some } v_1$
and $\text{dn2:decide} (\text{rho} (r + k) p_2) \neq \text{Some } v_2$
and $\text{dv2:decide} (\text{rho} (\text{Suc} (r + k)) p_2) = \text{Some } v_2$

shows $v_2 = v_1$

**proof**

from run $\text{dv1}$ obtain $r_1$

where $r_1 r < \text{Suc } r$

and $\text{dn1:decide} (\text{rho} r_1 p_1) \neq \text{Some } v_1$

and $\text{dv1':decide} (\text{rho} (\text{Suc} r_1) p_1) = \text{Some } v_1$

by (auto dest: decisionNonNullThenDecided)

from $r_1 r$ obtain $s$ where $r_1 r = \text{Suc } (r_1 + s)$

by (auto dest: less-imp-Suc-add)

then obtain $k'$ where $k': r + k = r_1 + k'$

by auto

with $\text{dn2 dv2}$

have $\text{dn2':decide} (\text{rho} (r_1 + k') p_2) \neq \text{Some } v_2$

and $\text{dv2':decide} (\text{rho} (\text{Suc} (r_1 + k')) p_2) = \text{Some } v_2$

by auto

from run $\text{run} \text{dv1} \text{dn2'} \text{dv2'}$

have $\text{rs0:step} r_1 = \text{Suc } 0$ and $\text{rks0:step} (r_1 + k') = \text{Suc } 0$

by (auto simp: mod-Suc step-def dest: decide-step)

have $\text{step} (r_1 + k') = \text{step} (\text{step} r_1 + k')$

unfolding step-def by (rule mod-add-left-eq)

with $\text{rs0 rks0}$ have $\text{step} k' = 0$ by (auto simp: step-def mod-Suc)

then obtain $k''$ where $k'' \ast nSteps$ by (auto simp: step-def)

with $\text{dn2'} \text{dv2'}$

have $\text{dn2'':decide} (\text{rho} (r_1 + k'' \ast nSteps) p_2) \neq \text{Some } v_2$

and $\text{dv2'':decide} (\text{rho} (\text{Suc} (r_1 + k'' \ast nSteps)) p_2) = \text{Some } v_2$

by auto

from $\text{rs0}$ have $\text{stp:step} (r_1 + k'' \ast nSteps) = \text{Suc } 0$

unfolding step-def by auto

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

**proof** (induct $k''$)

from $\text{stp}$ have $\text{step} (r_1 + 0 \ast nSteps) = \text{Suc } 0$

by (auto simp: step-def)

from run comm $\text{dn1 dv1'}$ show $\text{card} \{ q. \text{sendMsg Ute-M} (r_1 + 0 \ast nSteps) q p_1 \} (\text{rho} (r_1 + 0 \ast nSteps) q)$
\[ = \text{Vote (Some } v1) \} > E - \alpha \]

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

next

fix \( k'' \)

assume \( \text{ih}: E - \alpha < \)

\[ \text{card } \{ q. \text{sendMsg } \text{Ute-M} (r1 + k'' \ast nSteps) q p1 (\rho (r1 + k'' \ast nSteps)) q \} \]

\[ = \text{Vote (Some } v1) \} \]

from \( \text{rs0} \) have \( \text{stps: step } (r1 + k'' \ast nSteps) = \text{Suc 0} \)

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

with \( \text{run \ comm \ ih} \)

have \( E - \alpha < \)

\[ \text{card } \{ q. \text{sendMsg } \text{Ute-M} (\text{Suc (Suc (r1 + k'' \ast nSteps))) q p1 (\rho (\text{Suc (Suc (r1 + k'' \ast nSteps))) q) \}} \]

\[ = \text{Vote (Some } v1) \} \]

by (rule safety-inductive-argument)

thus \( E - \alpha < \)

\[ \text{card } \{ q. \text{sendMsg } \text{Ute-M} (r1 + \text{Suc } k'' \ast nSteps) q p1 (\rho (r1 + \text{Suc } k'' \ast nSteps) q) \}

\[ = \text{Vote (Some } v1) \} \]

by auto

qed

moreover

from \( \text{run} \)

have \( \forall q. \text{sendMsg } \text{Ute-M} (r1 + k'' \ast nSteps) q p1 (\rho (r1 + k'' \ast nSteps) q) \)

\[ = \text{sendMsg } \text{Ute-M} (r1 + k'' \ast nSteps) q p2 (\rho (r1 + k'' \ast nSteps) q) \]

by (auto simp: \text{Ute-SHOMachine-def} \text{Ute-sendMsg-def} \text{step-def} \text{send0-def} \text{send1-def})

moreover

from \( \text{run \ comm \ dn2'' \ dv2''} \)

have \( E - \alpha < \)

\[ \text{card } \{ q. \text{sendMsg } \text{Ute-M} (r1 + k'' \ast nSteps) q p2 (\rho (r1 + k'' \ast nSteps) q) \}

\[ = \text{Vote (Some } v2) \} \]

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

ultimately

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

qed

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

\text{theorem ute-agreement:}

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

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

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

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

\text{shows v = w \)

\text{proof –}

from \( \text{run \ p \ obtain \ k} \)

\text{where k1: decide (rho (Suc k) p) \( \neq \) decide (rho k p)}\]
and \( k_2 \): decide (\( \rho (\text{Suc } k) \) \( p \)) = Some \( v \) by (auto dest: decisionNonNullThenDecided)
from run \( q \) obtain \( l \)
  where \( l_1 \): decide (\( \rho (\text{Suc } l) \) \( q \)) \( \neq \) decide (\( \rho l q \))
  and \( l_2 \): decide (\( \rho (\text{Suc } l) \) \( q \)) = Some \( w \) by (auto dest: decisionNonNullThenDecided)
show \(?\)thesis
proof (cases \( k \leq l \))
  case True
  then obtain \( m \) where \( m \): \( l = k + m \) by (auto simp add: le_iff_add)
  from run \( \text{comm } k_2 l_1 l_2 m \) have \( w = v \)
  thus \(?\)thesis by simp
next
  case False
  hence \( l \leq k \) by simp
  then obtain \( m \) where \( m \): \( k = l + m \) by (auto simp add: le_iff_add)
  from run \( \text{comm } l_2 k_1 k_2 m \) show \(?\)thesis
    by (auto elim!: laterProcessDecidesSameValue)
qed

Main lemma for the proof of the Validity property.

lemma validity-argument:
  assumes run: \( \text{SHORun } Ute-M \rho HOs SHOs \)
  and comm: \( \forall r. \text{SHOcommPerRd } Ute-M (HOs r) (SHOs r) \)
  and init: \( \forall x. \text{((\( \rho 0 \) \( p \)) = v) \) \)
  and dw: decide (\( \rho r p \)) = Some \( w \)
  and \( \text{stp}: \text{step } r' = \text{Suc } 0 \)
  shows \( \text{card } \{ q. \text{sendMsg } Ute-M r' q p (\( \rho r' q \)) = \text{Vote } (\text{Some } v) \} > E - \alpha \)
proof —
  from \( \text{stp obtain } k \) where \( \text{stp:}\) = \( \text{Suc } 0 + k \ast n\text{Steps} \)
    unfolding step-def using mod-Suc mod-eqD by blast
  moreover
  have \( E - \alpha < \)
    \( \text{card } \{ q. \text{sendMsg } Ute-M (\text{Suc } 0 + k \ast n\text{Steps}) q p ((\( \rho (\text{Suc } 0 + k \ast n\text{Steps})) q) = \text{Vote } (\text{Some } v) \} \)
proof (induct \( k \))
  have \( \forall p p. \text{vote } ((\( \rho (\text{Suc } 0)) pp) = \text{Some } v \)
  proof
    fix \( p p \)
    from run obtain \( ppp \)
      where \( \text{ntpp:nextState } Ute-M 0 pp (\( \rho 0 pp \)) ppp (\( \rho (\text{Suc } 0) pp \)) \)
        and \( \text{mpp:}\)pppp \( \in \text{SHOmsgVectors } Ute-M 0 pp (\( \rho 0 ) (HOs 0 pp) (SHOs 0 pp) \)
      by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
    have \( \text{majv:}\)card \( \{ q. ppp q = \text{Some } (Val v) \} > T \)
    proof —
from run init have \( \forall q. \text{sendMsg Ute-M} 0 q \text{ pp (rho 0 q)} = \text{Val v} \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq
Ute-sendMsg-def send0-def step-def)

moreover
from comm have shoT:card (SHOs 0 pp \( \cap \) HOs 0 pp) > T
by (auto simp: Ute-SHOMachine-def Ute-commPerRd-def)

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

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

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

qed

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

thus \( E - \alpha < \)
card \( \{ q. \text{sendMsg Ute-M} (\text{Suc} 0 + 0 \ast nSteps) q \text{ p (rho (Suc} 0 + 0 \ast nSteps)} q\) = \text{Vote (Some v)}\}
using majE EltN by auto

next
fix \( k \)
assume ih:E - \( \alpha < \)
card \( \{ q. \text{sendMsg Ute-M} (\text{Suc} 0 + k \ast nSteps) q \text{ p (rho (Suc} 0 + k \ast nSteps)} q\) = \text{Vote (Some v)}\}

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

from run comm ih this
have E - \( \alpha < \)
card \( \{ q. \text{sendMsg Ute-M} (\text{Suc (Suc} 0 + k \ast nSteps)) q \text{ p} \)
\[(\rho \ (\text{Suc} \ (\text{Suc} \ 0 + k \ast \text{nSteps}))) \ q)\]  
\[= \text{Vote} \ (\text{Some} \ v)\]  
by (rule safety-inductive-argument)

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

qed

ultimately

show \(\exists v : \text{thesis by simp}\)

qed

The following theorem shows the Validity property of algorithm \(\mathcal{U}_{T,E,\alpha}\).

**Theorem ute-validity:**

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

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

and init: \(\forall p, x \ (\rho \ 0 \ p) = v\)

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

shows \(v = w\)

**Proof**

- from run dw obtain \(r1\)
  - where \(\text{dwr1: decide} \ ((\rho \ r1) \ p) \neq \text{Some} \ w\)
  - and \(\text{dwr1: decide} \ ((\rho \ (\text{Suc} \ r1)) \ p) = \text{Some} \ w\)
  - by (force dest: decisionNonNullThenDecided)

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

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

with assms

have \(E - \alpha < \)  
\[\text{card} \{q. \ \text{sendMsg} \ Ute-M \ r1 \ q \ p \ (\rho \ r1 \ q) = \text{Vote} \ (\text{Some} \ v)\}\]  
by (rule validity-argument)

moreover

from run comm dwr1 dwr1

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

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

ultimately

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

qed

### 8.6 Proof of Termination

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

**Lemma set-x-from-vote:**

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

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

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

and \(\pi: \forall p. \ \text{HOs} \ (\text{Suc} \ r) \ p = \text{SHOs} \ (\text{Suc} \ r) \ p\)
and \texttt{nxt: nextState Ute-M (Suc r) p (rho (Suc r) p) mu (rho (Suc (Suc r)) p)}

and \texttt{mu: \mu \in SHOmsgVectors Ute-M (Suc r) p (rho (Suc r)) (HOs (Suc r) p) (SHOs (Suc r) p)}

and \texttt{wp: \alpha < card \{qq. \mu qq = Some (Vote (Some vv))\}}

shows \texttt{x ((rho (Suc (Suc r))) p) = v}

\begin{tabbing}
proof \- \end{tabbing}

\begin{tabbing}
\hspace{1cm}from \texttt{nxt stp vp obtain wp} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}where \texttt{wp: \alpha < card \{qq. \mu qq = Some (Vote (Some wp))\}} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}and \texttt{wp: x (rho (Suc (Suc r)) p) = wp} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def) \hspace{1cm} \end{tabbing}

\begin{tabbing}
have \texttt{wp = v} \hspace{1cm} \end{tabbing}

\begin{tabbing}
proof \- \end{tabbing}

\begin{tabbing}
\hspace{1cm}from \texttt{xwp obtain pp where smw: \mu pp = Some (Vote (Some wp))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}by force \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}have \texttt{vote (rho (Suc r) pp) = Some wp} \hspace{1cm} \end{tabbing}

\begin{tabbing}
proof \- \end{tabbing}

\begin{tabbing}
\hspace{1cm}from \texttt{smw mu \pi} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}have \texttt{mu pp = Some (sendMsg Ute-M (Suc r) pp pp (rho (Suc r) pp))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}unfolding \texttt{SHOmsgVectors-def} by force \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}with \texttt{stp have \mu pp = Some (Vote (vote (rho (Suc r) pp))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def) \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}with \texttt{smw show \texttt{?thesis}} by auto \hspace{1cm} \end{tabbing}

qed

moreover

\begin{tabbing}
\hspace{1cm}from \texttt{wp obtain qq where smw: \mu qq = Some (Vote (Some v))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}by force \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}have \texttt{vote (rho (Suc r) qq) = Some v} \hspace{1cm} \end{tabbing}

\begin{tabbing}
proof \- \end{tabbing}

\begin{tabbing}
\hspace{1cm}from \texttt{smw mu \pi} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}have \texttt{mu qq = Some (sendMsg Ute-M (Suc r) qq pp (rho (Suc r) qq))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}unfolding \texttt{SHOmsgVectors-def} by force \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}with \texttt{stp have \mu qq = Some (Vote (vote (rho (Suc r) qq))} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def send1-def) \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}with \texttt{smw show \texttt{?thesis}} by auto \hspace{1cm} \end{tabbing}

qed

moreover

\begin{tabbing}
\hspace{1cm}from \texttt{run obtain \mu pp \mu qq} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}where \texttt{nextState Ute-M r pp (rho r pp) \mu pp (rho (Suc r) pp)} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}and \texttt{\mu pp \in SHOmsgVectors Ute-M r pp (rho r) (HOs r pp) (SHOs r pp)} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}and \texttt{nextState Ute-M r qq ((rho r) qq) \mu qq (rho (Suc r) qq)} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}and \texttt{\mu qq \in SHOmsgVectors Ute-M r qq (rho r) (HOs r qq) (SHOs r qq)} \hspace{1cm} \end{tabbing}

\begin{tabbing}
\hspace{1cm}unfolding \texttt{Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq} by blast \hspace{1cm} \end{tabbing}

ultimately

\begin{tabbing}
\hspace{1cm}show \texttt{?thesis using comm by (auto dest: common-vote)} \hspace{1cm} \end{tabbing}

qed

with \texttt{xp show \texttt{?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: \( \text{SHORun} \ Ute-M \ \rho \ \text{HOs} \ \text{SHOs} \)

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

**And** \( \pi : \forall p. \ \pi 0 = \text{HOs} \ (\text{Suc} \ r) \ p \land \pi 0 = \text{SHOs} \ (\text{Suc} \ r) \ p \)

**Obtains** \( v \) where

\[
\forall p. \ \mu p q.
\]

\[
[ q \in \text{SHOs} \ (\text{Suc} \ r) \ p \cap \text{HOs} \ (\text{Suc} \ r) \ p; \]

\[
\mu p' \in \text{SHOmsgVectors} \ Ute-M \ (\text{Suc} \ (\text{Suc} \ r)) \ p \ (\rho h (\text{Suc} \ (\text{Suc} \ r)))
\]

\[
(\text{HOs} \ (\text{Suc} \ (\text{Suc} \ r)) \ p) \ (\text{SHOs} \ (\text{Suc} \ (\text{Suc} \ r)) \ p)
\]

\[
] \Rightarrow \mu p' q = (\text{Some} \ (\text{Val} \ v))
\]

**Proof**

- from \( \pi \) have hosho: \( \forall p. \ \text{SHOs} \ (\text{Suc} \ r) \ p = \text{SHOs} \ (\text{Suc} \ r) \ p \cap \text{HOs} \ (\text{Suc} \ r) \ p \)
  - by simp

  have \( \forall p q. \ x (\rho h (\text{Suc} \ (\text{Suc} \ r)) \ p) = x (\rho h (\text{Suc} \ (\text{Suc} \ r)) \ q) \)
  - proof
    - fix \( p q \)
    - from run obtain \( \mu p \)

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

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

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

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

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

  from run obtain \( \mu q \)

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

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

  and \( \mu: \mu q \in \text{SHOmsgVectors} \ Ute-M \ (\text{Suc} \ r) \ q \ (\rho h (\text{Suc} \ (\text{Suc} \ r))) \)

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

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

  have \( \forall qq. \ \mu p qq = \mu q qq \)

  proof
    - fix \( qq \)
    - show \( \mu p qq = \mu q qq \)

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

    case False

    with \( \mu: \mu \pi \)

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

    unfolding \( \text{SHOmsgVectors-def} \) by auto

    from \( \mu: \mu \pi \)

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

    unfolding \( \text{SHOmsgVectors-def} \) by auto

    moreover

    from \( \mu: \mu \pi \)

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

    unfolding \( \text{SHOmsgVectors-def} \) by auto

117
ultimately

\[ \text{show } \text{thesis} \]
\[ \quad \text{by (auto simp: Ute-SHOMachine-def Ute-sendMsg-def step-def send0-def send1-def)} \]

next

\[ \text{case True} \]
\[ \quad \text{with } \mu \text{ have } qq \notin \text{HOs (Suc } r) \text{ } p \text{ unfolding SHOmsgVectors-def by auto} \]
\[ \quad \text{with } \pi \text{ } \mu q \text{ have } \mu q = \text{None unfolding SHOmsgVectors-def by auto} \]
\[ \quad \text{with } \text{True show } \text{thesis by simp} \]
\[ \text{qed} \]
\[ \text{qed} \]

hence \[ \text{vsets;} \forall v. \{ qq, \mu p qq = \text{Some (Vote (Some } v)) \} \]
\[ = \{ qq, \mu q qq = \text{Some (Vote (Some } v)) \} \]
\[ \text{by auto} \]

show \[ x (\rho (\text{Suc } (\text{Suc } r)) p) = x (\rho (\text{Suc } (\text{Suc } r)) q) \]
proof (cases \( \exists v. \alpha < \text{card}\{ qq, \mu p qq = \text{Some (Vote (Some } v))\}\), clarify)

fix \( v \)

assume \( \mu p \alpha < \text{card}\{ qq, \mu p qq = \text{Some (Vote (Some } v))\} \) by auto

with \( \text{run comm } stp \pi \text{ } \mu q \text{ have } x (\rho (\text{Suc } (\text{Suc } r)) q) = v \)
\[ \quad \text{by (auto dest: set-x-from-vote)} \]

moreover

from \( \text{vsets } \mu q \)

have \( \alpha < \text{card}\{ qq, \mu q qq = \text{Some (Vote (Some } v))\} \) by auto

with \( \text{run comm } stp \pi \text{ } \mu q \text{ have } x (\rho (\text{Suc } (\text{Suc } r)) q) = v \)
\[ \quad \text{by (auto dest: set-x-from-vote)} \]

ultimately

show \( x (\rho (\text{Suc } (\text{Suc } r)) p) = x (\rho (\text{Suc } (\text{Suc } r)) q) \)
\[ \quad \text{by auto} \]

next

assume \( \neg (\exists v. \alpha < \text{card}\{ qq, \mu q qq = \text{Some (Vote (Some } v))\}) \)

with \( \text{stp } \mu q \text{ have } x (\rho (\text{Suc } (\text{Suc } r)) q) = \text{undefined} \)
\[ \quad \text{by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def)} \]

moreover

from \( \text{vsets } \mu q \)

have \( \neg (\exists v. \alpha < \text{card}\{ qq, \mu q qq = \text{Some (Vote (Some } v))\}) \) by auto

with \( \text{stp } \mu q \text{ have } x (\rho (\text{Suc } (\text{Suc } r)) q) = \text{undefined} \)
\[ \quad \text{by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def next1-def)} \]

ultimately

show \( \text{thesis by simp} \)
\[ \text{qed} \]
\[ \text{qed} \]

then obtain \( v \) where \( \forall q. x (\rho (\text{Suc } (\text{Suc } r)) q) = v \) by blast

moreover

from \( \text{stp } \mu q \text{ have } \text{step } (\text{Suc } (\text{Suc } r)) = 0 \)
\[ \quad \text{by (auto simp: step-def mod-Suc)} \]
hence $\bigwedge p \mu p' q$.

$\begin{array}{l}
[q \in \text{SHOs} (\text{Suc} (\text{Suc} r)) \cap \text{HOs} (\text{Suc} (\text{Suc} r)) p; \\
\mu p' \in \text{SHOmsgVectors Ute-M} (\text{Suc} (\text{Suc} r)) p (\rho (\text{Suc} (\text{Suc} r))) \\
(\text{HOs} (\text{Suc} (\text{Suc} r)) p) (\text{SHOs} (\text{Suc} (\text{Suc} r)) p)
\end{array}$

$\implies \mu p' q = (\text{Some} (\text{Val} (x (\rho (\text{Suc} (\text{Suc} r)) q)))$ by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def send0-def)

ultimately have $\bigwedge p \mu p' q$.

$\begin{array}{l}
[q \in \text{SHOs} (\text{Suc} (\text{Suc} r)) \cap \text{HOs} (\text{Suc} (\text{Suc} r)) p; \\
\mu p' \in \text{SHOmsgVectors Ute-M} (\text{Suc} (\text{Suc} r)) p (\rho (\text{Suc} (\text{Suc} r))) \\
(\text{HOs} (\text{Suc} (\text{Suc} r)) p) (\text{SHOs} (\text{Suc} (\text{Suc} r)) p)
\end{array}$

$\implies \mu p' q = (\text{Some} (\text{Val} v))$ 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** $\mu p \in \text{SHOmsgVectors Ute-M} (\text{Suc} (\text{Suc} r)) p (\rho (\text{Suc} (\text{Suc} r)))$ $\text{HOs} (\text{Suc} (\text{Suc} r)) p (\text{SHOs} (\text{Suc} (\text{Suc} r)) p$ and $\text{nxtq: nextState Ute-M} r q (\rho q r q) \mu q (\rho (\text{Suc} (\text{Suc} r)) q)$

and $\text{vq: vote} (\rho (\text{Suc} r) q) = \text{Some} v$

and $\text{qsho: q \in SHOs} (\text{Suc} r) p \cap \text{HOs} (\text{Suc} r) p$

**shows** $\mu p q = (\text{Some} (\text{Vote} (\text{Some} v)))$

**proof**

from $\text{nxtq vq have step r = 0 by (auto simp: vote-step)}$

with $\text{map qsho have \mu p q = Some (Vote (\text{vote} (\rho (\text{Suc} r) q)))}$

by (auto simp: Ute-SHOMachine-def SHOmsgVectors-def Ute-sendMsg-def step-def send1-def mod-Suc)

with $\text{vq show \mu p q = Some (Vote (Som}

姜

We now prove the Termination property.

**theorem** ute-termination:

**assumes** $\text{run: SHORun Ute-M} \rho \text{HOs SHOs}$

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

and $\text{commG: SHOcommGlobal Ute-M} \text{HOs SHOs}$

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

**proof**

from $\text{commG}$

obtain $\Phi \pi r0$

where $rr: r0 = \text{Suc} (\text{nSteps} * \Phi)$

and $\pi: \forall p. p = \text{HOs} r0 p \land \pi = \text{SHOs} r0 p$

and $t: \forall p. \text{card} (\text{SHOs} (\text{Suc} r0) p \cap \text{HOs} (\text{Suc} r0) p) > T$

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 $\text{rr have stp: step r0 = Suc 0 by (auto simp: step-def)}$
obtain \( w \) where \( \forall p. (\text{vote} \ (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p)) = \text{Some} \ w \)

proof -
  have \( \exists w. \text{vote} \ (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p) = \text{Some} \ w \)
  proof
    fix \( p \)
    from \( \text{run} \ \text{stp} \ \text{obtain} \ \mu p \)
    where \( \text{nxt} : \text{nextState} \ \text{Ute-M} \ (\text{Suc} \ r0) \ p \ (\rho \ (\text{Suc} \ r0) \ p) \ \mu p \)
      (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p \)
    and \( \text{mup} : \mu p \in \text{SHOmsgVectors} \ \text{Ute-M} \ (\text{Suc} \ r0) \ p \ (\rho \ (\text{Suc} \ r0)) \)
      (\text{HOs} \ (\text{Suc} \ r0) \ p) \ (\text{SHOs} \ (\text{Suc} \ r0) \ p) \)
    by (auto simp: \text{Ute-SHOMachine-def} \text{SHORun-eq} \text{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 \( \text{run} \ \text{commR} \ \text{stp} \ \pi \ rr \)
    obtain \( v \) where
      \( \forall p. \text{vote} \ (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p) = \text{Some} \ w \)
      proof
        fix \( p \)
        from \( \text{abc} \)
        obtain \( \mu p \)
          where \( \text{nxt} : \text{nextState} \ \text{Ute-M} \ (\text{Suc} \ r0) \ p \ (\rho \ (\text{Suc} \ r0) \ p) \ \mu p \)
            (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p \)
          and \( \text{mup} : \mu p \in \text{SHOmsgVectors} \ \text{Ute-M} \ (\text{Suc} \ r0) \ p \ (\rho \ (\text{Suc} \ r0)) \)
            (\text{HOs} \ (\text{Suc} \ r0) \ p) \ (\text{SHOs} \ (\text{Suc} \ r0) \ p) \)
          by (auto simp: \text{Ute-SHOMachine-def} \text{SHORun-eq} \text{SHOnextConfig-eq})
        have \( \exists v. T < \text{card} \ \{qq. \mu p \ qq = \text{Some} \ (\text{Val} \ v)\} \)
        proof
          from \( \text{run} \ \text{commR} \ \text{stp} \ \pi \ rr \)
          obtain \( v \) where
            \( \forall qq. \text{vote} \ (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p) = \text{Some} \ w \)
            using \( \text{termination-argument-1} \) by blast
          with \( \text{mup} \) obtain \( v \) where
            \( \forall qq. \text{vote} \ (\rho \ (\text{Suc} \ (\text{Suc} \ r0))) \ p) = \text{Some} \ w \)
            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: \text{card mono})
          ultimately
          have \( \exists v. T < \text{card} \ \{qq. \mu p \ qq = \text{Some} \ (\text{Val} \ v)\} \)
          by auto
          thus \( \text{thesis} \) by auto
          qed
          with \( \text{stp} \ \text{next} \ \text{show} \ \exists w. \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ p) = \text{Some} \ w \)
          by (auto simp: \text{Ute-SHOMachine-def} \text{nextState-def} \text{Ute-nextState-def} \text{step-def} \text{mod-Suc next0-def})
          qed
          then obtain \( qq w \) where \( \text{qqw} : \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ qq) = \text{Some} \ w \)
          by blast
          have \( \forall pp. \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ pp) = \text{Some} \ w \)
          proof
            fix \( pp \)
            from \( \text{abc} \)
            obtain \( \text{wp} \) where \( \text{wpwp} : \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ pp) = \text{Some} \ wp \)
            proof
              fix \( pp \)
            qed
            with \( \text{stp} \ \text{next} \ \text{show} \ \exists w. \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ qq) = \text{Some} \ w \)
            by blast
            have \( \forall pp. \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ pp) = \text{Some} \ w \)
            proof
              fix \( pp \)
            from \( \text{abc} \)
            obtain \( \text{wp} \) where \( \text{wpwp} : \text{vote} \ ((\rho \ (\text{Suc} \ (\text{Suc} \ r0)))) \ pp) = \text{Some} \ wp \)
by blast
from run obtain \( \mu \cdot pp \cdot \mu qq \)
where \( \text{nextp: nextState Ute-M} (\text{Suc} r0) \) \( pp \cdot (\text{rho} (\text{Suc} r0)) pp \)
\( \mu pp \cdot (\text{rho} (\text{Suc} r0)) pp \)
and \( \text{map:} \mu pp \in \text{SHOmsgVectors Ute-M} (\text{Suc} r0) \) \( pp \cdot (\text{rho} (\text{Suc} r0)) pp \)
\( \text{HOs} (\text{Suc} r0) pp \cdot (\text{SHOs} (\text{Suc} r0)) pp \)
and \( \text{nextq: nextState Ute-M} (\text{Suc} r0) \) \( qq \cdot (\text{rho} (\text{Suc} r0)) qq \)
\( \mu qq \cdot (\text{rho} (\text{Suc} r0)) qq \)
and \( \text{map:} \mu qq \in \text{SHOmsgVectors Ute-M} (\text{Suc} r0) \) \( qq \cdot (\text{rho} (\text{Suc} r0)) qq \)
\( \text{HOs} (\text{Suc} r0) qq \cdot (\text{SHOs} (\text{Suc} r0)) qq \)
unfolding Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq by blast
from commR this \( \text{pwp} \cdot \text{qqw} \) have \( wp = w \)
by (auto dest: common-vote)
with \( \text{pwp show} \) \( \text{vote} \) \( ((\text{rho} (\text{Suc} r0)))) pp = \text{Some} w \)
by auto
qed
with that show ?thesis by auto
qed

from run obtain \( \mu pp' \)
where \( \text{nextp: nextState Ute-M} (\text{Suc} r0) \) \( p \cdot (\text{rho} (\text{Suc} r0)) p \)
\( \mu pp' \cdot (\text{rho} (\text{Suc} r0) p) p \)
and \( \text{map':} \mu pp' \in \text{SHOmsgVectors Ute-M} (\text{Suc} r0) \) \( p \cdot (\text{rho} (\text{Suc} r0) p) p \)
\( \text{HOs} (\text{Suc} r0) p \cdot (\text{SHOs} (\text{Suc} r0) p) p \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
have \( \forall qq. \) \( \text{qqw} \in \text{SHOs} (\text{Suc} r0) p \cdot (\text{HOs} (\text{Suc} r0) p) p \)
\( \Rightarrow \mu pp' qq = \text{Some} (\text{Vote} (\text{Some} w)) \)
proof –
fix \( qq \)
assume \( \text{qqsho:} qq \in \text{SHOs} (\text{Suc} r0) p \cdot (\text{HOs} (\text{Suc} r0) p) p \)
from run obtain \( \mu qq \) where
\( \text{nextq: nextState Ute-M} (\text{Suc} r0) \) \( qq \cdot (\text{rho} (\text{Suc} r0) qq) \)
\( \mu qq \cdot (\text{rho} (\text{Suc} r0) qq) \)
by (auto simp: Ute-SHOMachine-def SHORun-eq SHOnextConfig-eq)
from commR \( \text{map': nextq votew qqsho show} \) \( \mu pp' qq = \text{Some} (\text{Vote} (\text{Some} w)) \)
by (auto dest: termination-argument-2)
qed
hence \( \text{SHOs} (\text{Suc} r0) p \cdot (\text{HOs} (\text{Suc} r0) p) p \)
\( \subseteq \{ qq. \mu pp' qq = \text{Some} (\text{Vote} (\text{Some} w)) \} \)
by auto
hence \( \text{wsho: card} (\text{SHOs} (\text{Suc} r0) p) \cdot (\text{HOs} (\text{Suc} r0) p) p \)
\( \subseteq \text{card} \{ qq. \mu pp' qq = \text{Some} (\text{Vote} (\text{Some} w)) \} \)
by (auto simp: card_mono)
from \( \text{stp have} \) \( \text{step} (\text{Suc} r0) = \text{Suc} \emptyset \)
unfolding \( \text{step-def} \) by auto
with \( \text{nextp have} \) \( \text{next1} (\text{Suc} r0) p \cdot (\text{rho} (\text{Suc} r0) p) \mu pp' \)
\( (\text{rho} (\text{Suc} (\text{Suc} r0))) p \)
by (auto simp: Ute-SHOMachine-def nextState-def Ute-nextState-def)
moreover
from e have E < card (SHOs (Suc (Suc r0)) p ∩ HOs (Suc (Suc r0)) p)
  by auto
with wsho have majv:card {qq.pp' qq = Some (Vote (Some w))} > E
  by auto
ultimately
show thesis by (auto simp: next1-def)
qed

8.7 \( U_{T,E,\alpha} \) 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: \( \forall r. \ SHOcommPerRd \ Ute-M \ (HOs \ r) \ (SHOs \ r) \)
            and commG: SHOcommGlobal Ute-M HOs 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 commG]
by auto

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

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

end — context ute-parameters
9 Verification of the $A_{T,E,\alpha}$ Consensus algorithm

Algorithm $A_{T,E,\alpha}$ is presented in [3]. Like $U_{T,E,\alpha}$, it is an uncoordinated algorithm that tolerates value faults, and it is parameterized by values $T$, $E$, and $\alpha$ that serve a similar function as in $U_{T,E,\alpha}$, allowing the algorithm to be adapted to the characteristics of different systems. $A_{T,E,\alpha}$ can be understood as a variant of OneThirdRule tolerating Byzantine faults.

We formalize in Isabelle the correctness proof of the algorithm that appears in [3], using the framework of theory $HOModel$.

9.1 Model of the Algorithm

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

```isabelle
typedef Proc -- the set of processes
axiomatization where Proc-finite: OFCLASS(Proc, finite-class)
instance Proc :: finite by (rule Proc-finite)
```

abbreviation

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

The following record models the local state of a process.

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

The $x$ field of the initial state is unconstrained, but no decision has yet been taken.

```isabelle
definition Ate-initState where
  Ate-initState p st \equiv (decide st = None)
```

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

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

begin
The following are consequences of the assumptions on the parameters.

**Lemma** \( \text{majE} \): \( 2 \times (E - \alpha) \geq N \)

**Using** \( \text{TNaE \ TltN \ by \ auto} \)

**Lemma** \( \text{Egtl} \): \( E > \alpha \)

**Using** \( \text{majE \ EltN \ by \ auto} \)

**Lemma** \( \text{Tge2a} \): \( T \geq 2 \times \alpha \)

**Using** \( \text{TNaE \ EltN \ by \ auto} \)

At every round, each process sends its current \( x \). If it received more than \( T \) messages, it selects the smallest value and store it in \( x \). As in algorithm \( \text{OneThirdRule} \), we therefore require values to be linearly ordered.

If more than \( E \) messages holding the same value are received, the process decides that value.

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

\[
\text{mostOftenRcvd} (\text{msgs}:\text{Proc} \rightarrow \text{val option}) \equiv
\{ v \cdot \forall w. \text{card} \{qq. \text{msgs} qq = \text{Some} w\} \leq \text{card} \{qq. \text{msgs} qq = \text{Some} v\}\}
\]

**Definition** \( \text{Ate-sendMsg} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{val pstate} \Rightarrow \text{val} \)

**Where**

\( \text{Ate-sendMsg} r p q st \equiv x st \)

**Definition** \( \text{Ate-nextState} :: \text{nat} \Rightarrow \text{Proc} \Rightarrow (\text{val} :: \text{linorder}) \text{pstate} \Rightarrow (\text{Proc} \Rightarrow \text{val option}) \Rightarrow \text{val pstate} \Rightarrow \text{bool} \)

**Where**

\( \text{Ate-nextState} r p st msgs st' \equiv x st' \)

\[
\text{if card} \{q. \text{msgs} q \neq \text{None}\} > T \\
\text{then} x st' = \text{Min} (\text{mostOftenRcvd} \text{msgs}) \\
\text{else} x st' = x st \\
\land \left( \begin{array}{c}
\exists v. \text{card} \{q. \text{msgs} q = \text{Some} v\} > E \\
\land \text{decide} st' = \text{Some} v
\end{array} \right) \\
\lor \neg \left( \exists v. \text{card} \{q. \text{msgs} q = \text{Some} v\} > E \\
\land \text{decide} st' = \text{decide} st
\right)
\]

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

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

**Definition** \( \text{Ate-commPerRd} \) where

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

The global communication predicate stipulates the three following conditions:
• for every process $p$ there are infinitely many rounds where $p$ receives more than $T$ messages,

• for every process $p$ there are infinitely many rounds where $p$ receives more than $E$ uncorrupted messages,

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

**Definition**

$A_{T,E,\alpha}$-commGlobal where

$A_{T,E,\alpha}$-commGlobal HOs SHOs $\equiv$

$(\forall r. \exists r' > r. \text{card}(\text{HOs } r' p) > T)$

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

$\land (\forall r. \exists r' > r. \exists \pi_1 \pi_2$.

\hspace{1cm} \text{card} \pi_1 > E - \alpha$

$\land \text{card} \pi_2 > T$

$\land (\forall p \in \pi_1. \text{HOs } r' p = \pi_2 \land \text{SHOs } r' p \cap \text{HOs } r' p = \pi_2))$

### 9.3 The $A_{T,E,\alpha}$ Heard-Of Machine

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

**Definition**

$A_{T,E,\alpha}$-SHOMachine where

$A_{T,E,\alpha}$-SHOMachine $\equiv$

$(\lambda p. \text{Ate-initState } p)$,

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

$\text{CnextState} = (\lambda r p st msgs crd st' . \text{Ate-nextState } r p st msgs st')$,

$\text{SHOcommPerRd} = (\lambda HOs. \text{Proc } HO \Rightarrow \text{Proc } HO \Rightarrow \text{bool})$,

$\text{SHOcommGlobal} = \text{Ate-commGlobal}$

**Abbreviation**

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

**end** — locale ate-parameters

**end**

**theory** AteProof

**imports** AteDefs ./Reduction

**begin**

**context** ate-parameters

**begin**
9.4 Preliminary Lemmas

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

**lemma** decide-sent-msgs-threshold:

**assumes** run: SHORun Ate-M \( \rho \) HOs SHOs

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

**and** nep: decide \( (\rho \ r \ p) \neq \text{Some } v \)

**and** vp: decide \( (\rho \ (\text{Suc } r) \ p) = \text{Some } v \)

**shows** \( \text{card } \{ qq. \text{sendMsg Ate-M } r \ qq \ p \ (\rho \ r \ qq) = v \} > E - \alpha \)

**proof**

- from run obtain \( \mu p \)
  
  **where** \( \mu p : \mu p \in \text{SHOmsgVectors Ate-M } r \ p \ (\rho \ r \ p) (\text{SHOs } r \ p) \)

- and \( \text{nxt: nextState Ate-M } r \ p \ (\rho \ r \ p) \ \mu p \ (\rho \ (\text{Suc } r) \ p) \)
  
  by (auto simp: SHORun-eq SHOnextConfig-eq)

from \( \mu p \)

- have \( \{ qq. \mu p \ qq = \text{Some } v \} - (\text{HOs } r \ p - \text{SHOs } r \ p) \) \( \subseteq \{ qq. \text{sendMsg Ate-M } r \ qq \ p \ (\rho \ r \ qq) = v \} \)
  
  (is \text{?vrcvd} - ?ahop \subseteq ?vsentp)

  by (auto simp: SHOmsgVectors-def)

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

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

moreover

- from \( \text{nxt \ nep \ vp \ have} \text{ card } ?vrcvd > E \)
  
  by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)

moreover

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

ultimately

- show \( ?\text{thesis \ using} \ Egta \) by auto

**qed**

If more than \( E - \alpha \) processes send a value \( v \) to some process \( q \) at some round, then \( q \) will receive at least \( N + 2\alpha - E \) messages holding \( v \) at this round.

**lemma** other-values-received:

**assumes** \( \text{comm: SHOcommPerRd Ate-M } (\text{HOs } r) (\text{SHOs } r) \)

**and** \( \text{nxt: nextState Ate-M } r \ q \ (\rho \ r \ q) \ \mu q \ ((\rho \ (\text{Suc } r)) \ q) \)

**and** \( \mu q : \mu q \in \text{SHOmsgVectors Ate-M } r \ q \ (\rho \ r \ q) (\text{SHOs } r \ q) \)

**and** \( \text{vsent: card } \{ qq. \text{sendMsg Ate-M } r \ qq \ q \ (\rho \ r \ qq) = v \} > E - \alpha \)

  (is card ?vsent > -)

**shows** \( \text{card } \{ qq. \mu q \ qq \neq \text{Some } v \} \cap \text{HOs } r \ q \leq N + 2\alpha - E \)

**proof**

- from \( \text{nxt \ muq} \)

  **have** \( \{ qq. \mu q \ qq \neq \text{Some } v \} \cap \text{HOs } r \ q - (\text{HOs } r \ q - \text{SHOs } r \ q) \) \( \subseteq \{ qq. \text{sendMsg Ate-M } r \ qq \ q \ (\rho \ r \ qq) \neq v \} \)

  (is \text{?notvrcvd} - ?aho \subseteq ?notvsent)

126
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 \textit{comm} have \( \text{card } ?\text{aho} \leq \alpha \)

by \( \text{(auto simp: Ate-SHOMachine-def Ate-commPerRd-def)} \)

moreover

have \( 1 : \text{card } ?\text{notvsent} + \text{card } ?\text{vsent} = \text{card } (?\text{notvsent} \cup ?\text{vsent}) \)

by \( \text{(subst card-Un-Int)} \) \textit{auto}

have \( ?\text{vsent} \cup ?\text{notvsent} = (\text{UNIV::Proc set}) \) by \textit{auto}

hence \( \text{card } (\text{?notvsent} \cup ?\text{vsent}) = \text{N} \) by \textit{simp}

ultimately

show \( ?\text{thesis} \) using \textit{EltN Egta} by \textit{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}: \( \text{SHOcommPerRd Ate-M (HOs r) (SHOs r)} \)

\textbf{and} \textit{nxt}: \( \text{nextState Ate-M r q (rho r q) } \mu q ((\text{rho (Suc r)}) q) \)

\textbf{and} \textit{muq}: \( \mu q \in \text{SHOmsgVectors Ate-M r q (rho r) (HOs r q) (SHOs r q)} \)

\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 \textit{hodef}: \( \text{HOs r q = \{qq. } \mu q qq \neq \text{None}\} \)

unfolding \textit{SHOmsgVectors-def} by \textit{auto}

from \textit{comm \textit{nxt} \textit{muq} \textit{threshold-E}}

have \( \text{card } \{\text{\{qq. } \mu q qq \neq \text{Some } v\} \cap \text{HOs r q}\} \leq N + 2 \alpha - E \)

(is \( \text{card } ?\text{heardnotv} \leq -\) )

by \( \text{(rule other-values-received)} \)

moreover

have \( \text{card } ?\text{heardnotv} \geq T + 1 - \text{card } \{\text{qq. } \mu q qq = \text{Some } v\} \)

\textbf{proof} –

from \textit{muq}

have \( ?\text{heardnotv} = (\text{HOs r q}) - \{\text{qq. } \mu q qq = \text{Some } v\} \)

and \( \{\text{qq. } \mu q qq = \text{Some } v\} \subseteq \text{HOs r q} \)

unfolding \textit{SHOmsgVectors-def} by \textit{auto}

hence \( \text{card } ?\text{heardnotv} = \text{card } (\text{HOs r q}) - \text{card } \{\text{qq. } \mu q qq = \text{Some } v\} \)

by \( \text{(auto simp: card-Diff-subset)} \)

with \textit{hodef} \textit{threshold-T} show \( ?\text{thesis} \) by \textit{auto}

qed

ultimately

have \( \text{card } \{\text{qq. } \mu q qq = \text{Some } v\} > \text{card } ?\text{heardnotv} \)

using \textit{TNaE} by \textit{auto}

127
moreover
{
  fix w
  assume w: w ≠ v
  with hodef have { qq, μq qq = Some w } ⊆ ?heardno v by auto
  hence card { qq, μq qq = Some w } ≤ card ?heardno v 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)
        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)
        next
      qed
    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** \( \text{run}: \text{SHORun Ate-M} \ \rho \ \text{HOs SHOs} \)

**And** \( \text{comm} \): \( \forall r. \) \( \text{SHOcommPerRd} \ (\text{Ate-M}::(\text{Proc}, \text{val::linorder pstate}, \text{val}) \ \text{SHOMachine}) \)

\[ (\text{HOs} \ r) \ (\text{SHOs} \ r) \]

**And** \( d1: \text{decide} \ (\rho \ r \ p) \neq \text{Some} \ v \)

**And** \( d2: \text{decide} \ (\rho \ (\text{Suc} \ r) \ p) = \text{Some} \ v \)

**Shows** \( x \ (\rho \ (\text{Suc} \ k) \ q) \neq x \ (\rho \ (r + k) \ q) \)

**Proof**

**From** \( \text{comm} \)

**Have** \( \text{SHOcommPerRd} \ (\text{Ate-M}::(\text{Proc}, \text{val::linorder pstate}, \text{val}) \ \text{SHOMachine}) \)

\[ (\text{HOs} \ (r+k)) \ (\text{SHOs} \ (r+k)) \] ..

**Moreover**

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

**Where** \( \text{nxt: nextState} \ \text{Ate-M} \ (r+k) \ q \ (\rho \ (r+k) \ q) \ \mu q \ (\rho \ (r + \text{Suc} \ k) \ q) \)

**And** \( muq: \mu q \in \text{SHOMsgVectors Ate-M} \ (r+k) \ q \ (\rho \ (r+k)) \)

\[ (\text{HOs} \ (r+k) \ q) \ (\text{SHOs} \ (r+k) \ q) \]

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

**Moreover**

**From** \( \text{nxt qupdate} \)

**Have** \( \text{threshold-T}: \text{card} \ \{qq. \ \mu q \ qq \neq \text{None} \} > T \)

**And** \( xsmall: x \ (\rho \ (r + \text{Suc} \ k) \ q) = \text{Min} \ (\text{mostOftenRcvd} \ \mu q) \)

**By** \( (\text{auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def}) \)

**Moreover**

**Have** \( E - \alpha < \text{card} \ \{qq. \ x \ (\rho \ (r + k) \ qq) = v \} \)

**Proof** (\text{induct} \ k)

**From** \( \text{run comm} \ d1 \ d2 \)

**Have** \( E - \alpha < \text{card} \ \{qq. \ \text{sendMsg Ate-M} \ r \ qq \ p \ (\rho \ r \ qq) = v \} \)

**By** \( (\text{auto dest: decide-sent-msgs-threshold}) \)

**Thus** \( E - \alpha < \text{card} \ \{qq. \ x \ (\rho \ (r + 0) \ qq) = v \} \)

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

**Next**

**Fix** \( k \)

**Assume** \( E - \alpha < \text{card} \ \{qq. \ x \ (\rho \ (r + k) \ qq) = v \} \)

**With** \( \text{run comm} \) show \( E - \alpha < \text{card} \ \{qq. \ x \ (\rho \ (r + \text{Suc} \ k) \ qq) = v \} \)

**By** \( (\text{auto dest: common-x-induct}) \)

**QED**
A process that holds some decision \( v \) has decided \( v \) sometime in the past.

**lemma** decisionNonNullThenDecided:
**assumes** run: SHORun Ate-M rho HOs SHOs
and dec: decide (rho n p) = Some v
obtains m where m < n
  and decide (rho m p) \neq Some v
  and decide (rho (Suc m) p) = Some v

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

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

**with** 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** run: SHORun Ate-M rho HOs SHOs
and comm: \( \forall r. \ \text{SHOcommPerRd} \ Ate-M \ (\text{HOs} \ r) \ (\text{SHOs} \ r) \)
and initv: \( \forall q. \ x \ (\rho \ 0 \ q) = v \)
and dp: decide (rho r p) = Some w
**shows** \( w = v \)

**proof** –
\[
\begin{array}{l}
\{ \\
\text{fix } r \\
\text{have } \forall qq. \ \text{sendMsg} \ Ate-M \ r \ qq \ p \ (\rho \ r \ qq) = v \\
\text{proof (induct r)} \\
\text{from run initv show } \forall qq. \ \text{sendMsg} \ Ate-M \ 0 \ qq \ p \ (\rho \ 0 \ qq) = v \\
\text{by (auto simp: SHORun-eq HO\text{nextConfig-eq} Ate-SHOMachine-def Ate-sendMsg-def)}
\end{array}
\]
next
fix r
assume ih: \( \forall \, qq. \, \text{sendMsg Ate-M r qq p (rho r qq)} = v \)

have \( \forall \, qq. \, x (\rho (\text{Suc r}) \, qq) = v \)
proof
fix qq
from run obtain \( \mu qq \)
  where \( \text{nxt: nextState Ate-M r qq (rho r qq) \mu qq (rho (\text{Suc r}) \, qq)} \)
  and \( \mu qq \in \text{SHOmsgVectors Ate-M r qq (rho r) (HOs r qq) (SHOs r qq)} \)
by (auto simp: SHORun-eq SHOnextConfig-eq)
from nxt
have \( \text{card \{pp. \mu qq pp \neq None\} > T \land x (\rho (\text{Suc r}) \, qq) = \text{Min (mostOftenRcvd \mu qq)})} \)
  \( \lor (\text{card \{pp. \mu qq pp \neq None\} \leq T \land x (\rho (\text{Suc r}) \, qq) = x (\rho r qq))} \)
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
thus \( x (\rho (\text{Suc r}) \, qq) = v \)
proof safe
assume \( x (\rho (\text{Suc r}) \, qq) = x (\rho r qq) \)
with ih show \( ?\text{thesis} \)
by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
next
assume \( \text{threshold-T: T < card \{pp. \mu qq pp \neq None\}} \)
  \( \land xsmall: x (\rho (\text{Suc r}) \, qq) = \text{Min (mostOftenRcvd \mu qq)} \)
have \( \text{card \{pp. \exists w. \, w \neq v \land \mu qq pp = Some w\} \leq T \text{ div 2}} \)
proof -
from comm have \( 1: \text{card (HOs r qq - SHOs r qq)} \leq \alpha \)
  by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
moreover
from \( \mu \) ih
have \( \text{SHOs r qq \cap HOs r qq} \subseteq \{pp. \mu qq pp = Some v\} \)
  and \( \text{HOs r qq = \{pp. \mu qq pp \neq None\}} \)
by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def Ate-sendMsg-def)
have \( \{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = Some v\} \subseteq \text{HOs r qq - SHOs r qq} \)
  \( \subseteq \text{HOs r qq - SHOs r qq} \)
by auto
hence \( \text{card (\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = Some v\})} \leq \text{card (HOs r qq - SHOs r qq)} \)
  \( \leq \text{card (HOs r qq - SHOs r qq)} \)
by (auto simp:card-mono)
ultimately
have \( \text{card (\{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = Some v\})} \leq T \text{ div 2} \)
using Tge2a by auto
moreover
have \( \{pp. \mu qq pp \neq None\} - \{pp. \mu qq pp = Some v\} = \{pp. \exists w. \, w \neq v \land \mu qq pp = Some w\} \)
by auto
ultimately
show \( ?\text{thesis} \) by simp
qed
moreover
have \{ pp. \mu qq \neq None \} 
\quad = \{ pp. \mu qq \neq None \} \cup \{ pp. \exists w. w \neq v \land \mu qq = Some w \} 
\quad \land \{ pp. \mu qq = Some v \} \cap \{ pp. \exists w. w \neq v \land \mu qq = Some w \} = 
\{ \}
\quad by \ auto
hence \{ pp. \mu qq \neq None \} 
\quad = \{ pp. \mu qq = Some v \} + \{ pp. \exists w. w \neq v \land \mu qq = Some w \} 
\quad by (auto simp: card-Un-Int)
moreover
note threshold-T
ultimately
have \{ pp. \mu qq = Some v \} \cap \{ pp. \exists w. w \neq v \land \mu qq = Some w \} = \{ \}
\quad by (auto simp: card-mono)
ultimately
have zz:\wedge w. w \neq v \implies 
\quad card \{ pp. \mu qq = Some w \} < card \{ pp. \mu qq = Some v \} 
\quad by force
hence \wedge w. card \{ pp. \mu qq = Some v \} \leq card \{ pp. \mu qq = Some w \} 
\quad \implies w = v 
\quad by force
with zz have mostOftenRcvd \mu qq = \{ v \} 
\quad 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 
\quad by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
qed
)

note P = this

from run dp obtain rp
where rp: rp < r \decide (rho rp p) \neq Some w
\quad \decide (rho (Suc rp) p) = Some w 
\quad by (rule decisionNonNullThenDecided)
from run obtain μp
where nxt: nextState Ate-M rp p (rho rp p) μp (rho (Suc rp) p)
    and μw; μp ∈ SHOmsgVectors Ate-M rp p (rho rp p) (HOs rp p) (SHOs rp p)
by (auto simp: SHORun-eq SHOnextConfig-eq)
{
  fix w
  assume w: w ≠ v
  from comm have card (HOs rp p - SHOs rp p) ≤ α
    by (auto simp: Ate-SHOMachine-def Ate-commPerRd-def)
  moreover
from mu P have SHOs rp p ∩ HOs rp p ⊆ {pp. μp pp = Some v}
    and HOs rp p = {pp. μp pp ≠ None}
    by (auto simp: SHOmsgVectors-def)
  hence {pp. μp pp ≠ None} - {pp. μp pp = Some v} ⊆ HOs rp p - SHOs rp p
    by auto
  hence card ({pp. μp pp ≠ None} - {pp. μp pp = Some v}) ≤ card (HOs rp p - SHOs rp p)
    by (auto simp: card-mono)
  ultimately
have card ({pp. μp pp ≠ None} - {pp. μp pp = Some v}) < E
    using Egta by auto
  moreover
from w have {pp. μp pp = Some w}
    ⊆ {pp. μp pp ≠ None} - {pp. μp pp = Some v}
    by auto
  hence card {pp. μp pp = Some w} ≤ card ({pp. μp pp ≠ None} - {pp. μp pp = Some v})
    by (auto simp: card-mono)
  ultimately
have card {pp. μp pp = Some w} < E by simp
}
  hence PP: ∀w. card {pp. μp pp = Some w} ≥ E ⇒ w = v by force
from rp nxt mu have card {q. μp q = Some w} > E
  by (auto simp: SHOmsgVectors-def Ate-SHOMachine-def
  nextState-def Ate-nextState-def)
with PP show thesis by auto
qed

9.6 Proof of Agreement
If two processes decide at the same round, they decide the same value.

lemma common-decision:
  assumes run: SHORun Ate-M rho HOs SHOs
  and comm: SHOcommPerRd Ate-M (HOs r) (SHOs r)

and \( nvp \): decide \((\rho_r \text{ p}) \neq \text{ Some } v\)
and \( vp \): decide \((\rho_r \text{ (Suc } r) \text{ p}) = \text{ Some } v\)
and \( nwp \): decide \((\rho_r \text{ q}) \neq \text{ Some } w\)
and \( wp \): decide \((\rho_r \text{ (Suc } r) \text{ q}) = \text{ Some } w\)
shows \( w = v\)

proof –
  have \( gtn: \text{ card } \{qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = v\} + \text{ card } \{qq. \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq}) = w\} > N\)
  proof –
  from \( \text{ run comm nvp vp}\)
  have \( \text{ card } \{qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = v\} > E - \alpha\)
    by (rule decide-sent-mgs-threshold)
  moreover
  from \( \text{ run comm nwq wq}\)
  have \( \text{ card } \{qq. \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq}) = w\} > E - \alpha\)
    by (rule decide-sent-mgs-threshold)
  ultimately
  show \( \text{ thesis using majE by auto}\)
qed

show \( \text{ thesis}\)
proof (rule ccontr)
  assume \( vv: v \neq w\)
  have \( \forall qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq})\)
    by (auto simp: Ate-SHOMachine-def Ate-sendMsg-def)
  with \( vv\)
  have \( \{qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = v\} \cap \{qq. \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq}) = w\} = \{\}\)
    by auto
  with \( gtn\)
  have \( \text{ card } \{\{qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = v\} \cup \{qq. \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq}) = w\}\} > N\)
    by (auto simp: card-Un-Int)
  moreover
  have \( \text{ card } \{\{qq. \text{ sendMsg } Ate-M r qq p (\rho_r \text{ qq}) = v\} \cup \{qq. \text{ sendMsg } Ate-M r qq q (\rho_r \text{ qq}) = w\}\} \leq N\)
    by (auto simp: card-mono)
  ultimately
  show \( \text{ False by auto}\)
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 : \(\text{ assumes run: SHORun Ate-M rho HOs SHOs}\)
and \( \text{ comm}: \forall r. \text{ SHOcommPerRd Ate-M (HOs } r) (SHOs r)\)
and \( nd1\): decide \((\rho_r \text{ p}) \neq \text{ Some } v\)
and d1: decide (\( \rho (\text{Suc } r) \) p) = Some v
and nd2: decide (\( \rho (r+k) \) q) \neq Some w
and d2: decide (\( \rho (\text{Suc } (r+k)) \) q) = Some w
shows w = v
proof (rule ccontr)
  assume vdifw: w \neq v
  have kgt0: k > 0
  proof (rule ccontr)
    assume \( \neg k > 0 \)
    hence k = 0 by auto
  with run comm nd1 d1 nd2 d2 have w = v
  by (auto dest: common-decision)
with vdifw show False ..
qed

have 1: \{qq. sendMsg Ate-M r qq (\( \rho r qq \)) = v\}
 \cap \{qq. 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 \wedge 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 \( \Rightarrow \) x (\( \rho (r+k') qq \)) \neq w
  and ksk':\text{Suc } k' < k
  from ksk' have k' < k by simp
  with ih f 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 ksk:Suc (k - 1) = k by auto
ultimately
  have x (\( \rho (r + \text{Suc } (k - 1)) qq \)) \neq w by blast
with rkw ksk show False by simp
qed
then obtain k'
where k' < k
and \( w \): \[ x \cdot (\rho \cdot (r + \text{Suc } k')) \cdot qq = w \]
and \( qqupdatex \): \[ x \cdot (\rho \cdot (r + \text{Suc } k')) \cdot qq \neq x \cdot (\rho \cdot (r + k') \cdot qq) \]
by \textit{auto}
from run comm nd1 d1 qqupdatex
have \( x \cdot (\rho \cdot (r + \text{Suc } k')) \cdot qq = v \) by \textit{(rule common-x)}
with \( w v d i f w \) show \textit{False} by \textit{simp}
qed

from run comm nd1 d1 have \( \text{sentv} \cdot (\text{card } ?\text{sentv}) > E - \alpha \)
by \textit{(auto dest: decide-sent-msgs-threshold)}
from run comm nd2 d2 have \( \text{sentw} \cdot (\text{card } ?\text{sentw}) > E - \alpha \)
by \textit{(auto dest: decide-sent-msgs-threshold)}
with \( \text{sentv} \cdot \text{majE} \) have \( (\text{card } ?\text{sentv}) + (\text{card } ?\text{sentw}) > N \)
by \textit{simp}
with \( 1 \cdot v d i f w \) have \( 2 \cdot (\text{card } ?\text{sentv} \cup ?\text{sentw}) > N \)
by \textit{(auto simp: card-Un-Int)}
have \( (\text{card } ?\text{sentv} \cup ?\text{sentw}) \leq N \)
by \textit{(auto simp: card-mono)}
with \( 2 \) show \textit{False} by \textit{simp}
qed

The Agreement property is now an immediate consequence.

\textbf{theorem ate-agreement:}
\begin{itemize}
  \item assumes \textit{run}: \textit{SHORun Ate-M rho HOs SHOs}
  \item and \textit{comm}: \( \forall r. \textit{SHOcommPerRd Ate-M} (\textit{HOs} \ r) (\textit{SHOs} \ r) \)
  \item and \( p \): decide \( (\rho \cdot m \cdot p) = \text{Some } v \)
  \item and \( q \): decide \( (\rho \cdot n \cdot q) = \text{Some } w \)
  \item shows \( w = v \)
\end{itemize}
proof –
from \textit{run \( p \) obtain \( k \) where}
  \( k \cdot k < m \) decide \( (\rho \cdot k \cdot p) \neq \text{Some } v \) decide \( (\rho \cdot (\text{Suc } k) \cdot p) = \text{Some } v \)
by \textit{(rule decisionNonNullThenDecided)}
from \textit{run \( q \) obtain \( l \) where}
  \( l \cdot l < n \) decide \( (\rho \cdot l \cdot q) \neq \text{Some } w \) decide \( (\rho \cdot (\text{Suc } l) \cdot q) = \text{Some } w \)
by \textit{(rule decisionNonNullThenDecided)}
show \textit{?thesis}
proof \textit{(cases \( k \leq l \))}
  case \textit{True}
  then obtain \( i \) where \( l = k + i \) by \textit{(auto simp add: le-iff-add)}
  with \textit{run comm \( k \cdot l \) show \textit{?thesis}}
  by \textit{(auto dest: laterProcessDecidesSameValue)}
next
  case \textit{False}
  hence \( l \leq k \) by \textit{simp}
  then obtain \( i \) where \( m = l + i \) by \textit{(auto simp add: le-iff-add)}
  with \textit{run comm \( k \cdot l \) show \textit{?thesis}}
  by \textit{(auto dest: laterProcessDecidesSameValue)}
qed
qed
9.7 Proof of Termination

We now prove that every process must eventually decide, given the global and round-by-round communication predicates.

**theorem ate-termination:**

assumes run: SHORun Ate-M rho HOs SHOs
and commR: $\forall \ r. (SHO\text{commPerRd}::((\text{Proc}, '\text{val}::\text{linorder\ pstate}, '\text{val}) \text{SHO\machine})$

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

and commG: SHO\text{commGlobal} Ate-M HOs SHOs
shows $\exists r \ v. \ \text{decide} (\rho \ r\ p) = \text{Some\ v}$

**proof**

from 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 hosho: $\forall p \in \pi 1. (\text{HOs\ }r' p = \pi 2 \land \text{SHOs\ }r' p \cap \text{HOs\ }p = \pi 2)$
by (auto simp: Ate-SHOMachine-def Ate-commGlobal-def)

obtain $v$ where
P1: $\forall \ p. \ \text{card\ } \{qq. \ \mu p qq \neq None\} = \text{HOs\ }r' q$
and $\exists q. \ \mu q qq = \text{Some\ } (\text{sendMsg\ Ate-M\ }r' qq q (\rho\ Suc\ r' qq))$

by (auto simp: SHOmsgVectors-def)

from run obtain $\mu p$
where $\mu p \in \text{SHOmsgVectors\ Ate-M\ }r' p (\rho\ Suc\ r' p) (\text{SHOs\ }r' p)$
by (auto simp: SHORun-eq SHOnextConfig-eq)

from run obtain $\mu q$
where $\mu q \in \text{SHOmsgVectors\ Ate-M\ }r' q (\rho\ Suc\ r' q)$
and $\exists p. \ \mu q qq = \text{Some\ } (\text{sendMsg\ Ate-M\ }r' qq p (\rho\ Suc\ r' qq))$

by (auto simp: SHORun-eq SHOnextConfig-eq)

from $\mu p\ muq\ p\ q$
have $\{qq. \ \mu p qq \neq None\} = \text{HOs\ }r' q$
and $2: \{qq. \ \mu q qq = \text{Some\ } (\text{sendMsg\ Ate-M\ }r' qq q (\rho\ Suc\ r' qq))\}$
$\supset \text{SHOs\ }r' q \cap \text{HOs\ }r' q$
and $\{qq. \ \mu p qq \neq None\} = \text{HOs\ }r' p$
and $4: \{qq. \ \mu q qq = \text{Some\ } (\text{sendMsg\ Ate-M\ }r' qq p (\rho\ Suc\ r' qq))\}$
$\supset \text{SHOs\ }r' p \cap \text{HOs\ }r' p$
by (auto simp: SHOmsgVectors-def)

with $p\ q$ hosho
have \( \text{aa} : \pi^2 = \{ qq. \mu q qq \neq \text{None} \} \)
and \( \text{cc} : \pi^2 = \{ qq. \mu p qq \neq \text{None} \} \) by auto
from \( p q \) ho 2
have \( \text{bb} : \{ qq. \mu q qq = \text{Some} (\text{sendMsg Ate-M r'} qq q (\rho r' qq)) \} \supseteq \pi^2 \)
by auto
from \( p q \) ho 4
have \( \text{dd} : \{ qq. \mu p qq = \text{Some} (\text{sendMsg Ate-M r'} qq p (\rho r' qq)) \} \supseteq \pi^2 \)
by auto
have \( \text{Min} (\text{mostOftenRcvd } \mu p) = \text{Min} (\text{mostOftenRcvd } \mu q) \)

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 \( \text{auto simp: Ate-SHOMachine-def Ate-sendMsg-def} \)
with \( \text{aa bb cc dd} \)

moreover
from \( \text{aa bb cc dd} \)
have \( \{ qq. \mu p qq \neq \text{None} \} = \{ qq. \mu q qq \neq \text{None} \} \) by auto
hence \( \forall qq. \mu p qq = \text{None} \iff \mu q qq = \text{None} \) by blast
hence \( \forall qq. \mu p qq = \text{None} \iff \mu p qq = \mu q qq \) by auto
ultimately
have \( \forall qq. \mu p qq = \mu q qq \) by blast
thus \( ?\text{thesis} \) by \( \text{auto simp: mostOftenRcvd-def} \)
qed

with \( \pi^t \) \( aa \) \( nxnq \) \( nt \) \( cc \) \( nxnq \) show \( x (\rho (\text{Suc } r') p) = x (\rho (\text{Suc } r') q) \)
by \( \text{(auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)} \)
qed

then obtain \( v \) where \( \text{Pv} : \forall p \in \pi^1. x (\rho (\text{Suc } r') p) = v \) by blast

\{
fix \( pp \)
from \( \text{Pv} \) have \( \forall p \in \pi^1. \text{sendMsg Ate-M (Suc } r') p pp (\rho (\text{Suc } r') p) = v \)
by \( \text{(auto simp: Ate-SHOMachine-def Ate-sendMsg-def)} \)
hence \( \text{card } \pi^1 \leq \text{card } \{ qq. \text{sendMsg Ate-M (Suc } r') qq pp (\rho (\text{Suc } r') qq) = v \} \)
by \( \text{(auto intro: card-mono)} \)
with \( \pi^a \)
have \( E - \alpha < \text{card } \{ qq. \text{sendMsg Ate-M (Suc } r') qq pp (\rho (\text{Suc } r') qq) = v \} \)
by \( \text{simp} \)
\}
with that show \( ?\text{thesis} \) by blast
qed

\{
fix \( k pp \)
have \( E - \alpha < \text{card } \{ qq. \text{sendMsg Ate-M (Suc } r' + k) qq pp (\rho (\text{Suc } r' + k) qq) = v \} \)
(is \( ?P k \)
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 ppupdate: 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 µ pp
    where nxt: nextState Ate-M (Suc r' + k) pp (rho (Suc r' + k) pp) µ pp
      (rho (Suc r' + Suc k) pp)
    and mu: µ pp ∈ 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 ppupdate
  have threshold-T: card \{qq. µ pp qq ≠ None\} > T
    and zsmall: x (rho (Suc r' + Suc k) pp) = Min (mostOftenRcvd µ pp)
    by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)
  moreover
  from P2
  have E - α < card \{qq. sendMsg Ate-M (Suc r' + k) qq pp (rho (Suc r' + k) qq) = v\}.
  ultimately
  have mostOftenRcvd µ pp = \{v\} by (auto dest!: mostOftenRcvd-v)
  with zsmall
  have x (rho (Suc r' + Suc k) pp) = v by simp

  139
\[
\forall p. \exists k. x \ (\rho (\text{Suc } r' + \text{Suc } k) \ pp) = v
\]

**proof**

- **fix** \(pp\)
- **from** \(\text{commG}\) **have** \(\exists r'' > r'. \ \text{card} (\text{HOs } r'' \ pp) > T\)
  - **by** (auto simp: \(\text{Ate-SHOMachine-def}\ Ate-\text{commGlobal-def}\))
- **then obtain** \(k\) **where** \(\text{Suc } r' + k > r'\) **and** \( t : \text{card} (\text{HOs } (\text{Suc } r' + k) \ pp) > T\)
- **by** (auto dest: less-imp-Suc-add)
- **moreover**
- **from** \(\text{run}\) **obtain** \(\mu pp\)
  - **where** \(\text{nxt: nextState } Ate-M (\text{Suc } r' + k) \ pp (\rho (\text{Suc } r' + k) \ pp) \mu pp\)
    - \(\text{and } \mu: \mu pp \in \text{SHOmsgVectors } Ate-M (\text{Suc } r' + k) \ pp (\rho (\text{Suc } r' + k))\)
      - \(\text{(HOs } (\text{Suc } r' + k) \ pp) \text{ (SHOs } (\text{Suc } r' + k) \ pp)\)
  - **by** (auto simp: \(\text{SHORun-eq}\ SP\text{nextConfig-eq})
- **moreover**
- **from** \(\mu\) **have** \(\text{HOs } (\text{Suc } r' + k) \ pp = \{q. \mu pp q \neq \text{None}\}\)
  - **by** (auto simp: \(\text{SHOmsgVectors-def}\))
  - **with** \(\text{nxt } t\)
- **have** \(E - \alpha < \text{card} \{qq. \text{sendMsg } Ate-M (\text{Suc } r' + k) \ qq \ pp (\rho (\text{Suc } r' + k) \ qq) = v\}\)
  - **ultimately**
    - **have** \(\text{mostOftenRcvd } \mu pp = \{v\}\)
      - **using** \(\text{nxt } \mu\) **by** (auto dest!: mostOftenRcvd-v)
      - **with** \(\text{xsmball}\) **show** \(\text{thesis}\) **by** auto
- **qed**

**have** \(P5a: \forall pp. \exists rr. \forall k. x \ (\rho (\text{Suc } r' + \text{Suc } k) \ pp) = v\)

**proof**

- **fix** \(pp\)
- **from** \(P4\) **obtain** \(rk\) **where**
  - \(\text{xsize: } x \ (\rho (\text{Suc } r' + \text{Suc } rk) \ pp) = v\) \(\text{is } x \ (\rho ?rr \ pp) = v\)
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 \( xrrv \) show \( x (\rho (?rr + 0) pp) = v \) by simp
next
  fix \( k \)
  assume \( \text{ih} : x (\rho (?rr + k) pp) = v \)
  obtain \( k' \) where \( \mathit{rrk} : \text{Suc } r' + k' = ?rr + k \) by auto
  show \( x (\rho (?rr + \text{Suc } k) pp) = v \)
proof (rule ccontr)
    assume \( \text{nv} : x (\rho (?rr + \text{Suc } k) pp) \neq v \)
    with \( \mathit{rrk} \) \( \text{ih} \)
    have \( x (\rho (\text{Suc } r' + \text{Suc } k') pp) \neq x (\rho (\text{Suc } r' + k') pp) \)
    by (simp add: ac-simps)
    hence \( x (\rho (\text{Suc } r' + \text{Suc } k') pp) = v \) by (rule P3)
    with \( \mathit{rrk} \) \( \text{nv} \) show False by (simp add: ac-simps)
  qed
qed
qed
qed
thus \( \exists \mathit{rr}. \forall k. x (\rho (\mathit{rr} + k) pp) = v \) by blast
qed

from \( P5a \) have \( \exists F. \forall pp k. x (\rho (F pp + k) pp) = v \) by (rule choice)
then obtain \( \mathit{R} :: (\text{Proc} \Rightarrow \text{nat}) \)
  where \( \text{imgR} : \mathit{R} \cdot (\text{UNIV} :: \text{Proc set}) \neq \{\} \)
    and \( \mathit{R} : \forall pp k. x (\rho (\mathit{R} pp + k) pp) = v \)
by blast
def \( \mathit{rr} \equiv \text{Max} (\mathit{R} \cdot \text{UNIV}) \)

have \( P5: \forall r' > rr. \forall pp. x (\rho r' pp) = v \)
proof (clarify)
  fix \( r' pp \)
  assume \( r' : r' > rr \)
  hence \( r' > R pp \) by (auto simp: \( \mathit{rr-def} \))
  then obtain \( i \) where \( r' = R pp + i \)
  by (auto dest: less-imp-Suc-add)
  with \( \mathit{R} \) show \( x (\rho r' pp) = v \) by auto
qed

from \( \text{commG} \) have \( \exists r' > rr. \text{card} (\text{SHOs } r' p \cap \text{HOs } r' p) > E \)
by (auto simp: \( \text{Ate-SHOMachine-def} \text{ Ate-commGlobal-def} \))
with \( P5 \) obtain \( r' \)
  where \( r' > rr \)
    and \( \text{card} (\text{SHOs } r' p \cap \text{HOs } r' p) > E \)
    and \( \forall pp. \text{sendMsg } \text{Ate-M } r' pp p (\rho r' pp) = v \)
by (auto simp: \( \text{Ate-SHOMachine-def} \text{ Ate-sendMsg-def} \))

141
moreover
from run obtain \(\mu p\)
where \(\text{nxt: nextState Ate-M r' p (rho r' p) \mu p (rho (Suc r') p)}\)
and \(\mu: \mu p \in SHOmsgVectors Ate-M r' p (\rho r') (HOs r' p) (SHOs r' p)\)
by (auto simp: SHORun-eq SHOnextConfig-eq)

from \(\mu\) have \(\text{card (SHOs r' p \cap HOs r' p)}\)
\[\leq \text{card \{q. \mu p q = Some (sendMessage Ate-M r' q p (rho r' q))\}}\]
by (auto simp: SHOmsgVectors-def intro: card-mono)

ultimately
have \(\text{threshold-E: card \{q. \mu p q = Some v\} > E by auto}\)
with \(\text{nxt show ?thesis}\)
by (auto simp: Ate-SHOMachine-def nextState-def Ate-nextState-def)

qed

9.8 \(\mathcal{A}_{T,E,\alpha}\) Solves Weak Consensus

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

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

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

\text{theorem ate-weak-consensus-fg:}
\text{assumes run: fg-run Ate-M rho HOs SHOs (\lambda r q. undefined)}
and \(\text{commR: \forall r. SHOcommPerRd Ate-M (HOs r) (SHOs r)}\)
and \(\text{commG: SHOcommGlobal Ate-M HOs SHOs}\)
\text{shows weak-consensus \((\lambda p. x (state (rho 0)))\) decide (state \circ rho)}\n(is weak-consensus ?inits - -)

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

qed
10 Verification of the $EIGByz_f$ Consensus Algorithm

Lynch [12] presents $EIGByz_f$, a version of the exponential information gathering algorithm tolerating Byzantine faults, that works in $f$ rounds, and that was originally introduced in [1].

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

```haskell
theory EigbyzDefs
imports ../HOModel
begin

10.1 Tree Data Structure

The algorithm relies on propagating information about the initially proposed values among all the processes. This information is stored in trees whose branches are labeled by lists of (distinct) processes. For example, the interpretation of an entry $[p, q] \mapsto \text{Some } v$ is that the current process heard from process $q$ that it had heard from process $p$ that its proposed value is $v$. The value initially proposed by the process itself is stored at the root of the tree.

We introduce the type of labels, which encapsulate lists of distinct process identifiers and whose length is at most $f+1$.

```haskell
definition Label = \{xs::Proc list. length xs \leq Suc f \land distinct xs\}
```

```haskell
typedef Label = Label
by (auto simp: Label-def intro: exI[where x= []]) — the empty list is a label
```

There is a finite number of different labels.

```haskell
lemma finite-Label: finite Label
```
proof
  have Label ⊆ \{ xs. set xs ⊆ (UNIV::Proc set) \land length xs ≤ Suc f \}
    by (auto simp: Label-def)
  moreover
  have finite \{ xs. set xs ⊆ (UNIV::Proc set) \land length xs ≤ Suc f \}
    by (rule finite-lists-length-le) auto
  ultimately
  show ?thesis by (auto elim: finite-subset)
qed

lemma finite-UNIV-Label: finite (UNIV::Label set)
proof
  from finite-Label have finite (Abs-Label ' Label) by simp
  moreover
  { fix l::Label
    have l ∈ Abs-Label ' Label
      by (rule Abs-Label-cases) auto
  }
  hence (UNIV::Label set) = (Abs-Label ' Label) by auto
  ultimately show ?thesis by simp
qed

lemma finite-Label-set [iff]: finite (S :: Label set)
  using finite-UNIV-Label by (auto intro: finite-subset)

Utility functions on labels.

definition root-node where
  root-node ≡ Abs-Label []

definition length-lbl where
  length-lbl l ≡ length (Rep-Label l)

lemma length-lbl [intro]: length-lbl l ≤ Suc f
  unfolding length-lbl-def using Label-def Rep-Label by auto

definition is-leaf where
  is-leaf l ≡ length-lbl l = Suc f

definition last-lbl where
  last-lbl l ≡ last (Rep-Label l)

definition butlast-lbl where
  butlast-lbl l ≡ 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 \in set-lbl l \}

10.2 Model of the Algorithm

The following record models the local state of a process.

record 'val pstate =
vals :: Label ⇒ 'val option
newvals :: Label ⇒ 'val
decide :: 'val option

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

definition EIG-initState where
EIG-initState p st ≡
(∀ l. (vals st l = None) = (l ≠ root-node))
∧ decide st = None

type-synonym 'val Msg = Label ⇒ 'val option

At every round, every process sends its current vals tree to all processes. In fact, only the level of the tree corresponding to the round number is used (cf. definition of extend-vals below).

definition EIG-sendMsg where
EIG-sendMsg r p q st ≡ vals st

During the first \( f - 1 \) rounds, every process extends its tree vals according to the values received in the round. No decision is taken.

definition extend-vals where
extend-vals r p st msgs st' ≡ 
vals st' = (λ l.
if length-lbl l = Suc r ∧ msgs (last-lbl l) ≠ None
then (the (msgs (last-lbl l))) (butlast-lbl l)
else if length-lbl l = Suc r ∧ msgs (last-lbl l) = None then None
else vals st l)

definition next-main where
next-main r p st msgs st' ≡ extend-vals r p st msgs st' ∧ decide st' = None

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

\begin{verbatim}
fun fixupval :: 'val option ⇒ 'val where
  fixupval None = undefined
  | fixupval (Some v) = v
\end{verbatim}

\begin{verbatim}
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
\end{verbatim}

\begin{verbatim}
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))
\end{verbatim}

\begin{verbatim}
definition next-end where
  next-end r p st msgs st' ≡
  extend-vals r p st msgs st'
  ∧ check-newvals st'
  ∧ decide st' = Some (newvals st' root-node)
\end{verbatim}

The overall next-state relation is defined such that every process applies nextMain during rounds 0, \ldots, f\textminus1, and applies nextEnd during round f. After that, the algorithm terminates and nothing changes anymore.

\begin{verbatim}
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)
\end{verbatim}

### 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.

\begin{verbatim}
definition SKr :: Proc HO ⇒ Proc HO ⇒ Proc set where
  SKr HO SHO ≡ { q . ∀ p. q ∈ HO p ∩ SHO p}
\end{verbatim}

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 \textit{f} rounds really matter, since the algorithm terminates after that.

\begin{verbatim}
definition SK :: (nat ⇒ Proc HO) ⇒ (nat ⇒ Proc HO) ⇒ Proc set where
  SK HOs SHOs ≡ {q, r. q ∈ SKr (HOs r) (SHOs r)}
\end{verbatim}
The round-by-round predicate requires that the secure kernel at every round contains more than \((N+f) \div 2\) processes.

**definition** \textit{EIG-commPerRd} where
\[
EIG-commPerRd \ HO \ SHO \equiv \text{card} (SKr \ HO \ SHO) > (N + f) \div 2
\]

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** \textit{EIG-commGlobal} where
\[
EIG-commGlobal \ HOs \ SHOs \equiv \text{card} (SK \ HOs \ SHOs) \geq N - f
\]

The above communication predicates differ from Lynch’s presentation of \textit{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 \textit{EIG-commGlobal} is satisfied. The standard correctness proof assumes that \(N > 3f\), and therefore \(N - f > (N + f) \div 2\). Since moreover, for any \(r\), we obviously have
\[
\left( \bigcap_{p \in \Pi, r' \in \mathbb{N}} SHO(p, r') \right) \subseteq \left( \bigcap_{p \in \Pi} SHO(p, r) \right),
\]

it follows that any execution of \textit{EIGByz} \(_f\) where \(N > 3f\) also satisfies \textit{EIG-commPerRd} at any round. The standard correctness hypotheses thus imply our communication predicates.

However, our proof shows that \textit{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 \textit{EIG-commPerRd} always holds. Also, \textit{EIG-commGlobal} is satisfied because there are three processes from which every process receives the correct messages at all rounds. By our correctness proof, it follows that \textit{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 \textit{EIGByz} \(_f\) Heard-Of Machine

We now define the non-coordinated SHO machine for \textit{EIGByz} \(_f\) by assembling the algorithm definition and its communication-predicate.

**definition** \textit{EIG-SHOMachine} where
\[
\text{EIG-SHOMachine} = (\lambda p \ st \ \text{crd. EIG-initState} \ p \ st),
\]
sendMsg = EIG-sendMsg,
CnextState = (λ r p st msgs st’. EIG-nextState r p st msgs st’),
SHOcommPerRd = EIG-commPerRd,
SHOcommGlobal = EIG-commGlobal

abbreviation EIG-M ≡ (EIG-SHOMachine::(Proc, 'val pstate, 'val Msg) SHOMachine)

end

theory EigbyzProof
imports EigbyzDefs ../Majorities ../Reduction
begin

10.5 Preliminary Lemmas

Some technical lemmas about labels and trees.

lemma not-leaf-length:
  assumes l: ¬(is-leaf l)
  shows length-lbl l ≤ f
  using l length-lbl[of l] by (simp add: is-leaf-def)

lemma nil-is-Label: [] ∈ Label
  by (auto simp: Label-def)

lemma card-set-lbl: card (set-lbl l) = length-lbl l
  unfolding set-lbl-def length-lbl-def
  using Rep-Label[of l, unfolded Label-def]
  by (auto elim: distinct-card)

lemma Rep-Label-root-node [simp]: Rep-Label root-node = []
  using nil-is-Label by (simp add: root-node-def Abs-Label-inverse)

lemma root-node-length [simp]: length-lbl root-node = 0
  by (simp add: length-lbl-def)

lemma root-node-not-leaf: ¬(is-leaf root-node)
  by (simp add: is-leaf-def)

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

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

The label of a child is well-formed.

lemma Rep-Label-append:
  assumes l: ¬(is-leaf l)
  shows (Rep-Label l @ [p] ∈ Label) = (p ∉ set-lbl l)
    (is ?lhs = ?rhs is (?l' ∈ -) = -)
proof
  assume lhs: ?lhs thus ?rhs
  by (auto simp: Label-def set-lbl-def)
next
  assume p: ?rhs
  from l[THEN not-leaf-length] have length ?l' ≤ Suc f
    by (simp add: length-lbl-def)
moreover
  from Rep-Label[of l] have distinct (Rep-Label l)
    by (simp add: Label-def)
  with p have distinct ?l' by (simp add: set-lbl-def)
ultimately
  show ?lhs by (simp add: Label-def)
qed

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

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

The root node is never a child.

lemma children-not-root:
  assumes root-node ∈ 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 \( \text{label-children[OF assms]} \) by (auto simp: root-node-def)

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

**Lemma root-iff-no-child**: \( (l = \text{root-node}) = (\forall l'. l \notin \text{children } l') \) proof assume \( l = \text{root-node} \) thus \( \forall l'. l \notin \text{children } l' \) by (auto elim: children-not-root) next assume rhs: \( \forall l'. l \notin \text{children } l' \) show \( l = \text{root-node} \) proof (rule rev-exhaust[of \( \text{Rep-Label } l \)]) assume \( \text{Rep-Label } l = [] \) hence \( \text{Rep-Label } l = \text{Rep-Label } \text{root-node} \) by simp thus \( ?\text{thesis} \) by (simp only: Rep-Label-inject) next fix \( l' \ q \) assume \( l': \text{Rep-Label } l = l' @ [q] \) let \( ?l' = \text{Abs-Label } l' \) from \( \text{Rep-Label[of } l \} \) have \( l' \in \text{Label} \) by (simp add: Label-def) hence \( \text{repl' = Rep-Label } ?l' = l' \) by (rule Abs-Label-inverse) from \( \text{Rep-Label[of } l \} \) have \( l' @ [q] \in \text{Label} \) by (simp add: Label-def) with \( ?l' \) have \( \text{Rep-Label } l = \text{Rep-Label } (\text{Abs-Label } (l' @ [q])) ) by (simp add: Abs-Label-inverse) hence \( l = \text{Abs-Label } (l' @ [q]) \) by (simp add: Rep-Label-inject) moreover from \( \text{Rep-Label[of } l \} \) have \( \text{length } l' < \text{Suc } q \notin \text{set } l' \) by (auto simp: Label-def) moreover note repl' ultimately have \( l \in \text{children } ?l' \) by (auto simp: children-def is-leaf-def length-lbl-def set-lbl-def) with rhs show \( ?\text{thesis} \) by blast qed qed

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

**Lemma children-last-set**: assumes \( l: \neg(\text{is-leaf } l) \) 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 \( c: \text{Abs-Label } (\text{Rep-Label } l \ @ \ [p]) \in \text{children } l \)
  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.

lemma \( \text{last-lbl-inj-on-children: inj-on last-lbl } (\text{children } l) \)
proof (auto simp: inj-on-def)
  fix \( c \ c' \)
  assume \( c: c \in \text{children } l \) \( \text{and } c': c' \in \text{children } l \)
  and \( \text{eq: } \text{last-lbl } c = \text{last-lbl } c' \)
  from \( c \ c' \) obtain \( p \ p' \)
    where \( p: \text{Rep-Label } c = \text{Rep-Label } l \ @ \ [p] \)
    and \( p': \text{Rep-Label } c' = \text{Rep-Label } l \ @ \ [p'] \)
    by (auto dest!: label-children)
  from \( p \ p' \) eq have \( p = p' \) by (simp add: last-lbl-def)
  with \( p \ p' \) have \( \text{Rep-Label } c = \text{Rep-Label } c' \) 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 \( l \).

lemma \( \text{card-children:} \)
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 \( \text{card } (\text{UNIV} - \text{set-lbl } l) = \text{card } (\text{UNIV::Proc set}) - \text{card } (\text{set-lbl } l) \)
  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
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 $l'$: $l' \in$ children $l$
and shop: last-lbl $l'$ $\in$ SHOs (length-lbl $l$) $p$ $\cap$ HOs (length-lbl $l$) $p$
(is $\exists q \in$ SHOs ($\exists$len $l$) $p$ $\cap$ -)

**shows** vals (rho (\exists len $l$) $p$) $l'$ = vals (rho (\exists len $l$) $q$) $l$

**proof**

let $\exists r = \exists$len $l$
from run obtain $\mu p$
where nxt: nextState EIG-M $\exists r$ $p$ (rho $\exists r$ $p$) $\mu p$ (rho (Suc $\exists r$) $p$)
and mnu: $\mu p \in$ SHOmsgVectors EIG-M $\exists r$ $p$ (rho $\exists r$) (HOs $\exists r$ $p$) (SHOs $\exists r$ $p$)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with shop
have msl: $\exists q = \exists$ Some (vals (rho $\exists r$ $\exists q$))
by (auto simp: EIG-SHOMachine-def EIG-sendMsg-def SHOmsgVectors-def)
from nxt length-lbl[l of $l'$] children-length[OF $l'$]
have extend-vals $\exists r$ $p$ (rho $\exists r$ $p$) $\mu p$ (rho (Suc $\exists r$) $p$)
by (auto simp: EIG-SHOMachine-def nextState-def EIG-nextState-def
next-main-def next-end-def extend-vals-def)
with msl $l'$ show $\exists$thesis
by (auto simp: extend-vals-def children-length children-butlast-lbl)

qed

A process fixes the value vals $l$ of a label at state length-lbl $l$, and then never modifies the value.

**lemma** keep-vals:

**assumes** run: SHORun EIG-M rho HOs SHOs

**shows** vals (rho (length-lbl $l$ + $n$) $p$) $l$ = vals (rho (length-lbl $l$) $p$) $l$
(is $\exists v n = \exists vl$)

**proof** (induct $n$

show $\exists v 0 = \exists vl$ by simp

next

fix $n$
assume ih: $\exists v n = \exists vl$
let $\exists r = length-lbl l$ + $n$
from run obtain $\mu p$
where nxt: nextState EIG-M $\exists r$ $p$ (rho $\exists r$ $p$) $\mu p$ (rho (Suc $\exists r$) $p$)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq)
with ih show $\exists v (Suc n) = \exists vl$
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 \( \text{skr: last-lbl } l' \in SKr (HOs (length-lbl l)) (SHOs (length-lbl l)) \)

**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 \( \text{skr: last-lbl } l \in SKr (HOs (length-lbl t)) (SHOs (length-lbl t)) \)

and \( \text{skr':last-lbl } l' \in SKr (HOs (length-lbl l)) (SHOs (length-lbl l)) \)

**shows** vals (rho (length-lbl l') p) \( l' = \) vals (rho (length-lbl l) 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 \( \text{length-lbl } l \).

**lemma lynch-6-16-c:**

**assumes** commR: EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))

(is EIG-commPerRd (HOs ?r) -) \( - \)

and \( l \): \( \neg (\text{is-leaf } l) \)

**shows** card \( \{ l' \in \text{children } l. \text{last-lbl } l' \in SKr (HOs ?r) (SHOs ?r) \} > \text{card (children } l) \text{ div 2} \)

(is card ?lhs > -)

**proof**

let \( \text{?skr } = SKr (HOs ?r) (SHOs ?r) \)

**have** last-lbl \( ?\text{lhs } = \text{?skr } - \text{set-lbl } l \)

**proof**

from children-last-set[OF l]

**show** last-lbl \( ?\text{lhs } \subseteq \text{?skr } - \text{set-lbl } l \)

by (auto simp: children-length)

next

{ \( \text{fix } p \)

**assume** \( p \): \( p \in \text{?skr } p \not\in \text{set-lbl } l \)

**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 \( \text{?skr - set-lbl l} \subseteq \text{last-lbl ' ?lhs} \) by auto

qed

moreover
from \( \text{last-lbl-inj-on-children[of l]} \)
have \( \text{inj-on last-lbl ?lhs} \)
by (auto simp: inj-on-def)

ultimately
have \( \text{card ?lhs} = \text{card (set-lbl l)} \) by (auto dest: card-image)

also have \( \ldots \geq (\text{card ?skr}) - (\text{card (set-lbl l)} \) by auto

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 newvals decoration at round \( f + 1 \), then \( l \) itself is decorated with that same value.

lemma newvals-skr-uniform:
assumes run: \( \text{SHORun EIG-M rho HOs SHOs} \)
and commR: \( \text{EIG-commPerRd (HOs (length-lbl l)) (SHOs (length-lbl l))} \)
and notleaf: \( \neg (\text{is-leaf l}) \)
and unif: \( \forall l'. \exists l' \in \text{children l}, \text{last-lbl l'} \in \text{SKr (HOs (length-lbl l)) (SHOs (length-lbl l))} \)
\( \implies \text{newvals (rho (Suc f) p) l'} = v \)
shows newvals \( \text{(rho (Suc f) p) l} = v \)

proof

from unif
have \( \text{card \{l' \in children l. last-lbl l' \in SKr (HOs ?r) (SHOs ?r)\} \leq card \{l' \in children l. newvals (rho (Suc f) p) l' = v\}} \)
by (auto intro: card_mono)

with lynch-6-16-c[of HOs l SHOs, OF commR notleaf]
have maj: \( \text{has-majority v (newvals (rho (Suc f) p)) (children l)} \)
by (simp add: has-majority-def)

from run have \( \text{check-newvals (rho (Suc f) p)} \)
A node whose label \( l \) ends with a process which is well heard at round \( \text{length-lbl} \ l \) will have its \( \text{newvals} \) field set (at round \( f + 1 \)) to the “fixed-up” value given by \( \text{vals} \).

**Lemma lynch-6-16-d:**

**Assumes**
- \( \text{run} : \text{SHORun} \ EIG-M \ rho \ HOs \ SHOs \)
- \( \text{commR} : \forall r. \ EIG\text{-commPerRd} (\text{HOs} \ r) (\text{SHOs} \ r) \)
- \( \text{notroot} : l \in \text{children} \ t \)
- \( \text{skr} : \text{last-lbl} \ l \in \text{SKr} (\text{HOs} (\text{length-lbl} \ t)) (\text{SHOs} (\text{length-lbl} \ t)) \)

**Shows**
- \( \text{newvals} (\text{rho} (\text{Suc} \ f) \ p) \ l = \text{fixupval} (\text{vals} (\text{rho} (\text{Suc} \ f) \ p) \ l) \)
- \( (\text{is} \ ?P \ l) \)

**Using**
- \( \text{notroot} \ \text{skr} \)

**Proof** (induct Suc \( f - (?\text{len} \ l) \) arbitrary: \( l \ t \))

**Fix** \( l \ t \)

**Assume** \( 0 = \text{Suc} \ f - (?\text{len} \ l) \)

**With** \( \text{length-lbl}[\text{of} \ l] \) have \( \text{leaf} \) \( \text{is-leaf} \ l \) by (simp add: \text{is-leaf-def})

**From** \( \text{run} \) have \( \text{check-newvals} (\text{rho} (\text{Suc} \ f) \ p) \)

**By** (auto simp: \text{EIG-SHOMachine-def} \text{SHORun-eq} \text{SHOnextConfig-eq}
\text{nextState-def} \text{EIG-nextState-def} \text{next-end-def})

**With** \( \text{leaf} \) show \( ?P \ l \)

**By** (auto simp: \text{check-newvals-def} \text{is-leaf-def})

**Next**

**Fix** \( k \ l \ t \)

**Assume** \( i_h : \bigwedge l' \ t'. \)
\[
[k = \text{Suc} \ f - \text{length-lbl} \ l'; l' \in \text{children} \ t'; \ \text{last-lbl} \ l' \in \text{SKr} (\text{HOs} (?\text{len} \ t')) (\text{SHOs} (?\text{len} \ t'))] \implies ?P \ l'
\]

**And** \( flk : \text{Suc} \ k = \text{Suc} \ f - ?\text{len} \ l \)

**And** \( \text{notroot} : l \in \text{children} \ t \)

**And** \( \text{skr} : \text{last-lbl} \ l \in \text{SKr} (\text{HOs} (?\text{len} \ t)) (\text{SHOs} (?\text{len} \ t)) \)

**Let** \( ?v = \text{fixupval} (\text{vals} (\text{rho} (?\text{len} \ l) \ p) \ l) \)

**From** \( flk \) have \( \text{notlf} : \neg(\text{is-leaf} \ l) \) by (simp add: \text{is-leaf-def})

\[
\{ \\
\text{fix} \ l' \\
\text{assume} \ l': l' \in \text{children} \ l \\
\text{and} \ skr': \text{last-lbl} \ l' \in \text{SKr} (\text{HOs} (?\text{len} \ l)) (\text{SHOs} (?\text{len} \ l))
\}
from run notroot skr l skr'
have vals (rho (?len l') p) l' = vals (rho (?len l) p) l
  by (rule lynch-6-16-a)
moreover
from flk l' have k = Suc f - ?len l' by (simp add: children-length)
from this l' skr' have ?P l' by (rule ih)
ultimately
have newvals (rho (Suc f) p) l' = ?v
  using notroot l' by (simp add: children-length)
}

with run commR notlf show ?P l by (auto intro: newvals-skr-uniform)
qed

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

A label is common if all processes agree on the final value it is decorated with.

definition common where
common rho l ≡
  ∀ p q. newvals (rho (Suc f) p) l = newvals (rho (Suc f) q) l

The subtrees of a given label are all its possible extensions.

definition subtrees where
subtrees h ≡ { l. ∃ t. Rep-Label l = (Rep-Label h) @ t }

lemma children-in-subtree:
assumes l ∈ children h
shows l ∈ subtrees h
using label-children[OF assms] by (auto simp: subtrees-def)

lemma subtrees-refl [iff]: l ∈ subtrees l
by (auto simp: subtrees-def)

lemma subtrees-root [iff]: l ∈ subtrees root-node
by (auto simp: subtrees-def)

lemma subtrees-trans:
assumes l'' ∈ subtrees l' and l' ∈ subtrees l
shows l'' ∈ subtrees l
using assms by (auto simp: subtrees-def)

lemma subtrees-antisym:
assumes l ∈ subtrees l' and l' ∈ subtrees l
shows l' = l
using assms by (auto simp: subtrees-def Rep-Label-inject)

lemma subtrees-tree:
assumes \( l' : l \in \text{subtrees} \, l' \) and \( l'' : l \in \text{subtrees} \, l'' \)
shows \( l' \in \text{subtrees} \, l'' \lor l'' \in \text{subtrees} \, l' \)
using assms proof (auto simp: subtrees-def append-eq-append-conv2)

fix \( xs \)
assume \( \text{Rep-Label} \, l'' \, @ \, xs = \text{Rep-Label} \, l' \)

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

lemma subtrees-cases:
assumes \( l' : l'' : l' \in \text{subtrees} \, l' \)
and self : \( l' = l \implies P \)
and 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: Rep-Label-inject)
  next
    fix \( p \, t' \)
    assume \( \text{cons} : \, t = p \# \, t' \)
    from \( \text{Rep-Label} \, t : \, t \) have \( \text{length} \, (\text{Rep-Label} \, l \, @ \, t) \leq \text{Suc} \, f \)
    by (simp add: Label-def)

    with \( \text{cons} \) have \( \text{notleaf} : \, \neg (\text{is-leaf} \, l) \)
    by (auto simp: is-leaf-def length-lbl-def)

    let \( ? \, c = \text{Abs-Label} \, (\text{Rep-Label} \, t \, @ \, p) \)
    from \( \text{t \, cons \, Rep-Label} \, t \, @ \, p \) have \( \, p \notin \text{set-lbl} \, l \)
    by (auto simp: Label-def set-lbl-def)

    with \( \text{notleaf} \) have \( \, c \, : \, ? \, c \, \in \text{children} \, l \)
    by (auto simp: children-def)

    moreover
    from \( \text{notleaf} \, p \) have \( \text{Rep-Label} \, l \, @ \, p \in \text{Label} \)
    by (simp add: Rep-Label-append)

    hence \( \text{Rep-Label} \, ? \, c \, = \, (\text{Rep-Label} \, l \, @ \, p) \)
    by (simp add: Abs-Label-inverse)

    with \( \text{cons} \) have \( \text{t} \, \in \text{subtrees} \, \? \, c \)
    by (auto simp: subtrees-def)

    ultimately show \( \text{thesis} \) by blast
  qed
  thus \( \text{thesis} \) by (auto elim!: self child)
qed

lemma subtrees-leaf:
assumes \( l : \, \text{is-leaf} \, l \) and \( l' : \, l' \in \text{subtrees} \, l \)
shows \( l' = l \)

qed
using \( l' \) proof (rule subtrees-cases)

fix \( c \)

assume \( c \in \text{children } l \) — impossible

with \( l \) show \( ?\text{thesis} \) by (simp add: children-def)

qed

lemma children-subtrees-equal:
assumes \( c : c \in \text{children } l \) and \( c' : c' \in \text{children } l \)
and \( \text{sub} : c' \in \text{subtrees } c \)
shows \( c' = c \)

proof —
from assms have \( \text{Rep-Label } c' = \text{Rep-Label } c \)
by (auto simp: subtrees-def dest: label-children)
thus \( ?\text{thesis} \) by (simp add: Rep-Label-inject)

qed

A set \( C \) of labels is a 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

\[
\text{subcovering } C \ l \equiv \\
\forall s \in \text{subtrees } l. \ \text{is-leaf } s \longrightarrow (\exists h \in C. \ h \in \text{subtrees } l \land s \in \text{subtrees } h)
\]

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 \( EIG-M \rho \text{ HO} s \text{ SH} o s \)
and \( \forall r. \ \text{EIG-commPerRd } (\text{HO} s \ r) (\text{SH} o s \ r) \)
and \( l \in \text{children } t \)
and \( \text{last-lbl } l \in \text{SK} r (\text{HO} s (\text{length-lbl } t)) (\text{SH} o s (\text{length-lbl } t)) \)
shows \( \text{common } \rho \ l \)

using assms
by (auto simp: common-def lynch-6-16-d lynch-6-15
intro: arg-cong[where \( f = \text{fixupval} \)]

lemma lynch-6-18-b:
assumes ran: \( \text{SHORun } EIG-M \rho \text{ HO} s \text{ SH} o s \)
and \( \text{commG: EIG-commGlobal } \text{HO} s \text{ SH} o s \)
and \( \text{commR: } \forall r. \ \text{EIG-commPerRd } (\text{HO} s \ r) (\text{SH} o s \ r) \)
shows covering \( \{ l. \ \exists t. \ l \in \text{children } t \land \text{last-lbl } l \in (\text{SK } \text{HO} s \text{ SH} o s) \} \)

proof (clarsimp simp: subcovering-def)
fix \( l \)
assume is-leaf \( l \)
with 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 \( \exists q \in \text{set-lbl } l . \ q \in \text{SK HOs SHOs} \)
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 \ have \ l_1 \ @ \ [q] \in \text{Label} \)
by (simp add: Label-def)

hence \( \text{length-lbl } ?h \neq 0 \) by (simp add: length-lbl-inverse)

hence \( ?h \neq \text{root-node} \) by auto

then obtain \( t \) where \( \exists t \mid ?h \in \text{children } t \)
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 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 \( s \) children-in-subtree[OF \ e] have \( s \in \text{subtrees } l \)
by (rule subtrees-trans)

with leaf cov obtain \( h \) where \( h. \ h \in C \ l \in \text{subtrees } h \)
by (auto simp: subcovering-def)

with \( l \) obtain \( e' \) where \( e'. \ e' \in \text{children } l \ l \in \text{subtrees } e' \)
by (auto elim: subtrees-cases)

from \( s \in \text{subtrees } h \) \( \exists h \in \text{subtrees } e' \) 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**
  - run: $\text{SHORun EIG-M rho HOs SHOs}$
  - and cov: subcovering $C l$
  - and com: $\forall l' \in C. \text{common rho } l'$
  - shows common rho $l$

- **Using cov proof** (induct $Suc f \ - \ \text{length-lbl } l$ arbitrary; $l$)

  - **Fix** $l$
    - **Assume** $0: 0 = Suc f \ - \ \text{length-lbl } l$
      - and $C$: subcovering $C l$
      - **From** $0 \ \text{length-lbl}[of } l] \ \text{have is-leaf } l$ by (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 (auto simp: subcovering-def)
      - **Hence** $l \in C$ by (auto dest: subtreess-antisym)
      - **With** com show common rho $l$ ..
  - **Next**
    - **Fix** $k \ l$
      - **Assume** $k: Suc k = Suc f \ - \ \text{length-lbl } l$
        - and $C$: subcovering $C l$
        - and $ih: \lceil \lceil k = Suc f \ - \ \text{length-lbl } l'; \ \text{subcovering } C l'\rceil \implies \text{common rho } l'$
      - **Show** common rho $l$
      - **Proof** (cases $l \in C$)
        - case True
          - with com show ?thesis ..
        - **Next**
          - case False
            - with $C$ have $\forall e \in \text{children } l. \ \text{subcovering } C e$
              - by (blast intro: lynch-6-19-a)
            - **Moreover**
              - from $k$ have $\forall e \in \text{children } l. \ k = Suc f \ - \ \text{length-lbl } e$
                - by (auto simp: children-length)
            - **Ultimately**
              - have com-ch: $\forall e \in \text{children } l. \ \text{common rho } e$
                - by (blast intro: ih)
        - **Show** ?thesis
      - **Proof** (clarsimp simp: common-def)
        - **Fix** $p \ q$
          - from $k$ have notleaf: $\neg(is-leaf \ l)$ by (simp add: is-leaf-def)
          - let $?r = Suc f$
          - from com-ch
            - have $\forall e \in \text{children } l. \ \text{newvals } (\text{rho } $?r p) e = \text{newvals } (\text{rho } $?r q) e$
              - by (auto simp: common-def)
            - **Hence** $\forall w. \ \{e \in \text{children } l. \ \text{newvals } (\text{rho } $?r p) e = w\}$
              = $\{e \in \text{children } l. \ \text{newvals } (\text{rho } $?r q) e = w\}$
                - by auto
            - **Moreover**
from run
have check-newvals (rho ?r p) check-newvals (rho ?r q)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq nextState-def
EIG-nextState-def next-end-def)

with notleaf have
(∃ w. has-majority w (newvals (rho ?r p)) (children l)
 ∧ newvals (rho ?r p) l = w)
∨ ¬(∃ w. has-majority w (newvals (rho ?r p)) (children l))
 ∧ newvals (rho ?r p) l = undefined
(∃ w. has-majority w (newvals (rho ?r q)) (children l)
 ∧ newvals (rho ?r q) l = w)
∨ ¬(∃ w. has-majority w (newvals (rho ?r q)) (children l))
 ∧ newvals (rho ?r q) l = undefined
by (auto simp: check-newvals-def)
ultimately show newvals (rho ?r p) l = newvals (rho ?r q) l
by (auto simp: has-majority-def elim: abs-majoritiesE')
qed

The root of the tree is a common node.

lemma lynch-6-20:
assumes run: SHORun EIG-M rho HOs SHOs and commG: EIG-commGlobal HOs SHOs and commR: ∀ r. EIG-commPerRd (HOs r) (SHOs r)
shows common rho root-node
using run lynch-6-18-b[OF assms]
proof (rule lynch-6-19-b, clarify)
fix l t
assume l ∈ children t last-lbl l ∈ SK HOs SHOs
thus common rho l by (auto simp: SK-def elim: lynch-6-18-a[OF run commR])
qed

A decision is taken only at state f + 1 and then stays stable.

lemma decide:
assumes run: SHORun EIG-M rho HOs SHOs
shows decide (rho r p) =
(if r < Suc f then None
else Some (newvals (rho (Suc f) p) root-node))
is ?P r)
proof (induct r)
from run show ?P 0
by (auto simp: EIG-SHOMachine-def SHORun-eq HOinitConfig-eq
initState-def EIG-initState-def)

next
fix r
assume ih: ?P r
from run obtain μp
where EIG-nextState r p (rho r p) μp (rho (Suc r) p)
by (auto simp: EIG-SHOMachine-def SHORun-eq SHOnextConfig-eq nextState-def)

thus \( ?P (\text{Suc } r) \)

proof (auto simp: EIG-nextState-def next-main-def next-end-def)

assume \( \neg (r < f) \land r \neq f \)

with \( \text{ih} \)

show \( \text{decide } (\rho r p) = \text{Some } (\text{newvals } (\rho (\text{Suc } f) p) \text{ root-node}) \)

by simp

qed

qed

10.7 Proof of Agreement, Validity, and Termination

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

**theorem Agreement:**

assumes run: SHORun EIG-M \( \rho \) HOs SHOs

and \( \text{commG}: \text{EIG-commGlobal } \text{HOs SHOs} \)

and \( \text{commR}: \forall r. \text{EIG-commPerRd } (\text{HOs } r) (\text{SHOs } r) \)

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

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

shows \( v = w \)

using \( p \ q \text{ lynch-6-20}[\text{OF run commG commR}] \)

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-val, 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 \( \text{newvals} \). Finally, lemma \( \text{newvals-skr-uniform} \) lets us conclude.

**theorem Validity:**

assumes run: SHORun EIG-M \( \rho \) HOs SHOs

and \( \text{commR}: \forall r. \text{EIG-commPerRd } (\text{HOs } r) (\text{SHOs } r) \)

and \( \text{initv}: \forall q. \text{the } (\text{vals } (\rho 0 q) \text{ root-node}) = v \)

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

shows \( v = w \)

proof

have \( v: \forall q. \text{vals } (\rho 0 q) \text{ root-node} = \text{Some } v \)

proof

fix \( q \)

from \( \text{run have vals } (\rho 0 q) \text{ root-node} \neq \text{None} \)

by (auto simp: EIG-SHOMachine-def SHORun-eq HOinitConfig-eq initState-def EIG-initState-def)

then obtain \( w \) where \( w: \text{vals } (\rho 0 q) \text{ root-node} = \text{Some } w \)

by auto

from \( \text{initv have the } \text{(vals } (\rho 0 q) \text{ root-node}) = v \)

with \( w \) show \( \text{vals } (\rho 0 q) \text{ root-node} = \text{Some } v \) by simp
qed

let \( ?\text{len} = \text{length-lbl} \)
let \( ?r = \text{Suc } f \)

\[
\begin{align*}
&\{ \\
&\text{fix } l' \\
&\text{assume } l': l' \in \text{children root-node} \\
&\quad \text{and skr: last-lbl } l' \in \text{SKr } (\text{HOs 0 }) (\text{SHOs 0 }) \\
&\text{with run } v \ \text{have vals } (\text{rho } (\ ?\text{len } l') p) l' = \text{Some } v \\
&\quad \text{by (auto dest: sho-correct-vals simp: SKr-def)} \\
&\text{moreover} \\
&\text{from run commR } l' \text{ skr} \\
&\text{have newvals } (\text{rho } ?r p) l' = \text{fixupval } (\text{vals } (\text{rho } (\ ?\text{len } l') p) l') \\
&\quad \text{by (auto intro: lynch-6-16-d)} \\
&\text{ultimately} \\
&\text{have newvals } (\text{rho } ?r p) l' = v \text{ by simp} \\
&\} \\
\text{with run commR root-node-not-leaf} \\
\text{have newvals } (\text{rho } ?r p) \text{ root-node } = v \\
\quad \text{by (auto intro: newvals-skr-uniform)} \\
\text{with dp show } ?\text{thesis by (simp add: decide[OF run])} \\
\end{align*}
\]

qed

Termination is trivial for \( \text{EIGByz}_f \).

**Theorem** Termination:

**Assumes** \( \text{SHORun } \text{EIG-M } \text{rho } \text{HOs SHOs} \)

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

**Using** \( \text{assms by (auto simp: decide)} \)

### 10.8 \( \text{EIGByz}_f \) Solves Weak Consensus

Summing up, all (coarse-grained) runs of \( \text{EIGByz}_f \) for HO and SHO collections that satisfy the communication predicate satisfy the Weak Consensus property.

**Theorem** \( \text{eig-weak-consensus} \):

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

**And** \( \text{commR: } \forall r. \ EIG\text{-commPerRd } (\text{HOs } r) (\text{SHOs } r) \)

**And** \( \text{commG: } \text{EIG\text{-commGlobal } HOs SHOs} \)

**Shows** \( \text{weak-consensus } (\lambda p. \ \text{the } (\text{vals } (\text{rho } 0 p) \text{ root-node})) \ \text{decide } rho \)

**Unfolding** \( \text{weak-consensus-def} \)

**Using** \( \text{Validity[OF run commR]} \)

\( \text{Agreement[OF run commG commR]} \)

\( \text{Termination[OF run]} \)

**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**

- \( \text{run: fg-run EIG-M rho HOs SHOs (} \lambda r q. \text{undefined)} \)
- \( \text{and commR: } \forall r. \text{EIG-commPerRd (} \text{HOs r) (} \text{SHOs r) } \)
- \( \text{and commG: EIG-commGlobal HOs SHOs} \)

**Shows**

- \( \text{weak-consensus (} \lambda p. \text{the (vals (state (rho 0) p) root-node))} \)
- \( \text{decide (state } \circ \text{rho)} \)

(is weak-consensus ?inits - )

**Proof** (rule local-property-reduction[OF run weak-consensus-is-local])

fix crun

**Assume**

- \( \text{crun: CSHORun EIG-M crun HOs SHOs (} \lambda r q. \text{undefined)} \)
- \( \text{and init: crun 0 = state (rho 0)} \)

from crun have SHORun EIG-M crun HOs SHOs by (unfold SHORun-def)

from this commR commG

have weak-consensus (\( \lambda p. \text{the (vals (crun 0 p) root-node))} \) decide crun

by (rule eig-weak-consensus)

with init show weak-consensus ?inits decide crun

by (simp add: o-def)

qed

### 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


