Verification of Heard-Of Algorithms in Isabelle

Stephan Merz
July 30, 2009

Contents

1 Heard-Of Algorithms

2 Verification of the LastVoting Consensus Algorithm
   2.1 Formal Model of LastVoting ........................................ 5
   2.2 Proof of LastVoting: Preliminary Lemmas ....................... 8
   2.3 Boundedness and monotonicity of timestamps .................. 12
   2.4 Obvious facts about the algorithm ......................... 13
   2.5 Proof of Integrity ......................................... 19
   2.6 Proof of Agreement and Irrevocability ....................... 22
   2.7 Proof of liveness .................................... 30

theory CHO
imports Main
begin

1 Heard-Of Algorithms

We propose a generic representation of (coordinated) HO algorithms [1] in Isabelle/HOL.

An HO algorithm executes a sequence of rounds. A concrete algorithm is described by the following parameters:

- a type 'proc of processes whose extension is assumed to be finite,
- a type 'pst of local process states,
- a type 'msg of messages sent in the course of the algorithm,
- a predicate initState such that initState p st is true precisely of the initial states st of process p,
- a function sendMsg where sendMsg r p q st crd yields the message that process p sends to process q at round r, given its local state st and coordinator crd, and
- a predicate nextState where nextState r p st msgs crd st' characterizes the successor states st' of state st for process p at round r, where crd denotes the process that p believes to be the coordinator of round r and the function msgs :: 'proc ⇒ 'msg option represents the vector of messages that p received at round r,
• a communication predicate that constrains the heard-of and coordinator assignments (see below) that may occur during a run. For convenience, we split this predicate into a safety part that should hold at every round and a liveness part that should hold of the sequence of HO assignments.

An uncoordinated algorithm simply ignores the parameter crd of functions nextState and sendMsg. Similarly, the communication predicate does not refer to the coordinator assignment. The HO model assumes communication-closed rounds, that is, processes receive only messages sent for the round they are currently in. By a general result on the HO model, it can be assumed that each round is executed atomically. A snapshot of the system can therefore be represented by the local states of each process at the beginning of a round. The messages sent can be computed from the local state, so they do not have to be recorded explicitly.

We represent a system configuration as an array of process states. A system run is just an infinite sequence of configurations. At this generic level, process states are left parametric (represented by a type variable); they will be defined by particular algorithms. (For some reason type and record definitions cannot go inside locale definitions so we introduce them beforehand.)

```
types ('proc,'pst) run = nat ⇒ 'proc ⇒ 'pst

A heard-of assignment associates a set of processes with each process. The idea is that HO p designates the set of processes from which process p receives a message at the current round. A coordinator assignment associates a process (the coordinator) to each process.

types 'proc HO = 'proc ⇒ 'proc set

types 'proc coord = 'proc ⇒ 'proc

locale CHOAlgorithm =
  fixes initState :: 'proc ⇒ 'pst ⇒ bool
  and sendMsg :: nat ⇒ 'proc ⇒ 'proc ⇒ 'pst ⇒ 'proc ⇒ 'msg
  and nextState :: nat ⇒ 'proc ⇒ 'pst ⇒ ('proc ⇒ 'msg option) ⇒ 'proc ⇒ 'pst ⇒ bool
  and commSafe :: nat ⇒ 'proc HO ⇒ 'proc coord ⇒ bool
  and commLive :: (nat ⇒ 'proc HO) ⇒ (nat ⇒ 'proc coord) ⇒ bool
  assumes finiteProc: finite (UNIV::'proc set)

begin

By assumption finiteProc, any set of processes is finite.

  lemma finiteProcset [simp,intro]: finite (P::'proc set)
  using finiteProc by (blast intro:finite-subset)

Similarly, the range of any partial function from Proc is finite. (The Isabelle library contains a similar lemma for the range of a total function, a generalization of the following lemma could go to the standard library.)

  lemma finite-ran: finite (ran (f :: 'proc ⇒ 'a))

```
proof

let $?g = \lambda y. \text{case } y \text{ of } \text{None} \Rightarrow \text{arbitrary} \mid \text{Some } x \Rightarrow x$

have $\text{ran } f \subseteq ?g \cdot (\text{range } f)$

proof

fix $y$

assume $y \in \text{ran } f$

then obtain $x$ where $f x = \text{Some } y$ by (auto simp add: ran-def)

hence $y = ?g \cdot f x$ by simp

thus $y \in ?g \cdot (\text{range } f)$ by blast

qed

moreover have $\text{finite } (\text{?g } \cdot \text{range } f)$ by auto

ultimately show $\text{?thesis}$ by (rule finite-subset)

qed

Any two sets $S$ and $T$ of processes such that the sum of their cardinalities exceeds the number of processes have a non-empty intersection.

lemma majorities-intersect:

assumes $\text{crd}: \text{card } (\text{UNIV::'proc set}) < \text{card } (S::'proc set) + \text{card } (T::'proc set)$

shows $S \cap T \neq \{\}$

proof (clarify)

assume contra: $S \cap T = \{\}$

with $\text{crd}$ have $\text{card } (\text{UNIV::'proc set}) < \text{card } (S \cup T)$

by (auto simp add: card-Un-Int)

moreover have $\text{card } (S \cup T) \leq \text{card } (\text{UNIV::'proc set})$

by (simp add: card-mono)

ultimately show False

by simp

qed

lemma majoritiesE:

assumes $\text{crd}: \text{card } (\text{UNIV::'proc set}) < \text{card } (S::'proc set) + \text{card } (T::'proc set)$

obtains $p$ where $p \in S$ and $p \in T$

using $\text{crd}$ majorities-intersect by blast

Frequent special case

lemma majoritiesE':

assumes $S: \text{card } (S::'proc set) > (\text{card } (\text{UNIV::'proc set})) \text{ div } 2$

and $T: \text{card } (T::'proc set) > (\text{card } (\text{UNIV::'proc set})) \text{ div } 2$

obtains $p$ where $p \in S$ and $p \in T$

proof (rule majoritiesE)

from $S \ T$ show $\text{card } (\text{UNIV::'proc set}) < \text{card } S + \text{card } T$ by auto

qed

Because messages are not corrupted in the HO model and processes only react to messages sent at the current round, we need not explicitly represent the network state in the runs and use the following utility function to compute the messages that a process receives.

The function $\text{rcvdMsgs}$ computes the messages that process $p$ receives at round $r$, given a Heard-Of set, the collections of coordinators and process states, and a message send function. (This last parameter is useful in applications because $\text{rcvdMsgs}$ can be used with sub-functions of the overall message sending function used by the algorithm.)

definition $\text{rcvdMsgs}$ where
\[ \text{rcvdMsgs} \left( p \mapsto \text{proc} \right) \left( \text{HO} \mapsto \text{proc set} \right) \left( \text{coord} \mapsto \text{proc coord} \right) \left( \text{cfg} \mapsto \text{proc} \Rightarrow \text{pst} \right) \\
\quad \equiv \lambda q. \text{if } q \in \text{HO} \text{ then Some} \left( \text{send} q p \left( \text{cfg} q \right) \left( \text{coord} q \right) \right) \text{ else None} \]

An initial configuration is one where all processes are in an initial state.

**definition**

\[ \text{initConfig} \text{ where} \]

\[ \text{initConfig} \left( \text{cfg} \right) \equiv \forall p. \text{initState} p \left( \text{cfg} p \right) \]

The following definition characterizes successor configurations \( \text{cfg}' \) of a source configuration \( \text{cfg} \) at round \( r \), given assignments \( \text{HO} \) of heard-of sets and \( \text{coord} \) of coordinators.

**definition**

\[ \text{nextConfig} \text{ where} \]

\[ \text{nextConfig} \left( r \right) \left( \text{cfg} \right) \left( \text{HO} :: \text{proc HO} \right) \left( \text{coord} :: \text{proc coord} \right) \left( \text{cfg} \right) \equiv \forall p. \text{nextState} r p \left( \text{cfg} p \right) \left( \text{rcvdMsgs} p \left( \text{HO} p \right) \text{ coord} \left( \text{sendMsg} r \right) \left( \text{coord} p \right) \left( \text{cfg} p \right) \right) \]

Given heard-of and coordinator collections, i.e. a heard-of and coordinator assignment for each round, a run \( \rho \) of the algorithm is a sequence of configurations starting with an initial configuration and respecting the successor function \( \text{nextConfig} \).

**definition**

\[ \text{CHORun} \text{ where} \]

\[ \text{CHORun} \left( \rho \right) \left( \text{HOs} \right) \left( \text{coords} \right) \equiv \left( \text{initConfig} \left( \rho 0 \right) \right) \land \left( \forall r. \text{commSafe} r \left( \text{HOs} r \right) \left( \text{coords} r \right) \right) \land \left( \text{nextConfig} r \left( \rho r \right) \left( \text{HOs} r \right) \left( \text{coords} r \right) \left( \rho \left( \text{Suc} r \right) \right) \right) \land \text{commLive} \left( \text{HOs} \right) \left( \text{coords} \right) \]

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

**lemma ** \( \text{CHORun-0:} \)

\[ \text{assumes } \text{CHORun} \left( \rho \right) \left( \text{HOs} \right) \left( \text{coords} \right) \text{ and } \forall \text{cfg}. \text{initConfig} \left( \text{cfg} \right) \Rightarrow P \left( \text{cfg} \right) \]

\[ \text{shows } P \left( \rho 0 \right) \]

**using** \( \text{prems unfolding } \text{CHORun-def by blast} \)

**lemma ** \( \text{CHORun-Suc:} \)

\[ \text{assumes } \text{CHORun} \left( \rho \right) \left( \text{HOs} \right) \left( \text{coords} \right) \text{ and } \forall r. \left( \text{commSafe} r \left( \text{HOs} r \right) \left( \text{coords} r \right) ; \text{nextConfig} r \left( \rho r \right) \left( \text{HOs} r \right) \left( \text{coords} r \right) \left( \rho \left( \text{Suc} r \right) \right) \right) \Rightarrow P \left( r \right) \]

\[ \text{shows } P n \]

**using** \( \text{prems unfolding } \text{CHORun-def by blast} \)

**lemma ** \( \text{CHORun-induct:} \)

\[ \text{assumes } \text{run}: \text{CHORun} \left( \rho \right) \left( \text{HOs} \right) \left( \text{coords} \right) \text{ and } \text{init}: \text{initConfig} \left( \rho 0 \right) \Rightarrow P \left( 0 \right) \]

\[ \text{and step: } \forall r. \left( P \left( r \right) ; \text{commSafe} r \left( \text{HOs} r \right) \left( \text{coords} r \right) ; \text{nextConfig} r \left( \rho r \right) \left( \text{HOs} r \right) \left( \text{coords} r \right) \left( \rho \left( \text{Suc} r \right) \right) \right) \Rightarrow P \left( \text{Suc} r \right) \]

\[ \text{shows } P n \]

**using** \( \text{run unfolding } \text{CHORun-def by (induct n, auto elim: init step)} \)

end — locale CHOAlgorithm

end — theory CHO
theory LastVoting
imports CHO
begin

2 Verification of the LastVoting Consensus Algorithm

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

The LastVoting algorithm can be considered as a version of Lamport’s Paxos consensus algorithm [2] for the Heard-Of model. Following [1], we define the algorithm as an instance of the generic Heard-Of model.

2.1 Formal Model of LastVoting

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

axioms

procFinite: finite (UNIV::Proc set)

abbreviation

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

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

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

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

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

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

lemma phase-step: (phase r * 4) + step r = r
by (auto simp add: phase-def step-def)

The following record models the local state of a process.

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

Possible messages sent during the algorithm.

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

Characteristic predicates on messages.

**definition** isValStamp **where** isValStamp m ≡ ∃ 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 appropriate kind.

**fun val** **where**
val (ValStamp v ts) = v
val (Vote v) = v

**fun stamp** **where**
stamp (ValStamp v ts) = ts

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

**definition** initState **where**
initState p st ≡
(vote st = None) ∧ ¬(commit st) ∧ ¬(ready st) ∧ (timestamp st = 0) ∧ (decide st = None)

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

— processes from which values and timestamps were received

**definition** valStampsRcvd **where**
valStampsRcvd (msgs :: Proc → 'val msg) ≡
{ q . ∃ v ts. msgs q = Some (ValStamp v ts)}

**definition** highestStampRcvd **where**
highestStampRcvd msgs ≡ Max {ts . ∃ q v. (msgs::Proc → 'val msg) q = Some (ValStamp v ts)}

In step 0, each process sends its current x and timestamp values to its coordinator. A process that considers itself to be a coordinator updates its vote and commit fields if it has received messages from a majority of processes.

**definition** send0 **where**
send0 r p q st crd ≡
if q = crd then ValStamp (x st) (timestamp st) else Null

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

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

**definition** send1 **where**
send1 r p q st crd ≡
\[
\text{if } p = \text{crd} \land \text{commit } st \text{ then Vote (the (vote } st\text{)) else Null}
\]

**definition next1 where**

\[
\text{next1 } r \ p \ s t \ m s g s \ c r d \ s t' \equiv
\begin{align*}
&\text{if } m s g s \ c r d \neq \text{None } \land \text{isVote (the (msgs crd))} \\
&\text{then } s t' = s t \{ x := \text{val (the (msgs crd))}, \text{timestamp := Suc(phase } r) \} \\
&\text{else } s t' = s t
\end{align*}
\]

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

**definition send2 where**

\[
\text{send2 } r \ p \ q \ s t \ c r d \equiv
\begin{align*}
&\text{if } \text{timestamp } s t = \text{Suc(phase } r) \land q = \text{crd then (Ack::'val msg) else Null}
\end{align*}
\]

**definition acksRcvd where** — processes from which an acknowledgement was received

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

**definition next2 where**

\[
\text{next2 } r \ p \ s t \ m s g s \ c r d \ s t' \equiv
\begin{align*}
&\text{if } p = \text{crd } \land \text{card(acksRcvd msgs)} > N \text{ div } 2 \\
&\text{then } s t' = s t \{ \text{ready := True } \} \\
&\text{else } s t' = s t
\end{align*}
\]

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{commit} fields to false.

**definition send3 where**

\[
\text{send3 } r \ p \ q \ s t \ c r d \equiv
\begin{align*}
&\text{if } p = \text{crd } \land \text{ready } s t \text{ then Vote (the (vote } s t\text{)) else Null}
\end{align*}
\]

**definition next3 where**

\[
\text{next3 } r \ p \ s t \ m s g s \ c r d \ s t' \equiv
\begin{align*}
&\text{if } p = \text{crd } \land \text{card(acksRcvd msgs)} > N \text{ div } 2 \\
&\text{then decide } s t' = \text{Some (val (the (msgs crd}))} \\
&\text{else decide } s t' = \text{decide } s t \\
&\land (\text{if } p = \text{crd} \\
&\text{then } \neg(\text{ready } s t') \land \neg(\text{commit } s t') \\
&\text{else } (\text{ready } s t' = \text{ready } s t) \land (\text{commit } s t' = \text{commit } s t) \\
&\land (x s t' = x s t) \land (\text{vote } s t' = \text{vote } s t) \land (\text{timestamp } s t' = \text{timestamp } s t)
\end{align*}
\]

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

**definition sendMsg :: nat } \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{'val pstate} \Rightarrow \text{Proc} \Rightarrow \text{'val msg} where**

\[
\text{sendMsg (r::nat)} \equiv
\begin{align*}
&\text{if step } r = 0 \text{ then send0 } r \\
&\text{else if step } r = 1 \text{ then send1 } r \\
&\text{else if step } r = 2 \text{ then send2 } r \\
&\text{else send3 } r
\end{align*}
\]

**definition**

\[
\text{nextState :: nat } \Rightarrow \text{Proc} \Rightarrow \text{'val pstate} \Rightarrow \text{(Proc} \rightarrow \text{'val msg}) \Rightarrow \text{Proc} \Rightarrow \text{'val pstate} \Rightarrow \text{bool} where
\]

7
nextState \( r \) ≡ 
if \( \text{step} \ r = 0 \) then next0 \( r \) 
else if \( \text{step} \ r = 1 \) then next1 \( r \) 
else if \( \text{step} \ r = 2 \) then next2 \( r \) 
else next3 \( r \)

We now define the communication predicate for the LastVoting algorithm. The safety part is trivial: integrity and agreement are always ensured. However, coordinators are supposed to change only between phases. For the liveness part, Charron and Bost 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 [1], but obviously sufficient).

In fact, it is enough (as noted in the text of [1]) to require the existence of a single such phase.

**definition**

\[
LV\text{-commSafe} \quad \text{where} \\
LV\text{-commSafe} \ r \ (HO::\text{Proc} \ HO) \ (\text{coord}::\text{Proc} \ \text{coord}) \equiv \text{True}
\]

**definition**

\[
LV\text{-commLive} \quad \text{where} \\
LV\text{-commLive} \ HOs \ coords \equiv \\
(\forall \ r. \ \text{step} \ r \neq 3 \rightarrow coords (Suc \ r) = coords \ r) \\
\land (\exists (ph::\text{nat}). \ \exists (c::\text{Proc}). \\
(\forall p. \ \text{coords} (4 \cdot \text{ph} \cdot p) = c) \\
\land \ \text{card} (HOs (4 \cdot \text{ph}) \ c) > N \div 2 \land \ \text{card} (HOs (Suc (Suc (Suc (4 \cdot \text{ph})))) c) > N \div 2 \\
\land (\forall p. \ c \in HOs (Suc (Suc (Suc (Suc (4 \cdot \text{ph})))) p) \cap HOs (Suc (Suc (Suc (Suc (4 \cdot \text{ph})))) p))
\]

We instantiate the generic definition of Heard-Of algorithms for the LastVoting algorithm.

**interpretation**

\[
\text{CHOAlgorithm initState sendMsg nextState LV\text{-commSafe} LV\text{-commLive}}
\]

by (unfold-locales, rule procFinite)

### 2.2 Proof of LastVoting: Preliminary Lemmas

We begin by proving some rather obvious 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:

\[
\text{finite} \ \{ ts . \ \exists q v. \ (msgs::Proc \rightarrow \text{val msg}) q = \text{Some} \ (\text{ValStamp} \ v \ ts) \} \\
(\text{is finite} ?ts)
\]

**proof**

\[
\begin{align*}
\text{have} \ ?ts & = \text{stamp} ' \ \text{the} ' \ \text{msgs} ' \ \text{(valStampsRcvd msgs)} \ \text{by} \ (\text{force simp add: valStampsRcvd-def image-def}) \\
\text{thus} \ \text{?thesis} & \ \text{by auto}
\end{align*}
\]

**qed**

**lemma** highestStampRcvd-exists:

\[
\text{assumes} \ \text{nempty: valStampsRcvd msgs } \neq \ \{} \\
\text{obtains} \ p v \ \text{where} \ \text{msgs} p = \text{Some} \ (\text{ValStamp} \ v \ \text{highestStampRcvd msgs})
\]

**proof**

-
let \( ?ts = \{ ts . \exists q. \text{msgs} q = \text{Some (ValStamp} v ts)\} \)

from nempty have \( ?ts \neq \{\} \) by (auto simp add: valStampsRcvd-def)

with timeStampsRcvdFinite

have highestStampRcvd msgs \( \in ?ts \) unfolding highestStampRcvd-def by (rule Max-in)

then obtain p v where 
\( \text{msgs} p = \text{Some (ValStamp} v (\text{highestStampRcvd} \text{msgs}) \) 

by (auto simp add: highestStampRcvd-def)

with that show thesis .

qed

lemma highestStampRcvd-max:
assumes msgs p = Some (ValStamp v ts)
shows ts \( \leq \) highestStampRcvd msgs
using prems unfolding highestStampRcvd-def
by (blast intro: Max-ge timeStampsRcvdFinite)

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 rho HOs coords
and init: \( \forall p. \text{initState} p (\rho 0 p) \implies P 0 \)
and step0: \( \forall r. [\begin{array}{l}
\text{step } r = 0; P r; \text{phase} (\text{Suc} r) = \text{phase} r; \text{step} (\text{Suc} r) = 1; \\
\forall p. \text{next0} r p (\rho r p) \\
\text{rcvdMsgs} p (\text{HOs} r p) (\text{coords} r) (\rho r) (\text{send0} r) \\
\text{coords} r p \\
\text{rho} (\text{Suc} r) p \\
\end{array} ] \implies P (\text{Suc} r) \)

and step1: \( \forall r. [\begin{array}{l}
\text{step } r = 1; P r; \text{phase} (\text{Suc} r) = \text{phase} r; \text{step} (\text{Suc} r) = 2; \\
\forall p. \text{next1} r p (\rho r p) \\
\text{rcvdMsgs} p (\text{HOs} r p) (\text{coords} r) (\rho r) (\text{send1} r) \\
\text{coords} r p \\
\text{rho} (\text{Suc} r) p \\
\end{array} ] \implies P (\text{Suc} r) \)

and step2: \( \forall r. [\begin{array}{l}
\text{step } r = 2; P r; \text{phase} (\text{Suc} r) = \text{phase} r; \text{step} (\text{Suc} r) = 3; \\
\forall p. \text{next2} r p (\rho r p) \\
\text{rcvdMsgs} p (\text{HOs} r p) (\text{coords} r) (\rho r) (\text{send2} r) \\
\text{coords} r p \\
\text{rho} (\text{Suc} r) p \\
\end{array} ] \implies P (\text{Suc} r) \)

and step3: \( \forall r. [\begin{array}{l}
\text{step } r = 3; P r; \text{phase} (\text{Suc} r) = \text{Suc} (\text{phase} r); \text{step} (\text{Suc} r) = 0; \\
\forall p. \text{next3} r p (\rho r p) \\
\text{rcvdMsgs} p (\text{HOs} r p) (\text{coords} r) (\rho r) (\text{send3} r) \\
\text{coords} r p \\
\text{rho} (\text{Suc} r) p \\
\end{array} ] \implies P (\text{Suc} r) \)

shows P n

proof (rule CHORun-induct[OF run])

assume initConfig (\( \rho 0 \))

thus P 0 by (auto simp add: initConfig-def init)

next

fix r

assume ih: P r and nxt: nextConfig r (\( \rho r \)) (\text{HOs} r) (\text{coords} r) (\rho (\text{Suc} r))
have \( step \ r \in \{0,1,2,3\} \) by (auto simp add: step-def)
thus \( P \ (Suc \ r) \)
proof (auto)
  assume \( stp: \ step \ r = 0 \)
  hence \( stp': \ step \ (Suc \ r) = 1 \) by (auto simp add: step-def mod-Suc)
  from \( stp \) have \( ph: \ phase \ (Suc \ r) = phase \ r \) by (auto simp add: phase-def step-def)
  from \( \text{ih} \) \( nxt \ stp \ stp' \ ph \ show \ ?thesis \)
next
  by (intro step0, auto simp add: nextConfig-def nextState-def sendMsg-def)
next
  assume \( stp: \ step \ r = Suc \ 0 \)
  hence \( stp': \ step \ (Suc \ r) = 2 \) by (auto simp add: step-def mod-Suc)
  from \( stp \) have \( ph: \ phase \ (Suc \ r) = phase \ r \)
  unfolding step-def phase-def by presburger
  from \( \text{ih} \) \( nxt \ stp \ stp' \ ph \ show \ ?thesis \)
next
  by (intro step1, auto simp add: nextConfig-def nextState-def sendMsg-def)
next
  assume \( stp: \ step \ r = 3 \)
  hence \( stp': \ step \ (Suc \ r) = 0 \) by (auto simp add: step-def mod-Suc)
  from \( stp \) have \( ph: \ phase \ (Suc \ r) = Suc \ (phase \ r) \)
  unfolding step-def phase-def by presburger
  from \( \text{ih} \) \( nxt \ stp \ stp' \ ph \ show \ ?thesis \)
next
  by (intro step3, auto simp add: nextConfig-def nextState-def sendMsg-def)
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 \( \text{run}: \ CHORun \ \rho \ \text{HOs} \ \text{coords} \)  
and \( \text{step0}: \ [ \ step \ r = 0; \ step \ (Suc \ r) = 1; \ phase \ (Suc \ r) = phase \ r; \forall \ p. \ \text{next0} \ r \ p \ (\rho \ \text{r} \ \text{p}) \)  
  \( (\text{rcvdMmsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ \text{r} \ \text{p}) \ (\text{send0} \ r)) \)  
  \( (\text{coords} \ r \ p) \ (\rho \ (Suc \ r) \ p) \] \)  
  \( \implies \ P \ r \)
and \( \text{step1}: \ [ \ step \ r = 1; \ step \ (Suc \ r) = 2; \ phase \ (Suc \ r) = phase \ r; \forall \ p. \ \text{next1} \ r \ p \ (\rho \ \text{r} \ \text{p}) \)  
  \( (\text{rcvdMmsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ \text{r} \ \text{p}) \ (\text{send1} \ r)) \)  
  \( (\text{coords} \ r \ p) \ (\rho \ (Suc \ r) \ p) \] \)  
  \( \implies \ P \ r \)
and \( \text{step2}: \ [ \ step \ r = 2; \ step \ (Suc \ r) = 3; \ phase \ (Suc \ r) = phase \ r; \forall \ p. \ \text{next2} \ r \ p \ (\rho \ \text{r} \ \text{p}) \)  
  \( (\text{rcvdMmsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ \text{r} \ \text{p}) \ (\text{send2} \ r)) \)  
  \( (\text{coords} \ r \ p) \ (\rho \ (Suc \ r) \ p) \] \)  
  \( \implies \ P \ r \)
and \( \text{step3}: \ [ \ step \ r = 3; \ step \ (Suc \ r) = 0; \ phase \ (Suc \ r) = Suc \ (phase \ r); \forall \ p. \ \text{next3} \ r \ p \ (\rho \ \text{r} \ \text{p}) \)  
  \( (\text{rcvdMmsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ \text{r} \ \text{p}) \ (\text{send3} \ r)) \)  
  \( (\text{coords} \ r \ p) \ (\rho \ (Suc \ r) \ p) \] \)  
  \( \implies \ P \ r \)
proof
−
lemma LV-induct
proof.
case-distinction rules. When these variants are applicable, they help automating the Isabelle
state relation of that particular process. We prove corresponding variants of the induction and
Sometimes the assertion to prove talks about a specific process and follows from the next-
run
∈ 
auto simp add
have step r ∈ \{0,1,2,3\} by (auto simp add: step-def)
thus P r
proof (auto)
assumestp: step r = 0
hence stp': step (Suc r) = 1 by (auto simp add: step-def mod-Suc)
from stphave ph: phase (Suc r) = phase r by (auto simp add: phase-def step-def)
from nxtstp stp'ph show \?thesis
  by (intro step0, auto simp add: nextConfig-def nextState-def sendMsg-def)
next
assume stp: step r = Suc 0
hence stp': step (Suc r) = 2 by (auto simp add: step-def mod-Suc)
from stphave ph: phase (Suc r) = phase r
  unfolding step-def phase-def by presburger
from nxtstp stp'ph show \?thesis
  by (intro step1, auto simp add: nextConfig-def nextState-def sendMsg-def)
next
assume stp: step r = 2
hence stp': step (Suc r) = 3 by (auto simp add: step-def mod-Suc)
from stphave ph: phase (Suc r) = phase r
  unfolding step-def phase-def by presburger
from nxtstp stp'ph show \?thesis
  by (intro step2, auto simp add: nextConfig-def nextState-def sendMsg-def)
next
assume stp: step r = 3
hence stp': step (Suc r) = 0 by (auto simp add: step-def mod-Suc)
from stphave ph: phase (Suc r) = Suc (phase r)
  unfolding step-def phase-def by presburger
from nxtstp stp'ph show \?thesis
  by (intro step3, auto simp add: nextConfig-def nextState-def sendMsg-def)
qed
qed

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

lemma LV-induct':
  assumes run: CHORun rho HOs coords
  and init: initState p (rho 0 p) \implies P p 0
  and step0: \A r. \{ step r = 0; P p r; phase (Suc r) = phase r; step (Suc r) = 1;
    next0 r p (rho r p)
      (rcvdMsgs p (HOs r p) (coords r) (rho r) (send0 r))
      (coords r p) (rho (Suc r) p) \} \implies P p (Suc r)
  and step1: \A r. \{ step r = 1; P p r; phase (Suc r) = phase r; step (Suc r) = 2;
    next1 r p (rho r p)
      (rcvdMsgs p (HOs r p) (coords r) (rho r) (send1 r))
      (coords r p) (rho (Suc r) p) \} \implies P p (Suc r)
  and step2: \A r. \{ step r = 2; P p r; phase (Suc r) = phase r; step (Suc r) = 3;

11
next2 \( r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send2} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ (\text{Suc} \ r)\)

and step3: \( \forall r. \ [ step \ \ r = 3; \ P \ p \ r; \ phase \ (\text{Suc} \ r) = \text{Suc} \ (\text{phase} \ r); \ step \ (\text{Suc} \ r) = 0; \)
\(\text{next3} \ r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send3} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ (\text{Suc} \ r)\)

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

lemma \( \text{LV-Suc}'\):
assumes \( \text{run}: \text{CHORun} \ \rho \text{HOs} \text{coords} \)
and step0: \( [ \ step \ \ r = 0; \ step \ (\text{Suc} \ r) = 1; \ phase \ (\text{Suc} \ r) = \text{phase} \ r; \)
\(\text{next0} \ r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send0} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ r\)

and step1: \( [ \ step \ \ r = 1; \ step \ (\text{Suc} \ r) = 2; \ phase \ (\text{Suc} \ r) = \text{phase} \ r; \)
\(\text{next1} \ r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send1} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ r\)

and step2: \( [ \ step \ \ r = 2; \ step \ (\text{Suc} \ r) = 3; \ phase \ (\text{Suc} \ r) = \text{phase} \ r; \)
\(\text{next2} \ r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send2} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ r\)

and step3: \( [ \ step \ \ r = 3; \ step \ (\text{Suc} \ r) = 0; \ phase \ (\text{Suc} \ r) = \text{Suc} \ (\text{phase} \ r); \)
\(\text{next3} \ r \ p \ (\rho \ r \ p) \)
\((\text{rcvdMsgs} \ p \ (\text{HOs} \ r \ p) \ (\text{coords} \ r) \ (\rho \ r) \ (\text{send3} \ r))\)
\((\text{coords} \ r \ p) \ (\rho \ (\text{Suc} \ r) \ p) \]
\(\implies P \ p \ r\)

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

2.3 Boundedness and monotonicity of timestamps

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

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

Moreover, timestamps can only grow over time.

lemma \( \text{LV-timestamp-increasing}: \)
assumes \( \text{run}: \text{CHORun} \ \rho \text{HOs} \text{coords} \)
shows \( \text{timestamp} \ (\rho \ n \ p) \leq \ \text{timestamp} \ (\rho \ (\text{Suc} \ n) \ p)\)
\((\text{is} \ ?P \ p \ n \ \text{is} \ ?\text{ts} \ \leq -)\)
proof (rule \( \text{LV-Suc}[OF \ \text{run}, \ \text{where} \ P = ?P])\)

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

\textbf{fix} HO
\textbf{assume} stp: \text{step} n = 1
\textbf{and} nxt: \text{next1} n p (\rho n p)
\text{(rcvdmsgs} p (HOs n p) (coords n) (\rho n) (send1 n))
\text{(coords n p) (\rho (Suc n) p)}
\textbf{from} stp \textbf{have} ?ts \leq \text{phase} n
\textbf{using} LV-timestamp-bounded[\text{OF} \text{run}, \text{where} \ n=n, \ \text{where} \ p=p] \textbf{by} \ \text{auto}
\textbf{with} nxt \textbf{show} \ ?thesis \textbf{by} (\text{auto simp add: next1-def})
\textbf{qed} (\text{auto simp add: next0-def next2-def next3-def})

\textbf{lemma} LV-timestamp-monotonic:
\textbf{assumes} run: CHORun \rho HOs coords \text{and} \ le: \ m \leq n
\textbf{shows} timestamp (\rho m p) \leq \text{timestamp} (\rho n p)
\text{(is \ ?ts m \leq -)}
\textbf{proof} –
\textbf{from} le \textbf{obtain} k \textbf{where} k: n = m+k \textbf{by} (\text{auto simp add: le-iff-add})
\textbf{have} ?ts m \leq ?ts (m+k) (\text{is \ ?P k})
\textbf{proof} (\text{induct} k)
\textbf{case} 0 \textbf{show} \ ?P 0 \textbf{by} \text{simp}
\textbf{next}
\textbf{fix} k
\textbf{assume} ih: ?P k
\textbf{from} run \textbf{have} ?ts (m+k) \leq ?ts (m + Suc k) \textbf{by} (\text{auto simp add: LV-timestamp-increasing})
\textbf{with} ih \textbf{show} ?P (Suc k) \textbf{by} \text{simp}
\textbf{qed}
\textbf{with} k \textbf{show} \ ?thesis \textbf{by} \text{simp}
\textbf{qed}

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

\textbf{definition} procsBeyondTS \text{where} procsBeyondTS ts cfg \equiv \{ \ p . \ ts \leq \text{timestamp} (cfg p) \}

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

\textbf{lemma} procsBeyondTS-monotonic:
\textbf{assumes} run: CHORun \rho HOs coords
\textbf{and} \ p: p \in procsBeyondTS ts (\rho m) \text{and} \ le: \ m \leq (n:nat)
\textbf{shows} p \in procsBeyondTS ts (\rho n)
\textbf{proof} –
\textbf{from} p \textbf{have} ts \leq \text{timestamp} (\rho m p) (\text{is \ - \leq \ ?ts m})
\textbf{by} (\text{simp add: procsBeyondTS-def})
\textbf{moreover}
\textbf{from} run \textbf{le} \textbf{have} ?ts m \leq ?ts n \textbf{by} (\text{rule LV-timestamp-monotonic})
\textbf{ultimately show} ?thesis
\textbf{by} (\text{simp add: procsBeyondTS-def})
\textbf{qed}

\textbf{2.4 Obvious facts about the algorithm}

The following \textbf{lemmas} state some very obvious facts that follow “immediately” from the \text{definition of the algorithm}. We could prove them in one fell swoop by defining a big \text{invariant}, but it \text{appears more readable to prove them separately}.
Coordinators change only at step 3. This is an immediate consequence of the communication/coordinator predicate.

**lemma notStep3EqualCoord:**
- **assumes** `CHORun rho HOs coords and step r ≠ 3`
- **shows** `coords (Suc r) p = coords r p`
- **using** `assms by (auto simp add: CHORun-def LV-commLive-def)`

Votes only change at step 0.

**lemma notStep0EqualVote [rule-format]:**
- **assumes** `run: CHORun 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 add: next0-def next1-def next2-def next3-def)`

Commit status only changes at steps 0 and 3.

**lemma notStep03EqualCommit [rule-format]:**
- **assumes** `run: CHORun 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], auto simp add: next0-def next1-def next2-def next3-def)`

Timestamps only change at step 1.

**lemma notStep1EqualTimestamp [rule-format]:**
- **assumes** `run: CHORun rho HOs coords`
- **shows** `step r ≠ 1 ⟷ timestamp (rho (Suc r) p) = timestamp (rho r p) (is ?P p r)`
  - **by** `(rule LV-Suc'[OF run, where P=?P], auto simp add: next0-def next1-def next2-def next3-def)`

The `x` field only changes at step 1.

**lemma notStep1EqualX [rule-format]:**
- **assumes** `run: CHORun rho HOs coords`
- **shows** `step r ≠ 1 ⟷ x (rho (Suc r) p) = x (rho r p) (is ?P p r)`
  - **by** `(rule LV-Suc'[OF run, where P=?P], auto simp add: 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 rho HOs coords and cmt: commt (rho r p) and conds: [ 1 ≤ step r; coords r p = p; vote (rho r p) ≠ None; card {q . coords r q = p} > N div 2 ] ⟷ A`
- **shows** `A`
- **proof** –
have \( \text{commit}(\rho \ r \ p) \rightarrow \)
\( 1 \leq \text{step} \ r \land \text{coords} \ r \ p = p \land \text{vote}(\rho \ r \ p) \neq \text{None} \land \text{card}\{q . \text{coords} \ r \ q = p\} > N \div 2 \)
\( (\text{is} \ ?P \ r \ \text{is} - \rightarrow ?R \ r) \)

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

fix \( n \)
assume \( \text{nxt}: \text{next0} \ n \ p (\rho \ n \ p) (\text{rcvdMsgs} \ p (\text{HOs} \ n \ p) (\text{coords} \ n) (\rho \ n) (\text{send0} \ n)) (\text{coords} \ n \ p) (\rho \ (\text{Suc} \ n) \ p) \)
    and \( \text{ph}: \text{phase}(\text{Suc} \ n) = \text{phase} \ n \)
    and \( \text{stp}: \text{step} \ n = 0 \land \text{stp}' : \text{step}(\text{Suc} \ n) = 1 \)
    and \( \text{ih}: ?P \ p \ n \)
show \( ?P \ p \ (\text{Suc} \ n) \)

proof
assume \( \text{cm}' : \text{commit}(\rho \ (\text{Suc} \ n) \ p) \)
from \( \text{stp} \ \text{ih} \) have \( \text{cm}: \neg \text{commit}(\rho \ n \ p) \) by simp
with \( \text{nxt} \ \text{cm}' \)
have \( \text{coords} \ n \ p = p \land \text{vote}(\rho \ (\text{Suc} \ n) \ p) \neq \text{None} \)
    \( \land \text{card}(\text{valStampsRcvd}(\text{rcvdMsgs} \ p (\text{HOs} \ n \ p) (\text{coords} \ n) (\rho \ n) (\text{send0} \ n))) > N \div 2 \)
    by (auto simp add: next0-def)
moreover
have \( \text{valStampsRcvd}(\text{rcvdMsgs} \ p (\text{HOs} \ n \ p) (\text{coords} \ n) (\rho \ n) (\text{send0} \ n)) \subseteq \{q . \text{coords} \ n \ q = p\} \)
    by (auto simp add: valStampsRcvd-def rcvdMsgs-def send0-def)

hence \( \text{card}(\text{valStampsRcvd}(\text{rcvdMsgs} \ p (\text{HOs} \ n \ p) (\text{coords} \ n) (\rho \ n) (\text{send0} \ n))) \leq \text{card}\{q . \text{coords} \ n \ q = p\} \)
    by (auto intro: card-mono)
moreover
note \( \text{stp} \ \text{stp}' \ \text{run} \)
ultimately
show \( ?R \ (\text{Suc} \ n) \)
    by (auto simp add: notStep3EqualCoord)

qed
— the remaining cases are all solved by expanding the definitions

\text{qed} (\text{auto simp add: initState-def next1-def next2-def next3-def notStep3EqualCoord[OF run]})
with \( \text{cmt} \) show \( \text{?thesis} \) by (intro \text{conds}, auto)

\text{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.

\text{lemma currentTimestampE:}
\hspace{1em} \text{assumes run: CHORun rho HOs coords}
\hspace{1em} \text{and ts: timestamp}(\rho \ r \ p) = \text{Suc}(\text{phase} \ r)
\hspace{1em} \text{and \text{conds}: \[ 2 \leq \text{step} \ r ; \]
\hspace{2em} \text{commit}(\rho \ r \ (\text{coords} \ r \ p));
\hspace{2em} \text{x}(\rho \ r \ p) = \text{the}(\text{vote}(\rho \ r \ (\text{coords} \ r \ p)))
\hspace{1em} \text{\]} \implies A)

shows \( A \)
proof
let \( ?ts \ n = \text{timestamp}(\rho \ n \ p) \)
let \( ?crd \ n = \text{coords} \ n \ p \)
have \( ?ts \ r = \text{Suc}(\text{phase} \ r) \rightarrow 2 \leq \text{step} \ r \land \text{commit}(\rho \ r \ (\text{?crd} \ r)) \land x(\rho \ r \ p) = \text{the}(\text{vote}(\rho \ r \ (\text{?crd} \ r))) \)
(is \( ?Q \) \( p \) \( r \) is \( \neg \) \( \rightarrow \) \( ?R \) \( r \) )

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

— The assertion is trivially true initially because the timestamp is 0.

assume initState \( p \) (\( \rho \) 0 \( p \) ) thus \( ?Q \) \( p \) 0

by (auto simp add: 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 \( \text{stp}' \): step (Suc \( n \)) = 1

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

by auto

thus \( ?Q \) \( p \) (Suc \( n \)) by simp

next

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

fix \( n \)

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

and ph: phase (Suc \( n \)) = phase \( n \)

and nxt: next1 \( n \) \( p \) (rho \( n \) \( p \)) (rcvdMsgs \( p \) (HOs \( n \) \( p \)) (coords \( n \)) (rho \( n \)) (send1 \( n \)) (\( ?\text{crd} \) \( n \)) (rho (Suc \( n \)) \( p \))

show \( ?Q \) \( p \) (Suc \( n \))

proof

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

from run stp LV-timestamp-bounded\([where \ n=n]\) have \( \text{ts} \) \( n \) \( \leq \) phase \( n \) by auto

moreover

from run stp have vote (rho (Suc \( n \)) (\( ?\text{crd} \) (Suc \( n \)))) = vote (rho \( n \) (\( ?\text{crd} \) \( n \)))

by (auto simp add: notStep3EqualCoord notStep0EqualVote)

moreover

from run stp have commt (rho (Suc \( n \)) (\( ?\text{crd} \) (Suc \( n \)))) = commt (rho \( n \) (\( ?\text{crd} \) \( n \)))

by (auto simp add: notStep3EqualCoord notStep03EqualCommit)

moreover

note ts nxt stp' ph

ultimately

show \( ?R \) (Suc \( n \))

by (auto simp add: next1-def send1-def rcvdMsgs-def isVote-def)

qed

next

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

fix \( n \)

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

and ph: phase (Suc \( n \)) = phase \( n \)

and ih: \( ?Q \) \( p \) \( n \)

and nxt: next2 \( n \) \( p \) (rho \( n \) \( p \)) (rcvdMsgs \( p \) (HOs \( n \) \( p \)) (coords \( n \)) (rho \( n \)) (send2 \( n \)) (\( ?\text{crd} \) \( n \)) (rho (Suc \( n \)) \( p \))

show \( ?Q \) \( p \) (Suc \( n \))

proof

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

from run stp

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

by (auto simp add: notStep3EqualCoord notStep0EqualVote)

from run stp

have cnt: commt (rho (Suc \( n \)) (\( ?\text{crd} \) (Suc \( n \)))) = commt (rho \( n \) (\( ?\text{crd} \) \( n \)))

by (auto simp add: notStep3EqualCoord notStep03EqualCommit)

with vt ts ph stp stp' ih nxt

16
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)

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: \(CHORun \rho \ HOs \ coords \ and \ rdy: \ ready (\rho \ r \ p)\)
and conds: \[
\begin{align*}
\text{step} \(r\) & = 3; \\
\text{coords} \(r \ p\) & = \(p\); \\
\text{card} \(\{q . \text{coords} \(r \ q\) = \(p\) \& \text{timestamp} (\rho \ r \ q) = Suc (phase \(r\))\} > N \div \(\text{2}\)
\end{align*}
\]
shows \(P\)
proof —
  let \(\?qs \(n\) = \{q . \text{coords} \(n \ q\) = \(p\) \& \text{timestamp} (\rho \ n \ q) = Suc (phase \(n\))\}
  have ready \((\rho \ r \ p) \longrightarrow \text{step} \(r\) = 3 \& \text{coords} \(r \ p\) = \(p\) \& \text{card} (\?qs \(r\)) > N \div \(\text{2}\)
    (is \(?Q p \ r\) is - \(\longrightarrow \)?R \(p \ r\))
proof (rule \(LV\)-induct[of run, where \(P=\?Q\)])
  — the interesting case is step 2
  fix \(n\)
  assume \(stp\): step \(n\) = 2 and \(stp\)': step \((Suc \(n\))\) = 3
  and \(ih\): \(?Q p \ n\) and \(ph\): phase \((Suc \(n\))\) = phase \(n\)
  and \(nxt\): next2 \(n\) \(p\) \((\rho \ n \ p) \ (\text{rcvdMsgs} \(p \ (\text{HOs} \ n \ p) \ (\text{coords} \(n\)) \ (\rho \ n) \ (\text{send2} \(n\)) \ (\text{coords} \(n \ p\)) \ (\rho \ (Suc \(n\)) \ p)\)
  show \(?Q p (Suc \(n\))\)
proof
  assume \(rdy\): \(ready (\rho \ (Suc \(n\)) \ p)\)
  from \(stp \ ih\) have \(ndry\): \(\neg \text{ready} (\rho \ n \ p)\) by simp
  with \(rdy \ nxt\) have \(\text{coords} \(n \ p\) = \(p\)
    by (auto simp add: next2-def)
  with run \(stp\) have \(\text{coords} \(Suc \(n\)) \ p\) = \(p\)
    by (simp add: notStep3EqualCoord)
  let \(\?acks = \text{acksRcvd} (\text{rcvdMsgs} \(p \ (\text{HOs} \ n \ p) \ (\text{coords} \(n\)) \ (\rho \ n) \ (\text{send2} \(n\))\))
  from \(ndry \ rdy\) \(nxt\) have \(aRcvd\): \(\text{card} \?acks > N \div \(\text{2}\)
    by (auto simp add: next2-def)
  have \(\?acks \subseteq \?qs (Suc \(n\))\)
proof (clarify)
  fix \(q\)
  assume \(q\): \(q \in \?acks\)
  hence \(n\): \(\text{coords} \(n \ q\) = \(p\) \& \text{timestamp} (\rho \ n \ q) = Suc (phase \(n\))\)

17
A process decides only if the following conditions hold:

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

This is (essentially) Bernadette’s Lemma 3.

**Lemma decisionE:**

**Assumes** run: CHORun rho HOs coords
and dec: decide (rho (Suc r) p) ≠ decide (rho r p)
and conds: \[ \text{step } r = 3; \]
\[
\begin{aligned}
&\text{decide } (rho (Suc r) p) = \text{Some } (\text{the } (\text{vote } rho r (coords r p))) ; \\
&\text{ready } (rho r (coords r p)); \text{commit } (rho r (coords r p))
\end{aligned}
\]

**Shows** P

**Proof** —

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

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

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

(is ?Q p r)

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

— for step 3, we prove the thesis by expanding the relevant definitions

**Assume** next3 r p (?cfg p) (rcvdMsgs p (HOs r p) ?crd ?cfg (send3 r)) (?crd p) (?cfg' p)

and step r = 3

with dec show ?thesis

by (auto simp add: next3-def send3-def isVote-def rcvdMsgs-def)

**Next** — for the other steps, the proof is by contradiction because they don’t change the decision

**Assume** next0 r p (?cfg p) (rcvdMsgs p (HOs r p) ?crd ?cfg (send0 r)) (?crd p) (?cfg' p)

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

**Next**

**Assume** next1 r p (?cfg p) (rcvdMsgs p (HOs r p) ?crd ?cfg (send1 r)) (?crd p) (?cfg' p)

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

**Next**

**Assume** next2 r p (?cfg p) (rcvdMsgs p (HOs r p) ?crd ?cfg (send2 r)) (?crd p) (?cfg' p)
with \texttt{dec} show \texttt{thesis} by (auto simp add: \texttt{next2-def})

\begin{lstlisting}
with \texttt{run} have \texttt{card} \{q \cdot \texttt{crd} q = \texttt{crd} p \land \texttt{timestamp} (\texttt{cfg} q) = \text{Suc} (\text{phase} r)\} > \text{N div 2}
  by (rule \texttt{readyE})

— Hence there is at least one such process . . .

hence \texttt{card} \{q \cdot \texttt{crd} q = \texttt{crd} p \land \texttt{timestamp} (\texttt{cfg} q) = \text{Suc} (\text{phase} r)\} \neq 0
  by \texttt{arith}

then obtain \texttt{q} where \texttt{crd} \texttt{q} = \texttt{crd} \texttt{p} \land \texttt{timestamp} (\texttt{cfg} \texttt{q}) = \text{Suc} (\text{phase} r)
  by auto

— . . . and by a previous lemma the coordinator must have committed.

with \texttt{run} have \texttt{commt} (\texttt{cfg} (\texttt{crd} \texttt{p}))
  by (auto elim: \texttt{currentTimestampE})

with 1 show \texttt{thesis} by (blast intro: \texttt{conds})
\end{lstlisting}

\section{2.5 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.

\textbf{lemma} \texttt{integrityInvariant}:
\begin{itemize}
  \item \textbf{assumes} \texttt{run}: \texttt{CHORun} rho HOs coords
  \item \textbf{and} \texttt{inv}: \texttt{[ [ range} (x \circ (rho n)) \subseteq \texttt{range} (x \circ (rho 0));
    \texttt{range} (vote \circ (rho n)) \subseteq \{\texttt{None}\} \cup \texttt{Some} \cdot \texttt{range} (x \circ (rho 0));
    \texttt{range} (decide \circ (rho n)) \subseteq \{\texttt{None}\} \cup \texttt{Some} \cdot \texttt{range} (x \circ (rho 0))
  ] \implies A}
\end{itemize}
shows A

\textbf{proof} –
\begin{itemize}
  \item let \texttt{?x0} = \texttt{range} (x \circ rho 0)
  \item let \texttt{?x0opt} = \{\texttt{None}\} \cup \texttt{Some} \cdot \texttt{?x0}
  \item have \texttt{range} (x \circ rho n) \subseteq \texttt{?x0} \land
    \texttt{range} (vote \circ rho n) \subseteq \texttt{?x0opt} \land
    \texttt{range} (decide \circ rho n) \subseteq \texttt{?x0opt} (is \texttt{?Inv n} is \texttt{?X n} \land \texttt{?Vote n} \land \texttt{?Decide n})
\end{itemize}
\textbf{proof} (induct n)
\begin{itemize}
  \item from \texttt{run} show \texttt{?Inv 0}
    by (auto simp add: \texttt{CHORun-def initConfig-def initState-def})
  \item fix \texttt{n}
  \item assume \texttt{ih}: \texttt{?Inv n} \texttt{thus} \texttt{?Inv (Suc n)}
  \item proof (clarify)
    \begin{itemize}
      \item assume \texttt{x}: \texttt{?X n} \texttt{and} \texttt{vt}: \texttt{?Vote n} \texttt{and} \texttt{dec:} \texttt{?Decide n}
    \end{itemize}
\end{itemize}

Proof of first conjunct
\begin{itemize}
  \item have \texttt{x'}: \texttt{?X (Suc n)}
  \item proof (clarsimp)
    \begin{itemize}
      \item fix \texttt{p}
      \item from \texttt{run} show \texttt{x (rho (Suc n) p) \in range (\lambda q. x (rho 0 q)) (is \texttt{?P p n})}
        proof (rule \texttt{LV-Suc''[where} \texttt{P=?P]})
        — only \texttt{step1} is of interest
      \end{itemize}
  \end{itemize}
show \( ?\text{thesis} \)

**proof** (cases \( \rho (\text{Suc} \ n) \ p = \rho \ n \ p \))

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

next

case False
  with \( \text{nxt} \) have \( \text{cnt: commit} (\rho \ n \ (\text{coords} \ n \ p)) \)
    and \( \text{xp} : x \ (\rho (\text{Suc} \ n) \ p) = \text{the} (\rho \ n \ (\text{coords} \ n \ p)) \)
  by (auto simp add: next1-def send1-def rcvdMsgs-def isVote-def)

moreover
  from \( \text{vt} \) have \( \text{vote} (\rho \ n \ (\text{coords} \ n \ p)) \not= \text{None} \)
  by (rule commitE)

moreover
  note \( \text{xp} \)

ultimately
  show \( ?\text{thesis} \) by (force simp add: image-def)

qed

— the other steps don’t change \( x \) and therefore follow from the induction hypothesis

next
  assume \( \text{step} \ n = 0 \)
  with \( \text{run} \) have \( x \ (\rho (\text{Suc} \ n) \ p) = x \ (\rho \ n \ p) \)
    by (simp add: notStep1EqualX)
  with \( x \) show \( ?\text{thesis} \) by auto

next
  assume \( \text{step} \ n = 2 \)
  with \( \text{run} \) have \( x \ (\rho (\text{Suc} \ n) \ p) = x \ (\rho \ n \ p) \)
    by (simp add: notStep1EqualX)
  with \( x \) show \( ?\text{thesis} \) by auto

next
  assume \( \text{step} \ n = 3 \)
  with \( \text{run} \) have \( x \ (\rho (\text{Suc} \ n) \ p) = x \ (\rho \ n \ p) \)
    by (simp add: notStep1EqualX)
  with \( x \) show \( ?\text{thesis} \) by auto

qed

Proof of second conjunct

have \( \text{vt}' : ?\text{Vote} \ (\text{Suc} \ n) \)

**proof** (clarsimp simp add: image-def)

fix \( p \ v \)

assume \( v : \text{vote} (\rho (\text{Suc} \ n) \ p) = \text{Some} \ v \)

from \( \text{run} \) have \( \text{vote}(\rho (\text{Suc} \ n) \ p) = \text{Some} \ v \longrightarrow v \in \ ?x0opt \)
  by (auto simp add: isVote-def)

**proof** (rule LV-Suc'[where \( P=\ ?P \)])

— here only \( \text{step}0 \) is of interest

assume \( \text{nxt: next0} \ n \ p \ (\rho \ n \ p) \)
  \( \text{rcvdMsgs} \ p \ (\text{HOs} \ n \ p) \ (\text{coords} \ n \ p) \ (\rho \ n) \ (\text{send0} \ n) \)
  \( \ (\text{coords} \ n \ p) \ (\rho (\text{Suc} \ n) \ p) \)

show \( ?\text{thesis} \)

**proof** (cases \( \rho (\text{Suc} \ n) \ p = \rho \ n \ p \))

  case True
  from \( \text{vt} \) have \( \text{vote}(\rho \ n \ p) \in \ ?x0opt \)
    by (auto simp add: image-def)
  with \( \text{True} \) show \( ?\text{thesis} \) by auto

next

case False
  from \( \text{nxt} \) False \( v \) obtain \( q \) where \( v = x \ (\rho \ n \ q) \)
    by (auto simp add: next0-def send0-def rcvdMsgs-def)
  with \( x \) show \( ?\text{thesis} \) by (auto simp add: image-def)

20
qed
— the other cases don’t change the vote and therefore follow from the induction hypothesis

next
assume step \( n = 1 \)
with run have vote (\( \rho \) (Suc \( n \)) \( p \)) = vote (\( \rho \) \( n \) \( p \))
by (simp add: notStep0EqualVote)
moreover
from \( \text{vt} \) have vote (\( \rho \) \( n \) \( p \)) \( \in \) \( ?x0opt \) by (auto simp add: image-def)
ultimately
show \( ?\text{thesis} \) 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 \( \text{vt} \) have vote (\( \rho \) \( n \) \( p \)) \( \in \) \( ?x0opt \) by (auto simp add: image-def)
ultimately
show \( ?\text{thesis} \) 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 \( \text{vt} \) have vote (\( \rho \) \( n \) \( p \)) \( \in \) \( ?x0opt \) by (auto simp add: image-def)
ultimately
show \( ?\text{thesis} \) by auto

qed

Proof of third conjunct
have \( \text{dec'} \): \( \text{?Decide} \) (Suc \( n \))
proof (clarsimp simp add: image-def)
fix \( p \) \( v \)
assume \( \exists \) \( q \). \( v = x \) (\( \rho \) 0 \( q \))
show \( \exists \) \( q \). \( v = x \) (\( \rho \) 0 \( q \))
proof (cases decide (\( \rho \) (Suc \( n \)) \( p \)) = decide (\( \rho \) \( n \) \( p \)))

case False
let \( \text{?crd} = \text{coords} \) \( n \) \( p \)
from \( \text{False} \) run have
\( \text{d'}\): \text{decide} (\( \rho \) (Suc \( n \)) \( p \)) = Some (the (\( \text{vote} \) (\( \rho \) \( n \) ?\( \text{crd} \)))) and
\( \text{cmt}\): \text{commit} (\( \rho \) \( n \) ?\( \text{crd} \))
by (auto elim: decisionE)
from \( \text{vt} \) have \( \text{vtc}\): vote (\( \rho \) \( n \) ?\( \text{crd} \)) \( \in \) \( ?x0opt \) by (auto simp add: image-def)
from \( \text{run cmt} \) have vote (\( \rho \) \( n \) ?\( \text{crd} \)) \( \neq \) None by (rule commitE)
with \( \text{d'}\) \( \text{vt}\) show \( ?\text{thesis} \) by auto
qed
qed
from \( x' \) \( v' \) \( \text{dec'} \) show \( ?\text{thesis} \) by simp
qed
qed

with \( \text{inv}\) show \( ?\text{thesis} \) by simp
The Integrity theorem follows as an easy consequence.

**Theorem integrity:**
- **Assumes** `run: CHORun rho HOs coords and dec: decide (rho n p) = Some v`
- **Shows** `∃ q. v = x (rho 0 q)`

**Proof** –
- From `run have decide (rho n p) ∈ {None} ∪ Some ` (range (x o (rho 0)))`
  - By (rule `integrityInvariant`, `auto simp add: image-def`)
- With `dec` show `?thesis` by (`auto simp add: image-def`)

**2.6 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 rho HOs coords and dec: decide (rho (Suc r) p) ≠ decide (rho r p)`
- **Shows** `card (procsBeyondTS (Suc (phase r)) (rho r)) > N div 2`
- Using `run dec` proof (rule `decisionE`)
  - Lemma `decisionE` tells us that we are at step 3 and that the coordinator is ready.
- Let `?crd = coords r p`  
- Let `?qs = { q . coords 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 `∀ q. timestamp (rho r q) ≤ Suc (phase r)` by `auto`
  - Hence `?qs ⊆ procsBeyondTS (Suc (phase r)) (rho r)`
    - By (auto simp add: `procsBeyondTS-def`)
  - Hence `card ?qs ≤ 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 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 . coords r q = p} > N div 2` by (blast elim: `commitE`)
  - Moreover
  - From `run cmt' have card { q . coords r q = p'} > N div 2` by (blast elim: `commitE`)
  - Ultimately
    - Obtain `q` where `coords r q = p` and `p' = coords 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:**

proof

assume \( \text{commitThenVoteRecent} \)

\begin{align*}
&\text{assumes run: } \text{CHORun } \rho \text{ HOs coords} \\
&\text{and rdy: ready (} \rho r p \text{) and rdy': ready (} \rho r p' \text{)} \\
&\text{shows } p = p' \\
&\text{proof} \\
&\text{let } ?C p = \{ q . \text{coords } r q = p \wedge \text{timestamp (} \rho r q \text{)} = \text{Suc (phase } r) \} \\
&\text{from } \text{run rdy have } \text{card (} ?C p \text{)} > N \text{ div } 2 \text{ by (blast elim: readyE)} \\
&\text{moreover} \\
&\text{from } \text{run rdy'} \text{ have } \text{card (} ?C p' \text{)} > N \text{ div } 2 \text{ by (blast elim: readyE)} \\
&\text{ultimately} \\
&\text{obtain } q \text{ where } \text{coords } r q = p \text{ and } p' = \text{coords } r q \text{ by (auto elim: majoritiesE')} \\
&\text{thus } ?\text{thesis by simp} \\
&\text{qed}
\end{align*}

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:}

\begin{align*}
&\text{assumes run: } \text{CHORun } \rho \text{ HOs coords} \\
&\text{and maj: } \text{card (procsBeyondTS } ts \text{ (} \rho r \text{)) > N } \text{div } 2 \text{ and cmt: commit (} \rho r p \text{)} \\
&\text{shows } \exists q \in \text{procsBeyondTS } ts \text{ (} \rho r \text{), vote (} \rho r p \text{) = Some } (x \text{ (} \rho r q \text{))} \\
&(\text{is } \{ Q r \}) \\
&\text{proof} \\
&\text{rule LV-induct(OF run)}
\end{align*}

\textit{next0} establishes the property

\begin{align*}
&\text{fix } n \\
&\text{assume stp: step } n = 0 \\
&\text{and nxt: } \forall q. \text{next0 } n q (\rho n q) (\text{rcvdMsgs } p (\text{HOs } n p) (\text{coords } n) (\rho n) (\text{send0 } n) (\text{coords } n) q) \text{ (} \text{rho } (\text{Suc } n) q) \text{ (is } \forall q. \text{ ?nxt } q) \\
&\text{from nxt have } \text{nxt : ?nxt } p .. \\
&\text{show } ?P p (\text{Suc } n) \\
&\text{proof (clarify)} \\
&\text{assume mj: } \text{card (} ?\text{bynd } (\text{Suc } n) \text{)) > N } \text{div } 2 \text{ and ct: commit (} \rho (\text{Suc } n) p \text{)} \\
&\text{show } ?Q (\text{Suc } n) \\
&\text{proof} \\
&\text{let } ?\text{msgs = rcvdMsgs } p (\text{HOs } n p) (\text{coords } n) (\rho n) (\text{send0 } n) \\
&\text{from stp run have } ?\text{commit (} \rho n p \text{) by (auto elim: commitE)} \\
&\text{with } \text{nxt } \text{ct obtain } q v \text{ where} \\
&v: ?\text{msgs } q = \text{Some } (\text{ValStamp } v (\text{highestStampRcvd } ?\text{msgs})) \text{ and} \\
&\text{vote: } \text{vote (} \text{rho } (\text{Suc } n) p \text{) = Some } v \text{ and} \\
&\text{rcvd: } \text{card (} \text{valStampsRcvd } ?\text{msgs} \text{) > } N \text{ div } 2 \\
&\text{by (auto simp add: next0-def)} \\
&\text{from mj rcvd obtain } q' \text{ where} \\
&q1': q' \in ?\text{bynd } (\text{Suc } n) \text{ and } q2': q' \in \text{valStampsRcvd } ?\text{msgs} \\
&\text{by (rule majoritiesE')} \\
&\text{have } \text{timestamp (} \rho n q' \text{) } \leq \text{timestamp (} \rho n q \text{)} \\
&\text{proof} \\
&\text{from } q2' \text{ obtain } v' ts' \text{ where ts': } ?\text{msgs } q' = \text{Some } (\text{ValStamp } v' ts') \\
&\text{by (auto simp add: valStampsRcvd-def)} \\
&\text{hence } ts' \leq \text{highestStampRcvd } ?\text{msgs} \\
&\text{by (rule highestStampRcvd-max)} \\
&\text{moreover}
\end{align*}
We now prove that next1 preserves the property. Observe that next1 may establish a majority of processes with current timestamps, so we cannot just refer to the induction hypothesis. However, if that happens, there is at least one process with a fresh timestamp that copies the vote of the (only) committed coordinator, thus establishing the property.

fix \( n \)
assume \( \text{stp}: \text{step } n = 1 \)
and \( \text{nxt}: \forall q. \text{next1 } n \ q (\rho n \ q) (\text{rcvdMsgs } q (\text{HOs } n \ q) (\text{coords } n) (\rho n) (\text{send1 } n)) (\text{coords } n \ q) (\rho (\text{Suc } n) \ q) (\text{is } \forall q. \ ?\text{nxt } q) \)
and \( \text{ih}: ?\text{P } p \ n \)

from \( \text{nxt} \) have \( ?\text{nxt } p \ .. \)
show \( ?\text{P } p \ (\text{Suc } n) \)

proof (clarify)
assume \( \text{mj}\': \text{card } (?\text{bynd } (\text{Suc } n)) > N \text{ div } 2 \) and \( \text{ct}\': \text{commit } (\rho (\text{Suc } n) \ p) \)
from \( \text{run stp } \text{ct}\' \) have \( \text{ct}: \text{commit } (\rho n \ p) \)
by (simp add: notStep03EqualCommit)
from \( \text{run stp} \) have \( \text{vote}\': \text{vote } (\rho (\text{Suc } n) \ p) = \text{vote } (\rho n \ p) \)
by (simp add: notStep0EqualVote)
show \( \text{?Q } (\text{Suc } n) \)
proof (cases \( \exists q \in ?\text{bynd } (\text{Suc } n). \rho (\text{Suc } n) \ q \neq \rho n \ q \))
case True
— in this case the property holds because \( q \) updates its \( x \) field to the vote
then obtain \( q \) where \( q1: q \in ?\text{bynd } (\text{Suc } n) \) and \( \text{q2}: \rho (\text{Suc } n) \ q \neq \rho n \ q \ .. \)
from \( \text{nxt} \) have \( ?\text{nxt } q \ .. \)
with q2
have \( x' \colon x (\rho (\text{Suc } n) q) = \text{the} (\text{vote} (\rho n (\text{coords } n q))) \)
and coord \colon \text{commit} (\rho n (\text{coords } n q))
by (auto simp add: next1-def send1-def rcvdMsgs-def isVote-def)
from run ct have \( \text{vote} (\rho n p) \neq \text{None} \) by (rule commitE)
from run coord ct have coords n q = p by (rule committedProcsEqual)
with q1 \( x' \) \( \text{vote} \) vote' show \( \text{thesis} \) by auto
next

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

hence bynd: \( \text{bynd} (\text{Suc } n) = \text{bynd } n \)
proof (auto simp add: \text{procsBeyondTS-def})
fix r
assume ts: \( ts \leq \text{timestamp} (\rho n r) \)
from run have timestamp (\rho n r) \leq timestamp (\rho (\text{Suc } n r))
by (simp add: LV-timestamp-monotonic)
with ts show ts \leq timestamp (\rho (\text{Suc } n r)) by simp
qed

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

next

step2 preserves the property, via the induction hypothesis.

fix n
assume stp: \( \text{step } n = 2 \)
and nxt: \( \forall q. \text{next2 } n q (\rho n q) (\text{rcvdMsgs } q (HOs n q) (\text{coords } n) (\rho n) (\text{send2 } n)) (\text{coords } n q) (\rho (\text{Suc } n) q) (\text{is} \forall q. \text{?nxt } q) \)
and ih: \( \text{?P } p \ n \)
from nxt have nxp: \( \text{?nxt } p \) ..
show \( \text{?P } p (\text{Suc } n) \)
proof (clarify)
assume mj': \( \text{card} (\text{bynd } (\text{Suc } n)) > N \text{ div } 2 \) and ct': \text{commit} (\rho (\text{Suc } n) p)
from run stp ct' have ct': \text{commit} (\rho n p)
by (simp add: notStep3EqualCommit)
from run stp have vote': \text{vote} (\rho (\text{Suc } n) p) = \text{vote} (\rho n p)
by (simp add: notStep0EqualVote)
from run stp have \( \forall q. \text{timestamp} (\rho n p q) = \text{timestamp} (\rho n p q) \)
by (simp add: notStep1EqualTimestamp)

hence bynd': \( \text{bynd} (\text{Suc } n) = \text{bynd } n \)
by (auto simp add: \text{procsBeyondTS-def})
from run stp have \( \forall q. x (\rho n q) = x (\rho n q) \)
by (simp add: notStep1EqualX)
with bynd' vote' ct mj' ih show \( \text{?Q } (\text{Suc } n) \)
by auto
qed

the initial state and the \text{step3} transition are trivial because the \text{commit} flag cannot be set.

qed (auto elim: commitE[OF \text{run}])
with maj cmt show \( \text{thesis} \) 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 hold the decision value in their \( x \) field.

**Lemma XOfTimestampBeyondDecision:**

assumes \( \text{run: CHORun rho HOs coords} \)
and \( \text{dec: decide (rho (Suc r) p) \neq decide (rho r p)} \)
shows \( \forall q \in \text{procsBeyondTS (Suc (phase r)) (rho (r+k)).} \)
\( x (rho (r+k) q) = \text{the (decide (rho (Suc r) p))} \)

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

**Proof** (induct)

— 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 \( \text{run dec} \)

have \( \text{stp: step r = 3} \)
and \( v: \text{decide (rho (Suc r) p) = Some (the (vote (rho r (coords r p))))} \)
and \( \text{cnt: commt (rho r (coords r p))} \)
by (auto elim: decisionE)
from \( \text{stp} \) \( \text{LV-timestamp-bounded[OF run, where n=r]} \)

have \( \text{timestamp (rho r q) \leq Suc (phase r) by simp} \)

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

with \( \text{run} \)

have \( x: x (rho r q) = \text{the (vote (rho r (coords r q)))} \)
and \( \text{cnt': commt (rho r (coords r q))} \)
by (auto elim: currentTimestampE)
from \( \text{run cnt cnt'} \) have \( \text{coords r p = coords r q by (rule committedProcsEqual)} \)

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

qed

— induction step

fix \( k \)

assume \( \text{ih: } ?P p k \)

show \( ?P p (Suc k) \)

**Proof (clarify)**

fix \( q \)
assume \( q: q \in ?bynd (Suc k) \)

— distinguish the kind of transition—only step1 is interesting

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

**Proof (rule LV-Suc[‘OF run, where P=?X])**

fix \( \text{HO} \)
assume \( \text{stp: step (r+k) = 1} \)
and \( \text{nxt: next1 (r+k) q (rho (r+k) q)} \)
\( \text{(recvMsgs q (HOs (r+k) q) (coords (r+k)) (rho (r+k)) (send1 (r+k))}}) \)
\( \text{(coords (r+k) q) (rho (Suc (r+k)) q)} \)

show \( \text{?thesis} \)

**Proof (cases rho (Suc (r+k)) q = rho (r+k) q)**

case True
with \( q \) \( \text{ih} \) show \( \text{?thesis by (auto simp add: procsBeyondTS-def)} \)
We are now in position to prove agreement: if some process decides at step \( r \) and another (or possibly the same) process decides at step \( r+k \) then they decide the same value.

**Lemma** \( \text{laterProcessDecidesSameValue} \):
- **Assumes** \( \text{run} : \text{CHORun} \ rho \ HOs \ coords \)
and \( p \): decide \((\rho \; \text{Suc} \; r) \; p\) \(\neq\) decide \((\rho \; r) \; p\)
and \( q \): decide \((\rho \; \text{Suc} \; (r+k)) \; q\) \(\neq\) decide \((\rho \; (r+k)) \; q\)

shows

decide \((\rho \; \text{Suc} \; (r+k)) \; q\) = decide \((\rho \; \text{Suc} \; r) \; p\)

proof –

let ?bynd \( k = \text{procsBeyondTS} \; (\text{Suc} \; (\text{phase} \; r)) \; (\rho \; (r+k))\)
let ?qcrd = \text{coords} \; (r+k) \; q

from run \( p \) have notNone: decide \((\rho \; \text{Suc} \; r) \; p\) \(\neq\) None
by (auto elim: decisionE)
— process \( q \) decides on the vote of its coordinator

from run \( q \) have dec: decide \((\rho \; \text{Suc} \; (r+k)) \; q\) = Some \((\text{the} \; (\text{vote} \; (\rho \; (r+k) \; ?qcrd)))\)
and \( \text{cnt}: \text{commit} \; (\rho \; (r+k) \; ?qcrd)\)
by (auto elim: decisionE)
— that vote is the \( x \) field of some process \( q' \) with a recent timestamp

from run \( p \) have \( \text{card} \; (?\text{bynd} \; 0) > \text{N div} \; 2 \)
by (simp add: \text{decisionThenMajorityBeyondTS})

moreover
from run \( p \) have \(?\text{bynd} \; 0 \subseteq ?\text{bynd} \; k\) by (auto elim: \text{procsBeyondTS-monotonic})

hence \( \text{card} \; (?\text{bynd} \; 0) \leq \text{card} \; (?\text{bynd} \; k)\) by (auto intro: \text{card-mono})

ultimately
have maj: \( \text{card} \; (?\text{bynd} \; k) > \text{N div} \; 2\) by simp

from run maj \( \text{cnt} \) obtain \( q' \) where
\( q' 1: q' \in ?\text{bynd} \; k \) and \( q' 2: \text{vote} \; (\rho \; (r+k) \; ?qcrd) = \text{Some} \; (x \; (\rho \; (r+k) \; q'))\)
by (auto dest: \text{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: \text{XOOfTimestampBeyondDecision})
— which proves the assertion

with \( \text{dec} \; q' 2 \) notNone show \( ?\text{thesis} \) by auto

qed

A process that holds some decision \( v \) has decided \( v \) sometime in the past.

lemma \text{decisionNonNullThenDecided}:

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

shows \( \exists \; m < n. \; \text{decide} \; (\rho \; (\text{Suc} \; m) \; p) \neq \text{decide} \; (\rho \; m \; p) \)

\& \text{decide} \; (\rho \; (\text{Suc} \; m) \; p) = \text{Some} \; v

proof –

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

have \((\forall \; m < n. \; \text{dec} \; (\text{Suc} \; m) \neq \text{dec} \; (\text{Suc} \; m) \neq \text{Some} \; v) \longrightarrow ?\text{dec} \; m \neq \text{Some} \; v\)
(is \( ?P \; n \) is \( ?A \; n \longrightarrow :)\)

proof (induct \( n \))
from run show \( ?P \; 0\) by (auto simp add: \text{CHORun-def initConfig-def initState-def})

next

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

proof (clarify)

assume \( p: ?A \; (\text{Suc} \; n) \) and \( v: ?\text{dec} \; (\text{Suc} \; n) = \text{Some} \; v\)
from \( p \) have \( ?A \; n \) by simp
with \( \text{ih} \) have \( ?\text{dec} \; n \neq \text{Some} \; v \) by simp

moreover
from \( p \) have \( ?\text{dec} \; (\text{Suc} \; n) \neq \text{dec} \; (\text{Suc} \; n) \neq \text{Some} \; v \) by simp

ultimately
have \( ?\text{dec} \; (\text{Suc} \; n) \neq \text{Some} \; v \) by auto
with \( v \) show \( \text{False} \) by simp

qed
Irrevocability and Agreement follow as easy consequences.

theorem irrevocability:
  assumes run: CHORun rho HOs coords
  and p: decide (rho m p) = Some v
  shows decide (rho (m+k) p) = Some v
proof –
  from run p obtain n where 
    n1: n < m and 
    n2: decide (rho (Suc n) p) ≠ decide (rho n p) and 
    n3: decide (rho (Suc n) p) = Some v 
  by (auto dest: decisionNonNullThenDecided)
  have ∀ i. decide (rho (Suc (n+i)) p) = Some v (is ∀ i. ∀ dec i)
proof
  fix i
  show ∀ dec i
    proof (induct i)
    from n3 show ∀ dec 0 by simp
    next
    fix j
    assume ih: ∀ dec j
    show ∀ dec (Suc j)
      proof (rule ccontr)
        assume ctr: ¬ (∀ dec (Suc j))
        with ih have decide (rho (Suc (n + Suc j)) p) ≠ decide (rho (n + Suc j) p) 
          by simp
        with run n2 have decide (rho (Suc (n + Suc j)) p) = decide (rho (Suc n) p) 
          by (rule laterProcessDecidesSameValue)
        with ctr n3 show False by simp
      qed 
    qed 
  qed
  qed
  moreover 
  from n1 obtain j where m+k = Suc(n+j)
  by (auto dest: less-imp-Suc-add)
ultimately 
  show ∀ thesis by auto
qed

theorem agreement:
  assumes run: CHORun rho HOs coords
  and p: decide (rho m p) = Some v and q: decide (rho n q) = Some w
  shows v = w
proof –
  from run p obtain k where
    k1: decide (rho (Suc k) p) ≠ decide (rho k p) and 
    k2: decide (rho (Suc k) p) = Some v
  by (auto dest: decisionNonNullThenDecided)
  from run q obtain l where
    l1: decide (rho (Suc l) q) ≠ decide (rho l q) and 
    l2: decide (rho (Suc l) q) = Some w
  by (auto dest: decisionNonNullThenDecided)
  show ∀ thesis
proof (cases \(k \leq l\))

  case True
  then obtain \(m\) where \(m = k + m\) by (auto simp add: le_iff_add)
  from run \(k\) \(l\) \(m\) have decide \((\rho\ (Suc\ l)\ q) = decide\ (\rho\ (Suc\ k)\ p)\)
  by (auto elim: laterProcessDecidesSameValue)
  with \(k2\ l2\) show \(?thesis\ by\ simp\)

next

  case False
  hence \(l \leq k\) by simp
  then obtain \(m\) where \(m = k + m\) by (auto simp add: le_iff_add)
  from run \(l\) \(k1\) \(m\) have decide \((\rho\ (Suc\ k)\ p) = decide\ (\rho\ (Suc\ l)\ q)\)
  by (auto elim: laterProcessDecidesSameValue)
  with \(l2\ k2\) show \(?thesis\ by\ simp\)

qed

2.7 Proof of liveness

We now show that the communication predicate ensures termination of the algorithm: there exists some round \(r\) at which all processes have decided. In fact, the assumption ensures the existence of some phase during which there is a single coordinator that receives a majority of messages. Moreover, all processes receive the messages sent by the coordinator and therefore successfully execute the protocol, deciding at step 3 of that phase.

theorem decision:
  assumes run: \(\text{CHORun\ \rho\ HOs\ coords}\)
  shows \(\exists\ r.\ \forall\ p.\ \text{decide}\ (\rho\ r\ p) \neq \text{None}\)

proof –

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

from run obtain \(ph\ c\)
  where \(c\): \(\forall\ p.\ \text{coords}\ (4 * ph)\ p = c\)
  and maj0: \(\text{card}\ (\text{HOs}\ (4 * ph)\ c) > N\ div\ 2\)
  and maj2: \(\text{card}\ (\text{HOs}\ (\text{Suc}\ (\text{Suc}\ (4 * ph)))\ c) > N\ div\ 2\)
  and recv1: \(\forall\ p.\ c \in (\text{HOs}\ (\text{Suc}\ (4 * ph)))\ p\)
  and recv3: \(\forall\ p.\ c \in (\text{HOs}\ (\text{Suc}\ (\text{Suc}\ (\text{Suc}\ (4 * ph))))\ p)\)
  by (auto simp add: CHORun-def LV-commLive-def)

let \(?r = 4 * ph\)

let \(?r1 = \text{Suc}\ ?r\)
let \(?r2 = \text{Suc}\ (\text{Suc}\ ?r)\)
let \(?r3 = \text{Suc}\ (\text{Suc}\ (\text{Suc}\ ?r))\)
let \(?r4 = \text{Suc}\ (\text{Suc}\ (\text{Suc}\ (\text{Suc}\ ?r))))\)

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

from run \(c\) have c1: \(\forall\ p.\ \text{coords}\ ?r1\ p = c\)
  by (auto simp add: step-def notStep3EqualCoord)

with run \(c\) have c2: \(\forall\ p.\ \text{coords}\ ?r2\ p = c\)
  by (auto simp add: step-def mod-Suc notStep3EqualCoord)

with run \(c\) have c3: \(\forall\ p.\ \text{coords}\ ?r3\ p = c\)
  by (auto simp add: step-def mod-Suc notStep3EqualCoord)

The coordinator receives \(\text{ValStamp}\) 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: \(\text{commt}\ (\rho\ ?r1\ c)\ \text{is}\ \text{P}\ c\ (4 * ph))\)
proof (rule LV-Suc [OF run, where \( P=?P \)], auto simp add: step-def)
  assume next0 ?r c (rho ?r c) (rcvdMsgs c (HOs ?r c) (coords ?r) (rho ?r) (send0 ?r))
  with c maj0 show commt (rho (Suc ?r) c)
  by (auto simp add: next0-def send0-def valStampsRcvd-def rcvdMsgs-def)
qed

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

have 2: \( \forall p. \) timestamp (rho ?r2 p) = Suc ph
proof
  fix p
  let ?msgs = rcvdMsgs p (HOs ?r1 p) (coords ?r1) (rho ?r1) (send1 ?r1)
  let ?crd = coords ?r1 p
  from run 1 c1 rcv1 have end: ?msgs ?crd \( \neq \) None \& isVote (the (?msgs ?crd))
    by (auto elim: commitE simp add: rcvdMsgs-def send1-def isVote-def)
  show timestamp (rho ?r2 p) = Suc ph (is ?P p (Suc (4*ph)))
  proof (rule LV-Suc [OF run, where \( P=?P \)], auto simp add: step-def mod-Suc)
    assume next1 ?r1 p (rho ?r1 p) ?msgs ?crd (rho ?r2 p)
    with end show ?thesis
    by (auto simp add: next1-def phase-def)
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: \( \forall p. \) decide (rho ?r4 p) \( \neq \) None
proof
  fix p
  let ?msgs = rcvdMsgs p (HOs ?r3 p) (coords ?r3) (rho ?r3) (send3 ?r3)
  let ?crd = coords ?r3 p
  from run 3 c3 rcv3 have end: ?msgs ?crd \( \neq \) None \& isVote (the (?msgs ?crd))
    by (auto elim: readyE simp add: rcvdMsgs-def send3-def isVote-def)
  show decide (rho ?r4 p) \( \neq \) None (is ?P p (Suc (Suc (4*ph))))
  proof (rule LV-Suc [OF run, where \( P=?P \)], auto simp add: step-def mod-Suc)
    assume next2 ?r2 c (rho ?r2 c) (rcvdMsgs c (HOs ?r2 c) (coords ?r2) (rho ?r2) (send2 ?r2))
    with 2 c2 maj2 show ?thesis
    by (auto simp add: next2-def send2-def rcvdMsgs-def acksRcvd-def isAck-def phase-def)
qed

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

have 4: \( \forall p. \) decide (rho ?r4 p) = None
proof
  fix p
  let ?crd = coords ?r3 p
  from run 3 c3 rcv3 have end: ?msgs ?crd \( \neq \) None \& isVote (the (?msgs ?crd))
    by (auto elim: readyE simp add: rcvdMsgs-def send3-def isVote-def)
  show decide (rho ?r4 p) \( \neq \) None (is ?P p (Suc (Suc (4*ph))))
  proof (rule LV-Suc [OF run, where \( P=?P \)], auto simp add: step-def mod-Suc)
    assume next3 ?r3 p (rho ?r3 p) ?msgs ?crd (rho ?r4 p)
    with end show \( \exists v. \) decide (rho ?r4 p) = Some v
    by (auto simp add: next3-def)
qed

This immediately proves the assertion.

from 4 show ?thesis ..
qed
References
