Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff    Stephan Merz

September 19, 2015

Abstract

Disk Paxos [GL00] is an algorithm for building arbitrary fault-tolerant distributed systems. The specification of Disk Paxos has been proved correct informally and tested using the TLC model checker, but up to now, it has never been fully formally verified. In this work we have formally verified its correctness using the Isabelle theorem prover and the HOL logic system [NPW02], showing that Isabelle is a practical tool for verifying properties of TLA$^+$ specifications.

Contents

1 Introduction 2
2 The Disk Paxos Algorithm 3
   2.1 Informal description of the algorithm 4
   2.2 Disk Paxos and its TLA$^+$ Specification 4
3 Translating from TLA$^+$ to Isabelle/HOL 6
   3.1 Typed vs. Untyped 6
   3.2 Primed Variables 8
   3.3 Restructuring the specification 8
4 Structure of the Correctness Proof 9
   4.1 Going from Informal Proofs to Formal Proofs 10
5 Conclusion 11
A TLA$^+$ correctness specification 12
B Disk Paxos Algorithm Specification 13
1 Introduction

Algorithms for fault-tolerant distributed systems were first introduced to implement critical systems. Nevertheless, what good is such an algorithm if it has a design error? We need some kind of guarantee that the algorithm does not have a faulty design. Formal verification of its specification is one such guarantee.

Disk Paxos [GL00] is an algorithm for building arbitrary fault-tolerant distributed systems, that, due to its complexity, is difficult to reason about. It has been proved correct informally, and tested using the TLC model checker [Lam02]. The informal proof is rigorous but, as it is always the case with large informal proofs, it is easy to overlook details. Thus, one of the motivations of this work is to see if such a rigorous proof can be formalized in a contemporary theorem prover.

In [Pac01] part of the correctness proof (invariance of $HInv_1$ and $HInv_3$) was verified using the theorem prover ACL2 [KMM00]. An implicit assumption of this formalization is that all sets are finite, thus overlooking the fact that there is a missing conjunct in the Disk Paxos invariant (see Section 4).

We set the goal of formally verifying Disk Paxos correctness using the theorem prover Isabelle/HOL [NPW02]. In this way, we could gain more confidence in the correctness of Disk Paxos design and, at the same time, learn to what extent can Isabelle be a useful tool for proving the correctness of distributed systems using a real world example.
In Section 2 we give a brief description of the algorithm and its specification. In Section 3 we describe the translation from TLA\(^+\) to Isabelle/HOL and the problems that this translation originated. In Section 4 we discuss how our formal proofs relate to the informal ones in [GL00], and in Section 5 we conclude. The entire specification and all formal proofs can be found in the Appendix.

2 The Disk Paxos Algorithm

Disk Paxos is a variant of the classic Paxos algorithm [Lam98] for the implementation of arbitrary fault-tolerant systems with a network of processors and disks. It maintains consistency in the event of any number of non-Byzantine failures. This means that a processor may fail completely or pause for arbitrary long periods and then restart, remembering only that it has failed. A disk may become inaccessible to some or all processors, but it may not be corrupted. We say that a system is stable if all processes are either non-faulty or have failed completely (i.e. there are no new failed processes). Disk Paxos guarantees progress if the system is stable and there is at least one non-faulty processor that can read and write a majority of the disks. Consequently, the fundamental difference between Classic Paxos and Disk Paxos is that the former achieves redundancy by replicating processes while the latter replicates disks. Since disks are usually cheaper than processors, it is possible to obtain more redundancy at a lower cost.

Disk Paxos uses the state machine approach to solve the problem of implementing an arbitrary distributed system. The state machine approach [Sch90] is a general method that reduces this problem to solving a consensus problem. The distributed system is designed as a deterministic state machine that executes a sequence of commands, and a consensus algorithm ensures that, for each n, all processors agree on the n\(^{th}\) command. Hence, each processor p starts with an input value (a command), and it may output a value (the command to be executed). The problem is solved if all processors eventually output the same value and this value was a value of input[p] for some p (under certain assumptions, in our case that there exists at least one non-faulty processor that can write and read a majority of disks).

Progress of Disk Paxos relies on progress of a leader-election algorithm. It is easy to make a leader-election algorithm if the system stable, but very hard to devise one that works correctly even if the system is unstable. We are requiring stability to ensure progress, but actually only a weaker requirement is needed: progress of the underlying leader-election algorithm. Disk Paxos ensures that all outputs (if any) will be the same even if the leader-election algorithm fails.
2.1 Informal description of the algorithm.

The consensus algorithm of Disk Paxos is called Disk Synod. In it, each processor has an assigned block on each disk. Also it has a local memory that contains its current block (called the dblock), and other state variables (see figure 1). When a process \( p \) starts it contains an input value \( \text{input}[p] \) that will not be modified, except possibly when recovering from a failure.

Disk Synod is structured in two phases, plus one more phase for recovering from failures. In each phase, a processor writes its own block and reads each other processor’s block, on a majority of the disks. The idea is to execute ballots to determine:

**Phase 1:** whether a processor \( p \) can choose its own input value \( \text{input}[p] \) or must choose some other value. When this phase finishes a value \( v \) is chosen.

**Phase 2:** whether it can commit \( v \). When this phase is complete the process has committed value \( v \) and can output it (using variable \( \text{outpt} \)).

In either phase, a processor aborts its ballot if it learns that another processor has begun a higher-numbered ballot. The third phase (Phase 0) is for starting the algorithm or recovering from a failure.

In each block, processors maintain three values:

- **mbal**: The current ballot number.
- **bal**: The largest ballot number for which the processor entered phase 2.
- **inp**: The value the processor tried to commit in ballot number \( \text{bal} \).

For a complete description of the algorithm, see [GL00].

2.2 Disk Paxos and its TLA\(^+\) Specification

The specification of Disk Paxos is written in the TLA\(^+\) specification language [Lam02]. As it is usual with TLA\(^+\), the specification is organized into modules.

The specification of consensus is given in module Synod, which can be found in appendix A. In it there are only two variables: \( \text{input} \) and \( \text{output} \). To formalize the property stating that all processors should choose the same value and that this value should have been an input of a processor, we need variables that represent all past inputs and the value chosen as result. Consequently, an Inner submodule is introduced, which adds two variables: allInput and chosen. Our Synod module will be obtained by existentially quantifying these variables of the Inner module.

The specification of the algorithm is given in the HDiskSynod module. Hence, what we are going to prove is that the (translation to Isabelle/HOL
of the) Inner module is implied by the (translation to Isabelle/HOL of the) algorithm module HDiskSynod.

More concretely we have that the specification of the algorithm is:

$$\text{HDiskSynodSpec} \triangleq HInit \land \square[HNext]_{(vars, chosen, allInput)}$$

where $HInit$ describes the initial state of the algorithm and $HNext$ is the action that models all of its state transitions. The variable $vars$ is the tuple of all variables used in the algorithm.

Analogously, we have the specification of the Inner module:

$$\text{ISpec} \triangleq IInit \land \square[INext]_{(input, output, chosen, allInput)}$$

We define $ivars = (input, output, chosen, allInput)$. In order to prove that HDiskSynodSpec implies ISpec, we follow the structure of the proof given by Gafni and Lamport. We must prove two theorems:

**THEOREM R1** $HInit \Rightarrow IInit$

**THEOREM R2** $HInit \land \square[HNext]_{(vars, chosen, allInput)} \Rightarrow \square[INext]_{ivars}$

The proof of $R1$ is trivial. For $R2$, we use TLA proof rules [Lam02] that show that to prove $R2$, it suffices to find a state predicate $HInv$ for which we can prove:

**THEOREM R2a** $HInit \land \square[HNext]_{(vars, chosen, allInput)} \Rightarrow \square[HInv]$

**THEOREM R2b** $HInv \land HInv' \land HNext \Rightarrow INext \lor (\text{UNCHANGED} \ ivars)$

A predicate satisfying $HInv$ is said to be an invariant of HDiskSynodSpec. To prove $R2a$, we make $HInv$ strong enough to satisfy:
Again, we have TLA proof rules that say that $I_1$ and $I_2$ imply $R2a$. In summary, $R2b$, $I_1$, and $I_2$ together imply $\text{HDiskSynodSpec} \Rightarrow \text{ISpec}$.

Finding a predicate $HInv$ that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present $HInv$ as a conjunction of 6 predicates $HInv_1, \ldots, HInv_6$, where $HInv_1$ is a simple “type invariant” and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of $HInv_i$ by the algorithm’s next-state relation relies on all $HInv_j$ (for $j \leq i$) being true in the state before the transition. In our proofs we are going to use exactly the same tactic.

Before starting our proofs we have to translate all the specification and theorems above into the formal language of Isabelle/HOL.

### 3 Translating from TLA$^+$ to Isabelle/HOL

The translation from TLA$^+$ to Isabelle/HOL is pretty straightforward as Isabelle/HOL has equivalent counterparts for most of the constructs in TLA$^+$ (some representative examples are shown in table 1). Nevertheless, there are some semantic discrepancies. In the following, we discuss these differences, some of the options that one has when dealing with them, and the reasons for our choices$^1$.

#### 3.1 Typed vs. Untyped

TLA$^+$ is an untyped formalism. However, TLA$^+$ specifications usually have some type information, usually in the form of set membership or set inclusion. When translating these specifications to Isabelle/HOL, which is a typed formalism, one has to invent types that represent these sets of values.

---

$^1$There is no point in using the existing TLA encoding in Isabelle. Since the encoding is also based on HOL and we only prove safety, we would have gained nothing.
TLA⁺:

\[
\begin{align*}
\text{CONSTANT } & \text{Inputs} \\
\text{NotAnInput} & \triangleq \text{CHOOSE } c : c \not\in \text{Inputs} \\
\text{DiskBlock} & \triangleq \{m \in \text{Ballot} : p \in \text{Proc} \cup \{0\}, \\
& \quad \{b \in \text{Ballot} : p \in \text{Proc} \cup \{0\}, \\
& \quad \{i \in \text{Inputs} \cup \{\text{NotAnInput}\}\}
\end{align*}
\]

Isabelle/HOL:

\[
\begin{align*}
\text{typedef } & \text{InputsOrNi} \\
\text{consts} & \text{Inputs :: InputsOrNi set} \\
& \text{NotAnInput :: InputsOrNi} \\
\text{axioms} & \text{NotAnInput} \cdot \text{NotAnInput} \not\in \text{Inputs} \\
& \text{InputsOrNi} \cdot (\text{UNIV :: InputsOrNi set}) = \text{Inputs} \cup \{\text{NotAnInput}\}
\end{align*}
\]

\[
\begin{align*}
\text{record} & \text{DiskBlock} = \\
& \{m \in \text{nat} \\
& \quad b \in \text{nat} \\
& \quad i \in \text{Inputs} \cup \{\text{NotAnInput}\}
\end{align*}
\]

Figure 2: Untyped TLA⁺ vs. Typed Isabelle/HOL

This process is not automatic and requires some thought to find the right abstractions. Furthermore, simple types may not be expressive enough to represent exactly the set in the specification. In some cases, these sets could be modelled by algebraic datatypes, but this would make the specification more complex and less directly related to the original one. In this work, we have chosen to stick to simple types and add additional axioms to account for their lack of expressiveness.

For example, see figure 2. The type \text{InputsOrNi} models the members of the set \text{Inputs}, and the element \text{NotAnInput}. We record the fact that \text{NotAnInput} is not in \text{Inputs}, with axiom \text{NotAnInput}. Now, looking at the type of the \text{inp} field of the \text{DiskBlock} record in the TLA⁺ specification, we see that its type should be \text{InputsOrNi}. However, this is not the same type as \text{Inputs} \cup \{\text{NotAnInput}\}, as nothing prevents the \text{InputsOrNi} type from having more values. Consequently, we add the axiom \text{InputsOrNi} to establish that the only values of this type are the ones in \text{Inputs} and \text{NotAnInput}.

This example shows the kind of difficulties that can arise when trans-
TLA$^+$:

$$\text{Phase1or2Write}(p, d) \equiv$$
$$\land \ phase[p] \in \{1, 2\}$$
$$\land \ disk'[d] = [\text{disk except } ![d][p] = \text{dblock}[p]]$$
$$\land \ disksWritten'[d] = [\text{disksWritten except } ![d][p] = @ \cup \{d\}]$$
$$\land \ \text{UNCHANGED } (\text{input, output, phase, dblock, blocksRead})$$

Isabelle/HOL:

$$\text{Phase1or2Write} :: \text{state } \Rightarrow \text{state } \Rightarrow \text{Proc } \Rightarrow \text{Disk } \Rightarrow \text{bool}$$
$$\text{Phase1or2Write } s \ s' p d \equiv$$
$$\land \ disk'[d] = (\text{disk } s) (d := (\text{disk } s d) (p := \text{dblock } s p))$$
$$\land \ disksWritten'[d] = (\text{disksWritten } s) (p := (\text{disksWritten } s p) \cup \{d\})$$
$$\land \ \text{inpt } s' = \text{inpt } s \land \ \text{outpt } s' = \text{outpt } s$$
$$\land \ \text{phase } s' = \text{phase } s \land \ \text{dblock } s' = \text{dblock } s$$
$$\land \ \text{blocksRead } s' = \text{blocksRead } s$$

Figure 3: Translation of an action

3.2 Primed Variables

In TLA$^+$, to denote the value of a variable in the state resulting from an action, there exists the concept of primed variables. In Isabelle/HOL, there is no built-in notion of state, so we define the state as a record of the variables involved. Instead of defining a “priming” operator, we just define actions as predicates that take any two states, intended to be the previous state and the next state. In this way, $P_s s'$ will be true iff executing an action $P$ in the $s$ state could result in the $s'$ state. In figure 3 we can see how the action $\text{Phase1or2Write}$ is expressed in TLA$^+$ and in Isabelle/HOL.

3.3 Restructuring the specification

There are some minor changes that can be made to specifications to make the proofs in Isabelle easier.

One such change is the elimination of LET constructs by making them global definitions. This has two benefits, as it makes it possible to:

- Formulate lemmas that use these definitions.
- Selectively unfold these definitions in proofs, instead of adding Let-def to Isabelle’s simplifier, which unfolds all “let” constructs.
Another change that makes proofs easier is to break down actions into simpler actions (e.g. giving separate definitions for subactions corresponding to disjuncts). By making actions smaller, Isabelle has to deal with smaller formulas when we feed the prover with such an action.

For example, \textit{Phase1or2Read} is mainly a big if-then-else. We break it down into two simpler actions:

\[ \text{Phase1or2Read} \overset{\Delta}{=} \text{Phase1or2ReadThen} \lor \text{Phase1or2ReadElse} \]

In \textit{Phase1or2ReadThen} the condition of the if-then-else is present as a state formula (i.e. it is an enabling condition) while in \textit{Phase1or2ReadElse} we add the negation of this condition.

Another example is \textit{HInv2}, which we break down into:

\[ \text{HInv2} \overset{\Delta}{=} \text{Inv2a} \land \text{Inv2b} \land \text{Inv2c} \]

Not only we break it down into three conjuncts but, since these conjuncts are quantifications over some predicate, we give a separate definition for this predicate. For example for \textit{Inv2a}, and after translating to Isabelle/HOL, instead of writing:

\[
\text{Inv2a} \ s \equiv \forall \ p. \ \forall \ bk \in \text{blocksOf} \ s \ p \ldots
\]

we write:

\[
\begin{align*}
\text{Inv2a-innermost} & \ :: \ \text{state} \Rightarrow \ \text{Proc} \Rightarrow \ \text{DiskBlock} \Rightarrow \ \text{bool} \\
\text{Inv2a-innermost} \ s \ p \ bk & \equiv \ldots \\
\text{Inv2a-inner} & \ :: \ \text{state} \Rightarrow \ \text{Proc} \Rightarrow \ \text{bool} \\
\text{Inv2a-inner} \ s \ p & \equiv \forall \ bk \in \text{blocksOf} \ s \ p. \ \text{Inv2a-innermost} \ s \ p \ bk \\
\text{Inv2a} & \ :: \ \text{state} \Rightarrow \ \text{bool} \\
\text{Inv2a} \ s & \equiv \forall \ p. \ \text{Inv2a-inner} \ s \ p
\end{align*}
\]

Now we can express that we want to obtain the fact

\[
\text{Inv2a-innermost} \ s \ q \ (\text{dblock} \ s \ q)
\]

explicitly stating that we are interested in predicate \textit{Inv2a}, but only for some process \( q \) and block \( \text{dblock} \ s \ q \).

\section{Structure of the Correctness Proof}

In \cite{GL00}, a specification of correctness and a specification of the algorithm are given. Then, it is proved that the specification of the algorithm implies the specification of correctness. We will do the same for the translated specifications, maintaining the names of theorems and lemmas. It should be noted that only safety properties are given and proved.
4.1 Going from Informal Proofs to Formal Proofs

There are informal proofs for invariants $HInv_3-HInv_6$ and for theorem $R2b$ in [GL00]. These informal proofs are written in the structured-proof style that Lamport advocates [Lam95], and are rigorous and quite detailed. We based our formal proofs on these informal proofs, but in many cases a higher level of detail was needed. In some cases, the steps where too big for Isabelle to solve them automatically, and intermediate steps had to be proved first; in other cases, some of the facts relevant to the proofs were omitted in the informal proofs.

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase_1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv_4$ and $HInv_5$ hold in the previous state to prove lemma $I2f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate $I$ was an invariant of $Next$, we preferred proving the invariance of $I$ for each action, rather than a big theorem proving the invariance of $I$ for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $HInv_3$ for the $EndPhase_0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

The informal proofs start working with $Next$, and then do a case split in which each case implies some action. The structure of the formal proof was copied from the informal one, in the parts where the latter focused on a particular action. This structure could be easily maintained since we used Isabelle’s Isar proof language [Wen02, Nip03], a language for writing human-readable structured proofs.

Lamport’s use of a hierarchical scheme for naming subformulas of a formula would have been very useful, as we have to repeatedly extract subfacts from facts. These proofs were always solved automatically by the auto Isabelle tactic, but made the proofs longer and harder to understand. Automatic naming of subformulas of Isabelle definitions would be a very practical feature.
5 Conclusion

We formally verified the correctness of the Disk Paxos specification in Isabelle/HOL. We found some omissions in the informal proofs, including one that could not have been detected with finite-state model checking, but no outright errors. This formal proof gives us a greater confidence in the correctness of the algorithm.

This work was done in little more than two months, including learning Isabelle from scratch. Consequently, this work can be taken as evidence that formal verification of fault-tolerant distributed algorithms may be feasible for software verification in an industrial context.

Isabelle proved to be a very helpful tool for this kind of verification, although it would have been useful to have Lamport’s naming of subfacts to make proofs shorter and easier to write.

References


A TLA+ correctness specification

---

MODULE Synod

EXTENDS Naturals
CONSTANT N, Inputs
ASSUME (N ∈ Nat) ∧ (N > 0)
Proc ≜ 1..N
NotAnInput ≜ CHOOSE c: c ∉ Inputs
VARIABLES inputs, output

---

MODULE Inner

VARIABLES allInput, chosen

---

IInit ≜ ∧ input ∈ [Proc → Inputs]
∧ output = [p ∈ Proc → NotAnInput]
∧ chosen = NotAnInput
∧ allInput = input[p]: p ∈ Proc

IChoose(p) ≜ ∧ output[p] = NotAnInput
∧ IF chosen = NotAnInput
    THEN Ip ∈ allInput: ∧ chosen’ = ip
    ∧ output’ = [output except ![p] = ip]
ELSE ∧ output’ = [output except ![p] = chosen]
∧ UNCHANGED chosen
∧ UNCHANGED ⟨input, allInput⟩

IFail(p) ≜ ∧ output’ = [output except ![p] = NotAnInput]
∧ ∃ ip ∈ Inputs: ∧ input’ = [input except ![p] = ip]
∧ allInput’ = allInput ∪ {ip}

INext ≜ ∃ p ∈ Proc: IChoose(p) ∨ IFail(p)
ISpec ≜ IInit ∧ □[INext](input, output, chosen, allInput)

IS(chosen, allInput) ≜ INSTANCE Inner
SynodSpec ≜ ∃ chosen, allInput: IS(chosen, allInput)!ISpec
B Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedec InputsOrNi

typedec Disk

typedec Proc

axiomatization
 Inputs :: InputsOrNi set and
 NotAnInput :: InputsOrNi and
 Ballot :: Proc ⇒ nat set and
 IsMajority :: Disk set ⇒ bool

where
 NotAnInput: NotAnInput ∉ Inputs and
 InputsOrNi: (UNIV :: InputsOrNi set) = Inputs ∪ {NotAnInput} and
 Ballot-nzero: ∀ p. 0 ∉ Ballot p and
 Ballot-disj: ∀ p q. p ≠ q → (Ballot p) ∩ (Ballot q) = {} and
 Disk-isMajority: IsMajority(UNIV) and
 majorities-intersect:
 ∃ S T. IsMajority(S) ∧ IsMajority(T) → S ∩ T ≠ {}

lemma ballots-not-zero [simp]:
 b ∈ Ballot p → 0 < b
 ⟨proof⟩

lemma majority-nonempty [simp]: IsMajority(S) → S ≠ {}
 ⟨proof⟩

definition AllBallots :: nat set
 where AllBallots = (UN p. Ballot p)

record
 DiskBlock =
  mbal :: nat
  bal :: nat
  inp :: InputsOrNi

definition InitDB :: DiskBlock
 where InitDB = [| mbal = 0, bal = 0, inp = NotAnInput |]

record
 BlockProc =
  block :: DiskBlock
  proc :: Proc

record
 state =
\[
\begin{align*}
inpt &:: \text{Proc} \Rightarrow \text{InputsOrNi} \\
\text{outpt} &:: \text{Proc} \Rightarrow \text{InputsOrNi} \\
disk &:: \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \\
dblock &:: \text{Proc} \Rightarrow \text{DiskBlock} \\
\text{phase} &:: \text{Proc} \Rightarrow \text{nat} \\
disksWritten &:: \text{Proc} \Rightarrow \text{Disk set} \\
\text{blocksRead} &:: \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{BlockProc set} \\
\text{allInput} &:: \text{InputsOrNi set} \\
\text{chosen} &:: \text{InputsOrNi}
\end{align*}
\]

**definition** *hasRead* :: state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where
*hasRead* s p d q = (∃ br ∈ blocksRead s p d. proc br = q)

**definition** *allRdBlks* :: state ⇒ Proc ⇒ BlockProc set
where
*allRdBlks* s p = (\(UN\ d\). blocksRead s p d)

**definition** *allBlocksRead* :: state ⇒ Proc ⇒ DiskBlock set
where
*allBlocksRead* s p = block (\(allRdBlks\ s\ p\))

**definition** *Init* :: state ⇒ bool
where
\[
\begin{align*}
\text{Init} \ s & = \\
& (\text{range} \ (\text{inpt} \ s) \subseteq \text{Inputs} \\
& \& \ \text{outpt} \ s = (\lambda p. \text{NotAnInput}) \\
& \& \ \text{disk} \ s = (\lambda d p. \text{InitDB}) \\
& \& \ \text{phase} \ s = (\lambda p. 0) \\
& \& \ \text{dblock} \ s = (\lambda p. \text{InitDB}) \\
& \& \ \text{disksWritten} \ s = (\lambda p. \{\}) \\
& \& \ \text{blocksRead} \ s = (\lambda p d. \{\})
\end{align*}
\]

**definition** *InitializePhase* :: state ⇒ state ⇒ Proc ⇒ bool
where
\[
\begin{align*}
\text{InitializePhase} \ s \ s' \ p & = \\
& (\text{disksWritten} \ s' = (\text{disksWritten} \ s)(p := \{\}) \\
& \& \ \text{blocksRead} \ s' = (\text{blocksRead} \ s)(p := (\lambda d. \{\})))
\end{align*}
\]

**definition** *StartBallot* :: state ⇒ state ⇒ Proc ⇒ bool
where
\[
\begin{align*}
\text{StartBallot} \ s \ s' \ p & = \\
& (\text{phase} \ s p \in \{1,2\} \\
& \& \ \text{phase} \ s' = (\text{phase} \ s)(p := 1) \\
& \& \ (\exists b \in \text{Ballot} \ p. \\
& \quad \text{mbal} \ (\text{dblock} \ s p) < b \\
& \quad \& \ \text{dblock} \ s' = (\text{dblock} \ s)(p := (\text{dblock} \ s p)(\text{mbal} := b \ |)) \\
& \& \ \text{InitializePhase} \ s \ s' \ p \\
& \& \ \text{inpt} \ s' = \text{inpt} \ s \& \ \text{outpt} \ s' = \text{outpt} \ s \& \ \text{disk} \ s' = \text{disk} \ s)
\end{align*}
\]
\textbf{definition} \texttt{Phase1or2Write} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{Disk} \Rightarrow \texttt{bool} \\
\textbf{where} \\
\texttt{Phase1or2Write} \ s \ s' \ p \ d \ = \\
(\text{phase} \ s \ p \in \{1, \ 2\}) \land \ (\text{disk} \ s' = (\text{disk} \ s) \ (d := (\text{disk} \ s \ d) \ (p := \text{dblock} \ s \ p))) \land \\
(\text{disksWritten} \ s' = (\text{disksWritten} \ s) \ (p := (\text{disksWritten} \ s \ p) \cup \{d\})) \land \\
(\text{inpt} \ s' = \text{inpt} \ s \land \ (\text{outpt} \ s' = \text{outpt} \ s)) \land \\
(\text{phase} \ s' = \text{phase} \ s \land \ (\text{dblock} \ s' = \text{dblock} \ s) \land \ (\text{blocksRead} \ s' = \text{blocksRead} \ s)) \\
\textbf{definition} \texttt{Phase1or2ReadThen} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{Disk} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\textbf{where} \\
\texttt{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \ = \\
(d \in \text{disksWritten} \ s \ p) \land \\
(\text{mbal} \ (\text{disk} \ s \ d \ q) < \text{mbal} \ (\text{dblock} \ s \ p)) \land \\
(\text{blocksRead} \ s' = (\text{blocksRead} \ s) \ (p := (\text{blocksRead} \ s \ p) \ (d := \\
(\text{blocksRead} \ s \ p \ d) \cup \{\ (\text{block} = \text{disk} \ s \ d \ q, \ \\
\text{proc} = q)\})) \land \\
(\text{inpt} \ s' = \text{inpt} \ s \land (\text{outpt} \ s' = \text{outpt} \ s)) \land \\
(\text{disk} \ s' = \text{disk} \ s \land (\text{phase} \ s' = \text{phase} \ s) \land \\
(\text{dblock} \ s' = \text{dblock} \ s \land \ (\text{disksWritten} \ s' = \text{disksWritten} \ s)) \\
\textbf{definition} \texttt{Phase1or2ReadElse} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{Disk} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\textbf{where} \\
\texttt{Phase1or2ReadElse} \ s \ s' \ p \ d \ q \ = \\
(d \in \text{disksWritten} \ s \ p) \land \\
(\text{StartBallot} \ s \ s' \ p) \land \\
(\text{blocksSeen} \ s \ p \ q \ = \ \\
(\text{blocksRead} \ s \ p) \cup \{\ (\text{block} = \text{disk} \ s \ d \ q, \ \\
\text{proc} = q)\}) \land \\
(\text{inpt} \ s' = \text{inpt} \ s \land (\text{outpt} \ s' = \text{outpt} \ s)) \land \\
(\text{disk} \ s' = \text{disk} \ s \land (\text{phase} \ s' = \text{phase} \ s) \land \ \\
(\text{dblock} \ s' = \text{dblock} \ s \land \ (\text{disksWritten} \ s' = \text{disksWritten} \ s)) \\
\textbf{definition} \texttt{Phase1or2Read} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{Disk} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\textbf{where} \\
\texttt{Phase1or2Read} \ s \ s' \ p \ d \ q \ = \\
(\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q) \lor \\
(\text{Phase1or2ReadElse} \ s \ s' \ p \ d \ q) \\
\textbf{definition} \texttt{blocksSeen} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock} \ \texttt{set} \\
\textbf{where} \ 
\texttt{blocksSeen} \ s \ p = \text{allBlocksRead} \ s \ p \cup \{\text{dblock} \ s \ p\} \\
\textbf{definition} \texttt{nonInitBlks} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock} \ \texttt{set} \\
\textbf{where} \ 
\texttt{nonInitBlks} \ s \ p = \{\text{bs} . \ bs \in \text{blocksSeen} \ s \ p \land \ (\text{inp} \ bs \in \text{Inputs})\} \\
\textbf{definition} \texttt{maxBlk} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock} \\
\textbf{where} \ 
\texttt{maxBlk} \ s \ p = \\
(\text{SOME} \ b . \ b \in \text{nonInitBlks} \ s \ p \land (\forall c \in \text{nonInitBlks} \ s \ p . \ \text{bal} \ c \leq \text{bal} \ b)) \\
\textbf{definition} \texttt{EndPhase1} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\textbf{where} \ 
\texttt{EndPhase1} \ s \ s' \ p \ = \\
(\text{IsMajority} \ \{d . \ d \in \text{disksWritten} \ s \ p\)
\[ \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead} s p d q) \land \text{phase} s p = 1 \land \text{dblock} s' = (\text{dblock} s) (p := \text{dblock} s p) \]
\[
\begin{array}{l}
\land \text{bal} := \text{mbal}(\text{dblock} s p), \\
\land \text{inp} := \\
\left(\begin{array}{l}
\text{if nonInitBlks} s p = \{} \\
\text{then inpt} s p \\
\text{else inpt} s p
\end{array}\right) \\
\land \text{outpt} s' = \text{outpt} s \\
\land \text{phase} s' = (\text{phase} s) (p := \text{phase} s p + 1) \\
\land \text{InitializePhase} s s' p \\
\land \text{inpt} s' = \text{inpt} s \land \text{disk} s' = \text{disk} s
\end{array}
\]

**definition** EndPhase2 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool

**where**

\[ \text{EndPhase2} s s' p = (\text{IsMajority} \{d . d \in \text{disksWritten} s p \\
\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead} s p d q)\}) \land \text{phase} s p = 2 \land \text{outpt} s' = (\text{outpt} s) (p := \text{inp} (\text{dblock} s p)) \land \text{dblock} s' = \text{dblock} s \land \text{phase} s' = (\text{phase} s) (p := \text{phase} s p + 1) \land \text{InitializePhase} s s' p \land \text{inpt} s' = \text{inpt} s \land \text{disk} s' = \text{disk} s)\]

**definition** EndPhase1or2 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool

**where**

\[ \text{EndPhase1or2} s s' p = (\text{EndPhase1} s s' p \lor \text{EndPhase2} s s' p)\]

**definition** Fail :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool

**where**

\[ \text{Fail} s s' p = (\exists ip \in \text{Inputs}. \text{inpt} s' = (\text{inpt} s) (p := ip) \land \text{phase} s' = (\text{phase} s) (p := 0) \land \text{dblock} s' = (\text{dblock} s) (p := \text{InitDB}) \land \text{outpt} s' = (\text{outpt} s) (p := \text{NotAnInput}) \land \text{InitializePhase} s s' p \land \text{disk} s' = \text{disk} s)\]

**definition** Phase0Read :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) bool

**where**

\[ \text{Phase0Read} s s' p d = (\text{phase} s p = 0) \land \text{blocksRead} s' = (\text{blocksRead} s) (p := (\text{blocksRead} s p) (d := \text{blocksRead} s p d) \cup \{ (\text{block} = \text{disk} s d p, \text{proc} = p |)\}) \land \text{inpt} s' = \text{inpt} s \land \text{outpt} s' = \text{outpt} s \land \text{disk} s' = \text{disk} s \land \text{phase} s' = \text{phase} s \land \text{dblock} s' = \text{dblock} s \land \text{disksWritten} s' = \text{disksWritten} s)\]

16
definition EndPhase0 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase0 s s′ p =
(phase s p = 0
∧ IsMajority (\{ d. hasRead s p d p\})
∧ (∃ b ∈ Ballot p.
   (∀ r ∈ allBlocksRead s p. mbal r < b)
∧ dblock s′ = (dblock s) ( p:=
   (SOME r. r ∈ allBlocksRead s p
   ∧ (∀ s ∈ allBlocksRead s p. bal s ≤ bal r)) ( mbal := b ) ))
∧ InitializePhase s s′ p
∧ phase s′ = (phase s) ( p:= 1)
∧ inpt s′ = inpt s ∧ outpt s′ = outpt s ∧ disk s′ = disk s)

definition Next :: state ⇒ state ⇒ bool
where
Next s s′ = (∃ p.
  StartBallot s s′ p
∧ (∃ d. Phase0Read s s′ p d
  ∨ Phase1or2Write s s′ p d
  ∨ (∃ q. q ≠ p ∧ Phase1or2Read s s′ p d q))
∧ EndPhase1or2 s s′ p
∧ Fail s s′ p
∧ EndPhase0 s s′ p)

In the following, for each action or state name we name Hname the corresponding action that includes the history part of the HNext action or state predicate that includes history variables.

definition HInit :: state ⇒ bool
where
HInit s =
(Init s
∧ chosen s = NotAnInput
∧ allInput s = range (inpt s))

HNextPart is the part of the Next action that is concerned with history variables.

definition HNextPart :: state ⇒ state ⇒ bool
where
HNextPart s s′ =
(chosen s′ =
  (if chosen s ≠ NotAnInput ∨ (∀ p. outpt s′ p = NotAnInput )
  then chosen s
  else outpt s′ (SOME p. outpt s′ p ≠ NotAnInput))
∧ allInput s′ = allInput s ∪ (range (inpt s′)))

definition HNext :: state ⇒ state ⇒ bool
where
HNext s s′ =
We add $\text{HNextPart}$ to every action (rather than proving that $\text{Next}$ maintains the $\text{HInv}$ invariant) to make proofs easier.

**definition**

$\text{HPhase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q = (\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HEndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HEndPhase1} \ s \ s' \ p = (\text{EndPhase1} \ s \ s' \ p \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HStartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HStartBallot} \ s \ s' \ p = (\text{StartBallot} \ s \ s' \ p \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HPhase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}$ where

$\text{HPhase1or2Write} \ s \ s' \ p \ d = (\text{Phase1or2Write} \ s \ s' \ p \ d \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HPhase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ q = (\text{Phase1or2ReadElse} \ s \ s' \ p \ d \ q \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HEndPhase2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HEndPhase2} \ s \ s' \ p = (\text{EndPhase2} \ s \ s' \ p \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HFail} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HFail} \ s \ s' \ p = (\text{Fail} \ s \ s' \ p \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HPhase0Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}$ where

$\text{HPhase0Read} \ s \ s' \ p \ d = (\text{Phase0Read} \ s \ s' \ p \ d \ \land \ \text{HNextPart} \ s \ s')$

**definition**

$\text{HEndPhase0} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$\text{HEndPhase0} \ s \ s' \ p = (\text{EndPhase0} \ s \ s' \ p \ \land \ \text{HNextPart} \ s \ s')$

Since these definitions are the conjunction of two other definitions declaring them as simplification rules should be harmless.
declare HPhase1or2Write-def [simp]
declare HEndPhase2-def [simp]
declare HFail-def [simp]
declare HPhase0Read-def [simp]
declare HEndPhase0-def [simp]

end

C Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1 Invariant 1

This is just a type Invariant.

definition Inv1 :: state ⇒ bool where
Inv1 s = (∀ p.
inpt s p ∈ Inputs
∧ phase s p ≤ 3
∧ finite (allRdBlks s p))

definition HInv1 :: state ⇒ bool where
HInv1 s =
(Inv1 s
∧ allInput s ⊆ Inputs)

declare HInv1-def [simp]

We added the assertion that the set allRdBlks p is finite for every process p;
one may therefore choose a block with a maximum ballot number in action EndPhase1.

With the following the lemma, it will be enough to prove Inv1 s’ for every
action, without taking the history variables in account.

lemma HNextPart-Inv1: [ HInv1 s; HNextPart s s’; Inv1 s’ ] ⇒ HInv1 s’
⟨proof⟩

theorem HInit-HInv1: HInit s −→ HInv1 s
⟨proof⟩

lemma allRdBlks-finite:
assumes inv: HInv1 s
and asm: ∀ p. allRdBlks s’ p ⊆ insert bk (allRdBlks s p)
shows ∀ p. finite (allRdBlks s’ p)
⟨proof⟩
theorem HPhase1or2ReadThen-HInv1:
    assumes inv1: HInv1 s
    and  act: HPhase1or2ReadThen s s' p d q
    shows HInv1 s'
⟨proof⟩

theorem HEndPhase1-HInv1:
    assumes inv1: HInv1 s
    and  act: HEndPhase1 s s' p
    shows HInv1 s'
⟨proof⟩

theorem HStartBallot-HInv1:
    assumes inv1: HInv1 s
    and  act: HStartBallot s s' p
    shows HInv1 s'
⟨proof⟩

theorem HPhase1or2Write-HInv1:
    assumes inv1: HInv1 s
    and  act: HPhase1or2Write s s' p d
    shows HInv1 s'
⟨proof⟩

theorem HPhase1or2ReadElse-HInv1:
    assumes act: HPhase1or2ReadElse s s' p d q
    and  inv1: HInv1 s
    shows HInv1 s'
⟨proof⟩

theorem HEndPhase2-HInv1:
    assumes inv1: HInv1 s
    and  act: HEndPhase2 s s' p
    shows HInv1 s'
⟨proof⟩

theorem HFail-HInv1:
    assumes inv1: HInv1 s
    and  act: HFail s s' p
    shows HInv1 s'
⟨proof⟩

theorem HPhase0Read-HInv1:
    assumes inv1: HInv1 s
    and  act: HPhase0Read s s' p d
    shows HInv1 s'
⟨proof⟩

theorem HEndPhase0-HInv1:
assumes inv1: HInv1 s
and act: HEndPhase0 s s' p
shows HInv1 s'
(proof)

declare HInv1-def [simp del]

HInv1 is an invariant of HNext

lemma I2a:
assumes nxt: HNext s s'
and inv: HInv1 s
shows HInv1 s'
(proof)

end

theory DiskPaxos-Inv2 imports DiskPaxos-Inv1 begin

C.2 Invariant 2

The second invariant is split into three main conjuncts called Inv2a, Inv2b, and Inv2c. The main difficulty is in proving the preservation of the first conjunct.
definition rdBy :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ BlockProc set
where
rdBy s p q d = 
{br . br ∈ blocksRead s q d ∧ proc br = p}
definition blocksOf :: state ⇒ Proc ⇒ DiskBlock set
where
blocksOf s p = 
{dblock s p}
∪ {disk s d p | d ∈ UNIV}
∪ {block br | br . br ∈ (UN q d. rdBy s p q d) }
definition allBlocks :: state ⇒ DiskBlock set
where allBlocks s = (∪ p. blocksOf s p)
definition Inv2a-innermost :: state ⇒ Proc ⇒ DiskBlock ⇒ bool
where
Inv2a-innermost s p bk = 
(mbalf bk ∈ (Ballot p) ∪ {0})
∧ bal bk ∈ (Ballot p) ∪ {0}
∧ (bal bk = 0) = (inp bk = NotAnInput)
∧ bal bk ≤ mbalf bk
∧ inp bk ∈ (allInput s) ∪ {NotAnInput})
definition Inv2a-inner :: state ⇒ Proc ⇒ bool
  where Inv2a-inner s p = (∀ bk ∈ blocksOf s p. Inv2a-innermost s p bk)

definition Inv2a :: state ⇒ bool
  where Inv2a s = (∀ p. Inv2a-inner s p)

definition Inv2b-inner :: state ⇒ Proc ⇒ Disk ⇒ bool
  where
  Inv2b-inner s p d = ((d ∈ disksWritten s p ➔
    (phase s p ∈ {1,2} ∧ disk s d p = dblock s p))
  ∧ (phase s p ∈ {1,2} ➔
    (blocksRead s p d ≠ {} ➔ d ∈ disksWritten s p)
  ∧ ¬ hasRead s p d))

definition Inv2b :: state ⇒ bool
  where Inv2b s = (∀ p d. Inv2b-inner s p d)

definition Inv2c-inner :: state ⇒ Proc ⇒ bool
  where
  Inv2c-inner s p = ((phase s p = 0 ➔
    (dblock s p = InitDB
  ∧ disksWritten s p = {})
  ∧ (∀ d. ∀ br ∈ blocksRead s p d.
    proc br = p ∧ block br = disk s d))
  ∧ (phase s p ≠ 0 ➔
    (mbal(dblock s p) ∈ Ballot p
  ∧ bal(dblock s p) ∈ Ballot p ∪ {0})
  ∧ (∀ d. ∀ br ∈ blocksRead s p d.
    mbal(block br) < mbal(dblock s p)))
  ∧ (phase s p ∈ {2,3} ➔ bal(dblock s p) = mbal(dblock s p))
  ∧ outpt s p = (if phase s p = 3 then inp(dblock s p) else NotAnInput)
  ∧ chosen s ∈ allInput s ∪ {NotAnInput}
  ∧ (∀ p. inpt s p ∈ allInput s
  ∧ (chosen s = NotAnInput ➔ outpt s p = NotAnInput)))

definition Inv2c :: state ⇒ bool
  where Inv2c s = (∀ p. Inv2c-inner s p)

definition HInv2 :: state ⇒ bool
  where HInv2 s = (Inv2a s ∧ Inv2b s ∧ Inv2c s)

C.2.1 Proofs of Invariant 2 a

theorem HInit-Inv2a: HInit s ➔ Inv2a s
⟨proof⟩

For every action we define a action-blocksOf lemma. We have two cases: e-
ther the new blocksOf is included in the old blocksOf, or the new blocksOf is included in the old blocksOf union the new dblock. In the former case the assumption inv will imply the thesis. In the latter, we just have to prove the innermost predicate for the particular case of the new dblock. This particular case is proved in lemma action-Inv2a-dblock.

**lemma** HPhase1or2ReadThen-blocksOf:
\[ \begin{array}{l}
HPhase1or2ReadThen s s' p d q \implies blocksOf s' r \subseteq blocksOf s r
\end{array} \]
(proof)

**theorem** HPhase1or2ReadThen-Inv2a:
assumes inv: Inv2a s
and act: HPhase1or2ReadThen s s' p d q
shows Inv2a s'
(proof)

**lemma** InitializePhase-rdBy:
InitializePhase s s' p \implies rdBy s' pp qq dd \subseteq rdBy s pp qq dd
(proof)

**lemma** HStartBallot-blocksOf:
HStartBallot s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
(proof)

**lemma** HStartBallot-Inv2a-dblock:
assumes act: HStartBallot s s' p
and inv2a: Inv2a-innermost s p (dblock s p)
shows Inv2a-innermost s' p (dblock s' p)
(proof)

**lemma** HStartBallot-Inv2a-dblock-q:
assumes act: HStartBallot s s' p
and inv2a: Inv2a-innermost s q (dblock s q)
shows Inv2a-innermost s' q (dblock s' q)
(proof)

**theorem** HStartBallot-Inv2a:
assumes inv: Inv2a s
and act: HStartBallot s s' p
shows Inv2a s'
(proof)

**lemma** HPhase1or2Write-blocksOf:
\[ \begin{array}{l}
HPhase1or2Write s s' p d \implies blocksOf s' r \subseteq blocksOf s r
\end{array} \]
(proof)

**theorem** HPhase1or2Write-Inv2a:
assumes inv: Inv2a s
and act: HPhase1or2Write s s' p d
shows \( Inv2a \ s' \)
(\( \text{proof} \))

**theorem** \( H\text{Phase1or2ReadElse-Inv2a} \):
- **assumes** \( inv: Inv2a \ s \)
- **and** \( act: H\text{Phase1or2ReadElse} \ s \ s' \ p \ d \ q \)
- **shows** \( Inv2a \ s' \)
(\( \text{proof} \))

**lemma** \( H\text{EndPhase2-blocksOf} \):
\[
[ H\text{EndPhase2} \ s \ s' \ p ] \implies \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q
\]
(\( \text{proof} \))

**theorem** \( H\text{EndPhase2-Inv2a} \):
- **assumes** \( inv: Inv2a \ s \)
- **and** \( act: H\text{EndPhase2} \ s \ s' \ p \)
- **shows** \( Inv2a \ s' \)
(\( \text{proof} \))

**lemma** \( H\text{Fail-blocksOf} \):
\[
H\text{Fail} \ s \ s' \ p \implies \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \cup \{ \text{dblock} \ s' \ q \}
\]
(\( \text{proof} \))

**lemma** \( H\text{Fail-Inv2a-dblock-q} \):
- **assumes** \( act: H\text{Fail} \ s \ s' \ p \)
- **and** \( inv: Inv2a\text{-innermost} \ s \ q \ (\text{dblock} \ s \ q) \)
- **shows** \( Inv2a\text{-innermost} \ s' \ q \ (\text{dblock} \ s' \ q) \)
(\( \text{proof} \))

**theorem** \( H\text{Fail-Inv2a} \):
- **assumes** \( inv: Inv2a \ s \)
- **and** \( act: H\text{Fail} \ s \ s' \ p \)
- **shows** \( Inv2a \ s' \)
(\( \text{proof} \))

**lemma** \( H\text{Phase0Read-blocksOf} \):
\[
H\text{Phase0Read} \ s \ s' \ p \ d \implies \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q
\]
(\( \text{proof} \))

**theorem** \( H\text{Phase0Read-Inv2a} \):
- **assumes** \( inv: Inv2a \ s \)
- **and** \( act: H\text{Phase0Read} \ s \ s' \ p \ d \)
- **shows** \( Inv2a \ s' \)
(\( \text{proof} \))

**lemma** \( H\text{EndPhase0-blocksOf} \):
\[
H\text{EndPhase0} \ s \ s' \ p \implies \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \cup \{ \text{dblock} \ s' \ q \}
\]
(\( \text{proof} \))
lemma HEndPhase0-blocksRead: 
  assumes act: HEndPhase0 s s' p 
  shows \exists \, d. \, blocksRead s p d \neq \{
  \langle proof \rangle
EndPhase0 has the additional difficulty of having a choose expression. We prove that there exists an \( x \) such that the predicate of the choose expression holds, and then apply someI: \( \forall \, x \rightarrow \forall \, P \rightarrow P \circ Eps \, P \).

lemma HEndPhase0-some: 
  assumes act: HEndPhase0 s s' p 
and inv1: Inv1 s 
  shows (SOME b. \( b \in \text{allBlocksRead} \, s \, p \) \wedge (\\forall t \in \text{allBlocksRead} \, s \, p. \, bal \, t \leq bal \, b)) \wedge (\\forall t \in \text{allBlocksRead} \, s \, p. \, bal \, t \leq bal \, (SOME \, b. \, b \in \text{allBlocksRead} \, s \, p \wedge (\\forall t \in \text{allBlocksRead} \, s \, p. \, bal \, t \leq bal \, b)))
  \langle proof \rangle
lemma HEndPhase0-dblock-allBlocksRead: 
  assumes act: HEndPhase0 s s' p 
and inv1: Inv1 s 
  shows dblock s' p \in (\lambda x. \, x \circ (mbal := mbal(dblock s' p))) \circ allBlocksRead s p
  \langle proof \rangle
lemma HNextPart-allInput-or-NotAnInput: 
  assumes act: HNextPart s s' 
and inv2a: Inv2a-innermost s p (dblock s' p) 
  shows inp (dblock s' p) \in allInput s' \cup \{NotAnInput\}
  \langle proof \rangle
lemma HEndPhase0-Inv2a-allBlocksRead: 
  assumes act: HEndPhase0 s s' p 
and inv2a: Inv2a-inner s p 
and inv2c: Inv2c-inner s p 
  shows \forall t \in (\lambda x. \, x \circ (mbal := mbal(dblock s' p))) \circ allBlocksRead s p. \, Inv2a-innermost \, s \, p \, t
  \langle proof \rangle
lemma HEndPhase0-Inv2a-dblock: 
  assumes act: HEndPhase0 s s' p 
and inv1: Inv1 s 
and inv2a: Inv2a-inner s p 
and inv2c: Inv2c-inner s p 
  shows Inv2a-innermost s' p (dblock s' p)
  \langle proof \rangle
lemma HEndPhase0-Inv2a-dblock-q:
\begin{verbatim}
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a-inner s q
and inv2c: Inv2c-inner s p
shows Inv2a-innermost s' q (dblock s' q)
\end{verbatim}

\begin{verbatim}
theorem HEndPhase0-Inv2a:
assumes inv: Inv2a s
and act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows Inv2a s'
\end{verbatim}

\begin{verbatim}
lemma HEndPhase1-blocksOf:
  HEndPhase1 s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
\end{verbatim}

\begin{verbatim}
lemma maxBlk-in-nonInitBlks:
assumes b: b \in nonInitBlks s p
and inv1: Inv1 s
shows maxBlk s p \in nonInitBlks s p
\wedge (\forall c \in nonInitBlks s p. bal c \leq bal (maxBlk s p))
\end{verbatim}

\begin{verbatim}
lemma blocksOf-nonInitBlks:
(\forall p bk. bk \in blocksOf s p \implies P bk)
\implies bk \in nonInitBlks s p \implies P bk
\end{verbatim}

\begin{verbatim}
lemma maxBlk-allInput:
assumes inv: Inv2a s
and mblk: maxBlk s p \in nonInitBlks s p
shows inp (maxBlk s p) \in allInput s
\end{verbatim}

\begin{verbatim}
lemma HEndPhase1-dblock-allInput:
assumes act: HEndPhase1 s s' p
and inv1: HInv1 s
and inv2: Inv2a s
shows inp': inp (dblock s' p) \in allInput s'
\end{verbatim}

\begin{verbatim}
lemma HEndPhase1-Inv2a-dblock:
assumes act: HEndPhase1 s s' p
and inv1: HInv1 s
and inv2: Inv2a s
and inv2c: Inv2c-inner s p
\end{verbatim}

26
shows $\text{Inv2a-innermost } s' \ (\text{dblock } s' \ p)$

(\text{proof})

\textbf{lemma } H\text{EndPhase1-Inv2a-dblock-q:}
\textbf{ assumes } act: H\text{EndPhase1 } s \ s' \ p \\
\textbf{ and } inv1: H\text{Inv1 } s \\
\textbf{ and } inv: Inv2a \ s \\
\textbf{ and } inv2c: Inv2c-inner s \ p \\
\textbf{shows } Inv2a-innermost s' \ q \ (\text{dblock } s' \ q)$

(\text{proof})

\textbf{theorem } H\text{EndPhase1-Inv2a:}
\textbf{ assumes } act: H\text{EndPhase1 } s \ s' \ p \\
\textbf{ and } inv1: H\text{Inv1 } s \\
\textbf{ and } inv: Inv2a \ s \\
\textbf{ and } inv2c: Inv2c-inner s \ p \\
\textbf{shows } Inv2a \ s' 

(\text{proof})

\textbf{C.2.2 Proofs of Invariant 2 b}

Invariant 2b is proved automatically, given that we expand the definitions involved.

\textbf{theorem } H\text{Init-Inv2b: } H\text{Init } s \longrightarrow Inv2b \ s 

(\text{proof})

\textbf{theorem } H\text{Phase1or2ReadThen-Inv2b:} \\
[ Inv2b \ s; H\text{Phase1or2ReadThen } s \ s' \ p \ d \ q ] \\
\Longrightarrow Inv2b \ s' 

(\text{proof})

\textbf{theorem } H\text{StartBallot-Inv2b:} \\
[ Inv2b \ s; H\text{StartBallot } s \ s' \ p ] \\
\Longrightarrow Inv2b \ s' 

(\text{proof})

\textbf{theorem } H\text{Phase1or2Write-Inv2b:} \\
[ Inv2b \ s; H\text{Phase1or2Write } s \ s' \ p \ d ] \\
\Longrightarrow Inv2b \ s' 

(\text{proof})

\textbf{theorem } H\text{Phase1or2ReadElse-Inv2b:} \\
[ Inv2b \ s; H\text{Phase1or2ReadElse } s \ s' \ p \ d \ q ] \\
\Longrightarrow Inv2b \ s' 

(\text{proof})

\textbf{theorem } H\text{EndPhase1-Inv2b:} \\
[ Inv2b \ s; H\text{EndPhase1 } s \ s' \ p ] \Longrightarrow Inv2b \ s' 

(\text{proof})
**theorem** HFail-Inv2b:
\[ \left[ \text{Inv2b } s ; \text{HFail } s \ s' \ p \right] \implies \text{Inv2b } s' \]
\(\langle \text{proof} \rangle\)

**theorem** HEndPhase2-Inv2b:
\[ \left[ \text{Inv2b } s ; \text{HEndPhase2 } s \ s' \ p \right] \implies \text{Inv2b } s' \]
\(\langle \text{proof} \rangle\)

**theorem** HPhase0Read-Inv2b:
\[ \left[ \text{Inv2b } s ; \text{HPhase0Read } s \ s' \ p \ d \right] \implies \text{Inv2b } s' \]
\(\langle \text{proof} \rangle\)

**theorem** HEndPhase0-Inv2b:
\[ \left[ \text{Inv2b } s ; \text{HEndPhase0 } s \ s' \ p \right] \implies \text{Inv2b } s' \]
\(\langle \text{proof} \rangle\)

**C.2.3 Proofs of Invariant 2 c**

**theorem** HInit-Inv2c: HInit s \(\rightarrow\) Inv2c s
\(\langle \text{proof} \rangle\)

**lemma** HNextPart-Inv2c-chosen:
assumes hnp: HNextPart s s'
and inv2c: Inv2c s
and outpt': \(\forall p. \text{ outpt } s' \ p = (\text{if phase } s' \ p = 3 \ then \ inp(dblock s' \ p) \ else \ NotAnInput}\)
and inp-dblk: \(\forall p. \text{ inp } (dblock s' \ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}\)
sows chosen s' \(\in\) allInput s' \(\cup\) \{NotAnInput\}
\(\langle \text{proof} \rangle\)

**lemma** HNextPart-chosen:
assumes hnp: HNextPart s s'
sows chosen s' = NotAnInput \(\rightarrow\) (\(\forall p. \text{ outpt } s' \ p = \text{NotAnInput}\))
\(\langle \text{proof} \rangle\)

**lemma** HNextPart-allInput:
\[ \left[ \text{HNextPart } s \ s' ; \text{Inv2c } s \right] \implies \forall p. \text{ inpt } s' \ p \in \text{allInput } s' \]
\(\langle \text{proof} \rangle\)

**theorem** HPhase1or2ReadThen-Inv2c:
assumes inv: Inv2c s
and act: HPhase1or2ReadThen s s' p d q
and inv2a: Inv2a s
sows Inv2c s'
\(\langle \text{proof} \rangle\)
theorem HStartBallot-Inv2c:
  assumes inv: Inv2c s
  and act: HStartBallot s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
⟨proof⟩

theorem HPhase1or2Write-Inv2c:
  assumes inv: Inv2c s
  and act: HPhase1or2Write s s' p d
  and inv2a: Inv2a s
  shows Inv2c s'
⟨proof⟩

theorem HPhase1or2ReadElse-Inv2c:
  [[ Inv2c s; HPhase1or2ReadElse s s' p d q; Inv2a s ]] ⇒ Inv2c s'
⟨proof⟩

theorem HEndPhase1-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase1 s s' p
  and inv2a: Inv2a s
  and inv1: HInv1 s
  shows Inv2c s'
⟨proof⟩

theorem HEndPhase2-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase2 s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
⟨proof⟩

theorem HFail-Inv2c:
  assumes inv: Inv2c s
  and act: HFail s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
⟨proof⟩

theorem HPhase0Read-Inv2c:
  assumes inv: Inv2c s
  and act: HPhase0Read s s' p d
  and inv2a: Inv2a s
  shows Inv2c s'
⟨proof⟩

theorem HEndPhase0-Inv2c:
assumes inv: Inv2c s
and act: HEndPhase0 s s' p
and inv2a: Inv2a s
and inv1: Inv1 s
shows Inv2c s'
(proof)

theorem HInit-HInv2:
HInit s \implies HInv2 s
(proof)

HInv1 \land HInv2 is an invariant of HNext.

lemma I2b:
assumes nxt: HNext s s'
and inv: HInv1 s \land HInv2 s
shows HInv2 s'
(proof)

end

theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

This invariant says that if two processes have read each other’s block from
disk d during their current phases, then at least one of them has read the
other’s current block.

definition HInv3-L :: state \Rightarrow Proc \Rightarrow Proc \Rightarrow Disk \Rightarrow bool
where
HInv3-L s p q d = (phase s p \in \{1, 2\}
\land phase s q \in \{1, 2\}
\land hasRead s p d q
\land hasRead s q d p)

definition HInv3-R :: state \Rightarrow Proc \Rightarrow Proc \Rightarrow Disk \Rightarrow bool
where
HInv3-R s p q d = ((\langle block= dblock s q, proc= q\rangle \in blocksRead s p d
\land \langle block= dblock s p, proc= p\rangle \in blocksRead s q d)

definition HInv3-inner :: state \Rightarrow Proc \Rightarrow Proc \Rightarrow Disk \Rightarrow bool
where
HInv3-inner s p q d = (HInv3-L s p q d \rightarrow HInv3-R s p q d)

definition HInv3 :: state \Rightarrow bool
where
HInv3 s = (\forall p q d. HInv3-inner s p q d)

C.3.1 Proofs of Invariant 3

theorem HInit-HInv3: HInit s \implies HInv3 s
\textbf{lemma} \textit{InitPhase-HInv3-p}:
\[ \begin{array}{l}
\text{[ InitializePhase s s' p ; HInv3-L s' p q d ]} \\
\Rightarrow HInv3-R s' p q d
\end{array} \] (\textit{proof})

\textbf{lemma} \textit{InitPhase-HInv3-q}:
\[ \begin{array}{l}
\text{[ InitializePhase s s' q ; HInv3-L s' p q d ]} \\
\Rightarrow HInv3-R s' p q d
\end{array} \] (\textit{proof})

\textbf{lemma} \textit{HInv3-L-sym}:
\[ HInv3-L s p q d = \Rightarrow HInv3-L s q p d \] (\textit{proof})

\textbf{lemma} \textit{HInv3-R-sym}:
\[ HInv3-R s p q d = \Rightarrow HInv3-R s q p d \] (\textit{proof})

\textbf{lemma} \textit{Phase1or2ReadThen-HInv3-pq}:
\begin{itemize}
\item \textbf{assumes} act: Phase1or2ReadThen s s' p d q
\item \textbf{and} inv-L': HInv3-L s' p q d
\item \textbf{and} pq: p\neq q
\item \textbf{and} inv2b: Inv2b s
\item \textbf{shows} HInv3-R s' p q d
\end{itemize}
(\textit{proof})

\textbf{lemma} \textit{Phase1or2ReadThen-HInv3-hasRead}:
\[ \begin{array}{l}
\text{[ ¬hasRead s pp dd qq;}
\text{Phase1or2ReadThen s s' p d q;}
\text{pp\neq p \lor qq\neq q \lor dd\neq d]} \\
\Rightarrow \neg \text{hasRead s' pp dd qq}
\end{array} \] (\textit{proof})

\textbf{theorem} \textit{HPhase1or2ReadThen-HInv3}:
\begin{itemize}
\item \textbf{assumes} act: HPhase1or2ReadThen s s' p d q
\item \textbf{and} inv: HInv3 s
\item \textbf{and} pq: p\neq q
\item \textbf{and} inv2b: Inv2b s
\item \textbf{shows} HInv3 s'
\end{itemize}
(\textit{proof})

\textbf{lemma} \textit{StartBallot-HInv3-p}:
\[ \begin{array}{l}
\text{[ StartBallot s s' p; HInv3-L s' p q d ]} \\
\Rightarrow HInv3-R s' p q d
\end{array} \] (\textit{proof})

\textbf{lemma} \textit{StartBallot-HInv3-q}:
\[ \begin{array}{l}
\text{[ StartBallot s s' q; HInv3-L s' p q d ]} \\
\Rightarrow HInv3-R s' p q d
\end{array} \] (\textit{proof})
lemma StartBallot-HInv3-nL:
\[ \text{StartBallot } s \quad s'; \quad \neg \text{HInv3-L } s \quad p \quad q \quad d; \quad t
\neq p; \quad t \neq q \quad \Rightarrow \quad \neg \text{HInv3-L } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma StartBallot-HInv3-R:
\[ \text{StartBallot } s \quad s'; \quad \text{HInv3-R } s \quad p \quad q \quad d; \quad t
\neq p; \quad t \neq q \quad \Rightarrow \quad \text{HInv3-R } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma StartBallot-HInv3-t:
\[ \text{StartBallot } s \quad s'; \quad \text{HInv3-inner } s \quad p \quad q \quad d; \quad t
\neq p; \quad t \neq q \quad \Rightarrow \quad \text{HInv3-inner } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma StartBallot-HInv3:
\[ \text{assumes } \text{act}: \text{StartBallot } s \quad s' \quad t \quad \text{and } \text{inv}: \text{HInv3-inner } s \quad p \quad q \quad d \quad \text{shows } \text{HInv3-inner } s' \quad p \quad q \quad d \]
⟨proof⟩

theorem HStartBallot-HInv3:
\[ \text{HStartBallot } s \quad s'; \quad \text{HInv3 } s \quad \Rightarrow \quad \text{HInv3 } s' \]
⟨proof⟩

theorem HPhase1or2ReadElse-HInv3:
\[ \text{HPhase1or2ReadElse } s \quad s' \quad p \quad d \quad q; \quad \text{HInv3 } s \quad \Rightarrow \quad \text{HInv3 } s' \]
⟨proof⟩

theorem HPhase1or2Write-HInv3:
\[ \text{assumes } \text{act}: \text{HPhase1or2Write } s \quad s' \quad p \quad d \quad \text{and } \text{inv}: \text{HInv3 } s \quad \text{shows } \text{HInv3 } s' \]
⟨proof⟩

lemma EndPhase1-HInv3-p:
\[ \text{EndPhase1 } s \quad s' \quad p; \quad \text{HInv3-L } s' \quad p \quad q \quad d \quad \Rightarrow \quad \text{HInv3-R } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma EndPhase1-HInv3-q:
\[ \text{EndPhase1 } s \quad s' \quad q; \quad \text{HInv3-L } s' \quad p \quad q \quad d \quad \Rightarrow \quad \text{HInv3-R } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma EndPhase1-HInv3-nL:
\[ \text{EndPhase1 } s \quad s' \quad t; \quad \neg \text{HInv3-L } s \quad p \quad q \quad d; \quad t
\neq p; \quad t \neq q \quad \Rightarrow \quad \neg \text{HInv3-L } s' \quad p \quad q \quad d \]
⟨proof⟩

lemma EndPhase1-HInv3-R:
\[ \text{EndPhase1 } s s' t; \ H\text{Inv3-R } s p q d; \ t \neq p; \ t \neq q \] \[ \implies H\text{Inv3-R } s' p q d \]

⟨proof⟩

**Lemma EndPhase1-HInv3-t:**
\[ \text{EndPhase1 } s s' t; \ H\text{Inv3-inner } s p q d; \ t \neq p; \ t \neq q \] \[ \implies H\text{Inv3-inner } s' p q d \]

⟨proof⟩

**Lemma EndPhase1-HInv3:**
\begin{align*}
\text{assumes } & \text{act: EndPhase1 } s s' t \\
\text{and } & \text{inv: } H\text{Inv3-inner } s p q d \\
\text{shows } & H\text{Inv3-inner } s' p q d
\end{align*}

⟨proof⟩

**Theorem HEndPhase1-HInv3:**
\[ \text{EndPhase1 } s s' t; \ H\text{Inv3 } s s' \] \[ \implies H\text{Inv3 } s' \]

⟨proof⟩

**Lemma EndPhase2-HInv3-p:**
\[ \text{EndPhase2 } s s' p; \ H\text{Inv3-L } s' p q d \] \[ \implies H\text{Inv3-R } s' p q d \]

⟨proof⟩

**Lemma EndPhase2-HInv3-q:**
\[ \text{EndPhase2 } s s' q; \ H\text{Inv3-L } s' p q d \] \[ \implies H\text{Inv3-R } s' p q d \]

⟨proof⟩

**Lemma EndPhase2-HInv3-nL:**
\[ \text{EndPhase2 } s s' t; \neg H\text{Inv3-L } s p q d; \ t \neq p; \ t \neq q \] \[ \implies \neg H\text{Inv3-L } s' p q d \]

⟨proof⟩

**Lemma EndPhase2-HInv3-R:**
\[ \text{EndPhase2 } s s' t; \ H\text{Inv3-R } s p q d; \ t \neq p; \ t \neq q \] \[ \implies H\text{Inv3-R } s' p q d \]

⟨proof⟩

**Lemma EndPhase2-HInv3-t:**
\[ \text{EndPhase2 } s s' t; \ H\text{Inv3-inner } s p q d; \ t \neq p; \ t \neq q \] \[ \implies H\text{Inv3-inner } s' p q d \]

⟨proof⟩

**Lemma EndPhase2-HInv3:**
\begin{align*}
\text{assumes } & \text{act: EndPhase2 } s s' t \\
\text{and } & \text{inv: } H\text{Inv3-inner } s p q d \\
\text{shows } & H\text{Inv3-inner } s' p q d
\end{align*}

⟨proof⟩

**Theorem HEndPhase2-HInv3:**
\( \text{lemma Fail-HInv3-p:} \quad \left[ \begin{array}{l} \text{Fail } s s' p; \text{ HInv3 } s \end{array} \right] \implies \text{HInv3 } s' \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma Fail-HInv3-q:} \quad \left[ \begin{array}{l} \text{Fail } s s' q; \text{ HInv3-L } s' p q d \end{array} \right] \implies \text{HInv3-R } s' p q d \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma Fail-HInv3-nL:} \quad \left[ \begin{array}{l} \text{Fail } s s' t; \neg \text{HInv3-L } s p q d; \; t \neq p; \; t \neq q \end{array} \right] \implies \neg \text{HInv3-L } s' p q d \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma Fail-HInv3-R:} \quad \left[ \begin{array}{l} \text{Fail } s s' t; \text{ HInv3-R } s p q d; \; t \neq p; \; t \neq q \end{array} \right] \implies \text{HInv3-R } s' p q d \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma Fail-HInv3-t:} \quad \left[ \begin{array}{l} \text{Fail } s s' t; \text{ HInv3-inner } s p q d; \; t \neq p; \; t \neq q \end{array} \right] \implies \text{HInv3-inner } s' p q d \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma Fail-HInv3:} \) 

\begin{itemize}
  \item \text{assumes act: Fail } s s' t \\
  \item \text{and inv: HInv3-inner } s p q d \\
  \item \text{shows HInv3-inner } s' p q d
\end{itemize}

\( \langle \text{proof} \rangle \)

\( \text{theorem HFail-HInv3:} \quad \left[ \begin{array}{l} \text{HFail } s s' p; \text{ HInv3 } s \end{array} \right] \implies \text{HInv3 } s' \) 

\( \langle \text{proof} \rangle \)

\( \text{theorem HPhase0Read-HInv3:} \) 

\begin{itemize}
  \item \text{assumes act: HPhase0Read } s s' p d \\
  \item \text{and inv: HInv3 } s \\
  \item \text{shows HInv3 } s'
\end{itemize}

\( \langle \text{proof} \rangle \)

\( \text{lemma EndPhase0-HInv3-p:} \quad \left[ \begin{array}{l} \text{EndPhase0 } s s' p; \text{ HInv3-L } s' p q d \end{array} \right] \implies \text{HInv3-R } s' p q d \) 

\( \langle \text{proof} \rangle \)

\( \text{lemma EndPhase0-HInv3-q:} \quad \left[ \begin{array}{l} \text{EndPhase0 } s s' q; \text{ HInv3-L } s' p q d \end{array} \right] \)
\[ \implies HInv3-R \] s' p q d
\begin{proof}
\end{proof}

**lemma** EndPhase0-HInv3-nL:
\[ \begin{array}{c}
\text{EndPhase0} \; s \; s' \; t; \; \neg HInv3-L \; s \; p \; q \; d; \; t \neq p; \; t \neq q \\
\end{array} \implies \neg HInv3-L \; s' \; p \; q \; d
\begin{proof}
\end{proof}

**lemma** EndPhase0-HInv3-R:
\[ \begin{array}{c}
\text{EndPhase0} \; s \; s' \; t; \; HInv3-R \; s \; p \; q \; d; \; t \neq p; \; t \neq q \\
\end{array} \implies HInv3-R \; s' \; p \; q \; d
\begin{proof}
\end{proof}

**lemma** EndPhase0-HInv3-t:
\[ \begin{array}{c}
\text{EndPhase0} \; s \; s' \; t; \; HInv3-inner \; s \; p \; q \; d; \; t \neq p; \; t \neq q \\
\end{array} \implies HInv3-inner \; s' \; p \; q \; d
\begin{proof}
\end{proof}

**lemma** EndPhase0-HInv3:
\begin{itemize}
\item assumes act: EndPhase0 \; s \; s' \; t
\item and inv: HInv3-inner \; s \; p \; q \; d
\item shows HInv3-inner \; s' \; p \; q \; d
\end{itemize}
\begin{proof}
\end{proof}

**theorem** HEndPhase0-HInv3:
\[ \begin{array}{c}
\text{HEndPhase0} \; s \; s' \; p; \; HInv3 \; s \\
\end{array} \implies HInv3 \; s'
\begin{proof}
\end{proof}

\( HInv1 \land HInv2 \land HInv3 \) is an invariant of \( HNext \).

**lemma** I2c:
\begin{itemize}
\item assumes nxt: HNext \; s \; s'
\item and inv: HInv1 \; s \land HInv2 \; s \land HInv3 \; s
\item shows HInv3 \; s' \begin{proof}
\end{proof}
end
\end{itemize}

**theory** DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

**C.4 Invariant 4**

This invariant expresses relations among \( mbal \) and \( bal \) values of a processor and of its disk blocks. \( HInv4a \) asserts that, when \( p \) is not recovering from a failure, its \( mbal \) value is at least as large as the \( bal \) field of any of its blocks, and at least as large as the \( mbal \) field of its block on some disk in any majority set. \( HInv4b \) conjunct asserts that, in phase 1, its \( mbal \) value is actually greater than the \( bal \) field of any of its blocks. \( HInv4c \) asserts that, in phase 2, its \( bal \) value is the \( mbal \) field of all its blocks on some majority
set of disks. \( HInv4d \) asserts that the bal field of any of its blocks is at most as large as the mbal field of all its disk blocks on some majority set of disks.

**definition**

\[
\text{MajoritySet} ::= \text{Disk set set} \\
\text{where} \quad \text{MajoritySet} = \{ D. \ \text{IsMajority}(D) \}
\]

**definition**

\[
\text{HInv4a1} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4a1} s p = (\forall bk \in \text{blocksOf } s p. \ \text{bal } bk \leq \text{mbal}(\text{dblock } s p))
\]

**definition**

\[
\text{HInv4a2} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4a2} s p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ \text{mbal}(\text{disk } s d p) \leq \text{mbal}(\text{dblock } s p)) \wedge \text{bal}(\text{disk } s d p) \leq \text{bal}(\text{dblock } s p)))
\]

**definition**

\[
\text{HInv4a} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4a} s p = (\text{phase } s p \neq 0 \rightarrow \text{HInv4a1 } s p \land \text{HInv4a2 } s p)
\]

**definition**

\[
\text{HInv4b} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4b} s p = (\text{phase } s p = 1 \rightarrow (\forall bk \in \text{blocksOf } s p. \ \text{bal } bk < \text{mbal}(\text{dblock } s p)))
\]

**definition**

\[
\text{HInv4c} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4c} s p = (\text{phase } s p \in \{2,3\} \rightarrow (\exists D \in \text{MajoritySet}. \forall d \in D. \ \text{mbal}(\text{disk } s d p) = \text{bal}(\text{dblock } s p)))
\]

**definition**

\[
\text{HInv4d} ::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4d} s p = (\forall bk \in \text{blocksOf } s p. \ \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal}(\text{disk } s d p))
\]

**definition**

\[
\text{HInv4} ::= \text{state} \Rightarrow \text{bool} \\
\text{where} \quad \text{HInv4} s = (\forall p. \ \text{HInv4a } s p \land \text{HInv4b } s p \land \text{HInv4c } s p \land \text{HInv4d } s p)
\]

The initial state implies Invariant 4.

**theorem**

\[ \text{HInit-HInv4}: \ HInit \ s \Rightarrow \ HInv4 \ s \]

(proof)

To prove that the actions preserve \( HInv4 \), we do it for one conjunct at a time.

For each action \( actionss'q \) and conjunct \( x \in a, b, c, d \) of \( HInv4xs'p \), we prove two lemmas. The first lemma \( action-HInv4x-p \) proves the case of \( p = q \), while lemma \( action-HInv4x-q \) proves the other case.

C.4.1 Proofs of Invariant 4a

**lemma**

\[ \text{HStartBallot-HInv4a1}: \]

assumes \( act: \text{HStartBallot } s s' p \)

and \( inv: \text{HInv4a1 } s p \)

and \( inv2a: \text{Inv2a-inner } s' p \)
shows \( H_{\text{Inv4a1}} s' p \)
(proof)

lemma \( H_{\text{StartBallot-IInv4a2}} \):
  assumes act: \( H_{\text{StartBallot}} s s' p \)
  and inv: \( H_{\text{Inv4a2}} s p \)
  shows \( H_{\text{Inv4a2}} s' p \)
(proof)

lemma \( H_{\text{StartBallot-IInv4a-p}} \):
  assumes act: \( H_{\text{StartBallot}} s s' p \)
  and inv: \( H_{\text{Inv4a}} s p \)
  and inv2a: \( \text{Inv2a-inner} s' p \)
  shows \( H_{\text{Inv4a}} s' p \)
(proof)

lemma \( H_{\text{StartBallot-IInv4a-q}} \):
  assumes act: \( H_{\text{StartBallot}} s s' p \)
  and inv: \( H_{\text{Inv4a}} s q \)
  and pnq: \( p \neq q \)
  shows \( H_{\text{Inv4a}} s' q \)
(proof)

theorem \( H_{\text{StartBallot-IInv4a}} \):
  assumes act: \( H_{\text{StartBallot}} s s' p \)
  and inv: \( H_{\text{Inv4a}} s q \)
  shows \( H_{\text{Inv4a}} s' q \)
(proof)

lemma \( \text{Phase1or2Write-IInv4a1} \):
  \[
  \begin{array}{ll}
  \text{Phase1or2Write} s s' p d; & H_{\text{Inv4a1}} s q \\
  \end{array}
  \implies H_{\text{Inv4a1}} s' q 
  \]
(proof)

lemma \( \text{Phase1or2Write-IInv4a2} \):
  \[
  \begin{array}{ll}
  \text{Phase1or2Write} s s' p d; & H_{\text{Inv4a2}} s q \\
  \end{array}
  \implies H_{\text{Inv4a2}} s' q 
  \]
(proof)

theorem \( \text{HPhase1or2Write-IInv4a} \):
  assumes act: \( \text{HPhase1or2Write} s s' p d \)
  and inv: \( H_{\text{Inv4a}} s q \)
  shows \( H_{\text{Inv4a}} s' q \)
(proof)

lemma \( \text{HPhase1or2ReadThen-IInv4a1-p} \):
  assumes act: \( \text{HPhase1or2ReadThen} s s' p d q \)
  and inv: \( H_{\text{Inv4a1}} s p \)
  shows \( H_{\text{Inv4a1}} s' p \)
(proof)
lemma HPhase1or2ReadThen-HInv4a2:
\[ [ \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r; \ \text{HInv4a2} \ s \ q ] ] \rightarrow \text{HInv4a2} \ s' \ q 
\langle \text{proof} \rangle

lemma HPhase1or2ReadThen-HInv4a-p:
assumes act: \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r
and inv: \text{HInv4a} \ s \ p
and inv2b: \text{Inv2b} \ s
shows \text{HInv4a} \ s' \ p
\langle \text{proof} \rangle

lemma HPhase1or2ReadThen-HInv4a-q:
assumes act: \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r
and inv: \text{HInv4a} \ s \ q
and pnq: p \neq q
shows \text{HInv4a} \ s' \ q
\langle \text{proof} \rangle

theorem HPhase1or2ReadThen-HInv4a:
\[ [ \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r; \ \text{HInv4a} \ s \ q; \ \text{Inv2b} \ s ] ] \rightarrow \text{HInv4a} \ s' \ q 
\langle \text{proof} \rangle

theorem HPhase1or2ReadElse-HInv4a:
assumes act: \text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ r
and inv: \text{HInv4a} \ s \ q \ and \ inv2a: \text{Inv2a} \ s'
s shows \text{HInv4a} \ s' \ q 
\langle \text{proof} \rangle

lemma HEndPhase1-HInv4a1:
assumes act: \text{HEndPhase1} \ s \ s' \ p
and inv: \text{HInv4a1} \ s \ p
shows \text{HInv4a1} \ s' \ p
\langle \text{proof} \rangle

lemma HEndPhase1-HInv4a2:
assumes act: \text{HEndPhase1} \ s \ s' \ p
and inv: \text{HInv4a2} \ s \ p
and inv2a: \text{Inv2a} \ s
shows \text{HInv4a2} \ s' \ p
\langle \text{proof} \rangle

lemma HEndPhase1-HInv4a-p:
assumes act: \text{HEndPhase1} \ s \ s' \ p
and inv: \text{HInv4a} \ s \ p
and inv2a: \text{Inv2a} \ s
shows \text{HInv4a} \ s' \ p
\langle \text{proof} \rangle
lemma \texttt{HEndPhase1-HInv4a-q}:
assumes act: \texttt{HEndPhase1} s s' p
and inv: \texttt{HInv4a} s q
and pq: p \neq q
shows \texttt{HInv4a} s' q
(\textit{proof})

theorem \texttt{HEndPhase1-HInv4a}:
[ \texttt{HEndPhase1} s s' p; \texttt{HInv4a} s q; \texttt{Inv2a} s ] \implies \texttt{HInv4a} s' q
(\textit{proof})

theorem \texttt{HFail-HInv4a}:
[ \texttt{HFail} s s' p; \texttt{HInv4a} s q ] \implies \texttt{HInv4a} s' q
(\textit{proof})

theorem \texttt{HPhase0Read-HInv4a}:
[ \texttt{HPhase0Read} s s' p d; \texttt{HInv4a} s q ] \implies \texttt{HInv4a} s' q
(\textit{proof})

theorem \texttt{HEndPhase2-HInv4a}:
[ \texttt{HEndPhase2} s s' p; \texttt{HInv4a} s q ] \implies \texttt{HInv4a} s' q
(\textit{proof})

lemma \texttt{allSet}:
assumes aPQ: \forall a. \forall r \in P a. Q r and rb: rb \in P d
shows Q rb
(\textit{proof})

lemma \texttt{EndPhase0-44}:
assumes act: \texttt{EndPhase0} s s' p
and bk: bk \in \texttt{blocksOf s p}
and inv4d: \texttt{HInv4d} s p
and inv2c: \texttt{Inv2c-inner} s p
shows \exists d. \exists rb \in \texttt{blocksRead s p d}. \texttt{bal bk} \leq \texttt{mbal(block rb)}
(\textit{proof})

lemma \texttt{HEndPhase0-HInv4a1-p}:
assumes act: \texttt{HEndPhase0} s s' p
and inv2a': \texttt{Inv2a} s'
and inv2c: \texttt{Inv2c-inner} s p
and inv4d: \texttt{HInv4d} s p
shows \texttt{HInv4a1} s' p
(\textit{proof})

lemma \texttt{hasRead-allBlks}:
assumes inv2c: \texttt{Inv2c-inner} s p
and phase: phase s p = 0
shows \forall d \in \{d. \texttt{hasRead s p d p}\}. \texttt{disk s d p} \in \texttt{allBlocksRead s p}
(\textit{proof})

39
lemma HEndPhase0-41:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows \( \exists D \in \text{MajoritySet.} \forall d \in D. \ mbal(\text{disk} s d p) \leq mbal(\text{dblock} s' p) \)
\( \land \ bal(\text{disk} s d p) \leq bal(\text{dblock} s' p) \)
⟨proof⟩

lemma Majority-exQ:
assumes asm1: \( \exists D \in \text{MajoritySet.} \forall d \in D. P d \)
shows \( \forall D \in \text{MajoritySet.} \exists d \in D. P d \)
⟨proof⟩

lemma HEndPhase0-Hinv4a2-p:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows Hinv4a2 s' p
⟨proof⟩

lemma HEndPhase0-Hinv4a-p:
assumes act: HEndPhase0 s s' p
and inv2a: Inv2a s
and inv2: Inv2c s
and inv4d: Hinv4d s p
and inv1: Inv1 s
and inv: Hinv4a s p
shows Hinv4a s' p
⟨proof⟩

lemma HEndPhase0-Hinv4a-q:
assumes act: HEndPhase0 s s' p
and inv: Hinv4a s q
and pnq: p \neq q
shows Hinv4a s' q
⟨proof⟩

theorem HEndPhase0-Hinv4a:
\[
[ \text{HEndPhase0} s \ s' \ p; \ Hinv4a s \ q; \ Hinv4d s \ p; \\
\text{Inv2a} s; \ \text{Inv1} s; \ \text{Inv2a} s; \ \text{Inv2c} s]
\rightarrow Hinv4a s' q
\]
⟨proof⟩

C.4.2 Proofs of Invariant 4b

lemma blocksRead-allBlocksRead:
\( rb \in \text{blocksRead} s \ p \ d \rightarrow \text{block} \ rb \in \text{allBlocksRead} s \ p \)
lemma HEndPhase0-dblock-mbal:
\[ \text{HEndPhase0 } s \ s' \ p \] \[ \implies \forall \ br \in \text{allBlocksRead} \ s \ p. \ \mbal \ br < \mbal(\text{dblock} \ s' \ p) \]

lemma HEndPhase0-HInv4b-p-dblock:
\text{assumes} \ act: \text{HEndPhase0} \ s \ s' \ p \nand \ \text{inv1}: \text{Inv1} \ s \nand \ \text{inv2a}: \text{Inv2a} \ s \nand \ \text{inv2c}: \text{Inv2c-inner} \ s \ p \nshows \ \text{bal}(\text{dblock} \ s' \ p) < \mbal(\text{dblock} \ s' \ p) \n
lemma HEndPhase0-HInv4b-p-blocksOf:
\text{assumes} \ act: \text{HEndPhase0} \ s \ s' \ p \nand \ \text{inv4d}: \text{HInv4d} \ s \ p \nand \ \text{inv2c}: \text{Inv2c-inner} \ s \ p \nand \ \text{bk}: \text{bk} \in \text{blocksOf} \ s \ p \nshows \ \text{bal} \ \text{bk} < \mbal(\text{dblock} \ s' \ p) \n
lemma HEndPhase0-HInv4b-p:
\text{assumes} \ act: \text{HEndPhase0} \ s \ s' \ p \nand \ \text{inv4d}: \text{HInv4d} \ s \ p \nand \ \text{inv1}: \text{Inv1} \ s \nand \ \text{inv2a}: \text{Inv2a} \ s \nand \ \text{inv2c}: \text{Inv2c-inner} \ s \ p \nshows \ HInv4b \ s' \ p \n
lemma HEndPhase0-HInv4b-q:
\text{assumes} \ act: \text{HEndPhase0} \ s \ s' \ p \nand \ \text{pnq}: p \neq q \nand \ \text{inv}: HInv4b \ s \ q \nshows \ HInv4b \ s \ q \n
theorem HEndPhase0-HInv4b:
\text{assumes} \ act: \text{HEndPhase0} \ s \ s' \ p \nand \ \text{inv}: HInv4b \ s \ q \nand \ \text{inv4d}: HInv4d \ s \ p \nand \ \text{inv1}: \text{Inv1} \ s \nand \ \text{inv2a}: \text{Inv2a} \ s \nand \ \text{inv2c}: \text{Inv2c-inner} \ s \ p \nshows \ HInv4b \ s' \ q \n
41
lemma HStartBallot-HInv4b-p:
  assumes act: HStartBallot s s' p
  and inv2a: Inv2a-innermost s p (dblock s p)
  and inv4b: HInv4b s p
  shows HInv4b s' p
⟨proof⟩

lemma HStartBallot-HInv4b-q:
  assumes act: HStartBallot s s' p
  and pnq: p≠q
  and inv: HInv4b s q
  shows HInv4b s' q
⟨proof⟩

theorem HStartBallot-HInv4b:
  assumes act: HStartBallot s s' p
  and inv2a: Inv2a s
  and inv4b: HInv4b s q
  and inv4a: HInv4a s p
  shows HInv4b s' q
⟨proof⟩

theorem HPhase1or2Write-HInv4b:
  [ [ HPhase1or2Write s s' p d; HInv4b s q ] ] ⇒ HInv4b s' q
⟨proof⟩

lemma HPhase1or2ReadThen-HInv4b-p:
  assumes act: HPhase1or2ReadThen s s' p d q
  and inv: HInv4b s p
  shows HInv4b s' p
⟨proof⟩

lemma HPhase1or2ReadThen-HInv4b-q:
  assumes act: HPhase1or2ReadThen s s' p d q
  and inv: HInv4b s q
  and pnq: p≠q
  shows HInv4b s' q
⟨proof⟩

theorem HPhase1or2ReadThen-HInv4b:
  [ [ HPhase1or2ReadThen s s' p d q; HInv4b s r ] ] ⇒ HInv4b s' r
⟨proof⟩

theorem HPhase1or2ReadElse-HInv4b:
  [ [ HPhase1or2ReadElse s s' p d q; HInv4b s r ] ] ⇒ HInv4b s' r
⟨proof⟩
proof

lemma HEndPhase1-HInv4b-p:
  HEndPhase1 s s' p \implies HInv4b s' p

(\textit{proof})

lemma HEndPhase1-HInv4b-q:
  \textit{assumes} act: HEndPhase1 s s' p
  \textit{and} pmq: p \neq q
  \textit{and} inv: HInv4b s q
  \textit{shows} HInv4b s' q

(\textit{proof})

theorem HEndPhase1-HInv4b:
  \textit{assumes} act: HEndPhase1 s s' p
  \textit{and} inv: HInv4b s q
  \textit{shows} HInv4b s' q

(\textit{proof})

lemma HEndPhase2-HInv4b-p:
  HEndPhase2 s s' p \implies HInv4b s' p

(\textit{proof})

lemma HEndPhase2-HInv4b-q:
  \textit{assumes} act: HEndPhase2 s s' p
  \textit{and} pmq: p \neq q
  \textit{and} inv: HInv4b s q
  \textit{shows} HInv4b s' q

(\textit{proof})

theorem HEndPhase2-HInv4b:
  \textit{assumes} act: HEndPhase2 s s' p
  \textit{and} inv: HInv4b s q
  \textit{shows} HInv4b s' q

(\textit{proof})

lemma HFail-HInv4b-p:
  HFail s s' p \implies HInv4b s' p

(\textit{proof})

lemma HFail-HInv4b-q:
  \textit{assumes} act: HFail s s' p
  \textit{and} pmq: p \neq q
  \textit{and} inv: HInv4b s q
  \textit{shows} HInv4b s' q

(\textit{proof})

theorem HFail-HInv4b:
  \textit{assumes} act: HFail s s' p

(\textit{proof})
and $inv: HInv4b s \ q$

shows $HInv4b s' \ q$

\(\langle proof \rangle\)

**lemma** $HPhase0Read-HInv4b-p$:  
$HPhase0Read \ s \ s' \ p \ d \Rightarrow HInv4b \ s' \ p$

\(\langle proof \rangle\)

**lemma** $HPhase0Read-HInv4b-q$:  
assumes act: $HPhase0Read \ s \ s' \ p \ d$

and $pq: p \neq q$

and $inv: HInv4b \ s \ q$

shows $HInv4b \ s' \ q$

\(\langle proof \rangle\)

**theorem** $HPhase0Read-HInv4b$:  
assumes act: $HPhase0Read \ s \ s' \ p \ d$

and $inv: HInv4b \ s \ q$

shows $HInv4b \ s' \ q$

\(\langle proof \rangle\)

### C.4.3 Proofs of Invariant 4c

**lemma** $HStartBallot-HInv4c-p$:  
$[ [ HStartBallot \ s \ s' \ p; HInv4c \ s \ p ] ] \Rightarrow HInv4c \ s' \ p$

\(\langle proof \rangle\)

**lemma** $HStartBallot-HInv4c-q$:  
assumes act: $HStartBallot \ s \ s' \ p$

and $inv: HInv4c \ s \ q$

and $pq: p \neq q$

shows $HInv4c \ s' \ q$

\(\langle proof \rangle\)

**theorem** $HStartBallot-HInv4c$:  
$[ [ HStartBallot \ s \ s' \ p; HInv4c \ s \ q ] ] \Rightarrow HInv4c \ s' \ q$

\(\langle proof \rangle\)

**lemma** $HPhase1or2Write-HInv4c-p$:  
assumes act: $HPhase1or2Write \ s \ s' \ p \ d$

and $inv: HInv4c \ s \ p$

and $inv2c: Inv2c \ s$

shows $HInv4c \ s' \ p$

\(\langle proof \rangle\)

**lemma** $HPhase1or2Write-HInv4c-q$:  
assumes act: $HPhase1or2Write \ s \ s' \ p \ d$

and $inv: HInv4c \ s \ q$

and $pq: p \neq q$
shows $HInv4c s' q$

\begin{proof}
\end{proof}

**Theorem HPhase1or2Write-HInv4c:**
\begin{align*}
[& HPhase1or2Write s s' d; HInv4c s q; Inv2c s ] \\
\implies & HInv4c s' q \\
\end{align*}
\begin{proof}
\end{proof}

**Lemma HPhase1or2ReadThen-HInv4c-p:**
\begin{align*}
[ & HPhase1or2ReadThen s s' d q; HInv4c s p ] & \implies & HInv4c s' p \\
\end{align*}
\begin{proof}
\end{proof}

**Lemma HPhase1or2ReadThen-HInv4c-q:**
\begin{align*}
\text{assumes} & \text{ act: HPhase1or2ReadThen s s' d r} \\
\text{and} & \text{ inv: HInv4c s q} \\
\text{and} & \text{ pmq: p\neq q} \\
\text{shows} & \text{ HInv4c s' q} \\
\end{align*}
\begin{proof}
\end{proof}

**Theorem HPhase1or2ReadThen-HInv4c:**
\begin{align*}
[ & HPhase1or2ReadThen s s' d r; HInv4c s q ] \\
\implies & HInv4c s' q \\
\end{align*}
\begin{proof}
\end{proof}

**Theorem HPhase1or2ReadElse-HInv4c:**
\begin{align*}
[ & HPhase1or2ReadElse s s' d r; HInv4c s q ] & \implies & HInv4c s' q \\
\end{align*}
\begin{proof}
\end{proof}

**Lemma HEndPhase1-HInv4c-p:**
\begin{align*}
\text{assumes} & \text{ act: HEndPhase1 s s' p} \\
\text{and} & \text{ inv2b: Inv2b s} \\
\text{shows} & \text{ HInv4c s' p} \\
\end{align*}
\begin{proof}
\end{proof}

**Lemma HEndPhase1-HInv4c-q:**
\begin{align*}
\text{assumes} & \text{ act: HEndPhase1 s s' p} \\
\text{and} & \text{ inv: HInv4c s q} \\
\text{and} & \text{ pmq: p\neq q} \\
\text{shows} & \text{ HInv4c s' q} \\
\end{align*}
\begin{proof}
\end{proof}

**Theorem HEndPhase1-HInv4c:**
\begin{align*}
[ & HEndPhase1 s s' p; HInv4c s q; Inv2b s ] & \implies & HInv4c s' q \\
\end{align*}
\begin{proof}
\end{proof}

**Lemma HEndPhase2-HInv4c-p:**
\begin{align*}
[ & HEndPhase2 s s' p; HInv4c s p ] & \implies & HInv4c s' p \\
\end{align*}
\begin{proof}
\end{proof}
lemma HEndPhase2-HInv4c-q:
assumes act: HEndPhase2 s s' p
and inv: HInv4c s q
and pnq: p\neq q
shows HInv4c s' q
(proof)

theorem HEndPhase2-HInv4c:
[ [ HEndPhase2 s s' p; HInv4c s q ] ] \implies HInv4c s' q
(proof)

lemma HFail-HInv4c-p:
[ [ HFail s s' p; HInv4c s p ] ] \implies HInv4c s' p
(proof)

lemma HFail-HInv4c-q:
assumes act: HFail s s' p
and inv: HInv4c s q
and pnq: p\neq q
shows HInv4c s' q
(proof)

theorem HFail-HInv4c:
[ [ HFail s s' p; HInv4c s q ] ] \implies HInv4c s' q
(proof)

lemma HPhase0Read-HInv4c-p:
[ [ HPhase0Read s s' p d; HInv4c s p ] ] \implies HInv4c s' p
(proof)

lemma HPhase0Read-HInv4c-q:
assumes act: HPhase0Read s s' p d
and inv: HInv4c s q
and pnq: p\neq q
shows HInv4c s' q
(proof)

theorem HPhase0Read-HInv4c:
[ [ HPhase0Read s s' p d; HInv4c s q ] ] \implies HInv4c s' q
(proof)

lemma HEndPhase0-HInv4c-p:
[ [ HEndPhase0 s s' p; HInv4c s p ] ] \implies HInv4c s' p
(proof)

lemma HEndPhase0-HInv4c-q:
assumes act: HEndPhase0 s s' p
and inv: HInv4c s q
and pnq: p\neq q
shows $HInv4c\ s\ s’\ q$

(proof)

**Theorem HEndPhase0-HInv4c:**

\[
[\ HEndPhase0\ s\ s’\ p;\ HInv4c\ s\ q ] \implies HInv4c\ s’\ q
\]

(proof)

**C.4.4 Proofs of Invariant 4d**

**Lemma HStartBallot-HInv4d-p:**

assumes act: HStartBallot\ s\ s’\ p
and inv: HInv4d\ s\ q
shows HInv4d\ s’\ p

(proof)

**Lemma HStartBallot-HInv4d-q:**

assumes act: HStartBallot\ s\ s’\ p
and inv: HInv4d\ s\ q
and pnq: p\neq q
shows HInv4d\ s’\ q

(proof)

**Theorem HStartBallot-HInv4d:**

\[
[\ HStartBallot\ s\ s’\ p;\ HInv4d\ s\ q ] \implies HInv4d\ s’\ q
\]

(proof)

**Lemma HPhase1or2Write-HInv4d-p:**

assumes act: HPhase1or2Write\ s\ s’\ p\ d
and inv: HInv4d\ s\ q
and inv4a: HInv4a\ s\ p
shows HInv4d\ s’\ p

(proof)

**Lemma HPhase1or2Write-HInv4d-q:**

assumes act: HPhase1or2Write\ s\ s’\ p\ d
and inv: HInv4d\ s\ q
and pnq: p\neq q
shows HInv4d\ s’\ q

(proof)

**Theorem HPhase1or2Write-HInv4d:**

\[
[\ HPhase1or2Write\ s\ s’\ p\ d;\ HInv4d\ s\ q;\ HInv4a\ s\ p ] \implies HInv4d\ s’\ q
\]

(proof)

**Lemma HPhase1or2ReadThen-HInv4d-p:**

assumes act: HPhase1or2ReadThen\ s\ s’\ p\ d\ q
and inv: HInv4d\ s\ p
shows HInv4d\ s’\ p

(proof)
lemma \textit{HPhase1or2ReadThen-HInv4d-q}:
\begin{itemize}
  \item assumes act: \textit{HPhase1or2ReadThen} \(s \ s' p \ d \ r\)
  \item and inv: \textit{HInv4d} \(s q\)
  \item and \(pnq: p \neq q\)
  \item shows \(HInv4d \ s' q\)
\end{itemize}
(proof)

theorem \textit{HPhase1or2ReadThen-HInv4d}:
\[\begin{array}{ll}
\textit{HPhase1or2ReadThen} \(s \ s' p \ d \ r\); \textit{HInv4d} \(s q\) & \implies \textit{HInv4d} \(s' q\)
\end{array}\]
(proof)

theorem \textit{HPhase1or2ReadElse-HInv4d}:
\[\begin{array}{ll}
\textit{HPhase1or2ReadElse} \(s \ s' p \ d \ r\); \textit{HInv4d} \(s q\) & \implies \textit{HInv4d} \(s' q\)
\end{array}\]
(proof)

lemma \textit{HEndPhase1-HInv4d-p}:
\begin{itemize}
  \item assumes act: \textit{HEndPhase1} \(s \ s' p\)
  \item and inv: \textit{HInv4d} \(s p\)
  \item and inv2b: \textit{Inv2b} \(s\)
  \item and inv4c: \textit{HInv4c} \(s p\)
  \item shows \(HInv4d \ s' p\)
\end{itemize}
(proof)

lemma \textit{HEndPhase1-HInv4d-q}:
\begin{itemize}
  \item assumes act: \textit{HEndPhase1} \(s \ s' p\)
  \item and inv: \textit{HInv4d} \(s q\)
  \item and \(pnq: p \neq q\)
  \item shows \(HInv4d \ s' q\)
\end{itemize}
(proof)

theorem \textit{HEndPhase1-HInv4d}:
\[\begin{array}{ll}
\textit{HEndPhase1} \(s \ s' p\); \textit{HInv4d} \(s q\); \textit{Inv2b} \(s\); \textit{HInv4c} \(s p\) & \implies \textit{HInv4d} \(s' q\)
\end{array}\]
(proof)

lemma \textit{HEndPhase2-HInv4d-p}:
\begin{itemize}
  \item assumes act: \textit{HEndPhase2} \(s \ s' p\)
  \item and inv: \textit{HInv4d} \(s p\)
  \item shows \(HInv4d \ s' p\)
\end{itemize}
(proof)

lemma \textit{HEndPhase2-HInv4d-q}:
\begin{itemize}
  \item assumes act: \textit{HEndPhase2} \(s \ s' p\)
  \item and inv: \textit{HInv4d} \(s q\)
  \item and \(pnq: p \neq q\)
  \item shows \(HInv4d \ s' q\)
\end{itemize}
(proof)
Theorem HEndPhase2-HInv4d:
\[
\begin{array}{l}
\text{[ HEndPhase2 } s s' p; \text{ HInv4d } s q] \implies \text{ HInv4d } s' q
\end{array}
\]
(Proof)

Lemma HFail-HInv4d-p:
assumes act: HFail s s' p
and inv: HInv4d s p
shows HInv4d s' p
(Proof)

Lemma HFail-HInv4d-q:
assumes act: HFail s s' p
and inv: HInv4d s q
and pnq: p \neq q
shows HInv4d s' q
(Proof)

Theorem HFail-HInv4d:
\[
\begin{array}{l}
\text{[ HFail } s s' p; \text{ HInv4d } s q] \implies \text{ HInv4d } s' q
\end{array}
\]
(Proof)

Lemma HPhase0Read-HInv4d-p:
assumes act: HPhase0Read s s' p d
and inv: HInv4d s p
shows HInv4d s' p
(Proof)

Lemma HPhase0Read-HInv4d-q:
assumes act: HPhase0Read s s' p d
and inv: HInv4d s q
and pnq: p \neq q
shows HInv4d s' q
(Proof)

Theorem HPhase0Read-HInv4d:
\[
\begin{array}{l}
\text{[ HPhase0Read } s s' p d; \text{ HInv4d } s q] \implies \text{ HInv4d } s' q
\end{array}
\]
(Proof)

Lemma HEndPhase0-blocksOf2:
assumes act: HEndPhase0 s s' p
and inv2c: Inv2c-inner s p
shows allBlocksRead s p \subseteq blocksOf s p
(Proof)

Lemma HEndPhase0-HInv4d-p:
assumes act: HEndPhase0 s s' p
and inv: HInv4d s p
and inv2c: Inv2c s
and inv1: Inv1 s
shows $HInv4d \ s' \ p$

(\text{proof})

\textbf{lemma} $H\text{EndPhase0-HInv4d-q}$:
\textbf{assumes} act: $H\text{EndPhase0} \ s \ s' \ p$
\textbf{and} inv: $HInv4d \ s \ q$
\textbf{and} pnq: $p \neq q$
\textbf{shows} $HInv4d \ s' \ q$
(\text{proof})

\textbf{theorem} $H\text{EndPhase0-HInv4d}$:
\[ [H\text{EndPhase0} \ s \ s' \ p; HInv4d \ s \ q; Inv2c \ s; Inv1 \ s]\implies HInv4d \ s' \ q
(\text{proof})

Since we have already proved $HInv2$ is an invariant of $H\text{Next}$, $HInv1 \land HInv2 \land HInv4$ is also an invariant of $H\text{Next}$.

\textbf{lemma} $I2d$:
\textbf{assumes} nxt: $H\text{Next} \ s \ s'$
\textbf{and} inv: $HInv1 \ s \land HInv2 \ s \land HInv2' \ s' \land HInv4 \ s$
\textbf{shows} $HInv4 \ s'$
(\text{proof})

end

\textit{theory} DiskPaxos-Inv5 \textit{imports} DiskPaxos-Inv3 DiskPaxos-Inv4 begin

\textbf{C.5 Invariant 5}

This invariant asserts that, if a processor $p$ is in phase 2, then either its $bal$ and $inp$ values satisfy $\text{maxBalInp}$, or else $p$ must eventually abort its current ballot. Processor $p$ will eventually abort its ballot if there is some processor $q$ and majority set $D$ such that $p$ has not read $q$'s block on any disk $D$, and all of those blocks have $mbal$ values greater than $bal(dblocksp)$.

\textbf{definition} $\text{maxBalInp} :: state \Rightarrow \text{nat} \Rightarrow \text{InputsOrNi} \Rightarrow \text{bool}$
\textbf{where} $\text{maxBalInp} \ s \ b \ v = (\forall bk\in\text{allBlocks} \ s. \ b \leq bal \ bk \implies inp \ bk = v)$

\textbf{definition} $HInv5\text{-inner-R} :: state \Rightarrow \text{Proc} \Rightarrow \text{bool}$
\textbf{where} $HInv5\text{-inner-R} \ s \ p =$
\hspace{1em} ($\text{maxBalInp} \ s \ (bal(dblock \ s \ p)) \ (inp(dblock \ s \ p))$
\hspace{2em} $\lor (\exists D\in\text{MajoritySet}. \ \exists q. \ (\forall d\in D. \ bal(dblock \ s \ p) < mbal(disk \ s \ d \ q)$
\hspace{3em} $\land \neg\text{hasRead} \ s \ p \ d \ q))$

\textbf{definition} $HInv5\text{-inner} :: state \Rightarrow \text{Proc} \Rightarrow \text{bool}$
\textbf{where} $HInv5\text{-inner} \ s \ p = (\text{phase} \ s \ p = 2 \implies HInv5\text{-inner-R} \ s \ p)$

50
Definition \( HInv5 :: \text{state} \rightarrow \text{bool} \)
where \( HInv5 s = (\forall p. \text{HInv5-inner } s \ p) \)

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

Theorem \( \text{HInit-HInv5}: \text{HInit } s \implies \text{HInv5 } s \)

We will use the notation used in the proofs of invariant 4, and prove the lemma \( \text{action-HInv5-p} \) and \( \text{action-HInv5-q} \) for each action, for the cases \( p = q \) and \( p \neq q \) respectively.

Also, for each action we will define an \( \text{action-allBlocks} \) lemma in the same way that we defined \( -\text{blocksOf} \) lemmas in the proofs of \( HInv2 \). Now we prove that for each action the new \( allBlocks \) are included in the old \( allBlocks \) or, in some cases, included in the old \( allBlocks \) union the new \( dblock \).

Lemma \( \text{HStartBallot-HInv5-p} \):
- Assumes \( \text{act: HStartBallot } s \ s' \ p \)
- And \( \text{inv: HInv5-inner } s \ p \)
- Shows \( \text{HInv5-inner } s' \ p \) (proof)

Lemma \( \text{HStartBallot-blocksOf-q} \):
- Assumes \( \text{act: HStartBallot } s \ s' \ p \)
- And \( \text{p=q} \)
- Shows \( \text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \) (proof)

Lemma \( \text{HStartBallot-allBlocks} \):
- Assumes \( \text{act: HStartBallot } s \ s' \ p \)
- Shows \( \text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ \text{dblock } s' \ p \} \) (proof)

Lemma \( \text{HStartBallot-HInv5-q1} \):
- Assumes \( \text{act: HStartBallot } s \ s' \ p \)
- And \( \text{pq} \)
- And \( \text{inv5-1: maxBalInp } s \ (\text{bal(dblock } s \ q)) \ (\text{inp(dblock } s \ q)) \)
- Shows \( \text{maxBalInp } s' \ (\text{bal(dblock } s' \ q)) \ (\text{inp(dblock } s' \ q)) \) (proof)

Lemma \( \text{HStartBallot-HInv5-q2} \):
- Assumes \( \text{act: HStartBallot } s \ s' \ p \)
- And \( \text{pq} \)
- And \( \text{inv5-2: } \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal(dblock } s \ q) < \text{mbal(disk } s \ d \ qq) \ \wedge \neg \text{hasRead } s \ q \ d \ qq) \)
- Shows \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal(dblock } s' \ q) < \text{mbal(disk } s' \ d \ qq) \ \wedge \neg \text{hasRead } s' \ q \ d \ qq) \)
lemma HStartBallot-HInv5-q:
  assumes act: HStartBallot s s' p
  and inv: HInv5-inner s q
  and pnq: p ≠ q
  shows HInv5-inner s' q

(proof)

theorem HStartBallot-HInv5:
  \[\begin{array}{c}
  HStartBallot s s' p; HInv5-inner s q \\
  \end{array}\] \implies HInv5-inner s' q

(proof)

lemma HPhase1or2Write-HInv5-1:
  assumes act: HPhase1or2Write s s' p d
  and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
  shows maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))

(proof)

lemma HPhase1or2Write-HInv5-p2:
  assumes act: HPhase1or2Write s s' p d
  and inv4c: HInv4c s p
  and phase: phase s p = 2
  and inv5-2: \exists D ∈ MajoritySet. \exists q. (\forall d ∈ D. bal(dblock s q) < mbal(disk s d q))
  \land \lnot hasRead s p d q
  shows \exists D ∈ MajoritySet. \exists q. (\forall d ∈ D. bal(dblock s' p) < mbal(disk s' d q))
  \land \lnot hasRead s' p d q

(proof)

lemma HPhase1or2Write-HInv5-p:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv5-inner s p
  and inv4: HInv4c s p
  shows HInv5-inner s' p

(proof)

lemma HPhase1or2Write-allBlocks:
  assumes act: HPhase1or2Write s s' p d
  shows allBlocks s' ⊆ allBlocks s

(proof)

lemma HPhase1or2Write-HInv5-q2:
  assumes act: HPhase1or2Write s s' p d
  and pnq: p ≠ q
  and inv4a: HInv4a s p
  and inv5-2: \exists D ∈ MajoritySet. \exists qq. (\forall d ∈ D. bal(dblock s q) < mbal(disk s d qq))
  \land \lnot hasRead s q d qq

(52)
\( \exists D \in \text{MajoritySet}. \exists \ qq. (\forall d \in D. \ \text{bal}(\text{dblock } s' \ q) < \text{mbal}(\text{disk } s' \ d \ qq) \ \\
\wedge \neg \text{hasRead } s' \ q \ d \ qq) \)

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2Write-HInv5-q:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2Write \ s \ s' \ p \ d
  \item \textbf{and } inv: HInv5-inner \ s \ q
  \item \textbf{and } inv4a: HInv4a \ s \ p
  \item \textbf{and } pq: p \neq q
\end{itemize}
\textbf{shows } HInv5-inner \ s' \ q

\( (\text{proof}) \)

\textbf{theorem } HPhase1or2Write-HInv5:
\[ [ \text{HPhase1or2Write } s \ s' \ p \ d; \ HInv5-inner \ s \ q; \ HInv4c \ s \ p; \ HInv4a \ s \ p ] \implies \text{HInv5-inner } s' \ q \]

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2ReadThen-HInv5-1:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
  \item \textbf{and } inv5-1: \text{maxBalInp } s \ (\text{bal}(\text{dblock } s \ q)) \ (\text{inp}(\text{dblock } s \ q))
\end{itemize}
\textbf{shows } \text{maxBalInp } s' \ (\text{bal}(\text{dblock } s' \ q)) \ (\text{inp}(\text{dblock } s' \ q))

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2ReadThen-HInv5-p2:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
  \item \textbf{and } inv4c: HInv4c \ s \ p
  \item \textbf{and } inv2c: Inv2c s \ p
  \item \textbf{and } phase: \text{phase } s \ p = 2
  \item \textbf{and } inv5-2: \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\text{dblock } s \ p) < \text{mbal}(\text{disk } s \ d \ q) \ \\
\wedge \neg \text{hasRead } s' \ p \ d \ q)
\end{itemize}
\textbf{shows } \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s' \ d \ q) \ \\
\wedge \neg \text{hasRead } s' \ p \ d \ q)

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2ReadThen-HInv5-p:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
  \item \textbf{and } inv: HInv5-inner \ s \ p
  \item \textbf{and } inv4: HInv4c \ s \ p
  \item \textbf{and } inv2c: Inv2c s
\end{itemize}
\textbf{shows } HInv5-inner \ s' \ p

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2ReadThen-allBlocks:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
  \item \textbf{shows } \text{allBlocks } s' \subseteq \text{allBlocks } s
\end{itemize}

\( (\text{proof}) \)

\textbf{lemma } HPhase1or2ReadThen-HInv5-q2:
\begin{itemize}
  \item \textbf{assumes } act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
\end{itemize}
and \( p_{\neq} q \)
and \( inv4a: HInv4a s p \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s q) < \text{mbal}(disk s d qq)) \)
\( \wedge \neg \text{hasRead s q d qq} \)
shows \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s' q) < \text{mbal}(disk s' d qq)) \)
\( \wedge \neg \text{hasRead s' q d qq} \)

(proof)

lemma \( HPhase1or2ReadThen-HInv5-q \):
assumes act: \( HPhase1or2ReadThen s s' p d r \)
and inv: \( HInv5-inner s q \)
and inv4a: \( HInv4a s p \)
and \( p_{\neq} q \)
shows \( HInv5-inner s' q \)
(proof)

theorem \( HPhase1or2ReadThen-HInv5 \):
\[ [ \begin{array}{l} \text{HPhase1or2ReadThen s s' p d r; HInv5-inner s q; } \\
\text{Inv2c s; HInv4c s p; HInv4a s p } \end{array} ] \Rightarrow HInv5-inner s' q \]
(proof)

theorem \( HPhase1or2ReadElse-HInv5 \):
\[ [ \begin{array}{l} \text{HPhase1or2ReadElse s s' p d r; HInv5-inner s q } \end{array} ] \]
\( \Rightarrow HInv5-inner s' q \)
(proof)

lemma \( HEndPhase2-HInv5-p \):
\( HEndPhase2 s s' p \Rightarrow HInv5-inner s' p \)
(proof)

lemma \( HEndPhase2-allBlocks \):
assumes act: \( HEndPhase2 s s' p \)
shows allBlocks s' \( \subseteq \) allBlocks s
(proof)

lemma \( HEndPhase2-HInv5-q1 \):
assumes act: \( HEndPhase2 s s' p \)
and \( p_{\neq} q \)
and \( inv5-1: \text{maxBalInp s } \{(\text{bal}(dblock s q)) \} (\text{inp}(dblock s q)) \)
shows \( \text{maxBalInp s' } \{(\text{bal}(dblock s' q)) \} (\text{inp}(dblock s' q)) \)
(proof)

lemma \( HEndPhase2-HInv5-q2 \):
assumes act: \( HEndPhase2 s s' p \)
and \( p_{\neq} q \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s q) < \text{mbal}(disk s d qq)) \)
\( \wedge \neg \text{hasRead s q d qq} \)
shows $\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock~} s' q) < \text{mbal}(\text{disk~} s' d\ q) \\wedge \neg\text{hasRead s'~} q\ d\ qq) $

\langle proof \rangle

lemma HEndPhase2-HInv5-q:

assumes act: HEndPhase2 s s'\ p
and inv: HInv5-inner s q
and pnq: p\neq q

shows HInv5-inner s' q

\langle proof \rangle

theorem HEndPhase2-HInv5:

$[ H\text{EndPhase2} s\ s'\ p; H\text{Inv5-inner} s q ] \implies H\text{Inv5-inner} s' q

\langle proof \rangle

lemma HEndPhase2-HInv5-p:

assumes act: HEndPhase1 s s'\ p
and inv4: HInv4 s
and inv2a: Inv2a s
and inv2a': Inv2a s'
and inv2c: Inv2c s
and asm4: $\neg\text{maxBalInp s'} (\text{bal}(\text{dblock s' p})) (\text{inp}(\text{dblock s' p}))$

shows $\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock s' p}) < \text{mbal}(\text{disk s' d q}) \wedge \neg\text{hasRead s'~} p\ d\ q)$

\langle proof \rangle

lemma union-inclusion:

$[ A \subseteq A'; B \subseteq B' ] \implies A \cup B \subseteq A' \cup B'$

\langle proof \rangle

lemma HEndPhase1-blocksOf-q:

assumes act: HEndPhase1 s s'\ p
and pnq: p\neq q

shows blocksOf s' q \subseteq blocksOf s q

\langle proof \rangle

lemma HEndPhase1-allBlocks:

assumes act: HEndPhase1 s s'\ p

shows allBlocks s' \subseteq allBlocks s \cup \{ \text{dblock s' p} \}

\langle proof \rangle

lemma HEndPhase1-HInv5-q:

assumes act: HEndPhase1 s s'\ p
and inv: HInv5 s
and inv1: Inv1 s
and inv2a: Inv2a s'
and inv2a-q: Inv2a s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and phase’: phase s’ q = 2
and pnq: p ≠ q
and asm4: ¬maxBalInp s’ (bal(dblock s’ q)) (inp(dblock s’ q))
s Hows (\(\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(\text{dblock} s’ q) < \text{mbal}(\text{disk} s’ d qq) \land \neg \text{hasRead} s’ q d qq))\)

(\text{proof})

**Theorem HEndPhase1-HInv5:**
assumes act: HEndPhase1 s s’ p
and inv: HInv5 s
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2a’: Inv2a s’
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and inv4: HInv4 s
shows HInv5-inner s’ q
(\text{proof})

**Lemma HFail-HInv5-p:**
HFail s s’ p \(\Rightarrow\) HInv5-inner s’ p
(\text{proof})

**Lemma HFail-blocksOf-q:**
assumes act: HFail s s’ p
and pnq: p ≠ q
shows blocksOf s’ q \(\subseteq\) blocksOf s q
(\text{proof})

**Lemma HFail-allBlocks:**
assumes act: HFail s s’ p
shows allBlocks s’ \(\subseteq\) allBlocks s \(\cup\) \{dblock s’ p\}
(\text{proof})

**Lemma HFail-HInv5-q1:**
assumes act: HFail s s’ p
and pnq: p ≠ q
and inv2a: Inv2a-inner s’ q
and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
s Hows maxBalInp s’ (bal(dblock s’ q)) (inp(dblock s’ q))
(\text{proof})

**Lemma HFail-HInv5-q2:**
assumes act: HFail s s’ p
and pnq: p ≠ q
and inv5-2: \(\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d qq)\)

56
∧ ¬hasRead s q d qq)
shows ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
∧ ¬hasRead s q d qq)
⟨proof⟩

lemma HFail-HInv5-q:
assumes act: HFail s s′ p
and inv: HInv5-inner s q
and pnq: p ≠ q
and inv2a: Inv2a s′
shows HInv5-inner s′ q
⟨proof⟩

theorem HFail-HInv5:
[ HFail s s′ p; HInv5-inner s q; Inv2a s′ ] ⇒ HInv5-inner s′ q
⟨proof⟩

lemma HPhase0Read-HInv5-p:
HPhase0Read s s′ p d ⇒ HInv5-inner s′ q
⟨proof⟩

lemma HPhase0Read-allBlocks:
assumes act: HPhase0Read s s′ p d
shows allBlocks s′ ⊆ allBlocks s
⟨proof⟩

lemma HPhase0Read-HInv5-1:
assumes act: HPhase0Read s s′ p d
and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
shows maxBalInp s′ (bal(dblock s′ q)) (inp(dblock s′ q))
⟨proof⟩

lemma HPhase0Read-HInv5-q2:
assumes act: HPhase0Read s s′ p d
and pnq: p ≠ q
and inv5-2: ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
∧ ¬hasRead s q d qq)
shows ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s′ q) < mbal(disk s′ d qq)
∧ ¬hasRead s′ q d qq)
⟨proof⟩

lemma HPhase0Read-HInv5-q:
assumes act: HPhase0Read s s′ p d
and inv: HInv5-inner s q
and pnq: p ≠ q
shows HInv5-inner s′ q
⟨proof⟩
**Theorem** \( H\text{Phase}0\text{Read-HInv5}: \)

\[
\begin{align*}
\text{if } & H\text{Phase}0\text{Read } s s' p d; \text{HInv5-inner } s q \text{ then } H\text{Inv5-inner } s' q
\end{align*}
\]

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-HInv5-p}: \)

\( H\text{EndPhase}0 s s' p \implies \text{HInv5-inner } s' p \)

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-blocksOf-q}: \)

\text{assumes } \text{act: } H\text{EndPhase}0 s s' p \text{ and } p \neq q

\text{shows } \text{blocksOf } s' q \subseteq \text{blocksOf } s q

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-allBlocks}: \)

\text{assumes } \text{act: } H\text{EndPhase}0 s s' p

\text{shows } \text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ \text{dblock } s' p \}

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-HInv5-q1}: \)

\text{assumes } \text{act: } H\text{EndPhase}0 s s' p \text{ and } p \neq q \text{ and } \text{inv1: } \text{Inv1 } s

\text{shows } \text{maxBalInp } s' (\text{bal} (\text{dblock } s q)) (\text{inp} (\text{dblock } s q)) \text{ and } \neg \text{hasRead } s q d qq

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-HInv5-q2}: \)

\text{assumes } \text{act: } H\text{EndPhase}0 s s' p \text{ and } p \neq q \text{ and } \text{inv5-2: } \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \text{bal} (\text{dblock } s q) < \text{mbal} (\text{disk } s d q q) \wedge \neg \text{hasRead } s q d qq)

\text{shows } \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \text{bal} (\text{dblock } s' q) < \text{mbal} (\text{disk } s' d q q) \wedge \neg \text{hasRead } s' q d qq)

\( \langle \text{proof} \rangle \)

**Lemma** \( H\text{EndPhase}0\text{-HInv5-q}: \)

\text{assumes } \text{act: } H\text{EndPhase}0 s s' p \text{ and } \text{inv: } \text{HInv5-inner } s q \text{ and } \text{inv1: } \text{Inv1 } s \text{ and } p \neq q

\text{shows } \text{HInv5-inner } s' q

\( \langle \text{proof} \rangle \)

**Theorem** \( H\text{EndPhase}0\text{-HInv5}: \)

\[
\begin{align*}
\text{if } & H\text{EndPhase}0 s s' p; \text{HInv5-inner } s q; \text{Inv1 } s \text{ then } H\text{Inv5-inner } s' q
\end{align*}
\]

\( \langle \text{proof} \rangle \)
\(H_{inv1} \land H_{inv2} \land H_{inv3} \land H_{inv4} \land H_{inv5}\) is an invariant of \(H_{next}\).

**Lemma I2e**

assumes \(nxt : H_{next} s s'\)
and \(inv: H_{inv1} s \land H_{inv2} s \land H_{inv2} s' \land H_{inv3} s \land H_{inv4} s \land H_{inv5} s\)
shows \(H_{inv5} s'\)

(proof)

end

**theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin**

C.6 Lemma I2f

To prove the final conjunct we will use the predicate \(\text{valueChosen}(v)\). This predicate is true if \(v\) is the only possible value that can be chosen as output. It also asserts that, for every disk \(d\) in \(D\), if \(q\) has already read \(d_{sdp}\), then it has read a block with \(bal\) field at least \(b\).

**definition valueChosen :: state ⇒ InputsOrNi ⇒ bool where**

\(\text{valueChosen } s \ v =\)

\((\exists b \in (\text{UN } p. \text{Ballot } p). \ 
\text{maxBalInp } s \ b \ v \\
\land (\exists p. \exists D \in \text{MajoritySet}. (\forall \ d \in D. \ b \leq \text{bal}(s d p) \\
\land (\forall q.( \ 
\text{phase } s q = 1 \\
\land b \leq \text{mbal}(\text{dblock } s q) \\
\land \text{hasRead } s q d p \\
) \\
) \\
) \\
) → (\exists \ b_r \in \text{blocksRead } s q d. \ b \leq \text{bal}(\text{block } b_r))\)

(proof)

**lemma HEndPhase1-valueChosen-inp:**

assumes \(act: H_{EndPhase1} s s' q\)
and \(inv2a: \text{Inv2a } s\)
and \(asm1: b \in (\text{UN } p. \text{Ballot } p)\)
and \(bk\text{-blocksOf}: bk \in \text{blocksOf } s r\)
and \(bk: bk \in \text{blocksSeen } s q\)
and \(b\text{-bal}: b \leq \text{bal } bk\)
and \(asm3: \text{maxBalInp } s \ b \ v\)
and \(inv1: \text{Inv1 } s\)
shows \(\text{inp}(\text{dblock } s' q) = v\)
(proof)

**lemma HEndPhase1-maxBalInp:**

assumes \(act: H_{EndPhase1} s s' q\)
and \(asm1: b \in (\text{UN } p. \text{Ballot } p)\)
and \(asm2: D \in \text{MajoritySet}\)
and \(asm3: \text{maxBalInp } s \ b \ v\)
and \(asm4: \forall \ d \in D. \ b \leq \text{bal}(\text{disk } s d p)\)
\(\forall q. (\text{phase } s q = 1 \land b \leq \text{mbal(dblock } s q) \land \text{hasRead } s q d p) \rightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal(block } br))\)

and inv1: Inv1 s
and inv2a: Inv2a s
and inv2b: Inv2b s
shows maxBalInp s' b v
⟨proof⟩

lemma HEndPhase1-valueChosen2:
assumes act: HEndPhase1 s s' q
and asm4: \(\forall d \in D. \ b \leq \text{bal(disk } s d p)\)
\(\forall q. (\text{phase } s q = 1 \land b \leq \text{mbal(dblock } s q) \land \text{hasRead } s q d p) \rightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal(block } br))\) (is \(?P s) shows \(?P s'\)
⟨proof⟩

theorem HEndPhase1-valueChosen:
assumes act: HEndPhase1 s s' q
and vc: valueChosen s v
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2b: Inv2b s
and v-input: v \(\in\) Inputs
shows valueChosen s' v
⟨proof⟩

lemma HStartBallot-maxBalInp:
assumes act: HStartBallot s s' q
and asm3: maxBalInp s b v
shows maxBalInp s' b v
⟨proof⟩

lemma HStartBallot-valueChosen2:
assumes act: HStartBallot s s' q
and asm4: \(\forall d \in D. \ b \leq \text{bal(disk } s d p)\)
\(\forall q. (\text{phase } s q = 1 \land b \leq \text{mbal(dblock } s q) \land \text{hasRead } s q d p) \rightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal(block } br))\) (is \(?P s) shows \(?P s'\)
⟨proof⟩

theorem HStartBallot-valueChosen:
assumes act: HStartBallot s s' q
and \( vc: valueChosen s v \)
and \( v\text{-input}: v \in \text{Inputs} \)
shows \( valueChosen s' v \)
\( \langle \text{proof} \rangle \)

**lemma** \( H\text{Phase1or2Write-maxBalInp}: \)
assumes \( \text{act}: H\text{Phase1or2Write} s s' q d \)
and \( \text{asm3}: \text{maxBalInp} s b v \)
shows \( \text{maxBalInp} s' b v \)
\( \langle \text{proof} \rangle \)

**lemma** \( H\text{Phase1or2Write-valueChosen2}: \)
assumes \( \text{act}: H\text{Phase1or2Write} s s' pp d \)
and \( \text{asm2}: D \in \text{MajoritySet} \)
and \( \text{asm4}: \forall d \in D. \ b \leq \bal(disk s d p) \)
\( \land (\forall q. ( \phase s q = 1 \land b \leq \mbal(dblock s q) \land \hasRead s q d p )) \rightarrow (\exists br \in \text{blocksRead} s q d. \ b \leq \bal(block br))) \) (is \( ?P s \))
and \( \text{inv4}: H\text{inv4a} s pp \)
shows \( ?P s' \)
\( \langle \text{proof} \rangle \)

**theorem** \( H\text{Phase1or2Write-valueChosen}: \)
assumes \( \text{act}: H\text{Phase1or2Write} s s' q d \)
and \( \text{vc}: valueChosen s v \)
and \( v\text{-input}: v \in \text{Inputs} \)
and \( \text{inv4}: H\text{inv4a} s q \)
shows \( valueChosen s' v \)
\( \langle \text{proof} \rangle \)

**lemma** \( H\text{Phase1or2ReadThen-maxBalInp}: \)
assumes \( \text{act}: H\text{Phase1or2ReadThen} s s' q d p \)
and \( \text{asm3}: \text{maxBalInp} s b v \)
shows \( \text{maxBalInp} s' b v \)
\( \langle \text{proof} \rangle \)

**lemma** \( H\text{Phase1or2ReadThen-valueChosen2}: \)
assumes \( \text{act}: H\text{Phase1or2ReadThen} s s' q d pp \)
and \( \text{asm4}: \forall d \in D. \ b \leq \bal(disk s d p) \)
\( \land (\forall q. ( \phase s q = 1 \land b \leq \mbal(dblock s q) \land \hasRead s q d p )) \rightarrow (\exists br \in \text{blocksRead} s q d. \ b \leq \bal(block br))) \) (is \( ?P s \))
shows \( ?P s' \)
\( \langle \text{proof} \rangle \)

**theorem** \( H\text{Phase1or2ReadThen-valueChosen}: \)
assumes act: HPhase1or2ReadThen s s’ q d p
and vc: valueChosen s v
and v-input: v ∈ Inputs
shows valueChosen s’ v
(proof)

theorem HPhase1or2ReadElse-valueChosen:
 HPhase1or2ReadElse s s’ p d r; valueChosen s v; v ∈ Inputs
⇒ valueChosen s’ v
(proof)

lemma HEndPhase2-maxBalInp:
assumes act: HEndPhase2 s s’ q
and asm3: maxBalInp s b v
shows maxBalInp s’ b v
(proof)

lemma HEndPhase2-valueChosen2:
assumes act: HEndPhase2 s s’ q
and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
∧ (∀ q.( phase s q = 1
∧ b ≤ mbal(dblock s q)
∧ hasRead s q d p)) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br)))
shows ?P s
(proof)

theorem HEndPhase2-valueChosen:
assumes act: HEndPhase2 s s’ q
and vc: valueChosen s v
and v-input: v ∈ Inputs
shows valueChosen s’ v
(proof)

lemma HFail-maxBalInp:
assumes act: HFail s s’ q
and asm1: b ∈ (UN p. Ballot p)
and asm3: maxBalInp s b v
shows maxBalInp s’ b v
(proof)

lemma HFail-valueChosen2:
assumes act: HFail s s’ q
and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
∧ (∀ q.( phase s q = 1
∧ b ≤ mbal(dblock s q)
∧ hasRead s q d p)) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br)))
shows ?P s
(proof)
\(\text{proof}\)

\textbf{Theorem} \(H\text{Fail-valueChosen}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{Fail} \; s \; s' \; q\)
  \item \textbf{ and} \(\text{vc}: \text{valueChosen} \; s \; v\)
  \item \textbf{ and} \(\text{v-input}: \; v \in \text{Inputs}\)
  \item \textbf{ shows} \(\text{valueChosen} \; s' \; v\)
\end{enumerate}
\(\text{proof}\)

\textbf{Lemma} \(H\text{Phase0Read-maxBalInp}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{Phase0Read} \; s \; s' \; q \; d\)
  \item \textbf{ and} \(\text{asm3}: \text{maxBalInp} \; s \; b \; v\)
  \item \textbf{ shows} \(\text{maxBalInp} \; s' \; b \; v\)
\end{enumerate}
\(\text{proof}\)

\textbf{Lemma} \(H\text{Phase0Read-valueChosen2}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{Phase0Read} \; s \; s' \; q \; d\)
  \item \textbf{ and} \(\text{asm4}: \forall \; d \in D. \; \; b \leq \text{bal} \; (\text{disk} \; s \; d \; p)\)
\end{enumerate}
\(\text{proof}\)

\textbf{Theorem} \(H\text{Phase0Read-valueChosen}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{Phase0Read} \; s \; s' \; q \; d\)
  \item \textbf{ and} \(\text{vc}: \text{valueChosen} \; s \; v\)
  \item \textbf{ and} \(\text{v-input}: \; v \in \text{Inputs}\)
  \item \textbf{ shows} \(\text{valueChosen} \; s' \; v\)
\end{enumerate}
\(\text{proof}\)

\textbf{Lemma} \(H\text{EndPhase0-maxBalInp}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{EndPhase0} \; s \; s' \; q\)
  \item \textbf{ and} \(\text{asm3}: \text{maxBalInp} \; s \; b \; v\)
  \item \textbf{ and} \(\text{inv1}: \text{Inv1} \; s\)
  \item \textbf{ shows} \(\text{maxBalInp} \; s' \; b \; v\)
\end{enumerate}
\(\text{proof}\)

\textbf{Lemma} \(H\text{EndPhase0-valueChosen2}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{EndPhase0} \; s \; s' \; q\)
  \item \textbf{ and} \(\text{asm4}: \forall \; d \in D. \; \; b \leq \text{bal} \; (\text{disk} \; s \; d \; p)\)
\end{enumerate}
\(\text{proof}\)

\textbf{Lemma} \(H\text{EndPhase0-valueChosen}\):
\begin{enumerate}
  \item \textbf{ Assumes} \(\text{act}: H\text{EndPhase0} \; s \; s' \; q \; d\)
  \item \textbf{ and} \(\text{vc}: \text{valueChosen} \; s \; v\)
  \item \textbf{ and} \(\text{v-input}: \; v \in \text{Inputs}\)
  \item \textbf{ shows} \(\text{valueChosen} \; s' \; v\)
\end{enumerate}
\(\text{proof}\)
theorem HEndPhase0-valueChosen:
assumes act: HEndPhase0 s s' q
and vc: valueChosen s v
and v-input: v ∈ Inputs
and inv1: Inv1 s
shows valueChosen s' v
⟨proof⟩
end

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

The final conjunct of HInv asserts that, once an output has been chosen, valueChosen(chosen) holds, and each processor’s output equals either chosen or NotAnInput.

definition HInv6 :: state ⇒ bool
where
HInv6 s = ((chosen s ≠ NotAnInput → valueChosen s (chosen s))
∧ (∀ p. outpt s p ∈ {chosen s, NotAnInput}))

theorem HInit-HInv6: HInit s → HInv6 s
⟨proof⟩

lemma HEndPhase2-Inv6-1:
assumes act: HEndPhase2 s s' p
and inv: HInv6 s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and inv5: HInv5-inner s p
and chosen': chosen s' ≠ NotAnInput
shows valueChosen s' (chosen s')
⟨proof⟩

lemma valueChosen-equal-case:
assumes max-v: maxBalInp s b v
and Dmaj: D ∈ MajoritySet
and asm-v: ∀ d∈D. b ≤ bal (disk s d p)
and max-w: maxBalInp s ba w
and Damaj: Da ∈ MajoritySet
and asm-w: ∀ d∈Da. ba ≤ bal (disk s d pa)
and b-ba: ba ≤ ba
shows v=w
⟨proof⟩
lemma valueChosen-equal:
assumes v: valueChosen s v
and w: valueChosen s w
shows v = w ⟨proof⟩

lemma HEndPhase2-Inv6-2:
assumes act: HEndPhase2 s s' p
and inv: HInv6 s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and inv5: HInv5-inner s p
and asm: outpt s' r ≠ NotAnInput
shows outpt s' r = chosen s'
⟨proof⟩

theorem HEndPhase2-Inv6:
assumes act: HEndPhase2 s s' p
and inv: HInv6 s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and inv5: HInv5-inner s p
shows HInv6 s'
⟨proof⟩

lemma outpt-chosen:
assumes outpt: outpt s = outpt s'
and inv2c: Inv2c s
and nextp: HNextPart s s'
shows chosen s' = chosen s
⟨proof⟩

lemma outpt-Inv6:
[ outpt s = outpt s'; ∀ p. outpt s p ∈ {chosen s, NotAnInput};
  Inv2c s; HNextPart s s' ] → ∀ p. outpt s' p ∈ {chosen s', NotAnInput}
⟨proof⟩

theorem HStartBallot-Inv6:
assumes act: HStartBallot s s' p
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'
⟨proof⟩

theorem HPhase1or2Write-Inv6:
assumes act: HPhase1or2Write s s' p d
and inv: HInv6 s
and inv4: HInv4a s p
and \text{inv2c}: \text{Inv2c}\ s \\
\text{shows} \ H\text{Inv6}\ s' \\
\langle\text{proof}\rangle \\

\textbf{theorem} \ H\text{Phase1or2ReadThen-Inv6}: \\
\text{assumes act}: \ H\text{Phase1or2ReadThen}\ s\ s'\ p\ d\ q \\
and \text{inv}: \ H\text{Inv6}\ s \\
and \text{inv2c}: \text{Inv2c}\ s \\
\text{shows} \ H\text{Inv6}\ s' \\
\langle\text{proof}\rangle \\

\textbf{theorem} \ H\text{Phase1or2ReadElse-Inv6}: \\
\text{assumes act}: \ H\text{Phase1or2ReadElse}\ s\ s'\ p\ d\ q \\
and \text{inv}: \ H\text{Inv6}\ s \\
and \text{inv2c}: \text{Inv2c}\ s \\
\text{shows} \ H\text{Inv6}\ s' \\
\langle\text{proof}\rangle \\

\textbf{theorem} \ H\text{EndPhase1-Inv6}: \\
\text{assumes act}: \ H\text{EndPhase1}\ s\ s'\ p \\
and \text{inv}: \ H\text{Inv6}\ s \\
and \text{inv1}: \text{Inv1}\ s \\
and \text{inv2a}: \text{Inv2a}\ s \\
and \text{inv2b}: \text{Inv2b}\ s \\
and \text{inv2c}: \text{Inv2c}\ s \\
\text{shows} \ H\text{Inv6}\ s' \\
\langle\text{proof}\rangle \\

\textbf{lemma} \ \text{outpt-chosen-2}: \\
\text{assumes outpt}: \text{outpt}\ s' = (\text{outpt}\ s)\ (p := \text{NotAnInput}) \\
and \text{inv2c}: \text{Inv2c}\ s \\
and \text{nextp}: \ H\text{NextPart}\ s\ s' \\
\text{shows} \ \text{chosen}\ s = \text{chosen}\ s' \\
\langle\text{proof}\rangle \\

\textbf{lemma} \ \text{outpt-HInv6-2}: \\
\text{assumes outpt}: \text{outpt}\ s' = (\text{outpt}\ s)\ (p := \text{NotAnInput}) \\
and \ \forall \ p. \ \text{outpt}\ s\ p \in \{\text{chosen}\ s, \text{NotAnInput}\} \\
and \text{inv2c}: \text{Inv2c}\ s \\
and \text{nextp}: \ H\text{NextPart}\ s\ s' \\
\text{shows} \ \forall \ p. \ \text{outpt}\ s'\ p \in \{\text{chosen}\ s', \text{NotAnInput}\} \\
\langle\text{proof}\rangle \\

\textbf{theorem} \ H\text{Fail-Inv6}: \\
\text{assumes act}: \ H\text{Fail}\ s\ s'\ p \\
and \text{inv}: \ H\text{Inv6}\ s \\
and \text{inv2c}: \text{Inv2c}\ s \\
\text{shows} \ H\text{Inv6}\ s' \\
\langle\text{proof}\rangle \\

66
theorem HPhase0Read-Inv6:
assumes act: HPhase0Read s s' p d
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'
(proof)

theorem HEndPhase0-Inv6:
assumes act: HEndPhase0 s s' p
and inv: HInv6 s
and inv1: Inv1 s
and inv2c: Inv2c s
shows HInv6 s'
(proof)

HInv1 \land HInv2 \land HInv2' \land HInv3 \land HInv4 \land HInv5 \land HInv6 is an invariant
of HNext.

lemma I2f:
assumes nxt: HNext s s'
and inv: HInv1 s \land HInv2 s \land HInv2 s' \land HInv3 s \land HInv4 s \land HInv5 s \land HInv6 s
shows HInv6 s' (proof)

end

theory DiskPaxos-Invariant imports DiskPaxos-Inv6 begin

C.8 The Complete Invariant

definition HInv :: state \Rightarrow bool
where
HInv s = (HInv1 s
\land HInv2 s
\land HInv3 s
\land HInv4 s
\land HInv5 s
\land HInv6 s)

theorem I1:
HInit s \Rightarrow HInv s
(proof)

theorem I2:
assumes inv: HInv s
and nxt: HNext s s'
sows HInv s'
(proof)

67
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

record Istate =
  iinput :: Proc ⇒ InputsOrNi
  ioutput :: Proc ⇒ InputsOrNi
  ichosen :: InputsOrNi
  iallInput :: InputsOrNi set

definition IInit :: Istate ⇒ bool
where
  IInit s = (range (iinput s) ⊆ Inputs ∧ ioutput s = (λp. NotAnInput) ∧ ichosen s = NotAnInput ∧ iallInput s = range (iinput s))

definition IChoose :: Istate ⇒ Istate ⇒ Proc ⇒ bool
where
  IChoose s s' p = (ioutput s p = NotAnInput ∧ (if (ichosen s = NotAnInput) then (∃ ip ∈ iallInput s. ichosen s' = ip ∧ ioutput s' = (ioutput s) (p := ip)) else (ioutput s' = (ioutput s) (p:= ichosen s) ∧ ichosen s' = ichosen s)) ∧ iinput s' = iinput s ∧ iallInput s' = iallInput s)

definition IFail :: Istate ⇒ Istate ⇒ Proc ⇒ bool
where
  IFail s s' p = (ioutput s' = (ioutput s) (p:= NotAnInput) ∧ (∃ ip ∈ Inputs. iinput s' = (iinput s)(p:= ip) ∧ iallInput s' = iallInput s ∪ {ip}) ∧ ichosen s' = ichosen s)

definition INext :: Istate ⇒ Istate ⇒ bool
where
  INext s s' = (∃ p. IChoose s s' p ∨ IFail s s' p)

definition s2is :: state ⇒ Istate
where
  s2is s = (iinput = inpt s, ioutput = outpt s, ichosen = chosen s, iallInput = allInput s)

end
\textbf{theorem} \ R1: \\
\[
[HInit \ s; \ is = s2is \ s] \Rightarrow \ HInit \ is
\]
\langle proof \rangle

\textbf{theorem} \ R2b: \\
assumes inv: HInv \ s \\
and inv': HInv \ s' \\
and nxt: HNext \ s \ s' \\
and srel: is=s2is \ s \land is'=s2is \ s' \\
shows \ (\exists \ p. IFail \ is \ is' \ p \lor IChoose \ is \ is' \ p) \lor is = is'
\langle proof \rangle

\textbf{end}