Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff          Stephan Merz

June 11, 2019

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 $\text{input}[p]$ 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 \( input[p] \) 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 \( input[p] \) 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 \( outpt \)).

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 \( bal \).

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: \( input \) and \( output \). 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:

\[ \text{HDiskSynodSpec} \triangleq HInit \land \Box[HNext]_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \]

where \( HInit \) describes the initial state of the algorithm and \( HNext \) is the action that models all of its state transitions. The variable \( \text{vars} \) is the tuple of all variables used in the algorithm.

Analogously, we have the specification of the Inner module:

\[ \text{ISpec} \triangleq IInit \land \Box[INext]_{\langle \text{input}, \text{output}, \text{chosen}, \text{allInput} \rangle} \]

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

**THEOREM R1** \( HInit \Rightarrow IInit \)

**THEOREM R2** \( HInit \land \Box[HNext]_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \Box[INext]_{ivars} \)

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** \( HInit \land \Box[HNext]_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \Box[HInv] \)

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

A predicate satisfying \( HInv \) is said to be an invariant of \( \text{HDiskSynodSpec} \). To prove R2a, we make \( HInv \) strong enough to satisfy:
\[ \forall d \in D : \text{disk}(d[q]).\text{bal} = bk \]

\[ \exists d \in D. \text{bal}((\text{disk} \ s \ d \ q) = bk) \]

\[ \text{choose} \ x. P \ x \]

\[ \exists d \in D. \text{bal}((\text{disk} \ s \ d \ q) = bk) \]

\[ \forall x. \ P \ x \]

\[ \text{phase}' = \text{phase} \ \text{except} \ ![p] = 1 \]

\[ \text{phase}' = (\text{phase} \ s) (p := 1) \]

\[ \text{UN} \ p. \ \text{blocksOf} \ s \ p \]

\[ v \ s' = v \ s \]

Table 1: Examples of TLA^+ formulas and their counterparts in Isabelle/HOL.

\textbf{THEOREM I1} \quad H\text{Init} \Rightarrow H\text{Inv}

\textbf{THEOREM I2} \quad H\text{Inv} \land H\text{Next} \Rightarrow H\text{Inv}'

Again, we have TLA proof rules that say that \( I1 \) and \( I2 \) imply \( R2a \). In summary, \( R2b, I1, \) and \( I2 \) together imply \( H\text{DiskSynodSpec} \Rightarrow ISpec \).

Finding a predicate \( H\text{Inv} \) that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present \( H\text{Inv} \) as a conjunction of 6 predicates \( H\text{Inv}_1, \ldots, H\text{Inv}_6 \), where \( H\text{Inv}_1 \) is a simple “type invariant” and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of \( H\text{Inv}_i \) by the algorithm’s next-state relation relies on all \( H\text{Inv}_j \) (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+:

\[\text{CONSTANT } \text{Inputs} \]
\[\text{NotAnInput} \triangleq \text{CHOOSE } c : c \notin \text{Inputs} \]
\[\text{DiskBlock} \triangleq \text{mbal} : (\text{UNION Ballot}(p) : p \in \text{Proc}) \cup \{0\},\]
\[\text{bal} : (\text{UNION Ballot}(p) : p \in \text{Proc}) \cup \{0\},\]
\[\text{inp} : \text{Inputs} \cup \{\text{NotAnInput}\}\]

Isabelle/HOL:

typedec \text{InputsOrNi}

defines
\begin{align*}
\text{Inputs} &:: \text{InputsOrNi set} \\
\text{NotAnInput} &:: \text{InputsOrNi} \\
\end{align*}

axioms
\begin{align*}
\text{NotAnInput} &:: \text{NotAnInput} \notin \text{Inputs} \\
\text{InputsOrNi} &:: (\text{UNIV} :: \text{InputsOrNi set}) = \text{Inputs} \cup \{\text{NotAnInput}\} \\
\end{align*}

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

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

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

Isabelle/HOL:

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

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

\[
\text{Phase1or2Read} \triangleq \text{Phase1or2ReadThen} \lor \text{Phase1or2ReadElse}
\]

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

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

\[
\text{HInv2} \triangleq \text{Inv2a} \land \text{Inv2b} \land \text{Inv2c}
\]

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 \texttt{Inv2a}, and after translating to Isabelle/HOL, instead of writing:

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

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 \texttt{Inv2a}, but only for some process \texttt{q} and block (\texttt{dblock s q}).

4 Structure of the Correctness Proof

In [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 $I_2f$.

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

\begin{verbatim}
MODULE Synod

EXTENDS Naturals
CONSTANT N, Inputs
ASSUME (N \in Nat) \land (N > 0)
Proc \triangleq 1..N
NotAnInput \triangleq \text{CHOOSE } c : c \notin Inputs
VARIABLES inputs, output

MODULE Inner

VARIABLES allInput, chosen

IInit \triangleq \land \text{input} \in [\text{Proc} \rightarrow \text{Inputs}]
\land \text{output} = [p \in \text{Proc} \mapsto \text{NotAnInput}]
\land \text{chosen} = \text{NotAnInput}
\land \text{allInput} = \text{input}[p] : p \in \text{Proc}

IChoose(p) \triangleq
\land \text{output}[p] = \text{NotAnInput}
\land \text{IF chosen} = \text{NotAnInput}
  \text{THEN } \text{ip} \in \text{allInput} : \land \text{chosen'} = \text{ip}
  \land \text{output'} = [\text{output} \text{EXCEPT } ![p] = \text{ip}]
  \land \text{UNCHANGED chosen}
  \land \text{UNCHANGED } \langle \text{input, allInput} \rangle
\text{ELSE } \land \text{output'} = [\text{output} \text{EXCEPT } ![p] = \text{chosen}]
  \land \text{UNCHANGED chosen}
\land \text{UNCHANGED } \langle \text{input, allInput} \rangle

IFail(p) \triangleq \land \text{output'} = [\text{output EXCEPT } ![p] = \text{NotAnInput}]
\land \exists \text{ip} \in \text{Inputs} : \land \text{input'} = [\text{input EXCEPT } ![p] = \text{ip}]
\land \text{allInput'} = \text{allInput} \cup \{\text{ip}\}

INext \triangleq \exists p \in \text{Proc} : \text{IChoose(p) } \lor \text{IFail(p)}
ISpec \triangleq \text{IInit } \land \Box [\text{INext}](\text{input, output, chosen, allInput})

IS(chosen, allInput) \triangleq \text{INSTANCE Inner}
SynodSpec \triangleq \exists \text{chosen, allInput} : \text{IS(chosen, allInput)} \land ISpec
\end{verbatim}
B  Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedecl InputsOrNi
typedecl Disk
typedecl 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
majorizes-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 =

13
\[
inpt :: \text{Proc} \Rightarrow \text{InputsOrNi}
\]
\[
outpt :: \text{Proc} \Rightarrow \text{InputsOrNi}
\]
\[
disk :: \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock}
\]
\[
dbloc :: \text{Proc} \Rightarrow \text{DiskBlock}
\]
\[
\text{phase} :: \text{Proc} \Rightarrow \text{nat}
\]
\[
disksWritten :: \text{Proc} \Rightarrow \text{Disk set}
\]
\[
\text{blocksRead} :: \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{BlockProc set}
\]
\[
\text{allInput} :: \text{InputsOrNi set}
\]
\[
\text{chosen} :: \text{InputsOrNi}
\]

**Definition**

\[
\text{hasRead} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

\[
\text{where hasRead s p d q} = (\exists \ br \in\ \text{blocksRead s p d}. \ \text{proc br} = q)
\]

**Definition**

\[
\text{allRdBlks} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{BlockProc set}
\]

\[
\text{where allRdBlks s p} = (\bigcup d. \ \text{blocksRead s p d})
\]

**Definition**

\[
\text{allBlocksRead} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set}
\]

\[
\text{where allBlocksRead s p} = \ \text{block '}(\text{allRdBlks s p})
\]

**Definition**

\[
\text{Init} :: \text{state} \Rightarrow \text{bool}
\]

\[
\text{where Init s} =
\]
\[
(\text{range (inpt s) } \subseteq \text{Inputs}
\]
\[
\text{& outpt s} = (\lambda p. \text{NotAnInput})
\]
\[
\text{& disk s} = (\lambda d. \text{InitDB})
\]
\[
\text{& phase s} = (\lambda p. \text{0})
\]
\[
\text{& dbloc s} = (\lambda p. \text{InitDB})
\]
\[
\text{& disksWritten s} = (\lambda p. \{\})
\]
\[
\text{& blocksRead s} = (\lambda d. \{\})
\]

**Definition**

\[
\text{InitializePhase} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

\[
\text{where InitializePhase s s' p} =
\]
\[
(\text{disksWritten s'} = (\text{disksWritten s})(p := \{\})
\]
\[
\text{& blocksRead s'} = (\text{blocksRead s})(p := (\lambda d. \{\}))
\]

**Definition**

\[
\text{StartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

\[
\text{where StartBallot s s' p} =
\]
\[
(\text{phase s p } \in [1,2]
\]
\[
\text{& phase s'} = (\text{phase s})(p := 1)
\]
\[
\text{& } (\exists b \in \text{Ballot p}. 
\]
\[
\text{mbal (dbloc s p) < b}
\]
\[
\text{& dbloc s'} = (\text{dbloc s})(p := (\text{dbloc s p})(\text{mbal := b }))
\]
\[
\text{& InitializePhase s s' p}
\]
\[
\text{& inpt s'} = \text{inpt s} \text{& outpt s'} = \text{outpt s} \text{& disk s'} = \text{disk s})
\]

14
\textbf{definition} \texttt{Phase1or2Write} :: state \Rightarrow state \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{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} :: state \Rightarrow state \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\textbf{where} \\
\texttt{Phase1or2ReadThen} s s' p d q = \\
(\text{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 \{b | \text{block} = \text{disk} s d q, \text{proc} = \text{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} :: state \Rightarrow state \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\textbf{where} \\
\texttt{Phase1or2ReadElse} s s' p d q = \\
(\text{d} \in \text{disksWritten} s p \\
\land \text{StartBallot} s s' p)

\textbf{definition} \texttt{Phase1or2Read} :: state \Rightarrow state \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{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} :: state \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set} \\
\textbf{where} \texttt{blocksSeen} s p = \text{allBlocksRead} s p \cup \{\text{dblock} s p\}

\textbf{definition} \texttt{nonInitBlks} :: state \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set} \\
\textbf{where} \texttt{nonInitBlks} s p = \{b . \ b \in \text{blocksSeen} s p \land \text{inp} b s \in \text{Inputs}\}

\textbf{definition} \texttt{maxBlk} :: state \Rightarrow \text{Proc} \Rightarrow \text{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} :: state \Rightarrow state \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\textbf{where} \texttt{EndPhase1} s s' p = \\
(\text{IsMajority} \{d . d \in \text{disksWritten} s p}
\(\forall q \in \text{UNIV} - \{p\}. \text{hasRead} s p d q\) \\
\(\land \text{phase} s p = 1\) \\
\(\land \text{dblock} s' = (\text{dblock} s) (p := \text{dblock} s p)\) \\
\(\{\text{bal} := \text{mbal}(\text{dblock} s p),\) \\
\(\text{inp} :=\) \\
\(\text{(if nonInitBlks} s p = \{}\) \\
\(\text{then inpt} s p\) \\
\(\text{else inpt} s p\))} \\
\(\}\) \\
\(\land \text{outpt} s' = \text{outpt} s\) \\
\(\land \text{phase} s' = (\text{phase} s) (p := \text{phase} s p + 1)\) \\
\(\land \text{InitializePhase} s s' p\) \\
\(\land \text{inpt} s' = \text{inpt} s \land \text{disk} s' = \text{disk} s)\)

**definition** **EndPhase2** :: state ⇒ state ⇒ Proc ⇒ 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 ⇒ state ⇒ Proc ⇒ bool 
**where** 
\(\text{EndPhase1or2} s s' p = (\text{EndPhase1} s s' p \lor \text{EndPhase2} s s' p)\)

**definition** **Fail** :: state ⇒ state ⇒ Proc ⇒ 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 ⇒ state ⇒ Proc ⇒ Disk ⇒ bool 
**where** 
\(\text{Phase0Read} s s' p d =\) 
\(\{\text{phase} s p = 0\) \\
\(\land \text{blocksRead} s' = (\text{blocksRead} s) (p := (\text{blocksRead} s p) (d := \text{blocksRead} s p d\) \\
\(\lor \{\{\text{block} = \text{disk} s d p, \text{proc} = p\}\})\) \\
\(\land \text{inpt} s' = \text{inpt} s \land \text{outpt} s' = \text{outpt} s\) \\
\(\land \text{disk} s' = \text{disk} s \land \text{phase} s' = \text{phase} s\) \\
\(\land \text{dblock} s' = \text{dblock} s \land \text{disksWritten} s' = \text{disksWritten} s)\)
definition EndPhase0 :: state $\rightarrow$ state $\rightarrow$ Proc $\rightarrow$ bool
where
EndPhase0 s s' p =
  (phase s p = 0
  $\land$ IsMajority ($\{d.\ hasRead s p d p\}$)
  $\land$ ($\exists b \in$ Ballot p.
    ($\forall r \in$ allBlocksRead s p. mbal r < b)
    $\land$ dblock s' = (dblock s) ( p:=
      (SOME r. r $\in$ allBlocksRead s p
        $\land$ ($\forall s \in$ allBlocksRead s p. bal s $\leq$ bal r)) ( mbal := b | )))
  $\land$ InitializePhase s s' p
  $\land$ phase s' = (phase s) (p:= 1)
  $\land$ inpt s' = inpt s $\land$ outpt s' = outpt s $\land$ disk s' = disk s)

definition Next :: state $\rightarrow$ state $\rightarrow$ bool
where
Next s s' = ($\exists p.$
  StartBallot s s' p
  $\lor$ ($\exists d.$ Phase0Read s s' p d
    $\lor$ Phase1or2Write s s' p d
    $\lor$ ($\exists q. q \neq p \land$ Phase1or2Read s s' p d q))
  $\lor$ EndPhase1or2 s s' p
  $\lor$ Fail s s' p
  $\lor$ 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 $\rightarrow$ bool
where
HInit s =
  (Init s
  $\land$ chosen s = NotAnInput
  $\land$ allInput s = range (inpt s))

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

definition HNextPart :: state $\rightarrow$ state $\rightarrow$ bool
where
HNextPart s s' =
  (chosen s' =
    (if chosen s $\neq$ NotAnInput $\lor$ ($\forall p. outpt s' p =$ NotAnInput )
      then chosen s
      else outpt s' (SOME p. outpt s' p $\neq$ NotAnInput))
  $\land$ allInput s' = allInput s $\cup$ (range (inpt s')))
We add $H\text{NextPart}$ to every action (rather than proving that $\text{Next}$ maintains the $H\text{Inv}$ invariant) to make proofs easier.

**Definition**

$H\text{Phase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}$ where

$H\text{Phase1or2ReadThen} s s' p d q = (\text{Phase1or2ReadThen} s s' p d q \land H\text{NextPart} s s')$

**Definition**

$H\text{EndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{EndPhase1} s s' p = (\text{EndPhase1} s s' p \land H\text{NextPart} s s')$

**Definition**

$H\text{StartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{StartBallot} s s' p = (\text{StartBallot} s s' p \land H\text{NextPart} s s')$

**Definition**

$H\text{Phase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}$ where

$H\text{Phase1or2Write} s s' p d = (\text{Phase1or2Write} s s' p d \land H\text{NextPart} s s')$

**Definition**

$H\text{Phase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{Phase1or2ReadElse} s s' p d q = (\text{Phase1or2ReadElse} s s' p d q \land H\text{NextPart} s s')$

**Definition**

$H\text{EndPhase2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{EndPhase2} s s' p = (\text{EndPhase2} s s' p \land H\text{NextPart} s s')$

**Definition**

$H\text{Fail} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{Fail} s s' p = (\text{Fail} s s' p \land H\text{NextPart} s s')$

**Definition**

$H\text{Phase0Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}$ where

$H\text{Phase0Read} s s' p d = (\text{Phase0Read} s s' p d \land H\text{NextPart} s s')$

**Definition**

$H\text{EndPhase0} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$ where

$H\text{EndPhase0} s s' p = (\text{EndPhase0} s s' p \land H\text{NextPart} s s')$

Since these definitions are the conjunction of two other definitions declaring them as simplification rules should be harmless.

**Declare** $H\text{Phase1or2ReadThen-def} \ [\text{simp}]$

**Declare** $H\text{Phase1or2ReadElse-def} \ [\text{simp}]$

**Declare** $H\text{EndPhase1-def} \ [\text{simp}]$

**Declare** $H\text{StartBallot-def} \ [\text{simp}]$


```plaintext
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 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⟩
```

19
\textbf{theorem} \textit{HPhase1or2ReadThen-HInv1}:
assumes inv1: HInv1 s
and act: HPhase1or2ReadThen s s' s p d q
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HEndPhase1-HInv1}:
assumes inv1: HInv1 s
and act: HEndPhase1 s s' s p
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HStartBallot-HInv1}:
assumes inv1: HInv1 s
and act: HStartBallot s s' s p
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HPhase1or2Write-HInv1}:
assumes inv1: HInv1 s
and act: HPhase1or2Write s s' s p d
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HPhase1or2ReadElse-HInv1}:
assumes act: HPhase1or2ReadElse s s' s p d q
and inv1: HInv1 s
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HEndPhase2-HInv1}:
assumes inv1: HInv1 s
and act: HEndPhase2 s s' s p
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HFail-HInv1}:
assumes inv1: HInv1 s
and act: HFail s s' s p
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{HPhase0Read-HInv1}:
assumes inv1: HInv1 s
and act: HPhase0Read s s' s p d
shows HInv1 s'
(\textit{proof})

\textbf{theorem} \textit{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 \}
  \cup \{ disk s d p | d , d ∈ UNIV \}
  \cup \{ block br | br . br ∈ (UN q d . rdBy s p q d) \}

definition allBlocks :: state ⇒ DiskBlock set
where
  allBlocks s = (UN p . blocksOf s p)

definition Inv2a-innermost :: state ⇒ Proc ⇒ DiskBlock ⇒ bool
where
  Inv2a-innermost s p bk = 
  (mbal bk ∈ (Ballot p) ∪ \{0\})
  ∧ bal bk ∈ (Ballot p) ∪ \{0\}
  ∧ (bal bk = 0) = (inp bk = NotAnInput)
  ∧ bal bk ≤ mbal bk
  ∧ inp bk ∈ (allInput s) ∪ \{NotAnInput\})
definition \textit{Inv2a-inner} :: state \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
where \textit{Inv2a-inner} s p = (\forall bk \in \text{blocksOf} \ s \ p. \ \textit{Inv2a-innermost} \ s \ p \ bk)

definition \textit{Inv2a} :: state \Rightarrow \text{bool} \\
where \textit{Inv2a} s = (\forall p. \ \textit{Inv2a-inner} s \ p)

definition \textit{Inv2b-inner} :: state \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \\
where \textit{Inv2b-inner} s p d = \\
\begin{aligned}
& ((d \in \text{disksWritten} \ s \ p \ \longrightarrow \\
& \quad (\text{phase} \ s \ p \in \{1,2\} \land \text{disk} \ s \ d \ p \ = \ dBloc \ s \ p)) \\
& \land (\text{phase} \ s \ p \in \{1,2\} \ \longrightarrow \\
& \quad (\text{blocksRead} \ s \ p \ d \ \neq \ {} \ \longrightarrow \ d \in \text{disksWritten} \ s \ p)) \\
& \land \neg \text{hasRead} \ s \ p \ d)
\end{aligned}
\land
\begin{aligned}
& ((\text{phase} \ s \ p \neq 0 \ \longrightarrow \\
& \quad (\text{dbloc} \ s \ p \ = \ InitDB \\
& \land \text{disksWritten} \ s \ p \ = \ {} \\
& \land (\forall d. \forall \ br \in \text{blocksRead} \ s \ p \ d. \\
& \quad \text{proc} \ br \ = \ p \land \text{block} \ br \ = \ \text{disk} \ s \ d \ p))) \\
& \land (\text{phase} \ s \ p \neq 0 \ \longrightarrow \\
& \quad (\text{mbal}(\text{dbloc} \ s \ p) \in \\text{Ballot} \ p \\
& \land \text{bal}(\text{dbloc} \ s \ p) \in \\text{Ballot} \ p \cup \{0\} \\
& \land (\forall d. \forall \ br \in \text{blocksRead} \ s \ p \ d. \\
& \quad \text{mbal}(\text{block} \ br) < \text{mbal}(\text{dbloc} \ s \ p))))) \\
& \land (\text{phase} \ s \ p \in \{2,3\} \ \longrightarrow \ \text{bal}(\text{dbloc} \ s \ p) = \text{mbal}(\text{dbloc} \ s \ p)) \\
& \land \text{outpt} \ s \ p = (\text{if} \ \text{phase} \ s \ p = 3 \ \text{then} \ \text{inp}(\text{dbloc} \ s \ p) \ \text{else} \ \text{NotAnInput}) \\
& \land \text{chosen} \ s \in \text{allInput} \ s \cup \{\text{NotAnInput}\} \\
& \land (\forall p. \ \text{inpt} \ s \ p \in \text{allInput} \ s \\
& \land (\text{chosen} \ s = \text{NotAnInput} \ \longrightarrow \ \text{outpt} \ s \ p = \text{NotAnInput})))
\end{aligned}
\land
\begin{aligned}
& \land (\forall p. \ \text{inpt} \ s \ p \in \text{allInput} \ s \\
& \land (\text{chosen} \ s = \text{NotAnInput} \ \longrightarrow \ \text{outpt} \ s \ p = \text{NotAnInput}))
\end{aligned}
\newline

definition \textit{Inv2c-inner} :: state \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
where \textit{Inv2c-inner} s p = \\
\begin{aligned}
& ((\text{phase} \ s \ p = 0 \ \longrightarrow \\
& \quad (\text{dbloc} \ s \ p = \ InitDB \\
& \land \text{disksWritten} \ s \ p \ = \ {} \\
& \land (\forall d. \forall \ br \in \text{blocksRead} \ s \ p \ d. \\
& \quad \text{proc} \ br \ = \ p \land \text{block} \ br \ = \ \text{disk} \ s \ d \ p))))) \\
& \land (\text{phase} \ s \ p \neq 0 \ \longrightarrow \\
& \quad (\text{mbal}(\text{dbloc} \ s \ p) \in \\text{Ballot} \ p \\
& \land \text{bal}(\text{dbloc} \ s \ p) \in \\text{Ballot} \ p \cup \{0\} \\
& \land (\forall d. \forall \ br \in \text{blocksRead} \ s \ p \ d. \\
& \quad \text{mbal}(\text{block} \ br) < \text{mbal}(\text{dbloc} \ s \ p))) \\
& \land (\text{phase} \ s \ p \in \{2,3\} \ \longrightarrow \ \text{bal}(\text{dbloc} \ s \ p) = \text{mbal}(\text{dbloc} \ s \ p)) \\
& \land \text{outpt} \ s \ p = (\text{if} \ \text{phase} \ s \ p = 3 \ \text{then} \ \text{inp}(\text{dbloc} \ s \ p) \ \text{else} \ \text{NotAnInput}) \\
& \land \text{chosen} \ s \in \text{allInput} \ s \cup \{\text{NotAnInput}\} \\
& \land (\forall p. \ \text{inpt} \ s \ p \in \text{allInput} \ s \\
& \land (\text{chosen} \ s = \text{NotAnInput} \ \longrightarrow \ \text{outpt} \ s \ p = \text{NotAnInput}))
\end{aligned}
\land
\begin{aligned}
& \land (\forall p. \ \text{inpt} \ s \ p \in \text{allInput} \ s \\
& \land (\text{chosen} \ s = \text{NotAnInput} \ \longrightarrow \ \text{outpt} \ s \ p = \text{NotAnInput}))
\end{aligned}
\newline

definition \textit{Inv2c} :: state \Rightarrow \text{bool} \\
where \textit{Inv2c} s = (\forall p. \ \textit{Inv2c-inner} s \ p)

definition \textit{HInv2} :: state \Rightarrow \text{bool} \\
where \textit{HInv2} s = (\textit{Inv2a} s \land \textit{Inv2b} s \land \textit{Inv2c} s)

C.2.1 Proofs of Invariant 2 a

\textbf{theorem} \textit{HInit-Inv2a}: \textit{HInit} s \ \longrightarrow \ \textit{Inv2a} s
\langle \text{proof} \rangle

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

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

**theorem** HPhase1or2ReadThen-Inv2a:
\[
\begin{align*}
\text{assumes } & \text{inv: Inv2a } s \\
\text{and } & \text{act: HPhase1or2ReadThen } s s' p d q \\
\text{shows } & \text{Inv2a } s'
\end{align*}
\]
(\text{proof})

**lemma** InitializePhase-rdBy:
\[
\text{InitializePhase } s s' p \implies \text{rdBy } s' pp qq dd \subseteq \text{rdBy } s pp qq dd
\]
(\text{proof})

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

**lemma** HStartBallot-Inv2a-dblock:
\[
\begin{align*}
\text{assumes } & \text{act: HStartBallot } s s' p \\
\text{and } & \text{inv2a: Inv2a-innermost } s p (\text{dblock } s p) \\
\text{shows } & \text{Inv2a-innermost } s' p (\text{dblock } s' p)
\end{align*}
\]
(\text{proof})

**lemma** HStartBallot-Inv2a-dblock-q:
\[
\begin{align*}
\text{assumes } & \text{act: HStartBallot } s s' p \\
\text{and } & \text{inv2a: Inv2a-innermost } s q (\text{dblock } s q) \\
\text{shows } & \text{Inv2a-innermost } s' q (\text{dblock } s' q)
\end{align*}
\]
(\text{proof})

**theorem** HStartBallot-Inv2a:
\[
\begin{align*}
\text{assumes } & \text{inv: Inv2a } s \\
\text{and } & \text{act: HStartBallot } s s' p \\
\text{shows } & \text{Inv2a } s'
\end{align*}
\]
(\text{proof})

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

**theorem** HPhase1or2Write-Inv2a:
\[
\begin{align*}
\text{assumes } & \text{inv: Inv2a } s \\
\text{and } & \text{act: HPhase1or2Write } s s' p d
\end{align*}
\]

shows Inv2a s'
(proof)

**theorem** HPhase1or2ReadElse-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase1or2ReadElse s s' p d q
  shows Inv2a s'
(proof)

**lemma** HEndPhase2-blocksOf:
  HEndPhase2 s s' p \implies blocksOf s' q \subseteq blocksOf s q
(proof)

**theorem** HEndPhase2-Inv2a:
  assumes inv: Inv2a s
  and act: HEndPhase2 s s' p
  shows Inv2a s'
(proof)

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

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

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

**lemma** HPhase0Read-blocksOf:
  HPhase0Read s s' p d \implies blocksOf s' q \subseteq blocksOf s q
(proof)

**theorem** HPhase0Read-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase0Read s s' p d
  shows Inv2a s'
(proof)

**lemma** HEndPhase0-blocksOf:
  HEndPhase0 s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
(proof)
lemma HEndPhase0-blocksRead:
assumes act: HEndPhase0 s s' p
shows \exists d. blocksRead s p d \neq {}
⟨proof⟩

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: \( ?P \ ?x \Rightarrow ?P \ (Eps \ ?P) \).

lemma HEndPhase0-some:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
shows (SOME b. b \in allBlocksRead s p 
    \land (\forall t \in allBlocksRead s p. bal t \leq bal b)) \in allBlocksRead s p
    \land (\forall t \in allBlocksRead s p. bal t \leq bal (SOME b. b \in allBlocksRead s p 
    \land (\forall t \in allBlocksRead s p. bal t \leq bal b)))
⟨proof⟩

lemma HEndPhase0-dblock-allBlocksRead:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
shows dblock s' p \in (\lambda x. x (\lambda x. mbal:= mbal(dblock s' p))) \ \ \ allBlocksRead s p
⟨proof⟩

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\}
⟨proof⟩

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 (\lambda x. mbal:= mbal (dblock s' p))) \ \ \ allBlocksRead s p.
    Inv2a-innermost s p t
⟨proof⟩

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)
⟨proof⟩

lemma HEndPhase0-Inv2a-dblock-q:
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)`
⟨proof⟩

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'`
⟨proof⟩

lemma HEndPhase1-blocksOf:
`HEndPhase1 s s' p =⇒ blocksOf s' q ⊆ blocksOf s q ∪ {dblock s' q}`
⟨proof⟩

lemma maxBlk-in-nonInitBlks:
assumes `b: b ∈ nonInitBlks s p`
and `inv1: Inv1 s`
shows `maxBlk s p ∈ nonInitBlks s p`
∧ (∀ c∈ nonInitBlks s p. bal c ≤ bal (maxBlk s p))
⟨proof⟩

lemma blocksOf-nonInitBlks:
(∀ p bk. bk ∈ blocksOf s p =⇒ P bk)
⇒ bk ∈ nonInitBlks s p =⇒ P bk
⟨proof⟩

lemma maxBlk-allInput:
assumes `inv: Inv2a s`
and `mblk: maxBlk s p ∈ nonInitBlks s p`
shows `inp (maxBlk s p) ∈ allInput s`
⟨proof⟩

lemma HEndPhase1-dblock-allInput:
assumes `act: HEndPhase1 s s' p`
and `inv1: HInv1 s`
and `inv2: Inv2a s`
shows `inp': inp (dblock s' p) ∈ allInput s'`
⟨proof⟩

lemma HEndPhase1-Inv2a-dblock:
assumes `act: HEndPhase1 s s' p`
and `inv1: HInv1 s`
and `inv2: Inv2a s`
and `inv2c: Inv2c-inner s p`
shows \( \text{Inv2a-innermost } s' \ p \ (\text{dblock } s' \ p) \)

(\text{proof})

\text{lemma } \text{HEndPhase1-Inv2a-dblock-q}:
assumes act: \text{HEndPhase1 } s \ s' \ p
and inv1: \text{HInv1 } s
and inv: \text{Inv2a } s
and inv2c: \text{Inv2c-inner } s \ p
shows \( \text{Inv2a-innermost } s' \ q \ (\text{dblock } s' \ q) \)
(\text{proof})

\text{theorem } \text{HEndPhase1-Inv2a}:
assumes act: \text{HEndPhase1 } s \ s' \ p
and inv1: \text{HInv1 } s
and inv: \text{Inv2a } s
and inv2c: \text{Inv2c-inner } s \ p
shows \( \text{Inv2a } s' \)
(\text{proof})

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

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

\text{theorem } \text{HInit-Inv2b}: \text{HInit } s \rightarrow \text{Inv2b } s
(\text{proof})

\text{theorem } \text{HPhase1or2ReadThen-Inv2b}:
[ \text{Inv2b } s; \text{HPhase1or2ReadThen } s \ s' \ p \ d \ q ]
\implies \text{Inv2b } s'
(\text{proof})

\text{theorem } \text{HStartBallot-Inv2b}:
[ \text{Inv2b } s; \text{HStartBallot } s \ s' \ p ]
\implies \text{Inv2b } s'
(\text{proof})

\text{theorem } \text{HPhase1or2Write-Inv2b}:
[ \text{Inv2b } s; \text{HPhase1or2Write } s \ s' \ p \ d ]
\implies \text{Inv2b } s'
(\text{proof})

\text{theorem } \text{HPhase1or2ReadElse-Inv2b}:
[ \text{Inv2b } s; \text{HPhase1or2ReadElse } s \ s' \ p \ d \ q ]
\implies \text{Inv2b } s'
(\text{proof})

\text{theorem } \text{HEndPhase1-Inv2b}:
[ \text{Inv2b } s; \text{HEndPhase1 } s \ s' \ p ] \implies \text{Inv2b } s'
(\text{proof})
theorem HFail-Inv2b:
\[ \text{Inv2b } s ; \text{HFail } s \ s' \ p \] \[ \implies \text{Inv2b } s' \]
(proof)

theorem HEndPhase2-Inv2b:
\[ \text{Inv2b } s ; \text{HEndPhase2 } s \ s' \ p \] \[ \implies \text{Inv2b } s' \]
(proof)

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

theorem HEndPhase0-Inv2b:
\[ \text{Inv2b } s ; \text{HEndPhase0 } s \ s' \ p \] \[ \implies \text{Inv2b } s' \]
(proof)

C.2.3 Proofs of Invariant 2 c

theorem HInit-Inv2c: HInit \ s \implies \text{Inv2c } s
(proof)

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 \text{inp(dblock } s' \ p) \
else \text{NotAnInput})
and inp-dblk: \forall p. \text{inp } (\text{dblock } s' \ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}
shows chosen \ s' \in \text{allInput } s' \cup \{\text{NotAnInput}\}
(proof)

lemma HNextPart-chosen:
assumes hnp: HNextPart \ s \ s'
shows chosen \ s' = \text{NotAnInput} \implies (\forall p. \text{outpt } s' \ p = \text{NotAnInput})
(proof)

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

theorem HPhase1or2ReadThen-Inv2c:
assumes inv: Inv2c \ s
and act: HPhase1or2ReadThen \ s \ s' \ p \ d \ q
and inv2a: Inv2a \ s
shows Inv2c \ s'
(proof)

28
**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:**
\[ \text{Inv2c s}; \text{HPhase1or2ReadElse s s' p d q; Inv2a s} \] \implies \text{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 =⇒ HInv2 s
⟨proof⟩

HInv1 ∧ HInv2 is an invariant of HNext.

lemma I2b:
  assumes nxt: HNext s s'
  and inv: HInv1 s ∧ 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 ⇒ Proc ⇒ Proc ⇒ Disk ⇒ bool
where
  HInv3-L s p q d = (phase s p ∈ {1, 2}
                   ∧ phase s q ∈ {1, 2}
                   ∧ hasRead s p d q
                   ∧ hasRead s q d p)

definition HInv3-R :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ bool
where
  HInv3-R s p q d = ((block= dblock s q, proc= q) ∈ blocksRead s p d
                     ∨ (block= dblock s p, proc= p) ∈ blocksRead s q d)

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

definition HInv3 :: state ⇒ bool
where
  HInv3 s = (∀ p q d. HInv3-inner s p q d)

C.3.1 Proofs of Invariant 3

theorem HInit-HInv3: HInit s =⇒ HInv3 s

30
lemma InitPhase-HInv3-p:
\[ \begin{array}{c}
\text{InitializePhase } s \; s' \; p ; \ HInv3-L \ s' \; p \; q \; d \ \implies \ HInv3-R \ s' \; p \; q \; d \\
\end{array} \] (proof)

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

lemma HInv3-L-sym: HInv3-L \( s \; p \; q \; d \) = \( HInv3-L \; q \; p \; d \) (proof)

lemma HInv3-R-sym: HInv3-R \( s \; p \; q \; d \) = \( HInv3-R \; q \; p \; d \) (proof)

lemma Phase1or2ReadThen-HInv3-pq:
\[ \begin{array}{c}
\text{assumes act: } \text{Phase1or2ReadThen } s \; s' \; p \; d \; q \\
\text{and inv-L': } \ HInv3-L \ s' \; p \; q \; d \\
\text{and pq: } p \not= q \\
\text{and inv2b: } \text{Inv2b } s \\
\text{shows } \ HInv3-R \ s' \; p \; q \; d \\
\end{array} \] (proof)

lemma Phase1or2ReadThen-HInv3-hasRead:
\[ \begin{array}{c}
\text{¬hasRead } s \; pp \; dd \; qq; \\
\text{Phase1or2ReadThen } s \; s' \; p \; d \; q; \\
\text{pp} \not= p \lor \text{qq} \not= q \lor \text{dd} \not= d \\
\implies \text{¬hasRead } s' \; pp \; dd \; qq \\
\end{array} \] (proof)

theorem HPhase1or2ReadThen-HInv3:
\[ \begin{array}{c}
\text{assumes act: } \ HPhase1or2ReadThen \ s \; s' \; p \; d \; q \\
\text{and inv: } \ HInv3 \ s \\
\text{and pq: } p \not= q \\
\text{and inv2b: } \text{Inv2b } s \\
\text{shows } \ HInv3 \ s' \\
\end{array} \] (proof)

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

lemma StartBallot-HInv3-q:
\[ \begin{array}{c}
\text{StartBallot } s \; s' \; q ; \ HInv3-L \ s' \; p \; q \; d \ \implies \ HInv3-R \ s' \; p \; q \; d \\
\end{array} \] (proof)
lemma StartBallot-HInv3-nL:
\[
\begin{aligned}
&\text{StartBallot } s \ s' \ t; \ \neg \text{HInv3-L } s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies \ &\neg \text{HInv3-L } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

lemma StartBallot-HInv3-R:
\[
\begin{aligned}
&\text{StartBallot } s \ s' \ t; \ \text{HInv3-R } s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies \ &\text{HInv3-R } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

lemma StartBallot-HInv3-t:
\[
\begin{aligned}
&\text{StartBallot } s \ s' \ t; \ \text{HInv3-inner } s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies \ &\text{HInv3-inner } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

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

theorem HStartBallot-HInv3:
\[
\begin{aligned}
&\text{HStartBallot } s \ s' \ p; \ \text{HInv3 } s \\
\implies \ &\text{HInv3 } s'
\end{aligned}
\]
⟨proof⟩

theorem HPhase1or2ReadElse-HInv3:
\[
\begin{aligned}
&\text{HPhase1or2ReadElse } s \ s' \ p \ d \ q; \ \text{HInv3 } s \\
\implies \ &\text{HInv3 } s'
\end{aligned}
\]
⟨proof⟩

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

lemma EndPhase1-HInv3-p:
\[
\begin{aligned}
&\text{EndPhase1 } s \ s' \ p; \ \text{HInv3-L } s' \ p \ q \ d \\
\implies \ &\text{HInv3-R } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

lemma EndPhase1-HInv3-q:
\[
\begin{aligned}
&\text{EndPhase1 } s \ s' \ q; \ \text{HInv3-L } s' \ p \ q \ d \\
\implies \ &\text{HInv3-R } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

lemma EndPhase1-HInv3-nL:
\[
\begin{aligned}
&\text{EndPhase1 } s \ s' \ t; \ \neg \text{HInv3-L } s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies \ &\neg \text{HInv3-L } s' \ p \ q \ d
\end{aligned}
\]
⟨proof⟩

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

\textit{lemma} \text{EndPhase1-HInv3-t:}
\[(\text{EndPhase1 } s \ s' \ t; \ HInv3-inner \ s \ p \ q \ d; \ t\neq p; \ t\neq q) \implies HInv3-inner \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase1-HInv3:}
\[
\text{assumes act: EndPhase1 } s \ s' \ t \\
\text{and inv: HInv3-inner } s \ p \ q \ d \\
\text{shows } HInv3-inner \ s' \ p \ q \ d
\]

\textit{theorem} \text{HEndPhase1-HInv3:}
\[(\text{HEndPhase1 } s \ s' \ p; \ HInv3 \ s) \implies HInv3 \ s'\]

\textit{lemma} \text{EndPhase2-HInv3-p:}
\[(\text{EndPhase2 } s \ s' \ p; \ HInv3-L \ s' \ p \ q \ d) \implies HInv3-R \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase2-HInv3-q:}
\[(\text{EndPhase2 } s \ s' \ q; \ HInv3-L \ s' \ p \ q \ d) \implies HInv3-R \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase2-HInv3-nL:}
\[(\text{EndPhase2 } s \ s' \ t; \neg HInv3-L \ s \ p \ q \ d; \ t\neq p; \ t\neq q) \implies \neg HInv3-L \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase2-HInv3-R:}
\[(\text{EndPhase2 } s \ s' \ t; \ HInv3-R \ s \ p \ q \ d; \ t\neq p; \ t\neq q) \implies HInv3-R \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase2-HInv3-t:}
\[(\text{EndPhase2 } s \ s' \ t; \ HInv3-inner \ s \ p \ q \ d; \ t\neq p; \ t\neq q) \implies HInv3-inner \ s' \ p \ q \ d\]

\textit{lemma} \text{EndPhase2-HInv3:}
\[
\text{assumes act: EndPhase2 } s \ s' \ t \\
\text{and inv: HInv3-inner } s \ p \ q \ d \\
\text{shows } HInv3-inner \ s' \ p \ q \ d
\]

\textit{theorem} \text{HEndPhase2-HInv3:}
lemma Fail-HInv3-p:
\[[ \text{Fail } s s' \; \text{p}, \text{HInv3 } s ] \] \implies \text{HInv3 } s'
⟨proof⟩

lemma Fail-HInv3-q:
\[[ \text{Fail } s s' \; \text{q}, \text{HInv3-L } s p q d ] \] \implies \text{HInv3-R } s' p q d
⟨proof⟩

lemma Fail-HInv3-nL:
\[[ \text{Fail } s s' t; \neg \text{HInv3-L } s p q d; t \neq p; t \neq q ] \]

\implies \neg \text{HInv3-L } s' p q d
⟨proof⟩

lemma Fail-HInv3-R:
\[[ \text{Fail } s s' t; \text{HInv3-R } s p q d; t \neq p; t \neq q ] \]

\implies \text{HInv3-R } s' p q d
⟨proof⟩

lemma Fail-HInv3-t:
\[[ \text{Fail } s s' t; \text{HInv3-inner } s p q d; t \neq p; t \neq q ] \]

\implies \text{HInv3-inner } s' p q d
⟨proof⟩

lemma Fail-HInv3:
assumes act: \text{Fail } s s' t

and inv: \text{HInv3-inner } s p q d

shows \text{HInv3-inner } s' p q d
⟨proof⟩

theorem HFail-HInv3:
\[[ \text{HFail } s s' \; \text{p}, \text{HInv3 } s ] \] \implies \text{HInv3 } s'
⟨proof⟩

theorem HPhase0Read-HInv3:
assumes act: \text{HPhase0Read } s s' p d

and inv: \text{HInv3 } s

shows \text{HInv3 } s'
⟨proof⟩

lemma EndPhase0-HInv3-p:
\[[ \text{EndPhase0 } s s' \; \text{p}, \text{HInv3-L } s' p q d ] \]

\implies \text{HInv3-R } s' p q d
⟨proof⟩

lemma EndPhase0-HInv3-q:
\[[ \text{EndPhase0 } s s' \; \text{q}, \text{HInv3-L } s' p q d ] \]
\[ \Rightarrow H\text{Inv}3-R \ s' \ p \ q \ d \]

\textbf{proof}

\textbf{lemma} \ EndPhase0-HInv3-nL:
\[ \big[ \text{EndPhase0} \ s \ s' \ t; \neg H\text{Inv}3-L \ s \ p \ q \ d; \ t \neq p; \ t \neq q \big] \]
\[ \Rightarrow \neg H\text{Inv}3-L \ s' \ p \ q \ d \]

\textbf{proof}

\textbf{lemma} \ EndPhase0-HInv3-R:
\[ \big[ \text{EndPhase0} \ s \ s' \ t; H\text{Inv}3-R \ s \ p \ q \ d; \ t \neq p; \ t \neq q \big] \]
\[ \Rightarrow H\text{Inv}3-R \ s' \ p \ q \ d \]

\textbf{proof}

\textbf{lemma} \ EndPhase0-HInv3-t:
\[ \big[ \text{EndPhase0} \ s \ s' \ t; H\text{Inv}3-inner \ s \ p \ q \ d; \ t \neq p; \ t \neq q \big] \]
\[ \Rightarrow H\text{Inv}3-inner \ s' \ p \ q \ d \]

\textbf{proof}

\textbf{lemma} \ EndPhase0-HInv3:
\begin{align*}
\text{assumes} & \quad \text{act: EndPhase0} \ s \ s' \ t \\
\text{and} & \quad \text{inv: HInv3-inner} \ s \ p \ q \ d \\
\text{shows} & \quad H\text{Inv3-inner} \ s' \ p \ q \ d
\end{align*}

\textbf{proof}

\textbf{theorem} \ HEndPhase0-HInv3:
\[ \big[ \text{HEndPhase0} \ s \ s' \ p; \text{HInv3} \ s \big] \Rightarrow \text{HInv3} \ s' \]

\textbf{proof}

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

\textbf{lemma} \ I2c:
\begin{align*}
\text{assumes} & \quad \text{nxt: HNext} \ s \ s' \\
\text{and} & \quad \text{inv: HInv1} \ s \land HInv2 \ s \land HInv3 \ s \\
\text{shows} & \quad HInv3 \ s' \ (\text{proof})
\end{align*}

end

\textit{theory} \ DiskPaxos-Inv4 \ \textbf{imports} \ DiskPaxos-Inv2 \ \textbf{begin}

\textbf{C.4 Invariant 4}

This invariant expresses relations among \textit{mbal} and \textit{bal} values of a processor and of its disk blocks. \textit{HInv4a} asserts that, when \textit{p} is not recovering from a failure, its \textit{mbal} value is at least as large as the \textit{bal} field of any of its blocks, and at least as large as the \textit{mbal} field of its block on some disk in any majority set. \textit{HInv4b} conjunct asserts that, in phase 1, its \textit{mbal} value is actually greater than the \textit{bal} field of any of its blocks. \textit{HInv4c} asserts that, in phase 2, its \textit{bal} value is the \textit{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** MajoritySet :: Disk set set  
where MajoritySet = \{ D. IsMajority(D) \}

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

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

**definition** HInv4a :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool  
where \( HInv4a s p = (\forall bk \in \text{blocksOf } s \ p. \ bal bk = mbal (dblock s p)) \)

**definition** HInv4b :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool  
where \( HInv4b s p = (\forall bk \in \text{blocksOf } s \ p. \ bal bk < mbal (dblock s p)) \)

**definition** HInv4c :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool  
where \( HInv4c s p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal (disk s d p) = bal (dblock s p))) \)

**definition** HInv4d :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool  
where \( HInv4d s p = (\forall bk \in \text{blocksOf } s \ p. \ bal bk \leq mbal (disk s d p)) \)

**definition** HInv4 :: state \( \Rightarrow \) bool  
where \( HInv4 s = (\forall p. \ HInv4a s p \land HInv4b s p \land HInv4c s p \land HInv4d s p) \)

The initial state implies Invariant 4.

**theorem** 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-H\( HInv4x-p \) proves the case of \( p = q \), while lemma action-H\( HInv4x-q \) proves the other case.

**C.4.1 Proofs of Invariant 4a**

**lemma** HStartBallot-HInv4a1:  
assumes act: HStartBallot s s' p  
and inv: HInv4a1 s p  
and inv2a: Inv2a-inner s' p

36
lemma $\text{HStartBallot-HInv4a2}$:
assumes act: $\text{HStartBallot} s s'$
and inv: $\text{HInv4a2} s p$
shows $\text{HInv4a2} s' p$
(proof)

lemma $\text{HStartBallot-HInv4a-p}$:
assumes act: $\text{HStartBallot} s s'$
and inv: $\text{HInv4a} s p$
and inv2a: $\text{Inv2a-inner} s' p$
shows $\text{HInv4a} s' p$
(proof)

lemma $\text{HStartBallot-HInv4a-q}$:
assumes act: $\text{HStartBallot} s s'$
and inv: $\text{HInv4a} s q$
and inv2a: $\text{Inv2a} s'$
shows $\text{HInv4a} s' q$
(proof)

theorem $\text{HStartBallot-HInv4a}$:
assumes act: $\text{HStartBallot} s s'$
and inv: $\text{HInv4a} s q$
and inv2a: $\text{Inv2a} s'$
shows $\text{HInv4a} s' q$
(proof)

lemma $\text{Phase1or2Write-HInv4a1}$:
$[ \text{Phase1or2Write} s s' p d; \text{HInv4a1} s q ] \implies \text{HInv4a1} s' q$
(proof)

lemma $\text{Phase1or2Write-HInv4a2}$:
$[ \text{Phase1or2Write} s s' p d; \text{HInv4a2} s q ] \implies \text{HInv4a2} s' q$
(proof)

theorem $\text{HPhase1or2Write-HInv4a}$:
assumes act: $\text{HPhase1or2Write} s s' p d$
and inv: $\text{HInv4a} s q$
shows $\text{HInv4a} s' q$
(proof)

lemma $\text{HPhase1or2ReadThen-HInv4a1-p}$:
assumes act: $\text{HPhase1or2ReadThen} s s' p d q$
and inv: $\text{HInv4a1} s p$
shows $\text{HInv4a1} s' p$
(proof)
lemma HPhase1or2ReadThen-HInv4a2:
[ HPhase1or2ReadThen s s' p d r; HInv4a2 s q ] \implies HInv4a2 s' q
(proof)

lemma HPhase1or2ReadThen-HInv4a-p:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv4a s p
and inv2b: Inv2b s
shows HInv4a s' p
(proof)

lemma HPhase1or2ReadThen-HInv4a-q:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv4a s q
and p\neq q
shows HInv4a s' q
(proof)

theorem HPhase1or2ReadThen-HInv4a:
[ HPhase1or2ReadThen s s' p d r; HInv4a s q; Inv2b s ] \implies HInv4a s' q
(proof)

theorem HPhase1or2ReadElse-HInv4a:
assumes act: HPhase1or2ReadElse s s' p d r
and inv: HInv4a s q and inv2a: Inv2a s'
shows HInv4a s' q
(proof)

lemma HEndPhase1-HInv4a1:
assumes act: HEndPhase1 s s' p
and inv: HInv4a1 s p
shows HInv4a1 s' p
(proof)

lemma HEndPhase1-HInv4a2:
assumes act: HEndPhase1 s s' p
and inv: HInv4a2 s p
and inv2a: Inv2a s
shows HInv4a2 s' p
(proof)

lemma HEndPhase1-HInv4a-p:
assumes act: HEndPhase1 s s' p
and inv: HInv4a s p
and inv2a: Inv2a s
shows HInv4a s' p
(proof)
lemma \(HEndPhase1-HInv4a\): 
assumes act: \(HEndPhase1\ s s' p\) 
and inv: \(HInv4a\ s q\) 
and \(pq: p \neq q\) 
shows \(HInv4a\ s' q\) 
(proof)

theorem \(HEndPhase1-HInv4a\): 
\[
[\ \begin{array}{l}
HEndPhase1 \ s s' p; 
HInv4a \ s q; 
Inv2a \ s 
\end{array}\] \implies \(HInv4a\ s' q\) 
(proof)

theorem \(HFail-HInv4a\): 
\[
[\ \begin{array}{l}
HFail \ s s' p; 
HInv4a \ s q 
\end{array}\] \implies \(HInv4a\ s' q\) 
(proof)

theorem \(HPhase0Read-HInv4a\): 
\[
[\ \begin{array}{l}
HPhase0Read \ s s' p d; 
HInv4a \ s q 
\end{array}\] \implies \(HInv4a\ s' q\) 
(proof)

theorem \(HEndPhase2-HInv4a\): 
\[
[\ \begin{array}{l}
HEndPhase2 \ s s' p; 
HInv4a \ s q 
\end{array}\] \implies \(HInv4a\ s' q\) 
(proof)

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

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

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

lemma \(hasRead-allBlks\): 
assumes inv2c: \(Inv2c-inner\ s p\) 
and phase: \(phase\ s p = 0\) 
shows \(\forall \ d \in \{d. hasRead\ s p d p\}. disk\ s d p \in allBlocksRead\ s p\) 
(proof)
lemma \textit{HEndPhase0-41}:
\begin{itemize}
  \item \textbf{assumes act:} \textit{HEndPhase0} \textit{s s' p}
  \item \textbf{and inv1:} \textit{Inv1} \textit{s}
  \item \textbf{and inv2c:} \textit{Inv2c-inner} \textit{s p}
\end{itemize}
\textbf{shows} \exists D \in \textit{MajoritySet}. \forall d \in D. \textit{mbal(disk s d p)} \leq \textit{mbal(dblock s' p)}\wedge \textit{bal(disk s d p)} \leq \textit{bal(dblock s' p)}

(\textit{proof})

lemma \textit{Majority-exQ}:
\begin{itemize}
  \item \textbf{assumes asm1:} \exists D \in \textit{MajoritySet}. \forall d \in D. \textit{P d}
\end{itemize}
\textbf{shows} \forall D \in \textit{MajoritySet}. \exists d \in D. \textit{P d}

(\textit{proof})

lemma \textit{HEndPhase0-HInv4a2-p}:
\begin{itemize}
  \item \textbf{assumes act:} \textit{HEndPhase0} \textit{s s' p}
  \item \textbf{and inv1:} \textit{Inv1} \textit{s}
  \item \textbf{and inv2c:} \textit{Inv2c-inner} \textit{s p}
\end{itemize}
\textbf{shows} \textit{HInv4a2 s' p}

(\textit{proof})

lemma \textit{HEndPhase0-HInv4a-p}:
\begin{itemize}
  \item \textbf{assumes act:} \textit{HEndPhase0} \textit{s s' p}
  \item \textbf{and inv2a:} \textit{Inv2a} \textit{s}
  \item \textbf{and inv2:} \textit{Inv2c} \textit{s}
  \item \textbf{and inv4d:} \textit{HInv4d} \textit{s p}
  \item \textbf{and inv1:} \textit{Inv1} \textit{s}
  \item \textbf{and inv:} \textit{HInv4a} \textit{s p}
\end{itemize}
\textbf{shows} \textit{HInv4a s' p}

(\textit{proof})

lemma \textit{HEndPhase0-HInv4a-q}:
\begin{itemize}
  \item \textbf{assumes act:} \textit{HEndPhase0} \textit{s s' p}
  \item \textbf{and inv:} \textit{HInv4a} \textit{s q}
  \item \textbf{and pnq:} p \neq q
\end{itemize}
\textbf{shows} \textit{HInv4a s' q}

(\textit{proof})

theorem \textit{HEndPhase0-HInv4a}:
\begin{itemize}
  \item \textit{HEndPhase0 s s' p; HInv4a s q; HInv4d s p;}
  \item \textit{Inv2a s; Inv1 s; Inv2a s; Inv2c s}
\end{itemize}
\implies \textit{HInv4a s' q}

(\textit{proof})

C.4.2 Proofs of Invariant 4b

lemma \textit{blocksRead-allBlocksRead}:
\textit{rb} \in \textit{blocksRead s p d} \implies \textit{block rb} \in \textit{allBlocksRead s p}

40
lemma HEndPhase0-dblock-mbal:

\[ \forall \text{br} \in \text{allBlocksRead } s p. \text{mbal } \text{br} < \text{mbal}(\text{dblock } s'p) \]

lemma HEndPhase0-HInv4b-p-dblock:

assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows bal(dblock s' p) < mbal(dblock s' p)

lemma HEndPhase0-HInv4b-p-blocksOf:

assumes act: HEndPhase0 s s' p
and inv4d: HInv4d s p
and inv2c: Inv2c-inner s p
and bk: bk \in \text{blocksOf } s p
shows bal bk < mbal(dblock s' p)

lemma HEndPhase0-HInv4b-p:

assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows HInv4b s p

lemma HEndPhase0-HInv4b-q:

assumes act: HEndPhase0 s s' p
and pnq: p \neq q
and inv: HInv4b s q
shows HInv4b s' q

theorem HEndPhase0-HInv4b:

assumes act: HEndPhase0 s s' p
and inv: HInv4b s q
and inv4d: HInv4d s p
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows HInv4b s' q
lemma HStartBallot-HInv4b-p:
  assumes act: HStartBallot s s' p
  and inv2a: Inv2a-innermost s p (dblock s p)
  and inv4b: HInv4b s p
  and inv4a: HInv4a 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 r
  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 \Rightarrow HInv4b s' p
  \langle proof \rangle

lemma HEndPhase1-HInv4b-q:
  assumes act: HEndPhase1 s s' p
  and p\neq q
  and inv: HInv4b s q
  shows HInv4b s' q
  \langle proof \rangle

theorem HEndPhase1-HInv4b:
  assumes act: HEndPhase1 s s' p
  and inv: HInv4b s q
  shows HInv4b s' q
  \langle proof \rangle

lemma HEndPhase2-HInv4b-p:
  HEndPhase2 s s' p \Rightarrow HInv4b s' p
  \langle proof \rangle

lemma HEndPhase2-HInv4b-q:
  assumes act: HEndPhase2 s s' p
  and p\neq q
  and inv: HInv4b s q
  shows HInv4b s' q
  \langle proof \rangle

theorem HEndPhase2-HInv4b:
  assumes act: HEndPhase2 s s' p
  and inv: HInv4b s q
  shows HInv4b s' q
  \langle proof \rangle

lemma HFail-HInv4b-p:
  HFail s s' p \Rightarrow HInv4b s' p
  \langle proof \rangle

lemma HFail-HInv4b-q:
  assumes act: HFail s s' p
  and p\neq q
  and inv: HInv4b s q
  shows HInv4b s' q
  \langle proof \rangle

theorem HFail-HInv4b:
  assumes act: HFail s s' p
and \( \text{inv: HInv4b s q} \)
shows \( \text{HInv4b s' q} \)

(\(\text{proof}\))

**Lemma HPhase0Read-HInv4b-p:**
\[ HPhase0Read s s' p d \implies HInv4b s' p \]
(\(\text{proof}\))

**Lemma HPhase0Read-HInv4b-q:**
\begin{align*}
& \text{assumes act: } HPhase0Read s s' p d \\
& \text{and } p \neq q \\
& \text{and } \text{inv: HInv4b s q} \\
& \text{shows } HInv4b s' q \end{align*}
(\(\text{proof}\))

**Theorem HPhase0Read-HInv4b:**
\[ HPhase0Read s s' p d \implies HInv4b s' q \]
(\(\text{proof}\))

**C.4.3 Proofs of Invariant 4c**

**Lemma HStartBallot-HInv4c-p:**
\[ [ HStartBallot s s' p; HInv4c s p ] \implies HInv4c s' p \]
(\(\text{proof}\))

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

**Theorem HStartBallot-HInv4c:**
\[ [ HStartBallot s s' p; HInv4c s q ] \implies HInv4c s' q \]
(\(\text{proof}\))

**Lemma HPhase1or2Write-HInv4c-p:**
\begin{align*}
& \text{assumes act: } HPhase1or2Write s s' p d \\
& \text{and } \text{inv: HInv4c s p} \\
& \text{and } \text{inv2c: Inv2c s} \\
& \text{shows } HInv4c s' p \end{align*}
(\(\text{proof}\))

**Lemma HPhase1or2Write-HInv4c-q:**
\begin{align*}
& \text{assumes act: } HPhase1or2Write s s' p d \\
& \text{and } \text{inv: HInv4c s q} \\
& \text{and } p \neq q \end{align*}
shows $\text{HInv4c } s' \quad q$

(\text{proof})

\textbf{theorem} \text{HPhase1or2Write-HInv4c}: \[
[ \text{HPhase1or2Write } s \quad s' \quad p \quad d; \text{HInv4c } s \quad q; \text{Inv2c } s ] \Rightarrow \text{HInv4c } s' \quad q
\]

(\text{proof})

\textbf{lemma} \text{HPhase1or2ReadThen-HInv4c-p}: \[
[ \text{HPhase1or2ReadThen } s \quad s' \quad p \quad d \quad q; \text{HInv4c } s \quad p ] \Rightarrow \text{HInv4c } s' \quad p
\]

(\text{proof})

\textbf{lemma} \text{HPhase1or2ReadThen-HInv4c-q}: \[
\text{assumes act: HPhase1or2ReadThen } s \quad s' \quad p \quad d \quad r \\
\text{and inv: HInv4c } s \quad q \\
\text{and pnq: } p \neq q \\
\text{shows HInv4c } s' \quad q
\]

(\text{proof})

\textbf{theorem} \text{HPhase1or2ReadThen-HInv4c}: \[
[ \text{HPhase1or2ReadThen } s \quad s' \quad p \quad d \quad r; \text{HInv4c } s \quad q] \Rightarrow \text{HInv4c } s' \quad q
\]

(\text{proof})

\textbf{theorem} \text{HPhase1or2ReadElse-HInv4c}: \[
[ \text{HPhase1or2ReadElse } s \quad s' \quad p \quad d \quad r; \text{HInv4c } s \quad q] \Rightarrow \text{HInv4c } s' \quad q
\]

(\text{proof})

\textbf{lemma} \text{HEndPhase1-HInv4c-p}: \[
\text{assumes act: HEndPhase1 } s \quad s' \quad p \\
\text{and inv2b: Inv2b } s \\
\text{shows HInv4c } s' \quad p
\]

(\text{proof})

\textbf{lemma} \text{HEndPhase1-HInv4c-q}: \[
\text{assumes act: HEndPhase1 } s \quad s' \quad p \\
\text{and inv: HInv4c } s \quad q \\
\text{and pnq: } p \neq q \\
\text{shows HInv4c } s' \quad q
\]

(\text{proof})

\textbf{theorem} \text{HEndPhase1-HInv4c}: \[
[ \text{HEndPhase1 } s \quad s' \quad p; \text{HInv4c } s \quad q; \text{Inv2b } s ] \Rightarrow \text{HInv4c } s' \quad q
\]

(\text{proof})

\textbf{lemma} \text{HEndPhase2-HInv4c-p}: \[
[ \text{HEndPhase2 } s \quad s' \quad p; \text{HInv4c } s \quad p ] \Rightarrow \text{HInv4c } s' \quad p
\]

(\text{proof})
lemma HEndPhase2-HInv4c-q:
assumes act: HEndPhase2 s s' p
and inv: HInv4c s q
and pq: p\neq q
shows HInv4c s' q
⟨proof⟩

theorem HEndPhase2-HInv4c:
[ HEndPhase2 s s' p; HInv4c s q ] \Rightarrow HInv4c s' q
⟨proof⟩

lemma HFail-HInv4c-p:
[ HFail s s' p; HInv4c s p ] \Rightarrow HInv4c s' p
⟨proof⟩

lemma HFail-HInv4c-q:
assumes act: HFail s s' p
and inv: HInv4c s q
and pq: p\neq q
shows HInv4c s' q
⟨proof⟩

theorem HFail-HInv4c:
[ HFail s s' p; HInv4c s q ] \Rightarrow HInv4c s' q
⟨proof⟩

lemma HPhase0Read-HInv4c-p:
[ HPhase0Read s s' p d; HInv4c s p ] \Rightarrow HInv4c s' p
⟨proof⟩

lemma HPhase0Read-HInv4c-q:
assumes act: HPhase0Read s s' p d
and inv: HInv4c s q
and pq: p\neq q
shows HInv4c s' q
⟨proof⟩

theorem HPhase0Read-HInv4c:
[ HPhase0Read s s' p d; HInv4c s q ] \Rightarrow HInv4c s' q
⟨proof⟩

lemma HEndPhase0-HInv4c-p:
[ HEndPhase0 s s' p; HInv4c s p ] \Rightarrow HInv4c s' p
⟨proof⟩

lemma HEndPhase0-HInv4c-q:
assumes act: HEndPhase0 s s' p
and inv: HInv4c s q
and pq: p\neq q

46
shows $HInv4c\ s\ s'\ q$
(proof)

theorem $HEndPhase0-HInv4c$:
\[
[\ HEndPhase0\ s\ s'\ p;\ HInv4c\ s\ q]\ \Rightarrow\ HInv4c\ s'\ q
\]
(proof)

C.4.4 Proofs of Invariant 4d

lemma $HStartBallot-HInv4d-p$:
  assumes act: $HStartBallot\ s\ s'\ p$
  and inv: $HInv4d\ s\ p$
  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 $HStartBallot-HInv4d$:
\[
[\ HStartBallot\ s\ s'\ p;\ HInv4d\ s\ q]\ \Rightarrow\ HInv4d\ s'\ q
\]
(proof)

lemma $HPhase1or2Write-HInv4d-p$:
  assumes act: $HPhase1or2Write\ s\ s'\ p\ d$
  and inv: $HInv4d\ s\ p$
  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 $HPhase1or2Write-HInv4d$:
\[
[\ HPhase1or2Write\ s\ s'\ p\ d;\ HInv4d\ s\ q;\ HInv4a\ s\ p]\ \Rightarrow\ HInv4d\ s'\ q
\]
(proof)

lemma $HPhase1or2ReadThen-HInv4d-p$:
  assumes act: $HPhase1or2ReadThen\ s\ s'\ p\ d\ q$
  and inv: $HInv4d\ s\ p$
  shows $HInv4d\ s'\ p$
(proof)
lemma \text{HPhase1or2ReadThen-HInv4d-q}:
assumes act: \text{HPhase1or2ReadThen} s \ s' p d r
and inv: \text{HInv4d} s q
and pq: p\neq q
shows \text{HInv4d} s' q
⟨proof⟩

theorem \text{HPhase1or2ReadThen-HInv4d}:
[\text{HPhase1or2ReadThen} s \ s' p d r; \text{HInv4d} s q] \implies \text{HInv4d} s' q
⟨proof⟩

theorem \text{HPhase1or2ReadElse-HInv4d}:
[\text{HPhase1or2ReadElse} s \ s' p d r; \text{HInv4d} s q] \implies \text{HInv4d} s' q
⟨proof⟩

lemma \text{HEndPhase1-HInv4d-p}:
assumes act: \text{HEndPhase1} s \ s' p
and inv: \text{HInv4d} s p
and inv2b: \text{Inv2b} s
and inv4c: \text{HInv4c} s p
shows \text{HInv4d} s' p
⟨proof⟩

lemma \text{HEndPhase1-HInv4d-q}:
assumes act: \text{HEndPhase1} s \ s' p
and inv: \text{HInv4d} s q
and pq: p\neq q
shows \text{HInv4d} s' q
⟨proof⟩

theorem \text{HEndPhase1-HInv4d}:
[\text{HEndPhase1} s \ s' p; \text{HInv4d} s q; \text{Inv2b} s; \text{HInv4c} s p] \implies \text{HInv4d} s' q
⟨proof⟩

lemma \text{HEndPhase2-HInv4d-p}:
assumes act: \text{HEndPhase2} s \ s' p
and inv: \text{HInv4d} s p
shows \text{HInv4d} s' p
⟨proof⟩

lemma \text{HEndPhase2-HInv4d-q}:
assumes act: \text{HEndPhase2} s \ s' p
and inv: \text{HInv4d} s q
and pq: p\neq q
shows \text{HInv4d} s' q
⟨proof⟩
theorem HEndPhase2-HInv4d:
[ HEndPhase2 s s' p; HInv4d s q ] ⇒ HInv4d s' q
⟨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: p̸=q
shows HInv4d s' q
⟨proof⟩

theorem HFail-HInv4d:
[ HFail s s' p; HInv4d s q ] ⇒ HInv4d s' q
⟨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: p̸=q
shows HInv4d s' q
⟨proof⟩

theorem HPhase0Read-HInv4d:
[ HPhase0Read s s' p d; HInv4d s q ] ⇒ HInv4d s' q
⟨proof⟩

lemma HEndPhase0-blocksOf2:
assumes act: HEndPhase0 s s' p
and inv2c: Inv2c-inner s p
shows allBlocksRead s p ⊆ 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 $HInv4d\ s\ s'\ p$
(proof)

lemma $HEndPhase0-HInv4d-q$:
  assumes $act: HEndPhase0\ s\ s'\ p$
  and $inv: HInv4d\ s\ s'\ q$
  and $pq: p \neq q$
  shows $HInv4d\ s'\ q$
(proof)

theorem $HEndPhase0-HInv4d$:
\[ [\ HEndPhase0\ s\ s'\ p; \ HInv4d\ s\ s'\ q; \ Inv2c\ s; \ Inv1\ s ] \implies HInv4d\ s'\ q \]
(proof)

Since we have already proved $HInv2$ is an invariant of $HNext$, $HInv1 \land HInv2 \land HInv4$ is also an invariant of $HNext$.

lemma $I2d$:
  assumes $next: HNext\ s\ s'$
  and $inv: HInv1\ s \land HInv2\ s \land HInv2\ s' \land HInv4\ s$
  shows $HInv4\ s'$
(proof)

end

theory DiskPaxos-Inv5 imports DiskPaxos-Inv3 DiskPaxos-Inv4 begin

C.5 Invariant 5

This invariant asserts that, if a processor $p$ is in phase 2, then either its $bal$ and $inp$ values satisfy $maxBalInp$, 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 $bal(dblocksp)$.

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

definition $HInv5-inner-R$ :: $state \Rightarrow Proc \Rightarrow bool$
  where $HInv5-inner-R\ s\ p =$
    \[ (maxBalInp\ s\ (bal(dblock\ s\ p)))\ (inp(dblock\ s\ p)) \]
    \lor (\exists D\in MajoritySet.\ \exists q.\ (\forall d\in D.\ bal(dblock\ s\ p) < mbal(disk\ s\ d\ q) \land \neg hasRead\ s\ p\ d\ q)))

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

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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

theorem \( HInit\text{-}HInv5: HInit \ s \implies HInv5 \ s \)  
〈proof〉

We will use the notation used in the proofs of invariant 4, and prove the lemma \( \text{action-HInv5}\text{-}p \) and \( \text{action-HInv5}\text{-}q \) 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 \( \text{allBlocks} \) are included in the old \( \text{allBlocks} \) or, in some cases, included in the old \( \text{allBlocks} \) union the new \( \text{dblock} \).

lemma \( \text{HStartBallot-HInv5}\text{-p}: \)
  assumes \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)  
  and \( \text{inv}: \text{HInv5}\text{-inner} \ s \ p \)  
  shows \( \text{HInv5}\text{-inner} \ s' \ p \)  
〈proof〉

lemma \( \text{HStartBallot-blocksOf-q}: \)
  assumes \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)  
  and \( \text{pq}: p \neq q \)  
  shows \( \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \)  
〈proof〉

lemma \( \text{HStartBallot-allBlocks}: \)
  assumes \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)  
  shows \( \text{allBlocks} \ s' \subseteq \text{allBlocks} \ s \cup \{ \text{dblock} \ s' \ p \} \)  
〈proof〉

lemma \( \text{HStartBallot-HInv5}\text{-q1}: \)
  assumes \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)  
  and \( \text{pq}: p \neq q \)  
  and \( \text{inv5-1}: \text{maxBalImp} \ s \ (\text{bal}(\text{dblock} \ s \ q)) \ (\text{inp}(\text{dblock} \ s \ q)) \)  
  shows \( \text{maxBalImp} \ s' \ (\text{bal}(\text{dblock} \ s' \ q)) \ (\text{inp}(\text{dblock} \ s' \ q)) \)  
〈proof〉

lemma \( \text{HStartBallot-HInv5}\text{-q2}: \)
  assumes \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)  
  and \( \text{pq}: p \neq q \)  
  and \( \text{inv5-2}: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(\text{dblock} \ s \ q) < \text{mbal}(\text{disk} \ s \ d \ qq) \)  
  and \( \neg \text{hasRead} \ s \ q \ d \ qq \)  
  shows \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(\text{dblock} \ s' \ q) < \text{mbal}(\text{disk} \ s' \ d \ qq) \)  
  and \( \neg \text{hasRead} \ s' \ q \ d \ qq \)  
〈proof〉
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:
  ⟨ HStartBallot s s' p; HInv5-inner s q ⟩ ⟹ 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: ∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal(dblock s p) < mbal(disk s d q)
      ∧ ¬hasRead s p d q)
  shows ∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal(dblock s' p) < mbal(disk s' d q)
      ∧ ¬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: ∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
      ∧ ¬hasRead s q d qq)
  shows HInv5-inner s' q
⟨proof⟩
shows \( \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \quad \text{bal}(dblock s' q) < \text{mbal}(disk s' d qq) \wedge \neg \text{hasRead s' q d qq}) \)

(proof)

lemma \( H\text{Phase1or2Write-HInv5-q} \):
  assumes act: \( H\text{Phase1or2Write s s' p d} \)
  and inv: \( H\text{Inv5-inner s q} \)
  and inv4a: \( H\text{inv4a s p} \)
  and \( p \neq q \)
  shows \( H\text{Inv5-inner s' q} \)

(proof)

theorem \( H\text{Phase1or2Write-HInv5} \):
  \[ H\text{Phase1or2Write s s' p d}; H\text{Inv5-inner s q}; H\text{inv4c s p}; H\text{inv4a s p} \] \implies \( H\text{Inv5-inner s' q} \)

(proof)

lemma \( H\text{Phase1or2ReadThen-HInv5-1} \):
  assumes act: \( H\text{Phase1or2ReadThen s s' p d r} \)
  and inv5-1: \( \text{maxBalInp s} \left( \text{bal}(dblock s q) \right) \left( \text{inp}(dblock s q) \right) \)
  shows \( \text{maxBalInp s'} \left( \text{bal}(dblock s' q) \right) \left( \text{inp}(dblock s' q) \right) \)

(proof)

lemma \( H\text{Phase1or2ReadThen-HInv5-p2} \):
  assumes act: \( H\text{Phase1or2ReadThen s s' p d r} \)
  and inv4c: \( H\text{Inv4c s p} \)
  and inv2c: \( H\text{Inv2c s} \)
  and phase: \( \text{phase s p} = 2 \)
  and inv5-2: \( \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \quad \text{bal}(dblock s' p) < \text{mbal}(disk s' d q) \wedge \neg \text{hasRead s' p d q}) \)
  shows \( \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \quad \text{bal}(dblock s' p) < \text{mbal}(disk s' d q) \wedge \neg \text{hasRead s' p d q}) \)

(proof)

lemma \( H\text{Phase1or2ReadThen-HInv5-p} \):
  assumes act: \( H\text{Phase1or2ReadThen s s' p d r} \)
  and inv: \( H\text{Inv5-inner s p} \)
  and inv4: \( H\text{Inv4c s p} \)
  and inv2c: \( H\text{Inv2c s} \)
  shows \( H\text{Inv5-inner s' p} \)

(proof)

lemma \( H\text{Phase1or2ReadThen-allBlocks} \):
  assumes act: \( H\text{Phase1or2ReadThen s s' p d r} \)
  shows \( \text{allBlocks s'} \subseteq \text{allBlocks s} \)

(proof)

lemma \( H\text{Phase1or2ReadThen-HInv5-q2} \):
  assumes act: \( H\text{Phase1or2ReadThen s s' p d r} \)
and \( p\neq q \)
and \( inv^4a: HInv^4a s p \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock } s q) < m\text{bal}(\text{disk } d qq)) \)

\[\wedge \neg \text{hasRead } s q d qq\]

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

\[\wedge \neg \text{hasRead } s' q d qq\]

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

\textbf{lemma} \( H\text{Phase1or2ReadThen}-Hinv5-q\):
\textbf{assumes} act: \( H\text{Phase1or2ReadThen } s s' p d r \)
and inv: \( H\text{Inv5-inner } s q \)
and \( inv^4a: H\text{Inv4a } s p \)
and \( pnq: p \neq q \)
\textbf{shows} \( H\text{Inv5-inner } s' q \)

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

\textbf{theorem} \( H\text{Phase1or2ReadThen}-Hinv5:\)
\[ \begin{array}{l}
[ H\text{Phase1or2ReadThen } s s' p d r; H\text{Inv5-inner } s q; \\
\text{Inv2c } s; H\text{Inv4c } s p; H\text{Inv4a } s p ] \implies H\text{Inv5-inner } s' q
\end{array} \]

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

\textbf{theorem} \( H\text{Phase1or2ReadElse}-Hinv5:\)
\[ \begin{array}{l}
[ H\text{Phase1or2ReadElse } s s' p d r; H\text{Inv5-inner } s q ] \implies H\text{Inv5-inner } s' q
\end{array} \]

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

\textbf{lemma} \( H\text{EndPhase2}-Hinv5-p\):
\( H\text{EndPhase2 } s s' p \implies H\text{Inv5-inner } s' p \)

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

\textbf{lemma} \( H\text{EndPhase2-allBlocks}:\)
\textbf{assumes} act: \( H\text{EndPhase2 } s s' p \)
\textbf{shows} all\text{Blocks } s' \subseteq \text{all\text{Blocks } s} 

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

\textbf{lemma} \( H\text{EndPhase2-Hinv5-q1}:\)
\textbf{assumes} act: \( H\text{EndPhase2 } s s' p \)
and \( pnq: p \neq q \)
and \( inv5-1: \text{max\text{Bal\text{Inp}} } s (\text{bal}(\text{dblock } s q)) (\text{inp}(\text{dblock } s q)) \)
\textbf{shows} \( \text{max\text{Bal\text{Inp}} } s' (\text{bal}(\text{dblock } s' q)) (\text{inp}(\text{dblock } s' q)) \)

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

\textbf{lemma} \( H\text{EndPhase2-Hinv5-q2}:\)
\textbf{assumes} act: \( H\text{EndPhase2 } s s' p \)
and \( pnq: p \neq q \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock } s q) < m\text{bal}(\text{disk } d qq)) \)

\[\wedge \neg \text{hasRead } s q d qq\]

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

54
\[
\exists D \in \text{MajoritySet}. \exists q. \left( \forall d \in D. \quad \text{bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d q) \right) \\
\land \neg \text{hasRead } s' q d q)
\]

(proof)

lemma \text{HEndPhase2-HInv5-q}:
\begin{align*}
\text{assumes} & \quad \text{act}: \text{HEndPhase2 } s s' p \\
\text{and} & \quad \text{inv}: \text{HInv5-inner } s q \\
\text{and} & \quad \text{pnq}: p \neq q \\
\text{shows} & \quad \text{HInv5-inner } s' q
\end{align*}

(proof)

theorem \text{HEndPhase2-HInv5}:
\[
\left[ \text{HEndPhase2 } s s' p; \text{HInv5-inner } s q \right] \implies \text{HInv5-inner } s' q
\]

(proof)

lemma \text{HEndPhase1-HInv5-p}:
\begin{align*}
\text{assumes} & \quad \text{act}: \text{HEndPhase1 } s s' p \\
\text{and} & \quad \text{inv4}: \text{HInv4 } s \\
\text{and} & \quad \text{inv2a}: \text{Inv2a } s \\
\text{and} & \quad \text{inv2a'}: \text{Inv2a } s' \\
\text{and} & \quad \text{inv2c}: \text{Inv2c } s \\
\text{and} & \quad \text{asm4}': \neg \text{maxBalInp } s' (\text{bal}(\text{dblock } s' p)) (\text{inp}(\text{dblock } s' p)) \\
\text{shows} & \quad (\exists D \in \text{MajoritySet}. \exists q. \left( \forall d \in D. \quad \text{bal}(\text{dblock } s' p) < \text{mbal}(\text{disk } s' d q) \\
\land \neg \text{hasRead } s' p d q)\right)
\end{align*}

(proof)

lemma union-inclusion:
\[
\left[ A \subseteq A'; B \subseteq B' \right] \implies A \cup B \subseteq A' \cup B'
\]

(proof)

lemma \text{HEndPhase1-blocksOf-q}:
\begin{align*}
\text{assumes} & \quad \text{act}: \text{HEndPhase1 } s s' p \\
\text{and} & \quad \text{pnq}: p \neq q \\
\text{shows} & \quad \text{blocksOf } s' q \subseteq \text{blocksOf } s q
\end{align*}

(proof)

lemma \text{HEndPhase1-allBlocks}:
\begin{align*}
\text{assumes} & \quad \text{act}: \text{HEndPhase1 } s s' p \\
\text{shows} & \quad \text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ \text{dblock } s' p \}
\end{align*}

(proof)

lemma \text{HEndPhase1-HInv5-q}:
\begin{align*}
\text{assumes} & \quad \text{act}: \text{HEndPhase1 } s s' p \\
\text{and} & \quad \text{inv}: \text{HInv5 } s \\
\text{and} & \quad \text{inv1}: \text{Inv1 } s \\
\text{and} & \quad \text{inv2a}: \text{Inv2a } s' \\
\text{and} & \quad \text{inv2a-q}: \text{Inv2a } s \\
\text{and} & \quad \text{inv2b}: \text{Inv2b } s \\
\text{and} & \quad \text{inv2c}: \text{Inv2c } s
\end{align*}
and $inv3$: $HInv3\ s$
and $phase'$: $phase\ s'\ q = 2$
and $pnq$: $p \neq q$
and $asm4$: $\neg \text{maxBalInp}\ s' (\text{bal}(\text{dblock}\ s'\ q)) (\text{inp}(\text{dblock}\ s'\ q))$
shows $(\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock}\ s'\ q) < \text{mbal} (\text{disk}\ d\ qq))$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma $HFail-HInv5-p$:
$HFail\ s\ s'\ p \implies HInv5-\text{inner}\ s'\ p$

$\langle \text{proof} \rangle$

lemma $HFail-\text{blocksOf-q}$:
assumes act: $HFail\ s\ s'\ p$
and $pnq$: $p \neq q$
shows $\text{blocksOf}\ s'\ q \subseteq \text{blocksOf}\ s\ q$

$\langle \text{proof} \rangle$

lemma $HFail-\text{allBlocks}$:
assumes act: $HFail\ s\ s'\ p$
shows $\text{allBlocks}\ s' \subseteq \text{allBlocks}\ s \cup \{\text{dblock}\ s'\ p\}$

$\langle \text{proof} \rangle$

lemma $HFail-HInv5-\text{q1}$:
assumes act: $HFail\ s\ s'\ p$
and $pnq$: $p \neq q$
and $inv2a$: $Inv2a-\text{inner}\ s'\ q$
and $inv5-1$: $\text{maxBalInp}\ s\ (\text{bal}(\text{dblock}\ s\ q)) (\text{inp}(\text{dblock}\ s\ q))$
shows $\text{maxBalInp}\ s' (\text{bal}(\text{dblock}\ s'\ q)) (\text{inp}(\text{dblock}\ s'\ q))$

$\langle \text{proof} \rangle$

lemma $HFail-HInv5-\text{q2}$:
assumes act: $HFail\ s\ s'\ p$
and $pnq$: $p \neq q$
and $inv5-2$: $\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock}\ s\ q) < \text{mbal} (\text{disk}\ d\ qq)$

$\langle \text{proof} \rangle$
\[ \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d qq) \quad \land \neg \text{hasRead } s q d qq) \]

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

\langle \text{proof} \rangle

\text{lemma} \ HFail-Hinv5-q:
\text{assumes} \ act: HFail s s' p
\text{and} \ inv: Hinv5-inner s q
\text{and} \ pnq: p \neq q
\text{shows} Hinv5-inner s' q
\langle \text{proof} \rangle

\text{theorem} \ HFail-Hinv5:
[ HFail s s' p; Hinv5-inner s q; Inv2a s' ] \implies Hinv5-inner s' q
\langle \text{proof} \rangle

\text{lemma} \ HPhase0Read-Hinv5-p:
HPhase0Read s s' p d \implies Hinv5-inner s' p
\langle \text{proof} \rangle

\text{lemma} \ HPhase0Read-allBlocks:
\text{assumes} \ act: HPhase0Read s s' p d
\text{shows} \ allBlocks s' \subseteq allBlocks s
\langle \text{proof} \rangle

\text{lemma} \ HPhase0Read-Hinv5-1:
\text{assumes} \ act: HPhase0Read s s' p d
\text{and} \ inv5-1: \text{maxBalInp } s (\text{bal}(\text{dblock } s q)) (\text{inp}(\text{dblock } s q))
\text{shows} \text{maxBalInp } s' (\text{bal}(\text{dblock } s' q)) (\text{inp}(\text{dblock } s' q))
\langle \text{proof} \rangle

\text{lemma} \ HPhase0Read-Hinv5-2:
\text{assumes} \ act: HPhase0Read s s' p d
\text{and} \ pnq: p \neq q
\text{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) \quad \land \neg \text{hasRead } s q d qq)
\text{shows} \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) \quad \land \neg \text{hasRead } s' q d qq)
\langle \text{proof} \rangle

\text{lemma} \ HPhase0Read-Hinv5-q:
\text{assumes} \ act: HPhase0Read s s' p d
\text{and} \ inv: Hinv5-inner s q
\text{and} \ pnq: p \neq q
\text{shows} Hinv5-inner s' q
\langle \text{proof} \rangle
**Theorem** $\text{HPhase0Read-HInv5}$:

\[ [\text{HPhase0Read } s \ s' \ p \ d ]; \text{HInv5-inner } s \ s' q ] \implies \text{HInv5-inner } s' q \]

**Lemma** $\text{HEndPhase0-HInv5-p}$:

$\text{HEndPhase0 } s \ s' p \implies \text{HInv5-inner } s' p$

**Lemma** $\text{HEndPhase0-blocksOf-q}$:

**assumes** act: $\text{HEndPhase0 } s \ s' p$

**and** pnq: $p \neq q$

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

**Lemma** $\text{HEndPhase0-allBlocks}$:

**assumes** act: $\text{HEndPhase0 } s \ s' p$

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

**Lemma** $\text{HEndPhase0-HInv5-q1}$:

**assumes** act: $\text{HEndPhase0 } s \ s' p$

**and** pnq: $p \neq q$

**and** inv1: $\text{Inv1 } s$

**and** inv5-1: $\text{maxBalInp } s \ (\text{bal d} \text{block } s q) \ (\text{inp d} \text{block } s q))$

**shows** $\text{maxBalInp } s' \ (\text{bal d} \text{block } s' q) \ (\text{inp d} \text{block } s' q))$

**Lemma** $\text{HEndPhase0-HInv5-q2}$:

**assumes** act: $\text{HEndPhase0 } s \ s' p$

**and** pnq: $p \neq q$

**and** inv5-2: $\exists D \in \text{MajoritySet. } \exists q. (\forall d \in D. \text{bal d} \text{block } s q < \text{mbal } d \text{isk } s d q q)$

**shows** $\exists D \in \text{MajoritySet. } \exists q. (\forall d \in D. \text{bal d} \text{block } s' q < \text{mbal } d \text{isk } s' d q q)$

**Lemma** $\text{HEndPhase0-HInv5-q}$:

**assumes** act: $\text{HEndPhase0 } s \ s' p$

**and** inv: $\text{HInv5-inner } s q$

**and** inv1: $\text{Inv1 } s$

**and** pnq: $p \neq q$

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

**Theorem** $\text{HEndPhase0-HInv5}$:

\[ [\text{HEndPhase0 } s \ s' p \ s' q ] \implies \text{HInv5-inner } s' q \]
$HInv_1 \land HInv_2 \land HInv_3 \land HInv_4 \land HInv_5$ is an invariant of $HNext$.

**lemma I2e**:

assumes `nxt`: $HNext \ s \ s'$
and `inv`: $HInv_1 \ s \land HInv_2 \ s \land HInv_2 \ s' \land HInv_3 \ s \land HInv_4 \ s \land HInv_5 \ s$
shows $HInv_5 \ s'$
⟨proof⟩
end

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

C.6 Lemma I2f

To prove the final conjunct we will use the predicate $valueChosen(v)$. 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 $disksdp$, then
it has read a block with $bal$ field at least $b$.

definition valueChosen :: state ⇒ InputsOrNi ⇒ bool
where
valueChosen s v =
(∃ b ∈ (UN p. Ballot p).
maxBalInp s b v
∧ (∃ p. ∃ D ∈ MajoritySet. (∀ 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))))
))

**lemma HEndPhase1-valueChosen-inp**: 
assumes `act`: $HEndPhase1 \ s \ s' \ q$
and `inv2a`: $Inv2a \ s$
and `asm1`: $b ∈ (UN p. Ballot p)$
and `bk-blocksOf`: $bk ∈ blocksOf \ s \ r$
and `bk`: $bk ∈ blocksSeen \ s \ q$
and `b-bal`: $b ≤ bal \ bk$
and `asm3`: $maxBalInp \ s \ b \ v$
and `inv1`: $Inv1 \ s$
shows $inp(dblock \ s' \ q) = v$
⟨proof⟩

**lemma HEndPhase1-maxBalInp**: 
assumes `act`: $HEndPhase1 \ s \ s' \ q$
and `asm1`: $b ∈ (UN p. Ballot p)$
and `asm2`: $D ∈ MajoritySet$
and `asm3`: $maxBalInp \ s \ b \ v$
and `asm4`: $∀ d ∈ D. b ≤ bal(disk \ s \ d \ p)$

59
∧(∀q.( phase s q = 1 \\
∧ b ≤ mbal(dblock s q) \\
∧ hasRead s q d p ) → (∃br∈blocksRead s q d. b ≤ bal(block br)))

and inv1: Inv1 s
and inv2a: Inv2a s
and inv2b: Inv2b s
shows maxBalImp s' b v

⟨proof⟩

lemma HEndPhase1-valueChosen2:
assumes act: HEndPhase1 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))) (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 ∈ Inputs
shows valueChosen s' v
⟨proof⟩

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

lemma HStartBallot-valueChosen2:
assumes act: HStartBallot 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))) (is ?P s)

shows ?P s'
⟨proof⟩

theorem HStartBallot-valueChosen:
assumes act: HStartBallot s s' q

60
and \( vc : \text{valueChosen} s v \)
and \( v \)-input: \( v \in \text{Inputs} \)
shows \( \text{valueChosen} s' v \)
\( \langle \text{proof} \rangle \)

lemma \( \text{HPhase1or2Write-maxBalInp} \):
assumes act: \( \text{HPhase1or2Write} s s' q d \)
and asm3: \( \text{maxBalInp} s b v \)
shows \( \text{maxBalInp} s' b v \)
\( \langle \text{proof} \rangle \)

lemma \( \text{HPhase1or2Write-valueChosen2} \):
assumes act: \( \text{HPhase1or2Write} s s' pp d \)
and asm2: \( D \in \text{MajoritySet} \)
and asm4: \( \forall d \in D. \quad b \leq \text{bal} (\text{disk} s d p) \)
\( \land (\forall q . (\quad \text{phase} s q = 1 \)
\( \land b \leq \text{mbal} (\text{dblock} s q) \)
\( \land \text{hasRead} s q d p \)
\( ) \rightarrow (\exists \text{br} \in \text{blocksRead} s q d . \quad b \leq \text{bal} (\text{block} \text{br})) \) (is \( ?P s \))
and inv4: \( \text{HInv4a} s pp \)
shows \( ?P s' \)
\( \langle \text{proof} \rangle \)

theorem \( \text{HPhase1or2Write-valueChosen} \):
assumes act: \( \text{HPhase1or2Write} s s' q d \)
and \( vc : \text{valueChosen} s v \)
and \( v \)-input: \( v \in \text{Inputs} \)
and inv4: \( \text{HInv4a} s q \)
shows \( \text{valueChosen} s' v \)
\( \langle \text{proof} \rangle \)

lemma \( \text{HPhase1or2ReadThen-maxBalInp} \):
assumes act: \( \text{HPhase1or2ReadThen} s s' q d p \)
and asm3: \( \text{maxBalInp} s b v \)
shows \( \text{maxBalInp} s' b v \)
\( \langle \text{proof} \rangle \)

lemma \( \text{HPhase1or2ReadThen-valueChosen2} \):
assumes act: \( \text{HPhase1or2ReadThen} s s' q d pp \)
and asm4: \( \forall d \in D. \quad b \leq \text{bal} (\text{disk} s d p) \)
\( \land (\forall q . (\quad \text{phase} s q = 1 \)
\( \land b \leq \text{mbal} (\text{dblock} s q) \)
\( \land \text{hasRead} s q d p \)
\( ) \rightarrow (\exists \text{br} \in \text{blocksRead} s q d . \quad b \leq \text{bal} (\text{block} \text{br})) \) (is \( ?P s \))
shows \( ?P s' \)
\( \langle \text{proof} \rangle \)

theorem \( \text{HPhase1or2ReadThen-valueChosen} \):
\textbf{assumes} \( \text{act}: \) HPhase1or2ReadThen \( s \ s' \ q \ d \ p \)
\textbf{and} \( \text{vc}: \) valueChosen \( s \ v \)
\textbf{and} \( \text{v-input}: \) \( v \in \text{Inputs} \)
\textbf{shows} \( \text{valueChosen} \ s' \ v \)

\textbf{proof} \\

\textbf{theorem} HPhase1or2ReadElse-valueChosen:
\[ [ \text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ r; \ \text{valueChosen} \ s \ v; \ v \in \text{Inputs} ] \]
\[ \implies \text{valueChosen} \ s' \ v \]

\textbf{proof} \\

\textbf{lemma} HEndPhase2-maxBalInp:
\textbf{assumes} \( \text{act}: \) HEndPhase2 \( s \ s' \ q \)
\textbf{and} \( \text{asm3}: \) maxBalInp \( s \ b \ v \)
\textbf{shows} \( \text{maxBalInp} \ s' \ b \ v \)

\textbf{proof} \\

\textbf{lemma} HEndPhase2-valueChosen2:
\textbf{assumes} \( \text{act}: \) HEndPhase2 \( s \ s' \ q \)
\textbf{and} \( \text{asm4}: \) \( \forall \ d \in D. \ b \leq \text{bal}(\text{disk} \ s \ d \ p) \)
\textbf{and} \( \forall \ q. ( \text{phase} \ s \ q = 1 \)
\textbf{and} \( b \leq \text{mbal} (\text{dblock} \ s \ q) \)
\textbf{and} \( \text{hasRead} \ s \ q \ d \ p \)
\() \implies (\exists \ br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ br))) \) (is \( ?P \ s \))
\textbf{shows} \( ?P \ s' \)

\textbf{proof} \\

\textbf{theorem} HEndPhase2-valueChosen:
\textbf{assumes} \( \text{act}: \) HEndPhase2 \( s \ s' \ q \)
\textbf{and} \( \text{vc}: \) valueChosen \( s \ v \)
\textbf{and} \( \text{v-input}: \) \( v \in \text{Inputs} \)
\textbf{shows} \( \text{valueChosen} \ s' \ v \)

\textbf{proof} \\

\textbf{lemma} HFail-maxBalInp:
\textbf{assumes} \( \text{act}: \) HFail \( s \ s' \ q \)
\textbf{and} \( \text{asm1}: \) \( b \in (\text{UN} \ p. \ \text{Ballot} \ p) \)
\textbf{and} \( \text{asm3}: \) maxBalInp \( s \ b \ v \)
\textbf{shows} \( \text{maxBalInp} \ s' \ b \ v \)

\textbf{proof} \\

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

**Theorem HFail-valueChosen**:
- **Assumes** 
  - `act`: `HFail s s' q`  
  - `vc`: `valueChosen s v`  
  - `v-input`: `v ∈ Inputs`  
- **Shows** `valueChosen s' v`  


proof

**Lemma HPhase0Read-maxBalInp**:
- **Assumes** 
  - `act`: `HPhase0Read s s' q d`  
  - `asm3`: `maxBalInp s b v`  
- **Shows** `maxBalInp s' b v`  


proof

**Lemma HPhase0Read-valueChosen2**:
- **Assumes** 
  - `act`: `HPhase0Read s s' q d`  
  - `asm4`: `∀ d ∈ D. b ≤ bal(disk s d p)`  
  - `phase s q = 1`  
  - `b ≤ mbal(dblock s q)`  
  - `hasRead s q d p`  
  
- **Shows** `∀ P s. ?P s`  


proof

**Theorem HPhase0Read-valueChosen**:
- **Assumes** 
  - `act`: `HPhase0Read s s' q d`  
  - `vc`: `valueChosen s v`  
  - `v-input`: `v ∈ Inputs`  
- **Shows** `valueChosen s' v`  


proof

**Lemma HEndPhase0-maxBalInp**:
- **Assumes** 
  - `act`: `HEndPhase0 s s' q`  
  - `asm3`: `maxBalInp s b v`  
  - `inv1`: `Inv1 s`  
- **Shows** `maxBalInp s' b v`  


proof

**Lemma HEndPhase0-valueChosen2**:
- **Assumes** 
  - `act`: `HEndPhase0 s s' q`  
  - `asm4`: `∀ d ∈ D. b ≤ bal(disk s d p)`  
  - `∀ q. ( phase s q = 1`  
  - `b ≤ mbal(dblock s q)`  
  - `hasRead s q d p`  
  
- **Shows** `∀ P s. ?P s`  


proof
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 \( \Rightarrow \) bool
where
\[ HInv6 s = ((chosen s \neq NotAnInput \rightarrow valueChosen s (chosen s)) \land (\forall p. \text{outpt} s p \in \{chosen s, NotAnInput\})) \]

theorem \( HInit-HInv6 \): \( HInit s \Rightarrow HInv6 s \)
\( \langle \text{proof} \rangle \)

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 \( chosen': chosen s' \neq NotAnInput \)
shows \( valueChosen s' (chosen s') \)
\( \langle \text{proof} \rangle \)

lemma \( valueChosen-equal-case \):
assumes \( max-v: \text{maxBalInp} s b v \)
and \( Dmaj: D \in \text{MajoritySet} \)
and \( asm-v: \forall d \in D. b \leq bal (disk s d p) \)
and \( max-w: \text{maxBalInp} s ba w \)
and \( Damaj: Da \in \text{MajoritySet} \)
and \( asm-w: \forall d \in Da. ba \leq bal (disk s d pa) \)
and \( b-ba: b \leq ba \)
shows \( v=w \)
\( \langle \text{proof} \rangle \)
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 \( \text{inv2c}: \text{Inv2c} s \)

shows \( H\text{inv6} s' \)

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

**Theorem HPhase1or2ReadThen-Inv6:**

assumes act: HPhase1or2ReadThen s s' p d q

and inv: Hinv6 s

and inv2c: Inv2c s

shows Hinv6 s'

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

**Theorem HPhase1or2ReadElse-Inv6:**

assumes act: HPhase1or2ReadElse s s' p d q

and inv: Hinv6 s

and inv2c: Inv2c s

shows Hinv6 s'

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

**Theorem HEndPhase1-Inv6:**

assumes act: HEndPhase1 s s' p

and inv: Hinv6 s

and inv1: Inv1 s

and inv2a: Inv2a s

and inv2b: Inv2b s

and inv2c: Inv2c s

shows Hinv6 s'

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

**Lemma outpt-chosen-2:**

assumes outpt: outpt s' = (outpt s) (p := NotAnInput)

and inv2c: Inv2c s

and nextp: HNextPart s s'

shows chosen s = chosen s'

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

**Lemma outpt-Hinv6-2:**

assumes outpt: outpt s' = (outpt s) (p := NotAnInput)

and inv: \( \forall p. \) outpt s p \( \in \) \{chosen s, NotAnInput\}

and inv2c: Inv2c s

and nextp: HNextPart s s'

shows \( \forall p. \) outpt s' p \( \in \) \{chosen s', NotAnInput\}

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

**Theorem HFail-Inv6:**

assumes act: HFail s s' p

and inv: Hinv6 s

and inv2c: Inv2c s

shows Hinv6 s'

\( \langle \text{proof} \rangle \)
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 ∧ HInv2 ∧ HInv2' ∧ HInv3 ∧ HInv4 ∧ HInv5 ∧ HInv6 is an invariant
of HNext.

lemma I2f:
  assumes nxt: HNext s s'
  and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s ∧ HInv6 s
  shows HInv6 s' ⟨proof⟩
end

theory DiskPaxos-Invariant imports DiskPaxos-Inv6 begin

C.8 The Complete Invariant

definition HInv :: state ⇒ bool
where
  HInv s = (HInv1 s
  ∧ HInv2 s
  ∧ HInv3 s
  ∧ HInv4 s
  ∧ HInv5 s
  ∧ HInv6 s)

theorem I1:
  HInit s ⟹ HInv s
⟨proof⟩

theorem I2:
  assumes inv: HInv s
  and nxt: HNext s s'
  shows HInv s'
⟨proof⟩
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

definition Init :: Istate ⇒ bool
where
Init 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}
\textbf{theorem R1}: \\
\[ [ \text{HInit } s; \text{is } = \text{s2is } s] \implies \text{Hinit is} \] \\
\langle\text{proof}\rangle

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

\textbf{end}