Proving the Correctness of Disk Paxos in
Isabelle/HOL

Mauro Jaskelioff      Stephan Merz

August 16, 2018

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 \( HIv1 \) and \( HIv3 \)) 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 \textit{dblock}), and other state variables (see figure 1). When a process $p$ starts it contains an input value $\text{input}[p]$ that will not be modified, except possibly when recovering from a failure.

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

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

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

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

In each block, processors maintain three values:

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

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

2.2 Disk Paxos and its TLA$^+$ Specification

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

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

The specification of the algorithm is given in the $\text{HDiskSynod}$ 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:

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

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:

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

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

**Theorem R1**  \hspace{1em} $HInit \Rightarrow IInit$

**Theorem R2**  \hspace{1em} $HInit \land \Box[HNext]_{\{\text{vars}, \text{chosen}, \text{allInput}\}} \Rightarrow \Box[INext]_{\text{ivars}}$

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**  \hspace{1em} $HInit \land \Box[HNext]_{\{\text{vars}, \text{chosen}, \text{allInput}\}} \Rightarrow \Box[HInv]$

**Theorem R2b**  \hspace{1em} $HInv \land HInv' \land HNext \Rightarrow INext \lor (\text{UNCHANGED ivars})$

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

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

$\mathit{choose} \ x. P x$

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

$\mathit{phase}'(s) = (\mathit{phase}(s))(p := 1)$

$\mathit{UN} \ p. \ \mathit{blocksOf} \ s \ p$

$\mathit{UNCHANGED} \ v$

$v' = v$

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

THEOREM I1 $HInit \Rightarrow HInv$

THEOREM I2 $HInv \land HNext \Rightarrow HInv'$

Again, we have TLA proof rules that say that I1 and I2 imply R2a. In summary, R2b, I1, and I2 together imply HDiskSynodSpec $\Rightarrow ISpec$.

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

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

3 Translating from TLA+ to Isabelle/HOL

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

\textsuperscript{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⁺:

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

Isabelle/HOL:

\textit{typedef} \text{InputsOrNi}

\textit{consts}

\text{Inputs :: InputsOrNi set} \\
\text{NotAnInput :: InputsOrNi}

\textit{axioms}

\text{NotAnInput: NotAnInput \notin Inputs} \\
\text{InputsOrNi: (UNIV :: InputsOrNi set) = Inputs \cup \{NotAnInput\}}

\textit{record}

\text{DiskBlock =} \\
\quad \text{mbal :: nat} \\
\quad \text{bal :: nat} \\
\quad \text{inp :: InputsOrNi}

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

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

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

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

\[ \text{Phase1or2Write}(p, d) \triangleq \]
\[ \land \text{phase}[p] \in \{1, 2\} \]
\[ \land \text{disk}' = [\text{disk} \text{ EXCEPT } ![d][p] = \text{dblock}[p]] \]
\[ \land \text{disksWritten}' = [\text{disksWritten} \text{ EXCEPT } ![p] = \mathbb{0} \cup \{d\}] \]
\[ \land \text{UNCHANGED } (\text{input}, \text{output}, \text{phase}, \text{dblock}, \text{blocksRead}) \]

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

Another solution to this matter that is being currently investigated is the implementation of native (untyped) support for TLA⁺ in Isabelle, without relying on HOL.

### 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 \text{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 \text{Let-def} to Isabelle’s simplifier, which unfolds all “let” constructs.
Another change that makes proofs easier is to break down actions into simpler actions (e.g. giving separate definitions for subactions corresponding to disjuncts). By making actions smaller, Isabelle has to deal with smaller formulas when we feed the prover with such an action.

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

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

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

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

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

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

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

we write:

\[
\text{Inv2a-innermost} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \Rightarrow \text{bool}
\]

\[
\text{Inv2a-innermost} s p bk \equiv \ldots
\]

\[
\text{Inv2a-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

\[
\text{Inv2a-inner} s p \equiv \forall bk \in \text{blocksOf s p}. \text{Inv2a-innermost} s p bk
\]

\[
\text{Inv2a} :: \text{state} \Rightarrow \text{bool}
\]

\[
\text{Inv2a} s \equiv \forall p. \text{Inv2a-inner} s p
\]

Now we can express that we want to obtain the fact

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

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

```
MODULE Synod

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

MODULE Inner

VARIABLES allInput, chosen

IInit ∆= \land input ∈ [Proc → Inputs]
\land output = \{ p ∈ Proc \mapsto NotAnInput \}
\land chosen = NotAnInput
\land allInput = input[p] : p ∈ Proc

IChoose(p) ∆= \land output[p] = NotAnInput
\land IF chosen = NotAnInput
       THEN ip ∈ allInput : \land chosen' = ip
       \land output' = \{ output \mapsto \{ !p = ip \} \}
       ELSE \land output' = \{ output \mapsto \{ !p = chosen \} \}
       \land UNCHANGED chosen
       \land UNCHANGED \langle input, allInput \rangle

IFail(p) ∆= \land output' = \{ output \mapsto \{ !p = NotAnInput \} \}
\land \exists ip ∈ Inputs : \land input' = \{ input \mapsto \{ !p = ip \} \}
\land allInput' = allInput ∪ \{ ip \}

INext ∆= \exists p ∈ Proc : IChoose(p) ∨ IFail(p)
ISpec ∆= IInit ∧ □ INext <input, output, chosen, allInput>

IS(chosen, allInput) ∆= INSTANCE Inner
SynodSpec ∆= \exists chosen, allInput : IS(chosen, allInput)!ISpec
```
B  Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

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
  majorities-intersect:
    ∀ S T. IsMajority(S) ∧ IsMajority(T) → S ∩ T ≠ {}

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

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

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

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

definition InitDB :: DiskBlock
  where InitDB = (mbal = 0, bal = 0, inp = NotAnInput)

record
  BlockProc =
    block :: DiskBlock
    proc :: Proc

record
  state =
\text{inpt} :: \text{Proc} \Rightarrow \text{InputsOrNi}
\text{outpt} :: \text{Proc} \Rightarrow \text{InputsOrNi}
\text{disk} :: \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock}
\text{dblock} :: \text{Proc} \Rightarrow \text{DiskBlock}
\text{phase} :: \text{Proc} \Rightarrow \text{nat}
\text{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}

\text{definition} \text{hasRead} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\text{where} \text{hasRead} s p d q = (\exists \ br \in \text{blocksRead} s p d. \ proc br = q)

\text{definition} \text{allRdBlks} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{BlockProc set}
\text{where} \text{allRdBlks} s p = (\text{UN} d. \ \text{blocksRead} s p d)

\text{definition} \text{allBlocksRead} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set}
\text{where} \text{allBlocksRead} s p = \text{block } (^{\text{allRdBlks}} s p)

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

\text{definition} \text{InitializePhase} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\text{where} \text{InitializePhase} s s' p =
\text{(disksWritten} s' = (\text{disksWritten} s)(p := \{\})
\& \text{blocksRead} s' = (\text{blocksRead} s)(p := (\lambda d. \ \{\}))

\text{definition} \text{StartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\text{where} \text{StartBallot} s s' p =
\text{(phase} s p \in \{1, 2\}
\& \text{phase} s' = (\text{phase} s)(p := 1)
\& (\exists b \in \text{Ballot} p.
\text{mbal} (\text{dblock} s p) < b
\& \text{dblock} s' = (\text{dblock} s)(p := (\text{dblock} s p)(\text{mbal} := b \ |))
\& \text{InitializePhase} s s' p
\& \text{inpt} s' = \text{inpt} s \& \text{outpt} s' = \text{outpt} s \& \text{disk} s' = \text{disk} s)
definition Phase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool
where
Phase1or2Write s s' p d =
    (phase s p ∈ \{1, 2\}
    ∧ disk s' = (disk s) (d := (disk s d) (p := dblock s p))
    ∧ disksWritten s' = (disksWritten s) (p:= (disksWritten s p) ∪ \{d\})
    ∧ inpt s' = inpt s ∧ outpt s' = outpt s
    ∧ phase s' = phase s ∧ dblock s' = dblock s
    ∧ blocksRead s' = blocksRead s)

definition Phase1or2ReadThen :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where
Phase1or2ReadThen s s' p d q =
    (d ∈ disksWritten s p
    & mbal(disk s d q) < mbal(dblock s p)
    & blocksRead s' = (blocksRead s)(p := (blocksRead s p)(d :=
        (blocksRead s p d) ∪ \{block = disk s d q, proc = q \})))
    ∧ inpt s' = inpt s ∧ outpt s' = outpt s
    ∧ disk s' = disk s ∧ phase s' = phase s
    ∧ dblock s' = dblock s ∧ disksWritten s' = disksWritten s)

definition Phase1or2ReadElse :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where
Phase1or2ReadElse s s' p d q =
    (d ∈ disksWritten s p
    ∧ StartBallot s s' p)

definition Phase1or2Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where
Phase1or2Read s s' p d q =
    (Phase1or2ReadThen s s' p d q
    ∨ Phase1or2ReadElse s s' p d q)

definition blocksSeen :: state ⇒ Proc ⇒ DiskBlock set
where
blocksSeen s p = allBlocksRead s p ∪ \{dblock s p\}

definition nonInitBlks :: state ⇒ Proc ⇒ DiskBlock set
where
nonInitBlks s p = \{bs . bs ∈ blocksSeen s p ∧ inp bs ∈ Inputs\}

definition maxBlk :: state ⇒ Proc ⇒ DiskBlock
where
maxBlk s p =
    (SOME b . b ∈ nonInitBlks s p \(∀ c ∈ nonInitBlks s p . bal c ≤ bal b))

definition EndPhase1 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase1 s s' p =
    (IsMajority \{d . d ∈ disksWritten s p\)
∧ (∀ q ∈ UNIV − {p}. hasRead s p d q))
∧ phase s p = 1
∧ dblock s′ = (dblock s) (p := dblock s p
  (∥ bal := mbal(dblock s p),
   inp :=
     (if nonInitBlks s p = {} then inpt s p
      else inp (maxBlk s p))
  ∥)
∧ outpt s′ = outpt s
∧ phase s′ = (phase s) (p := phase s p + 1)
∧ InitializePhase s s′ p
∧ inpt s′ = inpt s ∧ disk s′ = disk s)

**definition** EndPhase2 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase2 s s′ p = (IsMajority {d . d ∈ disksWritten s p
  ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)})
∧ phase s p = 2
∧ outpt s′ = (outpt s) (p := inp (dblock s p))
∧ dblock s′ = dblock s
∧ phase s′ = (phase s) (p := phase s p + 1)
∧ InitializePhase s s′ p
∧ inpt s′ = inpt s ∧ disk s′ = disk s)

**definition** EndPhase1or2 :: state ⇒ state ⇒ Proc ⇒ bool
where EndPhase1or2 s s′ p = (EndPhase1 s s′ p ∨ EndPhase2 s s′ p)

**definition** Fail :: state ⇒ state ⇒ Proc ⇒ bool
where
Fail s s′ p = (∃ ip ∈ Inputs. inpt s′ = (inpt s) (p := ip))
∧ phase s′ = (phase s) (p := 0)
∧ dblock s′ = dblock s (p := InitDB)
∧ outpt s′ = (outpt s) (p := NotAnInput)
∧ InitializePhase s s′ p
∧ disk s′ = disk s)

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

**definition** Next :: state ⇒ state ⇒ bool
where
Next s s' = (\exists p.
  \text{StartBallot } s s' p
∧ (\exists d. \text{Phase0Read } s s' p d
∧ \text{Phase1or2Write } s s' p d
∧ (\exists q. q \neq p \∧ \text{Phase1or2Read } s s' p d q))
∧ \text{EndPhase1or2 } s s' p
∧ \text{Fail } s s' p
∧ \text{EndPhase0 } s s' p)

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

**definition** HInit :: state ⇒ bool
where
HInit s =
(Init s
∧ chosen s = \text{NotAnInput}
∧ allInput s = \text{range } (\text{inpt } s))

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

**definition** HNextPart :: state ⇒ state => bool
where
HNextPart s s' =
(\text{chosen } s' =
  (\text{if chosen } s \neq \text{NotAnInput} \lor (\forall p. \text{outpt } s' p = \text{NotAnInput} )
  \text{then chosen } s
  \text{else outpt } s' (\text{SOME } p. \text{outpt } s' p \neq \text{NotAnInput}))
∧ allInput s' = allInput s \cup (\text{range } (\text{inpt } s')))

**definition** HNext :: state ⇒ state ⇒ bool
where
HNext s s' =
(Next s s' \land HNextPart s s')

We add HNextPart to every action (rather than proving that Next maintains the HInv invariant) to make proofs easier.

definition HPhase1or2ReadThen :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool where
HPhase1or2ReadThen s s' p d q = (Phase1or2ReadThen s s' p d q \land HNextPart s s')

definition HEndPhase1 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool where
HEndPhase1 s s' p = (EndPhase1 s s' p \land HNextPart s s')

definition HStartBallot :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool where
HStartBallot s s' p = (StartBallot s s' p \land HNextPart s s')

definition HPhase1or2Write :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow bool where
HPhase1or2Write s s' p d = (Phase1or2Write s s' p d \land HNextPart s s')

definition HPhase1or2ReadElse :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool where
HPhase1or2ReadElse s s' p d q = (Phase1or2ReadElse s s' p d q \land HNextPart s s')

definition HEndPhase2 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool where
HEndPhase2 s s' p = (EndPhase2 s s' p \land HNextPart s s')

definition HFail :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool where
HFail s s' p = (Fail s s' p \land HNextPart s s')

definition HPhase0Read :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow bool where
HPhase0Read s s' p d = (Phase0Read s s' p d \land HNextPart s s')

definition HEndPhase0 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool where
HEndPhase0 s s' p = (EndPhase0 s s' p \land HNextPart s s')

Since these definitions are the conjunction of two other definitions declaring them as simplification rules should be harmless.
declare HPhase1or2ReadThen-def [simp]
declare HPhase1or2ReadElse-def [simp]
declare HEndPhase1-def [simp]
declare HStartBallot-def [simp]
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.
  inp t s p ∈ Inputs
∧ phase s p ≤ 3
∧ finite (allRdBlks s p))

definition HInv1 :: state ⇒ bool
where
HInv1 s =
  (Inv1 s
∧ allInput s ⊆ Inputs)
declare HInv1-def [simp]

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

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

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

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

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

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

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

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

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

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

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

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

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

declare HInv1-def [simp del]

HInv1 is an invariant of HNext

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

end

theory DiskPaxos-Inv2 imports DiskPaxos-Inv1 begin

C.2 Invariant 2

The second invariant is split into three main conjuncts called Inv2a, Inv2b, 
and Inv2c. The main difficulty is in proving the preservation of the first 
conjunct.
definition rdBy :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ BlockProc set 
where 
rdBy s p q d = 
{br . br ∈ blocksRead s q d ∧ proc br = p}
definition blocksOf :: state ⇒ Proc ⇒ DiskBlock set 
where 
blocksOf s p = 
{dblock s p} 
∪ {disk s d p | d . d ∈ UNIV} 
∪ {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 = 
(mbval 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** Inv2a-inner :: state ⇒ Proc ⇒ bool

*where* Inv2a-inner s p = (∀ bk ∈ blocksOf s p. Inv2a-innermost s p bk)

**definition** Inv2a :: state ⇒ bool

*where* Inv2a s = (∀ p. Inv2a-inner s p)

**definition** Inv2b-inner :: state ⇒ Proc ⇒ Disk ⇒ bool

*where*

Inv2b-inner s p d =
((d ∈ disksWritten s p →
  (phase s p ∈ {1, 2} ∧ disk s d p = dblock s p))
∧ (phase s p ∈ {1, 2} →
  (blocksRead s p d ≠ {} → d ∈ disksWritten s p)
∧ ¬ hasRead s p d))

**definition** Inv2b :: state ⇒ bool

*where* Inv2b s = (∀ p d. Inv2b-inner s p d)

**definition** Inv2c-inner :: state ⇒ Proc ⇒ bool

*where*

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

**definition** Inv2c :: state ⇒ bool

*where* Inv2c s = (∀ p. Inv2c-inner s p)

**definition** HInv2 :: state ⇒ bool

*where* HInv2 s = (Inv2a s ∧ Inv2b s ∧ Inv2c s)

C.2.1 Proofs of Invariant 2 a

**theorem** HInit-Inv2a: HInit s → Inv2a s

*⟨proof⟩*

For every action we define a action-blocksOf lemma. We have two cases: 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** \( \text{HPhase1or2ReadThen-blocksOf} \):
\[
\left[ \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q \right] \implies \text{blocksOf} \ s' \ r \subseteq \text{blocksOf} \ s \ r
\]
\( \langle \text{proof} \rangle \)

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

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

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

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

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

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

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

**theorem** \( \text{HPhase1or2Write-Inv2a} \):
\begin{align*}
\text{assumes} & \text{ inv: Inv2a } s \\
\text{and act:} & \text{HPhase1or2Write} \ s \ s' \ p \ d
\end{align*}
shows $\text{Inv2a } s'$

(\text{proof})

\textbf{theorem} \ H\text{Phase1or2ReadElse-Inv2a}:
assumes inv: $\text{Inv2a } s$
and act: $\text{HPhase1or2ReadElse } s s' p d q$
shows $\text{Inv2a } s'$
(\text{proof})

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

\textbf{theorem} \ H\text{EndPhase2-Inv2a}:
assumes inv: $\text{Inv2a } s$
and act: $\text{HEndPhase2 } s s' p$
shows $\text{Inv2a } s'$
(\text{proof})

\textbf{lemma} \ H\text{Fail-blocksOf}:
$\text{HFail } s s' p \implies \text{blocksOf } s' q \subseteq \text{blocksOf } s q \cup \{ \text{dblock } s' q \}$
(\text{proof})

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

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

\textbf{lemma} \ H\text{Phase0Read-blocksOf}:
$\text{HPhase0Read } s s' p d \implies \text{blocksOf } s' q \subseteq \text{blocksOf } s q$
(\text{proof})

\textbf{theorem} \ H\text{Phase0Read-Inv2a}:
assumes inv: $\text{Inv2a } s$
and act: $\text{HPhase0Read } s s' p d$
shows $\text{Inv2a } s'$
(\text{proof})

\textbf{lemma} \ H\text{EndPhase0-blocksOf}:
$\text{HEndPhase0 } s s' p \implies \text{blocksOf } s' q \subseteq \text{blocksOf } s q \cup \{ \text{dblock } s' q \}$
(\text{proof})
lemma HEndPhase0-blocksRead:
  assumes act: HEndPhase0 s s' p
  shows ∃ d. blocksRead s p d ≠ {}
⟨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

lemma HEndPhase0-some:
  assumes act: HEndPhase0 s s' p
  and inv1: Inv1 s
  shows (SOME b. b ∈ allBlocksRead s p
           ∧ (∀ t∈allBlocksRead s p. bal t ≤ bal b))
           ∈ allBlocksRead s p
  ∧ (∀ t∈allBlocksRead s p.
           bal t ≤ bal (SOME b. b ∈ allBlocksRead s p
           ∧ (∀ t∈allBlocksRead s p. bal t ≤ bal b)))
⟨proof⟩

lemma HEndPhase0-dblock-allBlocksRead:
  assumes act: HEndPhase0 s s' p
  and inv1: Inv1 s
  shows dblock s' p ∈ (λx. 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) ∈ allInput s' ∪ {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 ∀ t ∈ (λx. 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 mbk: 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 Inv2a-innermost s' p (dblock s' p)
⟨proof⟩

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

theorem HEndPhase1-Inv2a:
assumes act: HEndPhase1 s s' p
and inv1: HInv1 s
and inv: Inv2a s
and inv2c: Inv2c-inner s p
shows Inv2a s'
⟨proof⟩

C.2.2 Proofs of Invariant 2 b

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

theorem HInit-Inv2b: HInit s → Inv2b s
⟨proof⟩

theorem HPhase1or2ReadThen-Inv2b:
[ [ Inv2b s; HPhase1or2ReadThen s s' p d q ]
  → Inv2b s' ]
⟨proof⟩

theorem HStartBallot-Inv2b:
[ [ Inv2b s; HStartBallot s s' p ]
  → Inv2b s' ]
⟨proof⟩

theorem HPhase1or2Write-Inv2b:
[ [ Inv2b s; HPhase1or2Write s s' p d ]
  → Inv2b s' ]
⟨proof⟩

theorem HPhase1or2ReadElse-Inv2b:
[ [ Inv2b s; HPhase1or2ReadElse s s' p d q ]
  → Inv2b s' ]
⟨proof⟩

theorem HEndPhase1-Inv2b:
[ [ Inv2b s; HEndPhase1 s s' p ] → Inv2b s' ]
⟨proof⟩
theorem \textit{HFail-Inv2b}:
\[
\begin{array}{l}
\text{[ Inv2b s; HFail s s' p ]}
\end{array}
\]
\[
\Longrightarrow Inv2b s'
\]
\langle proof \rangle

theorem \textit{HEndPhase2-Inv2b}:
\[
\begin{array}{l}
\text{[ Inv2b s; HEndPhase2 s s' p ]}
\end{array}
\]
\[
\Longrightarrow Inv2b s'
\]
\langle proof \rangle

theorem \textit{HPhase0Read-Inv2b}:
\[
\begin{array}{l}
\text{[ Inv2b s; HPhase0Read s s' p d ]}
\end{array}
\]
\[
\Longrightarrow Inv2b s'
\]
\langle proof \rangle

theorem \textit{HEndPhase0-Inv2b}:
\[
\begin{array}{l}
\text{[ Inv2b s; HEndPhase0 s s' p ]}
\end{array}
\]
\[
\Longrightarrow Inv2b s'
\]
\langle proof \rangle

C.2.3 Proofs of Invariant 2 c

theorem \textit{HInit-Inv2c}:
\[
\text{HInit s} \Rightarrow Inv2c s
\]
\langle proof \rangle

lemma \textit{HNextPart-Inv2c-chosen}:
\begin{itemize}
  \item \textbf{assumes} hnp: HNextPart s s'
  \item \textbf{and} inv2c: Inv2c s
  \item \textbf{and} outpt': \forall p. outpt s' p = (if phase s' p = 3
    \textbf{then} inp(dblock s' p)
    \textbf{else} NotAnInput)
  \item \textbf{and} inp-dblk: \forall p. inp(dblock s' p) \in allInput s' \cup \{NotAnInput\}
  \item \textbf{shows} chosen s' \in allInput s' \cup \{NotAnInput\}
\end{itemize}
\langle proof \rangle

lemma \textit{HNextPart-chosen}:
\begin{itemize}
  \item \textbf{assumes} hnp: HNextPart s s'
  \item \textbf{shows} chosen s' = NotAnInput \Rightarrow (\forall p. outpt s' p = NotAnInput)
\end{itemize}
\langle proof \rangle

lemma \textit{HNextPart-allInput}:
\[
\begin{array}{l}
\text{[ HNextPart s s'; Inv2c s ]}
\end{array}
\]
\[
\forall p. \text{inpt s' p} \in \text{allInput s'}
\]
\langle proof \rangle

theorem \textit{HPhase1or2ReadThen-Inv2c}:
\begin{itemize}
  \item \textbf{assumes} inv: Inv2c s
  \item \textbf{and} act: HPhase1or2ReadThen s s' p d q
  \item \textbf{and} inv2a: Inv2a s
  \item \textbf{shows} Inv2c s'
\end{itemize}
\langle proof \rangle
theorem HStartBallot-Inv2c:
assumes inv: Inv2c s
and act: HStartBallot s s' p
and inv2a: Inv2a s
shows Inv2c s'
(proof)

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

theorem HPhase1or2ReadElse-Inv2c:
[ Inv2c s; HPhase1or2ReadElse s s' p d q; Inv2a s ] \to 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
lemma InitPhase-HInv3-p:
\[
\begin{array}{c}
\text{[ InitializePhase } s s' \ p p \ q q \ d d]\ 
\implies \ HInv3-R s' s q q d
\end{array}
\]
(\textit{proof})

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

lemma HInv3-L-sym: \( \text{HInv3-L } s p q d \implies \text{HInv3-L } s q p d \)
(\textit{proof})

lemma HInv3-R-sym: \( \text{HInv3-R } s p q d \implies \text{HInv3-R } s q p d \)
(\textit{proof})

lemma Phase1or2ReadThen-HInv3-pq:
\[
\text{assumes act}: \ Phase1or2ReadThen s s' p q d q
\text{and inv-L'}: \ HInv3-L s' p q d
\text{and pq}: \ p \neq q
\text{and inv2b}: \ Inv2b s
\text{shows} \ HInv3-R s' p q d
\]
(\textit{proof})

lemma Phase1or2ReadThen-HInv3-hasRead:
\[
\begin{array}{c}
\text{[ \neg hasRead } s p p d d q q; \\
\text{Phase1or2ReadThen } s s' p q d q; \\
p p \neq p \lor q q \neq q \lor d d \neq d]\ 
\implies \neg hasRead s' p p d d q q
\end{array}
\]
(\textit{proof})

theorem HPhase1or2ReadThen-HInv3:
\[
\text{assumes act}: \ HPhase1or2ReadThen s s' p q d q
\text{and inv}: \ HInv3 s
\text{and pq}: \ p \neq q
\text{and inv2b}: \ Inv2b s
\text{shows} \ HInv3 s'
\]
(\textit{proof})

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

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

31
lemma StartBallot-HInv3-nL:
\[
\begin{align*}
\texttt{StartBallot} \ s \ 's \ t; \neg \texttt{HInv3-L} \ s \ p \ q \ d; \ t \neq p; \ t \neq q \ \\
\implies \neg \texttt{HInv3-L} \ s' \ p \ q \ d
\end{align*}
\]
⟨proof⟩

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

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

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

theorem HStartBallot-HInv3:
\[
\begin{align*}
\texttt{HStartBallot} \ s \ 's \ p; \texttt{HInv3} \ s \ \\
\implies \texttt{HInv3} \ s'
\end{align*}
\]
⟨proof⟩

theorem HPhase1or2ReadElse-HInv3:
\[
\begin{align*}
\texttt{HPhase1or2ReadElse} \ s \ 's \ p \ d \ q; \texttt{HInv3} \ s \ \\
\implies \texttt{HInv3} \ s'
\end{align*}
\]
⟨proof⟩

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

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

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

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

lemma EndPhase1-HInv3-R:
lemma EndPhase1-HInv3-t:
[ EndPhase1 s s’ t; HInv3-inner s p q d; t ≠ p; t ≠ q ]
⇒ HInv3-inner s’ p q d
⟨proof⟩

lemma EndPhase1-HInv3:
assumes act: EndPhase1 s s’ t
and inv: HInv3-inner s p q d
shows HInv3-inner s’ p q d
⟨proof⟩

theorem HEndPhase1-HInv3:
[ HEndPhase1 s s’ p; HInv3 s ] ⇒ HInv3 s’
⟨proof⟩

lemma EndPhase2-HInv3-p:
[ EndPhase2 s s’ p; HInv3-L s’ p q d ] ⇒ HInv3-R s’ p q d
⟨proof⟩

lemma EndPhase2-HInv3-q:
[ EndPhase2 s s’ q; HInv3-L s’ p q d ] ⇒ HInv3-R s’ p q d
⟨proof⟩

lemma EndPhase2-HInv3-nL:
[ EndPhase2 s s’ t; ¬HInv3-L s p q d; t ≠ p; t ≠ q ]
⇒ ¬HInv3-L s’ p q d
⟨proof⟩

lemma EndPhase2-HInv3-R:
[ EndPhase2 s s’ t; HInv3-R s p q d; t ≠ p; t ≠ q ]
⇒ HInv3-R s’ p q d
⟨proof⟩

lemma EndPhase2-HInv3-t:
[ EndPhase2 s s’ t; HInv3-inner s p q d; t ≠ p; t ≠ q ]
⇒ HInv3-inner s’ p q d
⟨proof⟩

lemma EndPhase2-HInv3:
assumes act: EndPhase2 s s’ t
and inv: HInv3-inner s p q d
shows HInv3-inner s’ p q d
⟨proof⟩

theorem HEndPhase2-HInv3:
\[ \text{HEndPhase2 } s \ s' \ p; \ HInv3 \ s \] \[ \implies \ HInv3 \ s' \]

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

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

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

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

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

\textbf{lemma Fail-HInv3:}
\textbf{assumes} act: Fail \ s \ s' \ t \\
\textbf{and} inv: HInv3-inner \ s \ p \ q \ d \\
\textbf{shows} \ HInv3-inner \ s' \ p \ q \ d 

\textbf{theorem HFail-HInv3:}
\[ \text{HFail } s \ s' \ p; \ HInv3 \ s \] \[ \implies \ HInv3 \ s' \]

\textbf{theorem HPhase0Read-HInv3:}
\textbf{assumes} act: HPhase0Read \ s \ s' \ p \ d \\
\textbf{and} inv: HInv3 \ s \\
\textbf{shows} \ HInv3 \ s' 

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

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

\text{lemma} \ \text{EndPhase0-HInv3-nL:}
\[
\begin{array}{l}
\text{[ EndPhase0 } s \ s' \ t; \neg\text{HInv3-L} \ s \ p \ q \ d; \ t \not= p; \ t \not= q \ ] \\
\Rightarrow \neg\text{HInv3-L} \ s' \ p \ q \ d
\end{array}
\]

\text{lemma} \ \text{EndPhase0-HInv3-R:}
\[
\begin{array}{l}
\text{[ EndPhase0 } s \ s' \ t; \text{HInv3-R} \ s \ p \ q \ d; \ t \not= p; \ t \not= q \ ] \\
\Rightarrow \text{HInv3-R} \ s' \ p \ q \ d
\end{array}
\]

\text{lemma} \ \text{EndPhase0-HInv3-t:}
\[
\begin{array}{l}
\text{[ EndPhase0 } s \ s' \ t; \text{HInv3-inner} \ s \ p \ q \ d; \ t \not= p; \ t \not= q \ ] \\
\Rightarrow \text{HInv3-inner} \ s' \ p \ q \ d
\end{array}
\]

\text{lemma} \ \text{EndPhase0-HInv3:}
\[
\begin{array}{l}
\text{assumes act: EndPhase0 } s \ s' \ t \\
\text{and inv: HInv3-inner} \ s \ p \ q \ d \\
\text{shows} \ \text{HInv3-inner} \ s' \ p \ q \ d
\end{array}
\]

\text{theorem} \ \text{HEndPhase0-HInv3:}
\[
\begin{array}{l}
\text{[ HEndPhase0 } s \ s' \ p; \text{HInv3} \ s \ ] \\
\Rightarrow \text{HInv3} \ s'
\end{array}
\]

\text{HInv1} \land \text{HInv2} \land \text{HInv3} \ \text{is an invariant of HNext.}

\text{lemma} \ \text{I2c:}
\[
\begin{array}{l}
\text{assumes nxt: HNext } s \ s' \\
\text{and inv: HInv1} \ s \ \land \text{HInv2} \ s \ \land \text{HInv3} \ s \\
\text{shows} \ \text{HInv3} \ s' \ \langle \text{proof} \rangle
\end{array}
\]

\text{end}

\text{theory} \ \text{DiskPaxos-Inv4} \ \text{imports} \ \text{DiskPaxos-Inv2} \ \text{begin}

\text{C.4 Invariant 4}

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

**definition** \( \text{MajoritySet} :: \text{Disk set set} \)
where \( \text{MajoritySet} = \{ D. \text{IsMajority}(D) \} \)

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

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

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

**definition** \( HInv4b :: \text{state} \rightarrow \text{Proc} \rightarrow \text{bool} \)
where \( HInv4b s p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \text{mbal}(\text{disk} s d p) = \text{bal}(\text{disk} s d p))) \)

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

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

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

The initial state implies Invariant 4.

**theorem** \( HInit-HInv4 :: \text{HInit} s \Rightarrow HInv4 s \)

(\text{proof})

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

For each action $actsss'q$ and conjunct $x \in a, b, c, d$ of $HInv4xs'sp$, we prove two lemmas. The first lemma $\text{action-H}HInv4x-p$ proves the case of $p = q$, while lemma $\text{action-H}HInv4x-q$ proves the other case.

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

**lemma** \( \text{HStartBallot-HInv4a1} :: \)

\text{assumes} \( \text{act: HStartBallot s s'} \)
\text{and} \( \text{inv: HInv4a1 s p} \)
\text{and} \( \text{inv2a: Inv2a-inner s'} \)
shows $HInv4a1 \ s' \ p$

(proof)

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

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

lemma $HStartBallot-HInv4a-q$:  
assumes act: $HStartBallot \ s \ s' \ p$
and inv: $HInv4a \ s \ q$
and pnq: $p \neq q$
shows $HInv4a \ s' \ q$
(proof)

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

lemma $Phase1or2Write-HInv4a1$:  
$[\ [Phase1or2Write \ s \ s' \ p \ d \ ; \ HInv4a1 \ s \ q ] ] \ \Rightarrow \ HInv4a1 \ s' \ q$
(proof)

lemma $Phase1or2Write-HInv4a2$:  
$[\ [Phase1or2Write \ s \ s' \ p \ d \ ; \ HInv4a2 \ s \ q ] ] \ \Rightarrow \ HInv4a2 \ s' \ q$
(proof)

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

lemma $HPhase1or2ReadThen-HInv4a1-p$:  
assumes act: $HPhase1or2ReadThen \ s \ s' \ p \ d \ q$
and inv: $HInv4a1 \ s \ p$
shows $HInv4a1 \ s' \ p$
(proof)
lemma HPhase1or2ReadThen-HInv4a2:
\[
\begin{align*}
&\text{HPhase1or2ReadThen } s \ s' \ p \ d \ r, \ HInv4a2 \ s \ q \ \Rightarrow \ HInv4a2 \ s' \ q \\
\end{align*}
\] 
\langle proof \rangle

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

lemma HPhase1or2ReadThen-HInv4a-q:
\begin{align*}
&\text{assumes act: HPhase1or2ReadThen } s \ s' \ p \ d \ r \\
&\text{and inv: } HInv4a \ s \ q \\
&\text{and p≠q } \\
&\text{shows } HInv4a \ s' \ q \\
\langle proof \rangle
\end{align*}

theorem HPhase1or2ReadThen-HInv4a:
\begin{align*}
&\text{assumes act: HPhase1or2ReadThen } s \ s' \ p \ d \ r, \ HInv4a \ s \ q; \ Inv2b \ s \\
&\text{shows } HInv4a \ s' \ q \\
\langle proof \rangle
\end{align*}

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

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

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

lemma HEndPhase1-HInv4a-p:
\begin{align*}
&\text{assumes act: HEndPhase1 } s \ s' \ p \\
&\text{and inv: } HInv4a \ s \ p \\
&\text{and inv2a: Inv2a } s \\
&\text{shows } HInv4a \ s' \ p \\
\langle proof \rangle
\end{align*}
lemma HEndPhase1-HInv4a-q:
  assumes act: HEndPhase1 s s' p
  and inv: HInv4a s q
  and pnq: p ≠ q
  shows HInv4a s' q
⟨proof⟩

theorem HEndPhase1-HInv4a:
\[
\begin{align*}
& [ \ HEndPhase1 \ s \ s' \ p; \ HInv4a \ s \ q; \ Inv2a \ s \ ] \\
\Rightarrow & HInv4a \ s' \ q
\end{align*}
\langle proof \rangle
\]

theorem HFail-HInv4a:
\[
\begin{align*}
& [ \ HFail \ s \ s' \ p; \ HInv4a \ s \ q \ ] \\
\Rightarrow & HInv4a \ s' \ q
\end{align*}
\langle proof \rangle
\]

theorem HPhase0Read-HInv4a:
\[
\begin{align*}
& [ \ HPhase0Read \ s \ s' \ p \ d; \ HInv4a \ s \ q \ ] \\
\Rightarrow & HInv4a \ s' \ q
\end{align*}
\langle proof \rangle
\]

theorem HEndPhase2-HInv4a:
\[
\begin{align*}
& [ \ HEndPhase2 \ s \ s' \ p; \ HInv4a \ s \ q \ ] \\
\Rightarrow & HInv4a \ s' \ q
\end{align*}
\langle proof \rangle
\]

lemma allSet:
  assumes aPQ: ∀ a. ∀ r ∈ P a. Q r and rb: rb ∈ P d
  shows Q rb
⟨proof⟩

lemma EndPhase0-44:
  assumes act: EndPhase0 s s' p
  and bk: bk ∈ blocksOf s p
  and inv4d: HInv4d s p
  and inv2c: Inv2c-inner s p
  shows ∃ d. ∃ rb ∈ blocksRead s p d. bal bk ≤ 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 (∀ d ∈ {d. hasRead s p d p}. disk s d p ∈ allBlocksRead s p)
⟨proof⟩
lemma HEndPhase0-41:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows \( \exists D \in \text{MajoritySet}. \forall d \in D. \ mbal(\text{disk } s d p) \leq mbal(\text{dblock } s' p) \)
\( \land bal(\text{disk } s d p) \leq bal(\text{dblock } s' p) \)
⟨proof⟩

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

lemma HEndPhase0-HInv4a2-p:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows HInv4a2 s' p
⟨proof⟩

lemma HEndPhase0-HInv4a-p:
assumes act: HEndPhase0 s s' p
and inv2a: Inv2a s
and inv2: Inv2c s
and inv4d: HInv4d s p
and inv1: Inv1 s
and inv: HInv4a s p
shows HInv4a s' p
⟨proof⟩

lemma HEndPhase0-HInv4a-q:
assumes act: HEndPhase0 s s' p
and inv: HInv4a s q
and pnq: p \neq q
shows HInv4a s' q
⟨proof⟩

theorem HEndPhase0-HInv4a:
\[ \begin{align*}
& \text{HEndPhase0 s s' p; HInv4a s q; HInv4d s p;}\\
& \text{Inv2a s; Inv1 s; Inv2a s; Inv2c s}\\
\implies & \text{HInv4a s' q}
\end{align*} \]
⟨proof⟩

C.4.2 Proofs of Invariant 4b

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

lemma $HEndPhase0$-HInv4b-p-dblock:
\[
\begin{align*}
& \text{assumes} & \text{act}: & HEndPhase0 \ s \ s' \ p \\
& & \text{and inv1}: & \text{Inv1} \ s \\
& & \text{and inv2a}: & \text{Inv2a} \ s \\
& & \text{and inv2c}: & \text{Inv2c-inner} \ s \ p \\
& \text{shows} & \text{bal}(\text{dblock} \ s' \ p) < \text{mbal}(\text{dblock} \ s' \ p)
\end{align*}
\]

lemma $HEndPhase0$-HInv4b-p-blocksOf:
\[
\begin{align*}
& \text{assumes} & \text{act}: & HEndPhase0 \ s \ s' \ p \\
& & \text{and inv4d}: & \text{HInv4d} \ s \ p \\
& & \text{and inv2c}: & \text{Inv2c-inner} \ s \ p \\
& & \text{and bk}: & bk \in \text{blocksOf} \ s \ p \\
& \text{shows} & \text{bal} \ bk < \text{mbal}(\text{dblock} \ s' \ p)
\end{align*}
\]

lemma $HEndPhase0$-HInv4b-p:
\[
\begin{align*}
& \text{assumes} & \text{act}: & HEndPhase0 \ s \ s' \ p \\
& & \text{and inv4d}: & \text{HInv4d} \ s \ p \\
& & \text{and inv1}: & \text{Inv1} \ s \\
& & \text{and inv2a}: & \text{Inv2a} \ s \\
& & \text{and inv2c}: & \text{Inv2c-inner} \ s \ p \\
& \text{shows} & \text{HInv4b} \ s' \ p
\end{align*}
\]

lemma $HEndPhase0$-HInv4b-q:
\[
\begin{align*}
& \text{assumes} & \text{act}: & HEndPhase0 \ s \ s' \ p \\
& & \text{and pnq}: & p \neq q \\
& & \text{and inv}: & \text{HInv4b} \ s \ q \\
& \text{shows} & \text{HInv4b} \ s' \ q
\end{align*}
\]

theorem $HEndPhase0$-HInv4b:
\[
\begin{align*}
& \text{assumes} & \text{act}: & HEndPhase0 \ s \ s' \ p \\
& & \text{and inv}: & \text{HInv4b} \ s \ q \\
& & \text{and inv4d}: & \text{HInv4d} \ s \ p \\
& & \text{and inv1}: & \text{Inv1} \ s \\
& & \text{and inv2a}: & \text{Inv2a} \ s \\
& & \text{and inv2c}: & \text{Inv2c-inner} \ s \ p \\
& \text{shows} & \text{HInv4b} \ s' \ q
\end{align*}
\]
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; Inv2a s; HInv4a s p] ]
⇒ HInv4b s' r

42
lemma $H$EndPhase1-$H$Inv4b-p:
$H$EndPhase1 $s$ $s'$ $p$ $\Longrightarrow$ $H$Inv4b $s'$ $p$
(proof)

lemma $H$EndPhase1-$H$Inv4b-q:
assumes act: $H$EndPhase1 $s$ $s'$ $p$
and $pq$: $p \neq q$
and inv: $H$Inv4b $s$ $q$
shows $H$Inv4b $s'$ $q$
(proof)

theorem $H$EndPhase1-$H$Inv4b:
assumes act: $H$EndPhase1 $s$ $s'$ $p$
and inv: $H$Inv4b $s$ $s'$ $q$
shows $H$Inv4b $s'$ $q$
(proof)

lemma $H$EndPhase2-$H$Inv4b-p:
$H$EndPhase2 $s$ $s'$ $p$ $\Longrightarrow$ $H$Inv4b $s'$ $p$
(proof)

lemma $H$EndPhase2-$H$Inv4b-q:
assumes act: $H$EndPhase2 $s$ $s'$ $p$
and $pq$: $p \neq q$
and inv: $H$Inv4b $s$ $q$
shows $H$Inv4b $s'$ $q$
(proof)

theorem $H$EndPhase2-$H$Inv4b:
assumes act: $H$EndPhase2 $s$ $s'$ $p$
and inv: $H$Inv4b $s$ $s'$ $q$
shows $H$Inv4b $s'$ $q$
(proof)

lemma $H$Fail-$H$Inv4b-p:
$H$Fail $s$ $s'$ $p$ $\Longrightarrow$ $H$Inv4b $s'$ $p$
(proof)

lemma $H$Fail-$H$Inv4b-q:
assumes act: $H$Fail $s$ $s'$ $p$
and $pq$: $p \neq q$
and inv: $H$Inv4b $s$ $q$
shows $H$Inv4b $s'$ $q$
(proof)

theorem $H$Fail-$H$Inv4b:
assumes act: $H$Fail $s$ $s'$ $p
and \( \text{inv: } HInv4b \ s \ q \)
shows \( HInv4b \ s' \ q \)

(proof)

lemma HPhase0Read-HInv4b-p:
\( HPhase0Read \ s \ s' \ p \ d \) \( \Rightarrow \) \( HInv4b \ s' \ p \)
(proof)

lemma HPhase0Read-HInv4b-q:
assumes act: \( HPhase0Read \ s \ s' \ p \ d \)
and \( pnq: p\neq q \)
and \( \text{inv: } HInv4b \ s \ q \)
shows \( HInv4b \ s' \ q \)
(proof)

theorem HPhase0Read-HInv4b:
assumes act: \( HPhase0Read \ s \ s' \ p \ d \)
and \( \text{inv: } HInv4b \ s \ q \)
shows \( HInv4b \ s' \ q \)
(proof)

C.4.3 Proofs of Invariant 4c

lemma HStartBallot-HInv4c-p:
\[ \left[ \begin{array}{l}
\text{HStartBallot } s \ s' \ p; \ HInv4c \ s \ p
\end{array} \right] \Rightarrow HInv4c \ s' \ p \]
(proof)

lemma HStartBallot-HInv4c-q:
assumes act: \( HStartBallot \ s \ s' \ p \)
and \( \text{inv: } HInv4c \ s \ q \)
and \( pnq: p\neq q \)
shows \( HInv4c \ s' \ q \)
(proof)

theorem HStartBallot-HInv4c:
\[ \left[ \begin{array}{l}
\text{HStartBallot } s \ s' \ p; \ HInv4c \ s \ q
\end{array} \right] \Rightarrow HInv4c \ s' \ q \]
(proof)

lemma HPhase1or2Write-HInv4c-p:
assumes act: \( HPhase1or2Write \ s \ s' \ p \ d \)
and \( \text{inv: } HInv4c \ s \ p \)
and \( \text{inv2c: } Inv2c \ s \)
shows \( HInv4c \ s' \ p \)
(proof)

lemma HPhase1or2Write-HInv4c-q:
assumes act: \( HPhase1or2Write \ s \ s' \ p \ d \)
and \( \text{inv: } HInv4c \ s \ q \)
and \( pnq: p\neq q \)
shows \( H_{Inv4c} s' q \)

\[ \text{proof} \]

**Theorem HPhase1or2Write-HInv4c:**
\[ [ H_{Phase1or2Write} s' p \; d; \; H_{Inv4c} s q; \; Inv2c s ] \implies H_{Inv4c} s' q \]

\[ \text{proof} \]

**Lemma HPhase1or2ReadThen-HInv4c-p:**
\[ [ H_{Phase1or2ReadThen} s' p \; d \; q; \; H_{Inv4c} s' p ] \implies H_{Inv4c} s' p \]

\[ \text{proof} \]

**Lemma HPhase1or2ReadThen-HInv4c-q:**
assumes act: \( H_{Phase1or2ReadThen} s s' p \; d \; r \)
and inv: \( H_{Inv4c} s q \)
and \( p \neq q \)
shows \( H_{Inv4c} s' q \)

\[ \text{proof} \]

**Theorem HPhase1or2ReadThen-HInv4c:**
\[ [ H_{Phase1or2ReadThen} s s' p \; d \; q; \; H_{Inv4c} s q ] \implies H_{Inv4c} s' q \]

\[ \text{proof} \]

**Theorem HPhase1or2ReadElse-HInv4c:**
\[ [ H_{Phase1or2ReadElse} s s' p \; d \; r; \; H_{Inv4c} s q ] \implies H_{Inv4c} s' q \]

\[ \text{proof} \]

**Lemma HEndPhase1-HInv4c-p:**
assumes act: \( H_{EndPhase1} s s' p \)
and inv2b: \( Inv2b s \)
shows \( H_{Inv4c} s' p \)

\[ \text{proof} \]

**Lemma HEndPhase1-HInv4c-q:**
assumes act: \( H_{EndPhase1} s s' p \)
and inv: \( H_{Inv4c} s q \)
and \( p \neq q \)
shows \( H_{Inv4c} s' q \)

\[ \text{proof} \]

**Theorem HEndPhase1-HInv4c:**
\[ [ H_{EndPhase1} s s' p; \; H_{Inv4c} s q; \; Inv2b s ] \implies H_{Inv4c} s' q \]

\[ \text{proof} \]

**Lemma HEndPhase2-HInv4c-p:**
\[ [ H_{EndPhase2} s s' p; \; H_{Inv4c} s p ] \implies H_{Inv4c} s' p \]

\[ \text{proof} \]
lemma \texttt{HEndPhase2-HInv4c-q}:
\begin{itemize}
\item \textbf{assumes} \texttt{act}: \texttt{HEndPhase2} \ s \ s' \ p
\item \texttt{and} \ \texttt{inv}: \texttt{HInv4c} \ s \ q
\item \texttt{and} \ \texttt{pq}: \texttt{p} \neq \texttt{q}
\item \textbf{shows} \texttt{HInv4c} \ s' \ q
\end{itemize}
\begin{proof}
\end{proof}

\begin{theorem}
\texttt{HEndPhase2-HInv4c}:
\begin{itemize}
\item \texttt{\[ \texttt{HEndPhase2} \ s \ s' \ p; \texttt{HInv4c} \ s \ q \] \implies \texttt{HInv4c} \ s' \ q}
\end{itemize}
\begin{proof}
\end{proof}
\end{theorem}

lemma \texttt{HFail-HInv4c-p}:
\begin{itemize}
\item \texttt{\[ \texttt{HFail} \ s \ s' \ p; \texttt{HInv4c} \ s \ p \] \implies \texttt{HInv4c} \ s' \ p}
\end{itemize}
\begin{proof}
\end{proof}

lemma \texttt{HFail-HInv4c-q}:
\begin{itemize}
\item \textbf{assumes} \texttt{act}: \texttt{HFail} \ s \ s' \ p
\item \texttt{and} \ \texttt{inv}: \texttt{HInv4c} \ s \ q
\item \texttt{and} \ \texttt{pq}: \texttt{p} \neq \texttt{q}
\item \textbf{shows} \texttt{HInv4c} \ s' \ q
\end{itemize}
\begin{proof}
\end{proof}

\begin{theorem}
\texttt{HFail-HInv4c}:
\begin{itemize}
\item \texttt{\[ \texttt{HFail} \ s \ s' \ p; \texttt{HInv4c} \ s \ q \] \implies \texttt{HInv4c} \ s' \ q}
\end{itemize}
\begin{proof}
\end{proof}
\end{theorem}

lemma \texttt{HPhase0Read-HInv4c-p}:
\begin{itemize}
\item \texttt{\[ \texttt{HPhase0Read} \ s \ s' \ p \ d; \texttt{HInv4c} \ s \ p \] \implies \texttt{HInv4c} \ s' \ p}
\end{itemize}
\begin{proof}
\end{proof}

lemma \texttt{HPhase0Read-HInv4c-q}:
\begin{itemize}
\item \textbf{assumes} \texttt{act}: \texttt{HPhase0Read} \ s \ s' \ p \ d
\item \texttt{and} \ \texttt{inv}: \texttt{HInv4c} \ s \ q
\item \texttt{and} \ \texttt{pq}: \texttt{p} \neq \texttt{q}
\item \textbf{shows} \texttt{HInv4c} \ s' \ q
\end{itemize}
\begin{proof}
\end{proof}

\begin{theorem}
\texttt{HPhase0Read-HInv4c}:
\begin{itemize}
\item \texttt{\[ \texttt{HPhase0Read} \ s \ s' \ p \ d; \texttt{HInv4c} \ s \ q \] \implies \texttt{HInv4c} \ s' \ q}
\end{itemize}
\begin{proof}
\end{proof}
\end{theorem}

lemma \texttt{HEndPhase0-HInv4c-p}:
\begin{itemize}
\item \texttt{\[ \texttt{HEndPhase0} \ s \ s' \ p; \texttt{HInv4c} \ s \ p \] \implies \texttt{HInv4c} \ s' \ p}
\end{itemize}
\begin{proof}
\end{proof}

lemma \texttt{HEndPhase0-HInv4c-q}:
\begin{itemize}
\item \textbf{assumes} \texttt{act}: \texttt{HEndPhase0} \ s \ s' \ p
\item \texttt{and} \ \texttt{inv}: \texttt{HInv4c} \ s \ q
\item \texttt{and} \ \texttt{pq}: \texttt{p} \neq \texttt{q}
\end{itemize}
\begin{proof}
\end{proof}

46
shows $H_{Inv4c} s' q$

⟨proof⟩

**Theorem HEndPhase0-HInv4c:**

\[ [H_{EndPhase0} s s' p; H_{Inv4c} s q] \implies H_{Inv4c} s' q \]

⟨proof⟩

### C.4.4 Proofs of Invariant 4d

**Lemma HStartBallot-HInv4d-p:**

- **Assumes** act: $H_{StartBallot} s s' p$
- **And** inv: $H_{Inv4d} s p$
- **Shows** $H_{Inv4d} s' p$

⟨proof⟩

**Lemma HStartBallot-HInv4d-q:**

- **Assumes** act: $H_{StartBallot} s s' p$
- **And** inv: $H_{Inv4d} s q$
- **And** $pnq: p \neq q$
- **Shows** $H_{Inv4d} s' q$

⟨proof⟩

**Theorem HStartBallot-HInv4d:**

\[ [H_{StartBallot} s s' p; H_{Inv4d} s q] \implies H_{Inv4d} s' q \]

⟨proof⟩

**Lemma HPhase1or2Write-HInv4d-p:**

- **Assumes** act: $H_{Phase1or2Write} s s' p d$
- **And** inv: $H_{Inv4d} s p$
- **And** $inv4a: H_{Inv4a} s p$
- **Shows** $H_{Inv4d} s' p$

⟨proof⟩

**Lemma HPhase1or2Write-HInv4d-q:**

- **Assumes** act: $H_{Phase1or2Write} s s' p d$
- **And** inv: $H_{Inv4d} s q$
- **And** $pnq: p \neq q$
- **Shows** $H_{Inv4d} s' q$

⟨proof⟩

**Theorem HPhase1or2Write-HInv4d:**

\[ [H_{Phase1or2Write} s s' p d; H_{Inv4d} s q; H_{Inv4a} s p] \implies H_{Inv4d} s' q \]

⟨proof⟩

**Lemma HPhase1or2ReadThen-HInv4d-p:**

- **Assumes** act: $H_{Phase1or2ReadThen} s s' p d q$
- **And** inv: $H_{Inv4d} s p$
- **Shows** $H_{Inv4d} s' p$

⟨proof⟩
lemma HPhase1or2ReadThen-HInv4d-q:
  assumes act: HPhase1or2ReadThen s s' p d r
  and inv: HInv4d s q
  and pnq: p≠q
  shows HInv4d s' q
⟨proof⟩

theorem HPhase1or2ReadThen-HInv4d:
  ⟦ HPhase1or2ReadThen s s' p d r; HInv4d s q ⟧ ⇒ HInv4d s' q
⟨proof⟩

theorem HPhase1or2ReadElse-HInv4d:
  ⟦ HPhase1or2ReadElse s s' p d r; HInv4d s q ⟧ ⇒ HInv4d s' q
⟨proof⟩

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

lemma HEndPhase1-HInv4d-q:
  assumes act: HEndPhase1 s s' p
  and inv: HInv4d s q
  and pnq: p≠q
  shows HInv4d s' q
⟨proof⟩

theorem HEndPhase1-HInv4d:
  ⟦ HEndPhase1 s s' p; HInv4d s q; Inv2b s; HInv4c s p ⟧ ⇒ HInv4d s' q
⟨proof⟩

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

lemma HEndPhase2-HInv4d-q:
  assumes act: HEndPhase2 s s' p
  and inv: HInv4d s q
  and pnq: p≠q
  shows HInv4d s' q
⟨proof⟩
\textbf{theorem} \textit{HEndPhase2-HInv4d}: \\
\[ \text{[ } \text{HEndPhase2 } s \ s' \ p; \ \text{HInv4d } s \ q \text{]} \implies \text{HInv4d } s' \ q \]
\langle \text{proof} \rangle

\textbf{lemma} \textit{HFail-HInv4d-p}: \\
\textbf{assumes} act: \text{HFail } s \ s' \ p \\
\textbf{and} inv: \text{HInv4d } s \ p \\
\textbf{shows} \text{HInv4d } s' \ p \\
\langle \text{proof} \rangle

\textbf{lemma} \textit{HFail-HInv4d-q}: \\
\textbf{assumes} act: \text{HFail } s \ s' \ p \\
\textbf{and} inv: \text{HInv4d } s \ q \\
\textbf{and} \text{pnq: } p \neq q \\
\textbf{shows} \text{HInv4d } s' \ q \\
\langle \text{proof} \rangle

\textbf{theorem} \textit{HFail-HInv4d}: \\
\[ \text{[ } \text{HFail } s \ s' \ p; \ \text{HInv4d } s \ q \text{]} \implies \text{HInv4d } s' \ q \]
\langle \text{proof} \rangle

\textbf{lemma} \textit{HPhase0Read-HInv4d-p}: \\
\textbf{assumes} act: \text{HPhase0Read } s \ s' \ p \ d \\
\textbf{and} inv: \text{HInv4d } s \ p \\
\textbf{shows} \text{HInv4d } s' \ p \\
\langle \text{proof} \rangle

\textbf{lemma} \textit{HPhase0Read-HInv4d-q}: \\
\textbf{assumes} act: \text{HPhase0Read } s \ s' \ p \ d \\
\textbf{and} inv: \text{HInv4d } s \ q \\
\textbf{and} \text{pnq: } p \neq q \\
\textbf{shows} \text{HInv4d } s' \ q \\
\langle \text{proof} \rangle

\textbf{theorem} \textit{HPhase0Read-HInv4d}: \\
\[ \text{[ } \text{HPhase0Read } s \ s' \ p \ d; \ \text{HInv4d } s \ q \text{]} \implies \text{HInv4d } s' \ q \]
\langle \text{proof} \rangle

\textbf{lemma} \textit{HEndPhase0-blocksOf2}: \\
\textbf{assumes} act: \text{HEndPhase0 } s \ s' \ p \\
\textbf{and} inv2c: \text{Inv2c-inner } s \ p \\
\textbf{shows} \text{allBlocksRead } s \ p \subseteq \text{blocksOf } s \ p \\
\langle \text{proof} \rangle

\textbf{lemma} \textit{HEndPhase0-HInv4d-p}: \\
\textbf{assumes} act: \text{HEndPhase0 } s \ s' \ p \\
\textbf{and} inv: \text{HInv4d } s \ p \\
\textbf{and} inv2c: \text{Inv2c } s \\
\textbf{and} inv1: \text{Inv1 } s
shows \( H_{\text{Inv}4d} \ s \ p \)

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

**lemma** \( H_{\text{EndPhase0}}-H_{\text{Inv}4d}-q \):

**assumes** act: \( H_{\text{EndPhase0}} \ s \ s' \ p \)

**and** inv: \( H_{\text{Inv}4d} \ s \ q \)

**and** pq: \( p \neq q \)

**shows** \( H_{\text{Inv}4d} \ s' \ q \)

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

**theorem** \( H_{\text{EndPhase0}}-H_{\text{Inv}4d} \):

\( \begin{align*}
& [ \ H_{\text{EndPhase0}} \ s \ s' \ p; \\
& H_{\text{Inv}4d} \ s \ q; \\
& \text{Inv2c} \ s; \text{Inv1} \ s ] \implies H_{\text{Inv}4d} \ s' \ q 
\end{align*} \)

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

Since we have already proved \( H_{\text{Inv}2} \) is an invariant of \( H_{\text{Next}} \), \( H_{\text{Inv}1} \land H_{\text{Inv}2} \land H_{\text{Inv}4} \) is also an invariant of \( H_{\text{Next}} \).

**lemma** \( I2d \):

**assumes** act: \( H_{\text{Next}} \ s \ s' \)

**and** inv: \( H_{\text{Inv}1} \ s \land H_{\text{Inv}2} \ s \land H_{\text{Inv}2} \ s' \land H_{\text{Inv}4} \ s \)

**shows** \( H_{\text{Inv}4} \ s' \)

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

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

**definition** \( \text{maxBalInp} :: \text{state} \Rightarrow \text{nat} \Rightarrow \text{InputsOrNi} \Rightarrow \text{bool} \)

**where** \( \text{maxBalInp} \ s \ b \ v = (\forall bk \in \text{allBlocks} \ s. \ b \leq \text{bal}(bk) \implies \text{inp}(bk) = v) \)

**definition** \( H_{\text{Inv}5-inner-R} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

**where**

\( H_{\text{Inv}5-inner-R} \ s \ p = \\
(\text{maxBalInp} \ s \ (\text{bal} \text{(dblocksp)}) \ (\text{inp} \text{(dblocksp)}) ) \\
\lor (\exists D \in \text{MajoritySet}. \exists q. \forall d \in D. \text{bal}(\text{dblocksp}) < \text{mbal}(\text{disksp}) \land \neg \text{hasRead} \ s \ p \ d \ q) )\)

**definition** \( H_{\text{Inv}5-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

**where** \( H_{\text{Inv}5-inner} \ s \ p = (\text{phase} \ s \ p = 2 \implies H_{\text{Inv}5-inner-R} \ s \ p) \)
definition HInv5 :: state ⇒ bool
  where HInv5 s = (∀ p. HInv5-inner s p)

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

theorem HInit-HInv5: HInit s ⇒ HInv5 s
  ⟨proof⟩

We will use the notation used in the proofs of invariant 4, and prove the lemma action-HInv5-p and action-HInv5-q for each action, for the cases p = q and p ≠ q respectively.

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

lemma HStartBallot-HInv5-p:
  assumes act: HStartBallot s s' p
  and inv: HInv5-inner s p
  shows HInv5-inner s' p ⟨proof⟩

lemma HStartBallot-blocksOf-q:
  assumes act: HStartBallot s s' p
  and p≠q
  shows blocksOf s' q ⊆ blocksOf s q ⟨proof⟩

lemma HStartBallot-allBlocks:
  assumes act: HStartBallot s s' p
  shows allBlocks s' ⊆ allBlocks s ∪ {dblock s' p} ⟨proof⟩

lemma HStartBallot-HInv5-q1:
  assumes act: HStartBallot s s' p
  and p≠q
  and inv5-1: maxBalImp s (bal(dblock s q)) (inp(dblock s q))
  shows maxBalImp s' (bal(dblock s' q)) (inp(dblock s' q)) ⟨proof⟩

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

52
shows $\exists D \in \text{MajoritySet} \cdot \exists q. \ (\forall d \in D. \ \text{bal}(d\text{block } s' q) < m\text{bal}(d\text{isk } s' d q) \\
\wedge \neg \text{hasRead } s' q d q)$

(\text{proof})

\text{lemma } H\text{Phase1or2Write-HInv5-q:}
\text{assumes act: } H\text{Phase1or2Write } s s' p d 
\text{and inv: } H\text{Inv5-inner } s q 
\text{and inv4a: } H\text{Inv4a } s p 
\text{and pq: } p \neq q 
\text{shows } H\text{Inv5-inner } s' q 
(\text{proof})

\text{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 
(\text{proof})

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

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

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

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

\text{lemma } H\text{Phase1or2ReadThen-HInv5-q2:}
\text{assumes act: } H\text{Phase1or2ReadThen } s s' p d r
and \( pq \): \( p \neq q \)
and \( inv4a: HInv4a s p \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(dblock s q) < mbal(disk s d qq)) \wedge \neg \text{hasRead} s q d qq) \)
shows \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(dblock s' q) < mbal(disk s' d qq)) \wedge \neg \text{hasRead} s' q d qq) \)

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

**lemma** \( HPhase1or2ReadThen-HInv5-q \):
assumes act: \( HPhase1or2ReadThen s s' p d r \)
and inv: \( HInv5-inner s q \)
and inv4a: \( HInv4a s p \)
and \( pq: p \neq q \)
shows \( HInv5-inner s' q \)

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

**theorem** \( HPhase1or2ReadThen-HInv5 \):

\[ \begin{align*} & HPhase1or2ReadThen s s' p d r; HInv5-inner s q; \quad \text{inv2c} s; HInv4c s p; HInv4a s p \quad \Rightarrow \quad HInv5-inner s' q \end{align*} \]

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

**theorem** \( HPhase1or2ReadElse-HInv5 \):

\[ \begin{align*} & HPhase1or2ReadElse s s' p d r; HInv5-inner s q \quad \Rightarrow \quad HInv5-inner s' q \end{align*} \]

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

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

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

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

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

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

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

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

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

(proof)

**lemma** $H\text{EndPhase2-HInv5-q}$:
- **assumes** act: $H\text{EndPhase2 } s \ s' \ p$
- and inv: $H\text{inv5-inner } s \ q$
- and pnq: $p \neq q$
- **shows** $H\text{inv5-inner } s' \ q$

(proof)

**theorem** $H\text{EndPhase2-HInv5}$:
- $[ H\text{EndPhase2 } s \ s' \ p; H\text{inv5-inner } s \ q ] \implies H\text{inv5-inner } s' \ q$

(proof)

**lemma** $H\text{EndPhase1-HInv5-p}$:
- **assumes** act: $H\text{EndPhase1 } s \ s' \ p$
- and inv4: $H\text{inv4 } s$
- and inv2a: $H\text{inv2a } s$
- and inv2a\': $H\text{inv2a } s'$
- and inv2c: $H\text{inv2c } s$
- and asm4: $\neg \text{maxBalInp } s' \ (\text{bal}(\text{dblock } s' \ p)) \ (\text{inp}(\text{dblock } s' \ p))$
- **shows** $(\exists D \in \text{MajoritySet. } \exists q. \left( \forall d \in D. \text{ bal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s' \ d \ q) \right) \\
\land \neg \text{hasRead } s' \ p \ d \ q))$

(proof)

**lemma** union-inclusion:
- $[ A \subseteq A'; B \subseteq B' ] \implies A \cup B \subseteq A' \cup B'$

(proof)

**lemma** $H\text{EndPhase1-blocksOf-q}$:
- **assumes** act: $H\text{EndPhase1 } s \ s' \ p$
- and pnq: $p \neq q$
- **shows** $\text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q$

(proof)

**lemma** $H\text{EndPhase1-allBlocks}$:
- **assumes** act: $H\text{EndPhase1 } s \ s' \ p$
- **shows** $\text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ \text{dblock } s' \ p \}$

(proof)

**lemma** $H\text{EndPhase1-HInv5-q}$:
- **assumes** act: $H\text{EndPhase1 } s \ s' \ p$
- and inv: $H\text{inv5 } s$
- and inv1: $H\text{inv1 } s$
- and inv2a: $H\text{inv2a } s$
- and inv2a-q: $H\text{inv2a } s'$
- and inv2b: $H\text{inv2b } s$
- and inv2c: $H\text{inv2c } s$

55
and \( \text{inv3: } H\text{Inv3 } s \)
and \( \text{phase': phase } s' q = 2 \)
and \( \text{pnq: } p \neq q \)
and \( \text{asm4: } \neg \text{maxBalInp } s' (\text{bal}(\text{dblock } s' q)) (\text{inp}(\text{dblock } s' q)) \)
shows \( (\exists D \in \text{MajoritySet. } \exists qq. (\forall d \in D. \text{ bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d qq) \land \neg \text{hasRead } s' q d qq)) \)

\text{(proof)}

\text{theorem } \text{HEndPhase1-HInv5:}
\text{assumes act: } \text{HEndPhase1 } s s' p
\text{and inv: } H\text{Inv5 } s
\text{and inv1: Inv1 } s
\text{and inv2a: Inv2a } s
\text{and inv2a': Inv2a } s'
\text{and inv2b: Inv2b } s
\text{and inv2c: Inv2c } s
\text{and inv3: HInv3 } s
\text{and inv4: HInv4 } s
shows \( H\text{Inv5-inner } s' q \)
\text{(proof)}

\text{lemma } \text{HFail-HInv5-p:}
\text{HFail } s s' p \implies H\text{Inv5-inner } s' p
\text{(proof)}

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

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

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

\text{lemma } \text{HFail-HInv5-q2:}
\text{assumes act: } \text{HFail } s s' p
\text{and pnq: } p \neq q
\text{and 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))

56
\[ \exists D \in \text{MajoritySet. } \exists \text{qq. } (\forall d \in D. \text{ bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d \text{ qq}) \]
\[ \wedge \neg \text{hasRead } s' q d \text{ qq) } \]

\text{lemma HFail-HInv5-q:}
\text{assumes act: HFail } s s' p 
\text{and inv: HInv5-inner } s q 
\text{and p=q: p\neq q}
\text{and inv2: Inv2a } s' 
\text{shows HInv5-inner } s' q 
\langle \text{proof} \rangle

\text{theorem HFail-HInv5:}
\[ [ \text{ HFail } s s' p; \text{ HInv5-inner } s q; \text{ Inv2a } s' ] \implies \text{ HInv5-inner } s' q \]
\langle \text{proof} \rangle

\text{lemma HPhase0Read-HInv5-p:}
\text{HPhase0Read } s s' p d \implies \text{ 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 \text{allBlocks } s 
\langle \text{proof} \rangle

\text{lemma HPhase0Read-HInv5-1:}
\text{assumes act: HPhase0Read } s s' p d 
\text{and inv5-1: maxBalInp } s (\text{bal}(\text{dblock } s q)) (\text{inp}(\text{dblock } s q)) 
\text{shows 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 p=q: p\neq q}
\text{and inv5-2: } \exists D \in \text{MajoritySet. } \exists \text{qq. } (\forall d \in D. \text{ bal}(\text{dblock } s q) < \text{mbal}(\text{disk } s d \text{ qq}) 
\wedge \neg \text{hasRead } s q d \text{ qq) } 
\text{shows } \exists D \in \text{MajoritySet. } \exists \text{qq. } (\forall d \in D. \text{ bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d \text{ qq}) 
\wedge \neg \text{hasRead } s' q d \text{ 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 p=q: p\neq q}
\text{shows HInv5-inner } s' q 
\langle \text{proof} \rangle
theorem HPhase0Read-HInv5:
\[ \boxed{\text{HPhase0Read } s \ s' \ p \ d; \text{HInv5-inner } s \ q} \implies \text{HInv5-inner } s' \ q \]
⟨proof⟩

lemma HEndPhase0-HInv5-p:
\[ \text{HEndPhase0 } s \ s' \ p \implies \text{HInv5-inner } s' \ p \]
⟨proof⟩

lemma HEndPhase0-blocksOf-q:
\[ \text{assumes act: HEndPhase0 } s \ s' \ p \]
\[ \text{and pnq: } p \neq q \]
\[ \text{shows blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \]
⟨proof⟩

lemma HEndPhase0-allBlocks:
\[ \text{assumes act: HEndPhase0 } s \ s' \ p \]
\[ \text{shows allBlocks } s' \subseteq \text{allBlocks } s \cup \{\text{dblock } s' \ p\} \]
⟨proof⟩

lemma HEndPhase0-HInv5-q1:
\[ \text{assumes act: HEndPhase0 } s \ s' \ p \]
\[ \text{and pnq: } p \neq q \]
\[ \text{and inv1: } \text{Inv1 } s \]
\[ \text{and inv5-1: } \maxBalInp \ s \ (\text{bal} (\text{dblock } s \ q)) \ (\text{inp} (\text{dblock } s \ q)) \]
\[ \text{shows } \maxBalInp \ s' \ (\text{bal} (\text{dblock } s' \ q)) \ (\text{inp} (\text{dblock } s' \ q)) \]
⟨proof⟩

lemma HEndPhase0-HInv5-q2:
\[ \text{assumes act: HEndPhase0 } s \ s' \ p \]
\[ \text{and pnq: } p \neq q \]
\[ \text{and inv5-2: } \exists D \in \text{MajoritySet. } \exists qq. \ (\forall d \in D. \ \text{bal} (\text{dblock } s \ q)) < \text{mbal} (\text{disk } s \ d qq) \]
\[ \quad \land \neg \text{hasRead } s \ d \ qq \]
\[ \text{shows } \exists D \in \text{MajoritySet. } \exists qq. \ (\forall d \in D. \ \text{bal} (\text{dblock } s' \ q)) < \text{mbal} (\text{disk } s' \ d qq) \]
\[ \quad \land \neg \text{hasRead } s' \ d \ qq \]
⟨proof⟩

lemma HEndPhase0-HInv5-q:
\[ \text{assumes act: HEndPhase0 } s \ s' \ p \]
\[ \text{and inv: HInv5-inner } s \ q \]
\[ \text{and inv1: } \text{Inv1 } s \]
\[ \text{and pnq: } p \neq q \]
\[ \text{shows HInv5-inner } s' \ q \]
⟨proof⟩

theorem HEndPhase0-HInv5:
\[ \boxed{\text{HEndPhase0 } s \ s' \ p; \text{HInv5-inner } s \ q; \text{Inv1 } s} \implies \text{HInv5-inner } s' \ q \]
⟨proof⟩
HInv1 ∧ HInv2 ∧ HInv3 ∧ HInv4 ∧ HInv5 is an invariant of HNext.

lemma I2c:
assumes nxt: HNext s s'
and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s
shows HInv5 s'
(proof)
end

theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

C.6 Lemma I2f

To prove the final conjunct we will use the predicate \textit{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 \( \textit{disk}_s d \), then it has read a block with \textit{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 maxBalInp 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-maxBalInp:
assumes act: HStartBallot s s' q
and asm3: maxBalInp s b v
shows maxBalInp s' b v

-proof-

lemma HStartBallot-valueChosen2:
assumes act: HStartBallot s s' q
and asm4: ∀ 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
\[
\text{and } v_c: \text{valueChosen } s \ v \\
\text{and } v\text{-input: } v \in \text{Inputs} \\
\text{shows } \text{valueChosen } s' \ v \\
\langle \text{proof} \rangle
\]

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

**Lemma** \(H\text{Phase1or2Write-valueChosen2}:\)
\[
\text{assumes act: } H\text{Phase1or2Write } s \ s' \ pp \ d \\
\text{and } \text{asm2: } D \in \text{MajoritySet} \\
\text{and } \text{asm4: } \forall d \in D. \ b \leq \text{bal(disk } s \ d \ p) \\
\quad \wedge (\forall q.( \text{phase } s \ q = 1 \\
\quad \wedge b \leq \text{mbal(dblock } s \ q) \\
\quad \wedge \text{hasRead } s \ q \ d \ p)) \\
\text{and } \text{inv4: } H\text{inv4a } s \ pp \\
\text{shows } ?P \ s' \\
\langle \text{proof} \rangle
\]

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

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

**Lemma** \(H\text{Phase1or2ReadThen-valueChosen2}:\)
\[
\text{assumes act: } H\text{Phase1or2ReadThen } s \ s' \ q \ d \ pp \\
\text{and } \text{asm4: } \forall d \in D. \ b \leq \text{bal(disk } s \ d \ p) \\
\quad \wedge (\forall q.( \text{phase } s \ q = 1 \\
\quad \wedge b \leq \text{mbal(dblock } s \ q) \\
\quad \wedge \text{hasRead } s \ q \ d \ p)) \\
\text{shows } ?P \ s' \\
\langle \text{proof} \rangle
\]

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

theorem HPhase1or2ReadElse-valueChosen:
[ HPhase1or2ReadElse s s’ p d r; valueChosen s v; v∈ Inputs ]
⇒ valueChosen s’ v
⟨proof⟩

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

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

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

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

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

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

\textbf{lemma} \textit{HPhase0Read-maxBalInp}:  
\textbf{assumes} \textit{act}: \textit{HPhase0Read} \ s \ s' \ q \ d  
\textbf{and} \textit{asm3}: \textit{maxBalInp} \ s \ b \ v  
\textbf{shows} \textit{maxBalInp} \ s' \ b \ v  
(\textit{proof})

\textbf{lemma} \textit{HPhase0Read-valueChosen2}:  
\textbf{assumes} \textit{act}: \textit{HPhase0Read} \ s \ s' \ qq dd  
\textbf{and} \textit{asm4}: \ \forall \ d \in D. \ b \leq \textit{bal}(\textit{disk} \ s \ d \ p)  
\land (\forall \ q. (\  \textit{phase} \ s \ q = 1  
\land \ b \leq \textit{mbal}(\textit{dblock} \ s \ q)  
\land \textit{hasRead} \ s \ q \ d \ p  
) \rightarrow (\exists \ br \in \textit{blocksRead} \ s \ q \ d. \ b \leq \textit{bal}(\textit{block} \ br))) \ (\textit{is} \ \textit{?P} \ s)  
\textbf{shows} \ ?P \ s'  
(\textit{proof})

\textbf{theorem} \textit{HPhase0Read-valueChosen}:  
\textbf{assumes} \textit{act}: \textit{HPhase0Read} \ s \ s' \ q \ d  
\textbf{and} \textit{vc}: \textit{valueChosen} \ s \ v  
\textbf{and} \textit{v-input}: \ v \in \textit{Inputs}  
\textbf{shows} \textit{valueChosen} \ s' \ v  
(\textit{proof})

\textbf{lemma} \textit{HEndPhase0-maxBalInp}:  
\textbf{assumes} \textit{act}: \textit{HEndPhase0} \ s \ s' \ q  
\textbf{and} \textit{asm3}: \textit{maxBalInp} \ s \ b \ v  
\textbf{and} \textit{inv1}: \textit{Inv1} \ s  
\textbf{shows} \textit{maxBalInp} \ s' \ b \ v  
(\textit{proof})

\textbf{lemma} \textit{HEndPhase0-valueChosen2}:  
\textbf{assumes} \textit{act}: \textit{HEndPhase0} \ s \ s' \ q  
\textbf{and} \textit{asm4}: \ \forall \ d \in D. \ b \leq \textit{bal}(\textit{disk} \ s \ d \ p)  
\land (\forall \ q. (\  \textit{phase} \ s \ q = 1  
\land \ b \leq \textit{mbal}(\textit{dblock} \ s \ q)  
\land \textit{hasRead} \ s \ q \ d \ p  
) \rightarrow (\exists \ br \in \textit{blocksRead} \ s \ q \ d. \ b \leq \textit{bal}(\textit{block} \ br))) \ (\textit{is} \ \textit{?P} \ s)  
\textbf{shows} \ ?P \ s'  
(\textit{proof})

63
theorem HEndPhase0-valueChosen:
  assumes act: HEndPhase0 s s' q
  and vc: valueChosen s v
  and v-input: v ∈ Inputs
  and inv1: Inv1 s
  shows valueChosen s' v
⟨proof⟩

end

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

definition HInv6 :: state ⇒ bool
  where
  \[ HInv6 s = (\text{chosen s} \neq \text{NotAnInput} \rightarrow \text{valueChosen s} (\text{chosen s})) \]
  ∧ (∀ p. outpt s p ∈ \{\text{chosen s}, \text{NotAnInput}\})

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

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

lemma valueChosen-equal-case:
  assumes max-v: maxBalInp s b v
  and Dmaj: D ∈ MajoritySet
  and asm-v: \( \forall d \in D. \ b \leq bal (\text{disk s d p}) \)
  and max-w: maxBalInp s ba w
  and Danaj: Da ∈ MajoritySet
  and asm-w: \( \forall d \in Da. \ ba \leq bal (\text{disk s d pa}) \)
  and b-ba: b≤ba
  shows v=w
⟨proof⟩

64
lemma valueChosen-equal:
  assumes v: valueChosen s v
  and w: valueChosen s w
  shows v = w \langle \text{proof} \rangle

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 \neq NotAnInput
  shows outpt s' r = chosen s' \langle \text{proof} \rangle

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' \langle \text{proof} \rangle

lemma outpt-chosen:
  assumes outpt: outpt s = outpt s'
  and inv2c: Inv2c s
  and nextp: HNextPart s s'
  shows chosen s' = chosen s \langle \text{proof} \rangle

lemma outpt-Inv6:
  [ outpt s = outpt s'; \forall p. outpt s p \in \{chosen s, NotAnInput\};
    Inv2c s; HNextPart s s' ] \implies \forall p. outpt s' p \in \{chosen s', NotAnInput\}
  \langle \text{proof} \rangle

theorem HStartBallot-Inv6:
  assumes act: HStartBallot s s' p
  and inv: HInv6 s
  and inv2c: Inv2c s
  shows HInv6 s' \langle \text{proof} \rangle

theorem HPhase1or2Write-Inv6:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv6 s
  and inv4: HInv4a s p
\begin{proof}
The theorem HPhase1or2ReadThen-Inv6:
\begin{enumerate}
\item Assumptions: act: HPhase1or2ReadThen s s' p d q
\item Invs: HInv6 s
\item inv2c: Inv2c s
\end{enumerate}
shows HInv6 s'
\end{proof}

\begin{proof}
The theorem HPhase1or2ReadElse-Inv6:
\begin{enumerate}
\item Assumptions: act: HPhase1or2ReadElse s s' p d q
\item Invs: HInv6 s
\item inv2c: Inv2c s
\end{enumerate}
shows HInv6 s'
\end{proof}

\begin{proof}
The theorem HEndPhase1-Inv6:
\begin{enumerate}
\item Assumptions: act: HEndPhase1 s s' p
\item Invs: HInv6 s
\item inv1: Inv1 s
\item inv2a: Inv2a s
\item inv2b: Inv2b s
\item inv2c: Inv2c s
\end{enumerate}
shows HInv6 s'
\end{proof}

\begin{proof}
The lemma outpt-chosen-2:
\begin{enumerate}
\item Assumptions: outpt: outpt s' = (outpt s) (p:= NotAnInput)
\item Invs: inv2c: Inv2c s
\item nextp: HNextPart s s'
\end{enumerate}
shows chosen s = chosen s'
\end{proof}

\begin{proof}
The lemma outpt-HInv6-2:
\begin{enumerate}
\item Assumptions: outpt: outpt s' = (outpt s) (p:= NotAnInput)
\item Invs: \forall p. outpt s p \in \{ chosen s, NotAnInput \}
\item inv2c: Inv2c s
\item nextp: HNextPart s s'
\end{enumerate}
shows \forall p. outpt s' p \in \{ chosen s', NotAnInput \}
\end{proof}

\begin{proof}
The theorem HFail-Inv6:
\begin{enumerate}
\item Assumptions: act: HFail s s' p
\item Invs: HInv6 s
\item inv2c: Inv2c s
\end{enumerate}
shows HInv6 s'
\end{proof}
\begin{verbatim}
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⟩

end
\end{verbatim}
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

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

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

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

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

definition s2is :: state ⇒ Istate
where
s2is s = (iinput = inpt s,
 ioutput = outpt s,
 ichosen=chosen s,
 iallInput = allInput s)
**Theorem R1:**
\[
\left[ \mathit{HInit} \ s; \ is = s \mathit{2is} \ s \right] \implies \mathit{HInit} \ is
\]
\textit{(proof)}

**Theorem R2b:**
- **Assumes:** \( \mathit{inv} : \mathit{HInv} \ s \)
- **And:** \( \mathit{inv'} : \mathit{HInv} \ s' \)
- **And:** \( \mathit{nxt} : \mathit{HNext} \ s \ s' \)
- **And:** \( \mathit{srel} : is = s \mathit{2is} \ s \land is' = s \mathit{2is} \ s' \)
- **Shows:** \( \exists p. \ \mathit{IFail} \ is \ is' \ p \lor \mathit{IChoose} \ is \ is' \ p \lor is = is' \)
\textit{(proof)}

**end**