Abstract

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

Contents

1 Introduction 2

2 The Disk Paxos Algorithm 3
   2.1 Informal description of the algorithm. . . . . . . . . . . . . 4
   2.2 Disk Paxos and its TLA+ Specification . . . . . . . . . . . . . 4

3 Translating from TLA+ to Isabelle/HOL 6
   3.1 Typed vs. Untyped . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   3.2 Primed Variables . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   3.3 Restructuring the specification . . . . . . . . . . . . . . . . . . . 8

4 Structure of the Correctness Proof 9
   4.1 Going from Informal Proofs to Formal Proofs . . . . . . . . . 10

5 Conclusion 11

A TLA+ correctness specification 12

B Disk Paxos Algorithm Specification 13
1 Introduction

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

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

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

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

2 The Disk Paxos Algorithm

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

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

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

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

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

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

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

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

In each block, processors maintain three values:

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

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

2.2 Disk Paxos and its TLA$^+$ Specification

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

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

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

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

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

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

Analogously, we have the specification of the Inner module:

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

We define \text{ivars} = \langle \text{input}, \text{output}, \text{chosen}, \text{allInput} \rangle. 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 \quad \text{HInit} \Rightarrow \text{IInit} \\
THEOREM R2 \quad \text{HInit} \land \square[\text{HNext}]_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \square[\text{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 \text{HInv} for which we can prove:

THEOREM R2a \quad \text{HInit} \land \square[\text{HNext}]_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \square[\text{HInv}] \\
THEOREM R2b \quad \text{HInv} \land \text{HInv'} \land \text{HNext} \Rightarrow \text{INext} \lor \text{UNCHANGED ivars}

A predicate satisfying \text{HInv} is said to be an invariant of HDiskSynodSpec. To prove R2a, we make \text{HInv} strong enough to satisfy:
Again, we have TLA proof rules that say that I1 and I2 imply R2a. In summary, R2b, I1, and I2 together imply HDiskSynodSpec ⇒ 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 HInv1,...,HInv6, where HInv1 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 ≤ 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.

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

<table>
<thead>
<tr>
<th>TLA+</th>
<th>Isabelle/HOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>∃ d ∈ D : disk[d][q].bal = bk</td>
<td>∃ d ∈ D. bal(disk s d q) = bk</td>
</tr>
<tr>
<td>choose x. P x</td>
<td></td>
</tr>
<tr>
<td>phase′ = [phase except ![p] = 1]</td>
<td>phase s′ = (phase s)(p := 1)</td>
</tr>
<tr>
<td>UNION {blocksOf(p) : p ∈ Proc}</td>
<td>UN p. blocksOf s p</td>
</tr>
<tr>
<td>UNCHANGED v</td>
<td>v s′ = v s</td>
</tr>
</tbody>
</table>

THEOREM I1 HInit ⇒ HInv

THEOREM I2 HInv ∧ HNext ⇒ HInv′

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 choices1.

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.

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

\[
\begin{align*}
\text{CONSTANT } & \quad \text{Inputs} \\
\text{NotAnInput} & \quad \triangleq \text{CHOOSE } c : c \notin \text{Inputs} \\
\text{DiskBlock} & \quad \triangleq \left[ \begin{array}{l}
\text{mbal} : (\cup \text{Ballot}(p) : p \in \text{Proc}) \cup \{0\}, \\
\text{bal} : (\cup \text{Ballot}(p) : p \in \text{Proc}) \cup \{0\}, \\
\text{inp} : \text{Inputs} \cup \{\text{NotAnInput}\}\end{array} \right]
\end{align*}
\]

Isabelle/HOL:

\text{typedef} \quad \text{InputsOrNi}

\text{consts}

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

\text{axioms}

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

\text{record}

\begin{align*}
\text{DiskBlock} = \\
\text{mbal} & \quad : \text{nat} \\
\text{bal} & \quad : \text{nat} \\
\text{inp} & \quad : \text{InputsOrNi}
\end{align*}

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

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

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

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

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

Isabelle/HOL:

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

Figure 3: Translation of an action

3.2 Primed Variables

In TLA⁺, to denote the value of a variable in the state resulting from an action, there exists the concept of primed variables. In Isabelle/HOL, there is no built-in notion of state, so we define the state as a record of the variables involved. Instead of defining a “priming” operator, we just define actions as predicates that take any two states, intended to be the previous state and the next state. In this way, \( P ss' \) 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 \texttt{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 \texttt{Let-def} to Isabelle’s simplifier, which unfolds all “let” constructs.

8
Another change that makes proofs easier is to break down actions into simpler actions (e.g. giving separate definitions for subactions corresponding to disjuncts). By making actions smaller, Isabelle has to deal with smaller formulas when we feed the prover with such an action.

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

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

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

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

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

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

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

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

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 $EndPhase1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv_4$ and $HInv_5$ hold in the previous state to prove lemma $I2f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate $I$ was an invariant of $Next$, we preferred proving the invariance of $I$ for each action, rather than a big theorem proving the invariance of $I$ for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $HInv_3$ for the $EndPhase0$ 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 \(\doteq\) 1..N

NotAnInput \(\doteq\) CHOOSE c : c \(\notin\) Inputs

VARIABLES inputs, output

---

MODULE Inner

VARIABLES allInput, chosen

---

IInit \(\doteq\)

∧ input ∈ [Proc → Inputs]
∧ output = \([p ∈ Proc → NotAnInput]\]
∧ chosen = NotAnInput
∧ allInput = input[p] : p ∈ Proc

IChoose(p) \(\doteq\)

∧ output[p] = NotAnInput
∧ IF chosen = NotAnInput
    THEN ip ∈ allInput : ∧ chosen' = ip
        ∧ output' = [output EXCEPT ![p] = ip]
    ELSE ∧ output' = [output EXCEPT ![p] = chosen]
        ∧ UNCHANGED chosen
        ∧ UNCHANGED ⟨input, allInput⟩

IFail(p) \(\doteq\)

∧ output' = [output EXCEPT ![p] = NotAnInput]
∧ ∃ip ∈ Inputs : ∧ input' = [input EXCEPT ![p] = ip]
∧ allInput' = allInput \(∪\) \{ip\}

INext \(\doteq\)

∃p ∈ Proc : IChoose(p) \lor IFail(p)

ISpec \(\doteq\)

IInit \(\land\) □[INext](input, output, chosen, allInput)

---

IS(chosen, allInput) \(\doteq\) INSTANCE Inner

SynodSpec \(\doteq\)

∃chosen, allInput : IS(chosen, allInput) \(\land\) ISpec
B  Disk Paxos Algorithm Specification

description  DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedec InputsOrNi

typedec Disk

typedec Proc

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

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

 lemma ballots-not-zero [simp]:
   b ∈ Ballot p =⇒ 0 < b
〈proof〉

 lemma majority-nonempty [simp]: IsMajority(S) ⇒ S ≠ {}
〈proof〉

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

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

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

 record
 BlockProc =
  block :: DiskBlock
  proc :: Proc

 record
 state =
inpt :: Proc ⇒ InputsOrNi
outpt :: Proc ⇒ InputsOrNi
disk :: Disk ⇒ Proc ⇒ DiskBlock
dblock :: Proc ⇒ DiskBlock
phase :: Proc ⇒ nat
disksWritten :: Proc ⇒ Disk set
blocksRead :: Proc ⇒ Disk ⇒ BlockProc set

allInput :: InputsOrNi set
chosen :: InputsOrNi

definition hasRead :: state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where hasRead s p d q = (∃ br ∈ blocksRead s p d. proc br = q)
definition allRdBlks :: state ⇒ Proc ⇒ BlockProc set
where allRdBlks s p = (UN d. blocksRead s p d)
definition allBlocksRead :: state ⇒ Proc ⇒ DiskBlock set
where allBlocksRead s p = block ' (allRdBlks s p)
definition Init :: state ⇒ bool
where
Init s =
  (range (inpt s) ⊆ Inputs
& outpt s = (λp. NotAnInput)
& disk s = (λd p. InitDB)
& phase s = (λp. 0)
& dblock s = (λp. InitDB)
& disksWritten s = (λp. \{}
& blocksRead s = (λp d. \{\}))

definition InitializePhase :: state ⇒ state ⇒ Proc ⇒ bool
where
InitializePhase s s′ p =
  (disksWritten s′ = (disksWritten s)(p := \{\})
& blocksRead s′ = (blocksRead s)(p := (λd. \{\})))
definition StartBallot :: state ⇒ state ⇒ Proc ⇒ bool
where
StartBallot s s′ p =
  (phase s p ∈ \{1,2\}
& phase s′ = (phase s)(p := 1)
& (∃ b ∈ Ballot p.
  mbal (dblock s p) < b
  & dblock s′ = (dblock s)(p := (dblock s p)(mbal := b \]))
& InitializePhase s s′ p
& inpt s′ = inpt s & outpt s′ = outpt s & disk s′ = disk s)
definition Phase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool
where
Phase1or2Write s s′ p d =
(phase s ∈ {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 (dblock 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 [i]}}))
∧ 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 \(\{d. \text{ hasRead } s p d p\}\)
\& (\(\exists b \in \text{Ballot } p. \forall r \in \text{allBlocksRead } s p. \text{mbal } r < b\)
\& dblock s′ = (dblock s) \(p :=
(SOME r. r \in \text{allBlocksRead } s p \& \forall s \in \text{allBlocksRead } s p. \text{bal } s \leq \text{bal } r)\) (mbal := b))
\& InitializePhase s s′ p
\& phase s′ = (phase s) \(p := 1\)
\& inpt s′ = inpt s \& outpt s′ = outpt s \& disk s′ = disk s)

**definition** Next :: state ⇒ state ⇒ bool
where

Next s s′ = (\(\exists p. \\)
StartBallot s s′ p
\& (\(\exists d. \) Phase0Read s s′ p d
\& Phase1or2Write s s′ p d
\& (\(\exists q. q \neq p \& \) Phase1or2Read s s′ p d q))
\& EndPhase1or2 s s′ p
\& Fail s s′ p
\& EndPhase0 s s′ p)

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

**definition** HInit :: state ⇒ bool
where

HInit s = 
(Init s
\& chosen s = NotAnInput
\& allInput s = range (inpt s))

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

**definition** HNextPart :: state ⇒ state ⇒ bool
where

HNextPart s s′ = 
(chosen s′ = 
(if chosen s \neq NotAnInput \& \(\forall p. \) outpt s′ p = NotAnInput)
then chosen s
else outpt s′ (SOME p. outpt s′ p \neq NotAnInput))
\& allInput s′ = allInput s \cup (range (inpt s′)))

**definition** HNext :: state ⇒ state ⇒ bool
where

HNext s s′ =
\( (Next \ s \ s' \wedge HNextPart \ s \ s') \)

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

\textbf{definition}
\( HP\text{hase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( HP\text{hase1or2ReadThen} \ s \ s' \ p \ d \ q = (\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( H\text{EndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( H\text{EndPhase1} \ s \ s' \ p = (\text{EndPhase1} \ s \ s' \ p \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( H\text{StartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( H\text{StartBallot} \ s \ s' \ p = (\text{StartBallot} \ s \ s' \ p \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( HP\text{hase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \) where
\( HP\text{hase1or2Write} \ s \ s' \ p \ d = (\text{Phase1or2Write} \ s \ s' \ p \ d \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( HP\text{hase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( HP\text{hase1or2ReadElse} \ s \ s' \ p \ d \ q = (\text{Phase1or2ReadElse} \ s \ s' \ p \ d \ q \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( H\text{EndPhase2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( H\text{EndPhase2} \ s \ s' \ p = (\text{EndPhase2} \ s \ s' \ p \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( H\text{Fail} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( H\text{Fail} \ s \ s' \ p = (\text{Fail} \ s \ s' \ p \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( HP\text{hase0Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \) where
\( HP\text{hase0Read} \ s \ s' \ p \ d = (\text{Phase0Read} \ s \ s' \ p \ d \wedge H\text{NextPart} \ s \ s') \)

\textbf{definition}
\( H\text{EndPhase0} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where
\( H\text{EndPhase0} \ s \ s' \ p = (\text{EndPhase0} \ s \ s' \ p \wedge H\text{NextPart} \ s \ s') \)

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

\textbf{declare} \( HP\text{hase1or2ReadThen-def [simp]} \)
\textbf{declare} \( HP\text{hase1or2ReadElse-def [simp]} \)
\textbf{declare} \( H\text{EndPhase1-def [simp]} \)
\textbf{declare} \( H\text{StartBallot-def [simp]} \)
lemma allRdBlks-finite:
  assumes inv: HInv1 s
  and asm: \( \forall p. \text{allRdBlks } s' p \subseteq \text{insert } bk \text{ (allRdBlks } s \ p) \)
  shows \( \forall p. \text{finite } (\text{allRdBlks } s' p) \)
  (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 = 
(mb bal 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. input 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 blocksOf is included in the old blocksOf, or the new blocksOf is included in the old blocksOf union the new dblock. In the former case the assumption inv will imply the thesis. In the latter, we just have to prove the innermost predicate for the particular case of the new dblock. This particular case is proved in lemma action-Inv2a-dbloc.

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

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

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

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

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

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

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

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

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

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

lemma HEndPhase2-blocksOf:
  \[ HEndPhase2 s s' p \implies \text{blocksOf} s' q \subseteq \text{blocksOf} s q \]  
(proof)

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

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

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

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

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

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

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

EndPhase0 has the additional difficulty of having a choose expression. We prove that there exists an \( x \) such that the predicate of the choose expression holds, and then apply someI: \( ?P\ ?x \implies ?P\ (Eps\ ?P)\).

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

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

lemma HNextPart-allInput-or-NotAnInput:
assumes act: HNextPart s s'
and inv2a: Inv2a-innermost s p \( (\text{dblock } s'\ p) \)
shows \( \text{inp} (\text{dblock } s'\ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
⟨proof⟩

lemma HEndPhase0-Inv2a-allBlocksRead:
assumes act: HEndPhase0 s s' p
and inv2a: Inv2a-inner s p
and inv2c: Inv2c-inner s p
shows \( \forall t \in (\lambda x.\ x (|mbal:= mbal(\text{dblock } s'\ p)|)) \cdot \text{allBlocksRead } s\ p.\ \text{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 \implies \text{blocksOf s' q} \subseteq \text{blocksOf s q} \cup \{\text{dblock s' q}\}
⟨proof⟩

lemma maxBlk-in-nonInitBlks:
assumes b: b \in \text{nonInitBlks s p}
and inv1: Inv1 s
shows maxBlk s p \in \text{nonInitBlks s p}
\land (\forall c \in \text{nonInitBlks s p}. \text{bal c} \leq \text{bal (maxBlk s p)})
⟨proof⟩

lemma blocksOf-nonInitBlks:
(\forall p bk. bk \in \text{blocksOf s p} \implies P bk)
\implies bk \in \text{nonInitBlks s p} \implies P bk
⟨proof⟩

lemma maxBlk-allInput:
assumes inv: Inv2a s
and mbk: maxBlk s p \in \text{nonInitBlks s p}
shows inp (maxBlk s p) \in \text{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) \in \text{allInput s'}
⟨proof⟩

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

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

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

C.2.2 Proofs of Invariant 2 b

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

theorem $H\text{Init-Inv2b}$: $H\text{Init } s \rightarrow \text{Inv2b } s$
\langle proof \rangle

theorem $H\text{Phase1or2ReadThen-Inv2b}$:
$\begin{array}{c}
[ \text{Inv2b } s; \text{HPhase1or2ReadThen } s \ s' p \ d \ q ] \\
\rightarrow \text{Inv2b } s'
\end{array}$ 
\langle proof \rangle

theorem $H\text{StartBallot-Inv2b}$:
$\begin{array}{c}
[ \text{Inv2b } s; \text{HStartBallot } s \ s' p ] \\
\rightarrow \text{Inv2b } s'
\end{array}$ 
\langle proof \rangle

theorem $H\text{Phase1or2Write-Inv2b}$:
$\begin{array}{c}
[ \text{Inv2b } s; \text{HPhase1or2Write } s \ s' p \ d ] \\
\rightarrow \text{Inv2b } s'
\end{array}$ 
\langle proof \rangle

theorem $H\text{Phase1or2ReadElse-Inv2b}$:
$\begin{array}{c}
[ \text{Inv2b } s; \text{HPhase1or2ReadElse } s \ s' p \ d \ q ] \\
\rightarrow \text{Inv2b } s'
\end{array}$ 
\langle proof \rangle

theorem $H\text{EndPhase1-Inv2b}$:
$\begin{array}{c}
[ \text{Inv2b } s; \text{HEndPhase1 } s \ s' p ] \\
\rightarrow \text{Inv2b } s'
\end{array}$ 
\langle proof \rangle
\begin{verbatim}

\textbf{theorem} \textit{HFail-Inv2b}:
\[
\begin{array}{cl}
\text{\textbf{H}} & \text{Fail-Inv2b} \rightarrow \text{Inv2b} \ s' \\
\end{array}
\]
\langle \text{proof} \rangle

\textbf{theorem} \textit{HEndPhase2-Inv2b}:
\[
\begin{array}{cl}
\text{\textbf{H}} & \text{EndPhase2} \rightarrow \text{Inv2b} \ s' \\
\end{array}
\]
\langle \text{proof} \rangle

\textbf{theorem} \textit{HPhase0Read-Inv2b}:
\[
\begin{array}{cl}
\text{\textbf{H}} & \text{Phase0Read} \rightarrow \text{Inv2b} \ s' \\
\end{array}
\]
\langle \text{proof} \rangle

\textbf{theorem} \textit{HEndPhase0-Inv2b}:
\[
\begin{array}{cl}
\text{\textbf{H}} & \text{EndPhase0} \rightarrow \text{Inv2b} \ s' \\
\end{array}
\]
\langle \text{proof} \rangle

\textbf{C.2.3 Proofs of Invariant 2 c}

\textbf{theorem} \textit{HInit-Inv2c}: \textit{HInit} \rightarrow \text{Inv2c} \ s
\langle \text{proof} \rangle

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

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

\textbf{lemma} \textit{HNextPart-allInput}:
\[
\begin{array}{cl}
\text{\textbf{H}} & \textit{HNextPart} \ s \ s'; \textit{Inv2c} \ s \\
\end{array}
\]
\langle \text{proof} \rangle

\textbf{theorem} \textit{HPhase1or2ReadThen-Inv2c}:
\textbf{assumes} \textit{inv}: \textit{Inv2c} \ s \\
\textbf{and} \ \textit{act}: \textit{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q \\
\textbf{and} \ \textit{inv2a}: \textit{Inv2a} \ s \\
\textbf{shows} \ \textit{Inv2c} \ s'
\langle \text{proof} \rangle
\end{verbatim}
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 ] \implies 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

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 = (\(\text{block} = \text{dblock} s q, \text{proc} = q\) ∈ blocksRead s p d
∧ \(\text{block} = \text{dblock} s p, \text{proc} = p\) ∈ 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 = (∀ q p d. HInv3-inner s p q d)

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 = (\(\text{block} = \text{dblock} s q, \text{proc} = q\) ∈ blocksRead s p d
∧ \(\text{block} = \text{dblock} s p, \text{proc} = p\) ∈ 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 = (∀ q p 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:
\[\begin{align*}
\text{InitializePhase } s \quad s' \quad p \quad q \quad d \Rightarrow HInv3-R \quad s' \quad p \quad q \quad d
\end{align*}\]

lemma InitPhase-HInv3-q:
\[\begin{align*}
\text{InitializePhase } s \quad s' \quad q \quad p \quad d \Rightarrow HInv3-R \quad s' \quad p \quad q \quad d
\end{align*}\]

lemma HInv3-L-sym: \(HInv3-L \quad s \quad p \quad q \quad d \Rightarrow HInv3-L \quad s \quad q \quad p \quad d\)

lemma HInv3-R-sym: \(HInv3-R \quad s \quad p \quad q \quad d \Rightarrow HInv3-R \quad s \quad q \quad p \quad d\)

lemma Phase1or2ReadThen-HInv3-pq:
\[\begin{align*}
\text{assumes act: Phase1or2ReadThen } s \quad s' \quad p \quad d \quad q \\
\text{and inv-L': } HInv3-L \quad s' \quad p \quad q \quad d \\
\text{and } pq: p \neq q \\
\text{and inv2b: } Inv2b \quad s \\
\text{shows } HInv3-R \quad s' \quad p \quad q \quad d
\end{align*}\]

lemma Phase1or2ReadThen-HInv3-hasRead:
\[\begin{align*}
\text{assumes } \neg \text{hasRead } s \quad pp \quad dd \quad qq; \\
\text{Phase1or2ReadThen } s \quad s' \quad p \quad d \quad q; \\
\text{pp} \neq p \lor qq \neq q \lor dd \neq d \\
\Rightarrow \neg \text{hasRead } s' \quad pp \quad dd \quad qq
\end{align*}\]

theorem HPhase1or2ReadThen-HInv3:
\[\begin{align*}
\text{assumes act: HPhase1or2ReadThen } s \quad s' \quad p \quad d \quad q \\
\text{and } inv: HInv3 \quad s \\
\text{and } pq: p \neq q \\
\text{and } inv2b: Inv2b \quad s \\
\text{shows } HInv3 \quad s'
\end{align*}\]

lemma StartBallot-HInv3-p:
\[\begin{align*}
\text{StartBallot } s \quad s' \quad p \quad HInv3-L \quad s' \quad p \quad q \quad d \Rightarrow HInv3-R \quad s' \quad p \quad q \quad d
\end{align*}\]

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

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

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

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

theorem HStartBallot-HInv3:
\[ \begin{array}{l}
\text{HStartBallot } s \ s' p; \text{HInv3 } s \\
\Rightarrow \text{HInv3 } s'
\end{array} \]
⟨proof⟩

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

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

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

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

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

lemma EndPhase1-HInv3-R:
[EndPhase1 \( s s' t \); HInv3-R \( s p q d \); \( t \neq p \); \( t \neq q \)]
\[\Rightarrow\] HInv3-R \( s' p q d \)

⟨proof⟩

lemma EndPhase1-HInv3-t:
[EndPhase1 \( s s' t \); HInv3-inner \( s p q d \); \( t \neq p \); \( t \neq q \)]
\[\Rightarrow\] 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 \( s \)] \[\Rightarrow\] HInv3 \( s' \)
⟨proof⟩

lemma EndPhase2-HInv3-p:
[EndPhase2 \( s s' p \); HInv3-L \( s' p q d \)] \[\Rightarrow\] HInv3-R \( s' p q d \)
⟨proof⟩

lemma EndPhase2-HInv3-q:
[EndPhase2 \( s s' q \); HInv3-L \( s' p q d \)] \[\Rightarrow\] HInv3-R \( s' p q d \)
⟨proof⟩

lemma EndPhase2-HInv3-nL:
[EndPhase2 \( s s' t \); \neg\ HInv3-L \( s p q d \); \( t \neq p \); \( t \neq q \)]
\[\Rightarrow\] \neg\ HInv3-L \( s' p q d \)
⟨proof⟩

lemma EndPhase2-HInv3-R:
[EndPhase2 \( s s' t \); HInv3-R \( s p q d \); \( t \neq p \); \( t \neq q \)]
\[\Rightarrow\] HInv3-R \( s' p q d \)
⟨proof⟩

lemma EndPhase2-HInv3-t:
[EndPhase2 \( s s' t \); HInv3-inner \( s p q d \); \( t \neq p \); \( t \neq q \)]
\[\Rightarrow\] 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:
lemma Fail-HInv3-p:
\[ [ \text{Fail } s' \; p'; \text{HInv3-L } s' \; p \; q \; d ] \implies \text{HInv3-R } s' \; p \; q \; d \]
(\text{proof})

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

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

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

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

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

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

theorem HPhase0Read-HInv3:
\begin{itemize}
  \item assumes act: \text{HPhase0Read } s \; s' \; p \; d
  \item and inv: \text{HInv3 } s
\end{itemize}
\begin{itemize}
  \item shows \text{HInv3 } s'
\end{itemize}
(\text{proof})

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

lemma EndPhase0-HInv3-q:
\[ [ \text{EndPhase0 } s \; s' \; q; \text{HInv3-L } s' \; p \; q \; d ] \]
lemma EndPhase0-HInv3-R:
  \[
  \text{EndPhase0 } s \; s' \; t; \; \neg \text{HInv3-L } s \; p \; q \; d; \; t \neq p; \; t \neq q \implies \neg \text{HInv3-L } s' \; p \; q \; d
  \]
  \langle proof \rangle

lemma EndPhase0-HInv3-L:
  \[
  \text{EndPhase0 } s \; s' \; t; \; \neg \text{HInv3-L } s \; p \; q \; d; \; \neg \text{HInv3-R } s' \; p \; q \; d \implies \text{HInv3-L } s \; p \; q \; d
  \]
  \langle proof \rangle

lemma EndPhase0-HInv3-R:
  \[
  \text{EndPhase0 } s \; s' \; t; \; \text{HInv3-R } s \; p \; q \; d; \; t \neq p; \; t \neq q \implies \text{HInv3-R } s' \; p \; q \; d
  \]
  \langle proof \rangle

lemma EndPhase0-HInv3-L:
  \[
  \text{EndPhase0 } s \; s' \; t; \; \text{HInv3-L } s \; p \; q \; d; \; t \neq p; \; t \neq q \implies \text{HInv3-L } s' \; p \; q \; d
  \]
  \langle proof \rangle

lemma EndPhase0-HInv3-t:
  \[
  \text{EndPhase0 } s \; s' \; t; \; \text{HInv3-inner } s \; p \; q \; d; \; t \neq p; \; t \neq q \implies \text{HInv3-inner } s' \; p \; q \; d
  \]
  \langle proof \rangle

lemma EndPhase0-HInv3:
  \[
  \text{assumes act: } \text{EndPhase0 } s \; s' \; t \;
  \text{and inv: } \text{HInv3-inner } s \; p \; q \; d
  \]
  \text{shows } \text{HInv3-inner } s' \; p \; q \; d
  \langle proof \rangle

theorem HEndPhase0-HInv3:
  \[
  \text{HEndPhase0 } s \; s' \; p; \; \text{HInv3 } s \implies \text{HInv3 } s'
  \]
  \langle proof \rangle

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

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

end

theory DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

C.4 Invariant 4

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

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

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

**definition** HInv4a2 :: state => Proc => 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 :: state => Proc => bool  
where HInv4a s p =  \(\forall p.  \text{HInv4a1 } s p \land \text{HInv4a2 } s p\)

**definition** HInv4b :: state => Proc => bool  
where HInv4b s p =  \(\forall p.  \text{HInv4a1 } s p \land \text{HInv4a2 } s p\)

The initial state implies Invariant 4.

**theorem** HInit-HInv4: HInit s => HInv4 s  
(proof)

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

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

### C.4.1 Proofs of Invariant 4a

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

\begin{proof}
\end{proof}

\begin{lemma}
HStartBallot-HInv4a2:
assumes act: $HStartBallot \ s \ s' \ p$
and inv: $HInv4a2 \ s \ p$
shows $HInv4a2 \ s' \ p$
\end{lemma}

\begin{proof}
\end{proof}

\begin{lemma}
HStartBallot-HInv4a-p:
assumes act: $HStartBallot \ s \ s' \ p$
and inv: $HInv4a \ s \ p$
and inv2a: Inv2a-inner $s' \ p$
shows $HInv4a \ s' \ p$
\end{lemma}

\begin{proof}
\end{proof}

\begin{lemma}
HStartBallot-HInv4a-q:
assumes act: $HStartBallot \ s \ s' \ p$
and inv: $HInv4a \ s \ q$
and pnq: $p \neq q$
shows $HInv4a \ s' \ q$
\end{lemma}

\begin{proof}
\end{proof}

\begin{theorem}
HStartBallot-HInv4a:
assumes act: $HStartBallot \ s \ s' \ p$
and inv: $HInv4a \ s \ q$
and inv2a: Inv2a $s'$
shows $HInv4a \ s' \ q$
\end{theorem}

\begin{proof}
\end{proof}

\begin{lemma}
Phase1or2Write-HInv4a:
\begin{align*}
\{ \ & \text{Phase1or2Write} \ s \ s' \ p \ d; \ HInv4a1 \ s \ q \} \\
\implies & \ HInv4a1 \ s' \ q
\end{align*}
\end{lemma}

\begin{proof}
\end{proof}

\begin{lemma}
Phase1or2Write-HInv4a:
\begin{align*}
\{ \ & \text{Phase1or2Write} \ s \ s' \ p \ d; \ HInv4a2 \ s \ q \} \\
\implies & \ HInv4a2 \ s' \ q
\end{align*}
\end{lemma}

\begin{proof}
\end{proof}

\begin{theorem}
HPhase1or2Write-HInv4a:
assumes act: $HPhase1or2Write \ s \ s' \ p \ d$
and inv: $HInv4a \ s \ q$
shows $HInv4a \ s' \ q$
\end{theorem}

\begin{proof}
\end{proof}

\begin{lemma}
HPhase1or2ReadThen-HInv4a1-p:
assumes act: $HPhase1or2ReadThen \ s \ s' \ p \ d \ q$
and inv: $HInv4a1 \ s \ p$
shows $HInv4a1 \ s' \ p$
\end{lemma}

\begin{proof}
\end{proof}
lemma HPhase1or2ReadThen-HInv4a2:
\[ [ \text{HPhase1or2ReadThen } s s' p d r; \text{HInv4a2 } s q ] \implies \text{HInv4a2 } s' q \]
(proof)

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

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

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

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

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

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

lemma HEndPhase1-HInv4a-p:
assumes act: HEndPhase1 \ s \ s' \ p
and inv: HInv4a \ s \ p
and inv2a: Inv2a \ s
shows HInv4a \ s' \ p
(proof)
lemma \ HE\text{EndPhase1-}HInv4a-q:
assumes act: HE\text{ndPhase1} s s' p
and inv: HInv4a s q
and pnq: p \neq q
shows HInv4a s' q
\langle proof \rangle

theorem \ HE\text{ndPhase1-}HInv4a:
[ \ HE\text{ndPhase1} s s' p; HInv4a s q; Inv2a s ] \Longrightarrow HInv4a s' q
\langle proof \rangle

theorem \ H\text{Fail-}HInv4a:
[ \ H\text{Fail} s s' p; HInv4a s q ] \Longrightarrow HInv4a s' q
\langle proof \rangle

theorem \ H\text{Phase0Read-}HInv4a:
[ \ H\text{Phase0Read} s s' p d; HInv4a s q ] \Longrightarrow HInv4a s' q
\langle proof \rangle

theorem \ HE\text{ndPhase2-}HInv4a:
[ \ HE\text{ndPhase2} s s' p; HInv4a s q ] \Longrightarrow HInv4a s' q
\langle proof \rangle

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

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

lemma \ HE\text{ndPhase0-}HInv4a1-p:
assumes act: HE\text{ndPhase0} s s' p
and inv2a': Inv2a s'
and inv2c: Inv2c-inner s p
and inv4d: HInv4d s p
shows HInv4a1 s' p
\langle proof \rangle

lemma \ hasRead-allBlks:
assumes inv2c: Inv2c-inner s p
and phase: phase s p = 0
shows (\forall d \in \{ d. \ hasRead s p d p \}, disk s d p \in allBlocksRead s p)
\langle proof \rangle
lemma $HEndPhase0$-I1:
assumes act: $HEndPhase0$ s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows $\exists D \in \text{MajoritySet. } \forall d \in D. \quad \text{mbal} (\text{disk} s d p) \leq \text{mbal} (\text{dblock} s' p) \\
\wedge \text{bal} (\text{disk} s d p) \leq \text{bal} (\text{dblock} s' p)$
(\proof)

lemma Majority-exQ:
assumes asm1: $\exists D \in \text{MajoritySet. } \forall d \in D. \quad P d$
shows $\forall D \in \text{MajoritySet. } \exists d \in D. \quad P d$
(\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:
[ $HEndPhase0$ s s' p; HInv4a s q; HInv4d s p; 
  Inv2a s; \quad Inv1 s; \quad Inv2a s; \quad Inv2c s]$ 
$\Rightarrow$ HInv4a s' q 
(\proof)

C.4.2 Proofs of Invariant 4b

lemma blocksRead-allBlocksRead:
rb \in \text{blocksRead} s p d \Rightarrow \text{block } rb \in \text{allBlocksRead} s p
lemma $H\text{EndPhase0-dblock-mbal}$:

\[
H\text{EndPhase0 s s'} p \implies \forall br \in \text{allBlocksRead s p}. \ mbal br < mbal(dblock s' p)
\]

lemma $H\text{EndPhase0-HInv4b-p-dblock}$:

\begin{align*}
\text{assumes} & \quad \text{act: } H\text{EndPhase0 s s'} p \\
\text{and} & \quad \text{inv1: } Inv1 s \\
\text{and} & \quad \text{inv2a: } Inv2a s \\
\text{and} & \quad \text{inv2c: } Inv2c\text{-inner s p} \\
\text{shows} & \quad \text{bal(dblock s' p) < mbal(dblock s' p)}
\end{align*}

lemma $H\text{EndPhase0-HInv4b-p-blocksOf}$:

\begin{align*}
\text{assumes} & \quad \text{act: } H\text{EndPhase0 s s'} p \\
\text{and} & \quad \text{inv4d: } H\text{Inv4d s p} \\
\text{and} & \quad \text{inv2c: } Inv2c\text{-inner s p} \\
\text{and} & \quad \text{bk: } bk \in \text{blocksOf s p} \\
\text{shows} & \quad \text{bal bk < mbal(dblock s' p)}
\end{align*}

lemma $H\text{EndPhase0-HInv4b-p}$:

\begin{align*}
\text{assumes} & \quad \text{act: } H\text{EndPhase0 s s'} p \\
\text{and} & \quad \text{inv4d: } H\text{Inv4d s p} \\
\text{and} & \quad \text{inv1: } Inv1 s \\
\text{and} & \quad \text{inv2a: } Inv2a s \\
\text{and} & \quad \text{inv2c: } Inv2c\text{-inner s p} \\
\text{shows} & \quad H\text{Inv4b s' p}
\end{align*}

lemma $H\text{EndPhase0-HInv4b-q}$:

\begin{align*}
\text{assumes} & \quad \text{act: } H\text{EndPhase0 s s'} p \\
\text{and} & \quad \text{pnq: } p \neq q \\
\text{and} & \quad \text{inv: } H\text{Inv4b s q} \\
\text{shows} & \quad H\text{Inv4b s' q}
\end{align*}

theorem $H\text{EndPhase0-HInv4b}$:

\begin{align*}
\text{assumes} & \quad \text{act: } H\text{EndPhase0 s s'} p \\
\text{and} & \quad \text{inv: } H\text{Inv4b s q} \\
\text{and} & \quad \text{inv4d: } H\text{Inv4d s p} \\
\text{and} & \quad \text{inv1: } Inv1 s \\
\text{and} & \quad \text{inv2a: } Inv2a s \\
\text{and} & \quad \text{inv2c: } Inv2c\text{-inner s p} \\
\text{shows} & \quad H\text{Inv4b 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:
\[ [ HPhase1or2Write s s' p d; HInv4b s q ] \implies HInv4b s' q \]
⟨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:
\[ [ HPhase1or2ReadThen s s' p d q; HInv4b s r ] \implies HInv4b s' r \]
⟨proof⟩

theorem HPhase1or2ReadElse-HInv4b:
\[ [ HPhase1or2ReadElse s s' p d q; HInv4b s r; Inv2a s; HInv4a s p ] \implies HInv4b s' r \]
lemma HEndPhase1-HInv4b-p:
  HEndPhase1 s s' p \Rightarrow HInv4b s' p

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

theorem HEndPhase1-HInv4b:
  assumes act: HEndPhase1 s s' p
  and inv: HInv4b s q
  shows HInv4b s' q

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

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

theorem HEndPhase2-HInv4b:
  assumes act: HEndPhase2 s s' p
  and inv: HInv4b s q
  shows HInv4b s' q

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

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

theorem HFail-HInv4b:
  assumes act: HFail s s' p
and $inv: HInv_{4b} s q$
shows $HInv_{4b} s' q$
(proof)

lemma $HPhase0Read-HInv_{4b}$-p:
$HPhase0Read s s' p d \implies HInv_{4b} s' p$
(proof)

lemma $HPhase0Read-HInv_{4b}$-q:
assumes act: $HPhase0Read s s' p d$
and $pnq: p \neq q$
and $inv: HInv_{4b} s q$
shows $HInv_{4b} s' q$
(proof)

theorem $HPhase0Read-HInv_{4b}$:
assumes act: $HPhase0Read s s' p d$
and $inv: HInv_{4b} s q$
shows $HInv_{4b} s' q$
(proof)

C.4.3 Proofs of Invariant 4c

lemma $HStartBallot-HInv_{4c}$-p:
$[HStartBallot s s' p; HInv_{4c} s p] \implies HInv_{4c} s' p$
(proof)

lemma $HStartBallot-HInv_{4c}$-q:
assumes act: $HStartBallot s s' p$
and $inv: HInv_{4c} s q$
and $pnq: p \neq q$
shows $HInv_{4c} s' q$
(proof)

theorem $HStartBallot-HInv_{4c}$:
$[HStartBallot s s' p; HInv_{4c} s q] \implies HInv_{4c} s' q$
(proof)

lemma $HPhase1or2Write-HInv_{4c}$-p:
assumes act: $HPhase1or2Write s s' p d$
and $inv: HInv_{4c} s p$
and $inv2c: Inv_{2c} s$
shows $HInv_{4c} s' p$
(proof)

lemma $HPhase1or2Write-HInv_{4c}$-q:
assumes act: $HPhase1or2Write s s' p d$
and $inv: HInv_{4c} s q$
and $pnq: p \neq q$
shows $HInv4c \ s \ s' \ q$

\begin{proof}
\end{proof}

**Theorem** $HPhase1or2Write-HInv4c$:

\[
[HPhase1or2Write \ s \ s' \ p \ d; \ HInv4c \ s \ q; \ Inv2c \ s] \Rightarrow HInv4c \ s' \ q
\]

\begin{proof}
\end{proof}

**Lemma** $HPhase1or2ReadThen-HInv4c-p$:

\[
[HPhase1or2ReadThen \ s \ s' \ p \ d \ q; \ HInv4c \ s \ p] \Rightarrow HInv4c \ s' \ p
\]

\begin{proof}
\end{proof}

**Lemma** $HPhase1or2ReadThen-HInv4c-q$:

assumes act: $HPhase1or2ReadThen \ s \ s' \ p \ d \ r$
and inv: $HInv4c \ s \ q$
and pnq: $p \neq q$
shows $HInv4c \ s' \ q$

\begin{proof}
\end{proof}

**Theorem** $HPhase1or2ReadThen-HInv4c$:

\[
[HPhase1or2ReadThen \ s \ s' \ p \ d \ r; \ HInv4c \ s \ q] \Rightarrow HInv4c \ s' \ q
\]

\begin{proof}
\end{proof}

**Theorem** $HPhase1or2ReadElse-HInv4c$:

\[
[HPhase1or2ReadElse \ s \ s' \ p \ d \ r; \ HInv4c \ s \ q] \Rightarrow HInv4c \ s' \ q
\]

\begin{proof}
\end{proof}

**Lemma** $HEndPhase1-HInv4c-p$:

assumes act: $HEndPhase1 \ s \ s' \ p$
and inv2b: $Inv2b \ s$
shows $HInv4c \ s' \ p$

\begin{proof}
\end{proof}

**Lemma** $HEndPhase1-HInv4c-q$:

assumes act: $HEndPhase1 \ s \ s' \ p$
and inv: $HInv4c \ s \ q$
and pnq: $p \neq q$
shows $HInv4c \ s' \ q$

\begin{proof}
\end{proof}

**Theorem** $HEndPhase1-HInv4c$:

\[
[HEndPhase1 \ s \ s' \ p; \ HInv4c \ s \ q; \ Inv2b \ s] \Rightarrow HInv4c \ s' \ q
\]

\begin{proof}
\end{proof}

**Lemma** $HEndPhase2-HInv4c-p$:

\[
[HEndPhase2 \ s \ s' \ p; \ HInv4c \ s \ p] \Rightarrow HInv4c \ s' \ p
\]

\begin{proof}
\end{proof}

45
lemma \text{HEndPhase2-Inv4c-q}:
assumes act: HEndPhase2 s s' p
and inv: \text{Inv4c} s q
and pnq: p \neq q
shows \text{Inv4c} s' q
\langle proof \rangle

theorem \text{HEndPhase2-Inv4c}:
\[ \{ \text{HEndPhase2} s s' p; \text{Inv4c} s q \} \implies \text{Inv4c} s' q \]
\langle proof \rangle

lemma \text{HFail-Inv4c-p}:
\[ \{ \text{HFail} s s' p; \text{Inv4c} s p \} \implies \text{Inv4c} s' p \]
\langle proof \rangle

lemma \text{HFail-Inv4c-q}:
assumes act: HFail s s' p
and inv: \text{Inv4c} s q
and pnq: p \neq q
shows \text{Inv4c} s' q
\langle proof \rangle

theorem \text{HFail-Inv4c}:
\[ \{ \text{HFail} s s' p; \text{Inv4c} s q \} \implies \text{Inv4c} s' q \]
\langle proof \rangle

lemma \text{HPhase0Read-Inv4c-p}:
\[ \{ \text{HPhase0Read} s s' p d; \text{Inv4c} s p \} \implies \text{Inv4c} s' p \]
\langle proof \rangle

lemma \text{HPhase0Read-Inv4c-q}:
assumes act: HPhase0Read s s' p d
and inv: \text{Inv4c} s q
and pnq: p \neq q
shows \text{Inv4c} s' q
\langle proof \rangle

theorem \text{HPhase0Read-Inv4c}:
\[ \{ \text{HPhase0Read} s s' p d; \text{Inv4c} s q \} \implies \text{Inv4c} s' q \]
\langle proof \rangle

lemma \text{HEndPhase0-Inv4c-p}:
\[ \{ \text{HEndPhase0} s s' p; \text{Inv4c} s p \} \implies \text{Inv4c} s' p \]
\langle proof \rangle

lemma \text{HEndPhase0-Inv4c-q}:
assumes act: HEndPhase0 s s' p
and inv: \text{Inv4c} s q
and pnq: p \neq q

46
shows $HInv^4c \ s' \ q$

(proof)

theorem $HEndPhase0-HInv^4c$:

\[ [ \ HEndPhase0 \ s \ s' \ p; \ HInv^4c \ s \ q ] \implies HInv^4c \ s' \ q \]

(proof)

C.4.4 Proofs of Invariant 4d

lemma $HStartBallot-HInv^4d-p$:

assumes act: $HStartBallot \ s \ s' \ p$

and inv: $HInv^4d \ s \ p$

shows $HInv^4d \ s' \ p$

(proof)

lemma $HStartBallot-HInv^4d-q$:

assumes act: $HStartBallot \ s \ s' \ p$

and inv: $HInv^4d \ s \ q$

and $pnq$: $p \neq q$

shows $HInv^4d \ s' \ q$

(proof)

theorem $HStartBallot-HInv^4d$:

\[ [ \ HStartBallot \ s \ s' \ p; \ HInv^4d \ s \ q ] \implies HInv^4d \ s' \ q \]

(proof)

lemma $HPhase1or2Write-HInv^4d-p$:

assumes act: $HPhase1or2Write \ s \ s' \ p \ d$

and inv: $HInv^4d \ s \ p$

and inv4a: $HInv^4a \ s \ p$

shows $HInv^4d \ s' \ p$

(proof)

lemma $HPhase1or2Write-HInv^4d-q$:

assumes act: $HPhase1or2Write \ s \ s' \ p \ d$

and inv: $HInv^4d \ s \ q$

and $pnq$: $p \neq q$

shows $HInv^4d \ s' \ q$

(proof)

theorem $HPhase1or2Write-HInv^4d$:

\[ [ \ HPhase1or2Write \ s \ s' \ p \ d; \ HInv^4d \ s \ q; \ HInv^4a \ s \ p ] \implies HInv^4d \ s' \ q \]

(proof)

lemma $HPhase1or2ReadThen-HInv^4d-p$:

assumes act: $HPhase1or2ReadThen \ s \ s' \ p \ d \ q$

and inv: $HInv^4d \ s \ p$

shows $HInv^4d \ s' \ p$

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

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

theorem HPhase1or2ReadElse-HInv4d:
  \[ HPhase1or2ReadElse s s' p d r; HInv4d s q \implies HInv4d s' q \]
⟨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 p\neq q
  shows HInv4d s' q
⟨proof⟩

theorem HEndPhase1-HInv4d:
  \[ HEndPhase1 s s' p; HInv4d s q; Inv2b s; HInv4c s p \]  
  \implies 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 p\neq q
  shows HInv4d s' q
⟨proof⟩
theorem HEndPhase2-HInv4d:
\[ \[ \text{HEndPhase2 } s \ s' \ p; \ \text{HInv4d } s \ q \] \implies \text{HInv4d } s' \ q \]
\langle proof \rangle

lemma HFail-HInv4d-p:
assumes act: HFail s s' p
and inv: HInv4d s p
shows HInv4d s' p
\langle proof \rangle

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

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

lemma HPhase0Read-HInv4d-p:
assumes act: HPhase0Read s s' p d
and inv: HInv4d s p
shows HInv4d s' p
\langle proof \rangle

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

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

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

lemma HEndPhase0-HInv4d-p:
assumes act: HEndPhase0 s s' p
and inv: HInv4d s p
and inv2c: Inv2c s
and inv1: Inv1 s
shows $HInv^4_d s' \ s \ p$
(proof)

lemma $HEndPhase0-HInv^4_d-q$
assumes $act\ : HEndPhase0 \ s \ s' \ p$
and $inv\ : HInv^4_d \ s \ q$
and $pq\ : p \neq q$
shows $HInv^4_d s' \ q$
(proof)

theorem $HEndPhase0-HInv^4_d$
$$\begin{bmatrix}
HEndPhase0 \ s \ s' \ p
& HInv^4_d \ s \ q
& Inv2c \ s
& Inv1 \ s
\end{bmatrix} \Rightarrow HInv^4_d s' \ q$$
(proof)

Since we have already proved $HInv^2$ is an invariant of $HNext$, $HInv^1 \land HInv^2 \land HInv^4$ is also an invariant of $HNext$.

lemma $I2d$
assumes $nxt\ : HNext \ s \ s'$
and $inv\ : HInv^1 \ s \land HInv^2 \ s \land HInv^2 \ s' \land HInv^4 \ s$
shows $HInv^4 \ s'$
(proof)

end

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

C.5 Invariant 5

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

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

definition $HInv5-inner-R$ :: $state \ \Rightarrow \ Proc \ \Rightarrow \ bool$
where $HInv5-inner-R \ s \ p =$
\begin{align*}
& (maxBalInp \ s \ (bal(dblocksp)) \ (inp(dblocksp))) \\
& \lor (\exists D \in \MajoritySet. \ \exists q. \ (\forall d \in D. \ bal(dblocksp) < mbal(diskspd)) \land \ \neg \ hasRead \ s \ p \ d \ q))
\end{align*}

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


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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

\[ \text{theorem \ HInit-HInv5: } \text{HInit} \ s = \Rightarrow \text{HInv5} \ s \]  

\[ \langle \text{proof} \rangle \]

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

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

\[ \text{lemma \ HStartBallot-HInv5-p:} \]
  \[ \text{assumes act: } \text{HStartBallot} \ s \ s' \ p \]
  \[ \text{and inv: } \text{HInv5-inner} \ s \ p \]
  \[ \text{shows } \text{HInv5-inner} \ s' \ p \langle \text{proof} \rangle \]

\[ \text{lemma \ HStartBallot-blocksOf-q:} \]
  \[ \text{assumes act: } \text{HStartBallot} \ s \ s' \ p \]
  \[ \text{and pq: } p \neq q \]
  \[ \text{shows } \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \langle \text{proof} \rangle \]

\[ \text{lemma \ HStartBallot-allBlocks:} \]
  \[ \text{assumes act: } \text{HStartBallot} \ s \ s' \ p \]
  \[ \text{shows } \text{allBlocks} \ s' \subseteq \text{allBlocks} \ s \cup \{\text{dblock} \ s' \ p\} \langle \text{proof} \rangle \]

\[ \text{lemma \ HStartBallot-HInv5-q1:} \]
  \[ \text{assumes act: } \text{HStartBallot} \ s \ s' \ p \]
  \[ \text{and pq: } p \neq q \]
  \[ \text{and inv5-1: } \text{maxBalInp} \ s \ (\text{bal}(\text{dblock} \ s \ q)) \ (\text{inp}(\text{dblock} \ s \ q)) \]
  \[ \text{shows } \text{maxBalInp} \ s' \ (\text{bal}(\text{dblock} \ s' \ q)) \ (\text{inp}(\text{dblock} \ s' \ q)) \langle \text{proof} \rangle \]

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

⟨proof⟩

theorem HStartBallot-HInv5:
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: \exists D ∈ MajoritySet. \exists q. (\forall d ∈ D. bal(dblock s p) < mbal(disk s d q)
∧ ¬ hasRead s p d q)
shows \exists D ∈ MajoritySet. \exists q. (\forall 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: HInv4 c 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: HInv4 a s p
and inv5-2: \exists D ∈ MajoritySet. \exists qq. (\forall d ∈ D. bal(dblock s q) < mbal(disk s d qq)
∧ ¬ hasRead s q d qq)
\[ \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \quad \text{bal}(d\text{block} \ s' \ q) < \text{mbal}(d\text{isk} \ s' \ d \ qq) \\
\wedge \neg \text{hasRead} \ s' \ q \ d \ qq) \]

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2Write-}\text{H}\text{Inv5-q}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2Write} \ s \ s' \ p \ d
  \item \text{and} \ inv: \ H\text{Inv5-inner} \ s \ q
  \item \text{and} \ inv4a: \ H\text{Inv4a} \ s \ p
  \item \text{and} \ p\neq q
\end{itemize}
\textbf{shows} \ H\text{Inv5-inner} \ s' \ q

(\text{proof})

\textbf{theorem} \ H\text{Phase1or2Write-}\text{H}\text{Inv5}:
\begin{itemize}
  \item [ 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
\end{itemize}

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2ReadThen-}\text{H}\text{Inv5-1}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ r
  \item \text{and} \ inv5-1: \ maxBalInp \ s \ (\text{bal}(d\text{block} \ s \ q)) \ (\text{inp}(d\text{block} \ s \ q))
\end{itemize}
\textbf{shows} \ maxBalInp \ s' \ (\text{bal}(d\text{block} \ s' \ q)) \ (\text{inp}(d\text{block} \ s' \ q))

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2ReadThen-}\text{H}\text{Inv5-p2}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ r
  \item \text{and} \ inv4c: \ H\text{Inv4c} \ s \ p
  \item \text{and} \ inv2c: \ H\text{Inv2c-inner} \ s \ p
  \item \text{and} \ phase: \ phase \ s \ p = 2
  \item \text{and} \ inv5-2: \ \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \quad \text{bal}(d\text{block} \ s \ p) < \text{mbal}(d\text{isk} \ s \ d \ q) \\
\wedge \neg \text{hasRead} \ s \ p \ d \ q)
\end{itemize}
\textbf{shows} \ \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \quad \text{bal}(d\text{block} \ s' \ p) < \text{mbal}(d\text{isk} \ s' \ d \ q) \\
\wedge \neg \text{hasRead} \ s' \ p \ d \ q)

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2ReadThen-}\text{H}\text{Inv5-p}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ r
  \item \text{and} \ inv: \ H\text{Inv5-inner} \ s \ p
  \item \text{and} \ inv4: \ H\text{Inv4c} \ s \ p
  \item \text{and} \ inv2c: \ H\text{Inv2c-inner} \ s \ p
\end{itemize}
\textbf{shows} \ H\text{Inv5-inner} \ s' \ p

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2ReadThen-allBlocks}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ r
\end{itemize}
\textbf{shows} \ allBlocks \ s' \subseteq \ allBlocks \ s

(\text{proof})

\textbf{lemma} \ H\text{Phase1or2ReadThen-}\text{H}\text{Inv5-q2}:
\begin{itemize}
  \item \textbf{assumes} act: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ r
\end{itemize}

\textbf{lemma} H\text{Phase1or2ReadThen-}\text{H}\text{Inv5-q2}:
and \( pnq: p \neq q \)
and \( inv4a: HInv4a s p \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s q) < \text{mbal}(\text{disk} s d qq) \quad \land \neg \text{hasRead} s q d qq) \)
shows \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s' q) < \text{mbal}(\text{disk} s' d qq) \quad \land \neg \text{hasRead} s' q d qq) \)

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

**Lemma HPhase1or2ReadThen-HInv5-q:**
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv5-inner s q
and inv4a: HInv4a s p
and pnq: \( p \neq q \)
shows HInv5-inner s' q
\( \langle \text{proof} \rangle \)

**Theorem HPhase1or2ReadThen-HInv5:**
\[ \begin{align*}
\{ \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r; \text{HInv5-inner} \ s \ q; \\
\text{Inv2c} \ s; \text{HInv4c} \ s \ p; \text{HInv4a} \ s \ p \} \Rightarrow \text{HInv5-inner} \ s' \ q
\end{align*} \]
\( \langle \text{proof} \rangle \)

**Theorem HPhase1or2ReadElse-HInv5:**
\[ \begin{align*}
\{ \text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ r; \text{HInv5-inner} \ s \ q \} \Rightarrow \text{HInv5-inner} \ s' \ q
\end{align*} \]
\( \langle \text{proof} \rangle \)

**Lemma HEndPhase2-HInv5-p:**
HEndPhase2 s s' p \( \Rightarrow \) 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 \( pnq: p \neq q \)
and \( inv5-1: \text{maxBalInp} \ s (\text{bal}(dblock s q)) (\text{inp}(dblock s q)) \)
shows maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
\( \langle \text{proof} \rangle \)

**Lemma HEndPhase2-HInv5-q2:**
assumes act: HEndPhase2 s s' p
and \( pnq: p \neq q \)
and \( inv5-2: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \quad \text{bal}(dblock s q) < \text{mbal}(\text{disk} s d qq) \quad \land \neg \text{hasRead} s q d qq) \)
\( \langle \text{proof} \rangle \)

54
shows $\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(d\text{block } s' \ q) < \text{mbal}(d\text{isk } s' \ d \ qq) \wedge \neg\text{hasRead } s' \ q \ d \ qq)$

\[ \langle \text{proof} \rangle \]

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

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

\[ \langle \text{proof} \rangle \]

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

\[ \langle \text{proof} \rangle \]

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

\[ \langle \text{proof} \rangle \]

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

lemma $\text{HEndPhase1-allBlocks}$:
\begin{align*}
\text{assumes } & \text{act}: \text{HEndPhase1 } s \ s' \ p \\
\text{shows } & \text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ d\text{block } s' \ p \} \\
\langle \text{proof} \rangle
\end{align*}

lemma $\text{HEndPhase1-HInv5-q}$:
\begin{align*}
\text{assumes } & \text{act}: \text{HEndPhase1 } s \ s' \ p \\
\text{and } & \text{inv}: \text{HInv5 } s \\
\text{and } & \text{inv1}: \text{Inv1 } s \\
\text{and } & \text{inv2a}: \text{Inv2a } s' \\
\text{and } & \text{inv2a-q}: \text{Inv2a' } s \\
\text{and } & \text{inv2b}: \text{Inv2b } s \\
\text{and } & \text{inv2c}: \text{Inv2c } s
\end{align*}
and $inv_3$: $Hinv_3\ s$

and $phase'$: $phase\ s'\ q = 2$

and $pq: p\neq q$

and $asm_4$: $\neg\maxBalInp\ s'(\text{bal}(\text{dblock}\ s'\ q))\ (\text{inp}(\text{dblock}\ s'\ q))$

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

\neg\text{hasRead}\ s'\ q\ d\ qq))$

(proof)

\textbf{theorem} $HEndPhase1-HInv5$:  
\textbf{assumes} act: $HEndPhase1\ s\ s'\ p$

and $inv_1$: $ Inv_1\ s$

and $inv_2$: $ Inv_2\ a\ s$

and $inv_2'$: $ Inv_2\ a\ s'$

and $inv_2b$: $ Inv_2\ b\ s$

and $inv_2c$: $ Inv_2\ c\ s$

and $inv_3$: $Hinv_3\ s$

and $inv_4$: $Hinv_4\ s$

shows $Hinv_5-\text{inner}\ s'\ q$

(proof)

\textbf{lemma} $HFail-HInv5-p$:  
$HFail\ s\ s'\ p \implies Hinv_5-\text{inner}\ s'\ p$

(proof)

\textbf{lemma} $HFail-blocksOf-q$:  
\textbf{assumes} act: $HFail\ s\ s'\ p$

and $pq: p\neq q$

shows $blocks_\ s'\ q \subseteq blocks_\ s\ q$

(proof)

\textbf{lemma} $HFail-allBlocks$:  
\textbf{assumes} act: $HFail\ s\ s'\ p$

shows $allBlocks_\ s' \subseteq allBlocks_\ s \cup \{\text{dblock}_\ s'\ p\}$

(proof)

\textbf{lemma} $HFail-HInv5-q1$:  
\textbf{assumes} act: $HFail\ s\ s'\ p$

and $pq: p\neq q$

and $inv_2a$: $ Inv_2a-\text{inner}\ s'\ q$

and $inv_5-1$: $\maxBalInp\ s(\text{bal}(\text{dblock}\ s\ q))\ (\text{inp}(\text{dblock}\ s\ q))$

shows $\maxBalInp\ s'(\text{bal}(\text{dblock}\ s'\ q))\ (\text{inp}(\text{dblock}\ s'\ q))$

(proof)

\textbf{lemma} $HFail-HInv5-q2$:  
\textbf{assumes} act: $HFail\ s\ s'\ p$

and $pq: p\neq q$

and $inv_5-2$: $\exists D\in\text{MajoritySet}.\ \exists qq.\ (\forall d\in D.\ \text{bal}(\text{dblock}\ s\ q) < \text{mbal}(\text{disk}\ s\ d\ qq)$
\(\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock } s' q) < \text{mbal}(\text{disk } s' d qq) \wedge \neg \text{hasRead } s q d qq)\)

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

\textbf{lemma HFail-HInv5-q:}
\begin{itemize}
\item \textbf{assumes} act: HFail s s' p
\item \textbf{and} inv: HInv5-inner s q
\item \textbf{and} pnq: p \neq q
\item \textbf{shows} HInv5-inner s' q
\end{itemize}
\(\langle \text{proof} \rangle\)

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

\textbf{lemma HPhase0Read-HInv5-p:}
\[ \text{HPhase0Read } s s' p d \implies \text{HInv5-inner } s' q \]
\(\langle \text{proof} \rangle\)

\textbf{lemma HPhase0Read-allBlocks:}
\begin{itemize}
\item \textbf{assumes} act: HPhase0Read s s' p d
\item \textbf{shows} allBlocks s' \subseteq allBlocks s
\end{itemize}
\(\langle \text{proof} \rangle\)

\textbf{lemma HPhase0Read-HInv5-1:}
\begin{itemize}
\item \textbf{assumes} act: HPhase0Read s s' p d
\item \textbf{and} inv5-1: \text{maxBal} s (\text{bal}(\text{dblock } s q)) (\text{inp}(\text{dblock } s q))
\item \textbf{shows} \text{maxBal} s' (\text{bal}(\text{dblock } s' q)) (\text{inp}(\text{dblock } s' q))
\end{itemize}
\(\langle \text{proof} \rangle\)

\textbf{lemma HPhase0Read-HInv5-q2:}
\begin{itemize}
\item \textbf{assumes} act: HPhase0Read s s' p d
\item \textbf{and} pnq: p \neq q
\item \textbf{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) \wedge \neg \text{hasRead } s q d qq)\]
\item \textbf{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) \wedge \neg \text{hasRead } s' q d qq)
\end{itemize}
\(\langle \text{proof} \rangle\)

\textbf{lemma HPhase0Read-HInv5-q:}
\begin{itemize}
\item \textbf{assumes} act: HPhase0Read s s' p d
\item \textbf{and} inv: HInv5-inner s q
\item \textbf{and} pnq: p \neq q
\item \textbf{shows} HInv5-inner s' q
\end{itemize}
\(\langle \text{proof} \rangle\)
theorem HPhase0Read-HInv5:
\[[ HPhase0Read \ s \ s' \ p \ d; \ HInv5-inner \ s \ q ] \implies HInv5-inner \ s' \ q\]
⟨proof⟩

lemma HEndPhase0-HInv5-p:
HEndPhase0 \ s \ s' \ p \implies HInv5-inner \ s' \ p
⟨proof⟩

lemma HEndPhase0-blocksOf-q:
assumes act: HEndPhase0 \ s \ s' \ p
and pq: p\neq q
shows blocksOf \ s' \ q \subseteq blocksOf \ s \ q
⟨proof⟩

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

lemma HEndPhase0-HInv5-q1:
assumes act: HEndPhase0 \ s \ s' \ p
and pq: p\neq q
and inv1: Inv1 \ s
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 HEndPhase0-HInv5-q2:
assumes act: HEndPhase0 \ s \ s' \ p
and pq: p\neq q
and inv5-2: \exists \ D \in MajoritySet. \ \exists \ qq. (\forall d \in D. bal(dblock \ s \ q) < mbal(disk \ s \ d \ qq)
\land \neg hasRead \ s \ q \ d \ qq)
shows \exists \ D \in MajoritySet. \ \exists \ qq. (\forall d \in D. bal(dblock \ s' \ q) < mbal(disk \ s' \ d \ qq)
\land \neg hasRead \ s' \ q \ d \ qq)
⟨proof⟩

lemma HEndPhase0-HInv5-q:
assumes act: HEndPhase0 \ s \ s' \ p
and inv: HInv5-inner \ s \ q
and inv1: Inv1 \ s
and pq: p\neq q
shows HInv5-inner \ s' \ q
⟨proof⟩

theorem HEndPhase0-HInv5:
\[[ HEndPhase0 \ s \ s' \ p; \ HInv5-inner \ s \ q; Inv1 \ s ] \implies HInv5-inner \ s' \ q\]
⟨proof⟩
\( HInv_1 \land HInv_2 \land HInv_3 \land HInv_4 \land HInv_5 \) is an invariant of \( HNext \).

**lemma I2e:**

**assumes** \( nxt: HNext s s' \)

**and** \( inv: HInv_1 s \land HInv_2 s \land HInv_2 s' \land HInv_3 s \land HInv_4 s \land HInv_5 s \)

**shows** \( HInv_5 s' \)

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

end

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

**C.6 Lemma I2f**

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

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

where

\[
valueChosen s v = \exists b \in (\bigcup p. \text{Ballot } p). \text{maxBalInp } s \ b \ v \\
\land (\exists p. \exists D \in \text{MajoritySet}. (\forall d \in D. b \leq \text{bal}(disk s d p)) \\
\land (\forall q. (\text{phase } s q = 1 \\
\land b \leq \text{mbal}(\text{dblock } s q) \\
\land \text{hasRead } s q d p)) \\
\rightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal}(\text{block } br))
\]

**lemma HEndPhase1-valueChosen-inp:**

**assumes** \( act: HEndPhase1 s s' q \)

**and** \( inv2a: Inv2a s \)

**and** \( asm1: b \in (\bigcup p. \text{Ballot } p) \)

**and** \( bk-blocksOf: bk \in \text{blocksOf } s r \)

**and** \( bk: bk \in \text{blocksSeen } s q \)

**and** \( b-bal: b \leq \text{bal } bk \)

**and** \( asm3: \text{maxBalInp } s \ b \ v \)

**and** \( inv1: Inv1 s \)

**shows** \( \text{inp}(\text{dblock } s' q) = v \)

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

**lemma HEndPhase1-maxBalInp:**

**assumes** \( act: HEndPhase1 s s' q \)

**and** \( asm1: b \in (\bigcup p. \text{Ballot } p) \)

**and** \( asm2: D \in \text{MajoritySet} \)

**and** \( asm3: \text{maxBalInp } s \ b \ v \)

**and** \( asm4: \forall d \in D. b \leq \text{bal}(\text{disk } s d p) \)
\[(\forall q . (\text{phase} s q = 1 \land b \leq \text{mbal} (\text{dblock} s q) \land \text{hasRead} s q d p) \implies (\exists br \in \text{blocksRead} s q d. b \leq \text{bal} (\text{block} br)))\]

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

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

shows \(?P s'\)

\[\text{theorem } \text{HEndPhase1-valueChosen}\]
assumes act: HEndPhase1 s s' q
and vc: valueChosen s v
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2b: Inv2b s
and v-input: v \in Inputs
shows valueChosen s' v

\[\text{lemma } \text{HStartBallot-maxBalImp}\]
assumes act: HStartBallot s s' q
and asm3: maxBalImp s b v
shows maxBalImp s' b v

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

shows \(?P s'\)

\[\text{theorem } \text{HStartBallot-valueChosen}\]
assumes act: HStartBallot s s' q
lemma \textit{HPhase1or2Write-maxBalInp}:
\begin{itemize}
\item assumes \textit{act}: \textit{HPhase1or2Write} $s$ $s'$ $q$ $d$
\item and \textit{asm3}: \textit{maxBalInp} $s$ $b$ $v$
\end{itemize}
shows \textit{maxBalInp} $s'$ $b$ $v$
\hspace{1cm} (proof)

lemma \textit{HPhase1or2Write-valueChosen2}:
\begin{itemize}
\item assumes \textit{act}: \textit{HPhase1or2Write} $s$ $s'$ $pp$ $d$
\item and \textit{asm2}: $D \in \text{MajoritySet}$
\item and \textit{asm4}: $\forall d \in D. \ b \leq \text{bal}(\text{disk} s d p)$
\item \hspace{1cm} $\land (\forall q. (\text{phase} s q = 1$
\item \hspace{1cm} $\land b \leq \text{mbal}(\text{dblock} s q)$
\item \hspace{1cm} $\land \text{hasRead} s q d p$)
\item \hspace{1cm} $\rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br)))$ (is $\mathcal{P} s$)
\end{itemize}
\begin{itemize}
\item and \textit{inv4}: \textit{HInv4a} $s$ $pp$
\item shows $\mathcal{P} s'$
\end{itemize}
\hspace{1cm} (proof)

theorem \textit{HPhase1or2Write-valueChosen}:
\begin{itemize}
\item assumes \textit{act}: \textit{HPhase1or2Write} $s$ $s'$ $q$ $d$
\item and \textit{vc}: \textit{valueChosen} $s$ $v$
\item and \textit{v-input}: $v \in \text{Inputs}$
\item and \textit{inv4}: \textit{HInv4a} $s$ $q$
\item shows \textit{valueChosen} $s'$ $v$
\end{itemize}
\hspace{1cm} (proof)

lemma \textit{HPhase1or2ReadThen-maxBalInp}:
\begin{itemize}
\item assumes \textit{act}: \textit{HPhase1or2ReadThen} $s$ $s'$ $q$ $d$ $p$
\item and \textit{asm3}: \textit{maxBalInp} $s$ $b$ $v$
\end{itemize}
shows \textit{maxBalInp} $s'$ $b$ $v$
\hspace{1cm} (proof)

lemma \textit{HPhase1or2ReadThen-valueChosen2}:
\begin{itemize}
\item assumes \textit{act}: \textit{HPhase1or2ReadThen} $s$ $s'$ $q$ $d$ $pp$
\item and \textit{asm4}: $\forall d \in D. \ b \leq \text{bal}(\text{disk} s d p)$
\item \hspace{1cm} $\land (\forall q. (\text{phase} s q = 1$
\item \hspace{1cm} $\land b \leq \text{mbal}(\text{dblock} s q)$
\item \hspace{1cm} $\land \text{hasRead} s q d p$)
\item \hspace{1cm} $\rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br)))$ (is $\mathcal{P} s$)
\item shows $\mathcal{P} s'$
\end{itemize}
\hspace{1cm} (proof)

theorem \textit{HPhase1or2ReadThen-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))) (is ?P s)
s 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))) (is ?P s)
s shows ?P s'

\textbf{theorem} \texttt{HFail-valueChosen}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HFail s s' q}
\item \texttt{and} \texttt{vc: valueChosen s v}
\item \texttt{and} \texttt{v-input: v ∈ Inputs}
\item \texttt{shows} \texttt{valueChosen s' v}
\end{itemize}
\texttt{⟨proof⟩}

\textbf{lemma} \texttt{HPhase0Read-maxBalInp}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HPhase0Read s s' q d}
\item \texttt{and} \texttt{asm3: maxBalInp s b v}
\item \texttt{shows} \texttt{maxBalInp s' b v}
\end{itemize}
\texttt{⟨proof⟩}

\textbf{lemma} \texttt{HPhase0Read-valueChosen2}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HPhase0Read s s' q d}
\item \texttt{and} \texttt{asm4: \(\forall d ∈ D. \ b ≤ bal(disk s d p)\)}
\item \texttt{and} \texttt{phase s q = 1}
\item \texttt{and} \texttt{b ≤ mbal(dblock s q)}
\item \texttt{and} \texttt{hasRead s q d p}
\item \texttt{shows} \texttt{?P s'}
\end{itemize}
\texttt{⟨proof⟩}

\textbf{theorem} \texttt{HPhase0Read-valueChosen}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HPhase0Read s s' q d}
\item \texttt{and} \texttt{vc: valueChosen s v}
\item \texttt{and} \texttt{v-input: v ∈ Inputs}
\item \texttt{shows} \texttt{valueChosen s' v}
\end{itemize}
\texttt{⟨proof⟩}

\textbf{lemma} \texttt{HEndPhase0-maxBalInp}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HEndPhase0 s s' q}
\item \texttt{and} \texttt{asm3: maxBalInp s b v}
\item \texttt{and} \texttt{inv1: Inv1 s}
\item \texttt{shows} \texttt{maxBalInp s' b v}
\end{itemize}
\texttt{⟨proof⟩}

\textbf{lemma} \texttt{HEndPhase0-valueChosen2}:
\begin{itemize}
\item \texttt{assumes} \texttt{act: HEndPhase0 s s' q}
\item \texttt{and} \texttt{asm4: \(\forall d ∈ D. \ b ≤ bal(disk s d p)\)}
\item \texttt{and} \texttt{phase s q = 1}
\item \texttt{and} \texttt{b ≤ mbal(dblock s q)}
\item \texttt{and} \texttt{hasRead s q d p}
\item \texttt{shows} \texttt{?P s'}
\end{itemize}
\texttt{⟨proof⟩}
theorem HEndPhase0-valueChosen:
  assumes act: HEndPhase0 s s' q
  and vc: valueChosen s v
  and v-input: v ∈ Inputs
  and inv1: Inv1 s
  shows valueChosen s' v
⟨proof⟩

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

The final conjunct of HInv asserts that, once an output has been cho-

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

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

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

64
lemma valueChosen-equal:
assumes v: valueChosen s v
and w: valueChosen s w
shows v = w \ (proof)

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

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

lemma outpt-chosen:
assumes outpt: outpt s = outpt s'
and inv2c: Inv2c s
and nextp: HNextPart s s'
shows chosen s' = chosen s
\ (proof)

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

theorem HStartBallot-Inv6:
assumes act: HStartBallot s s' p
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'
\ (proof)

theorem HPhase1or2Write-Inv6:
assumes act: HPhase1or2Write s s' p d
and inv: HInv6 s
and inv4: HInv4a s p

65
and \( \text{inv2c}: \text{Inv2c } s \)

shows \( \text{HInv6 } s' \)

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

\textbf{theorem} \( \text{HPHase1or2ReadThen-Inv6}: \)

\textbf{assumes} \( \text{act}: \text{HPHase1or2ReadThen } s \ s' \ p \ d \ q \)

\textbf{and} \( \text{inv}: \text{HInv6 } s \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{shows} \( \text{HInv6 } s' \)

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

\textbf{theorem} \( \text{HPHase1or2ReadElse-Inv6}: \)

\textbf{assumes} \( \text{act}: \text{HPHase1or2ReadElse } s \ s' \ p \ d \ q \)

\textbf{and} \( \text{inv}: \text{HInv6 } s \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{shows} \( \text{HInv6 } s' \)

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

\textbf{theorem} \( \text{HEndPhase1-Inv6}: \)

\textbf{assumes} \( \text{act}: \text{HEndPhase1 } s \ s' \ p \)

\textbf{and} \( \text{inv}: \text{HInv6 } s \)

\textbf{and} \( \text{inv1}: \text{Inv1 } s \)

\textbf{and} \( \text{inv2a}: \text{Inv2a } s \)

\textbf{and} \( \text{inv2b}: \text{Inv2b } s \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{shows} \( \text{HInv6 } s' \)

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

\textbf{lemma} \( \text{outpt-chosen-2}: \)

\textbf{assumes} \( \text{outpt}: \text{outpt } s' = (\text{outpt } s) \ (p:= \text{NotAnInput}) \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{and} \( \text{nextp}: \text{HNextPart } s \ s' \)

\textbf{shows} \( \text{chosen } s = \text{chosen } s' \)

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

\textbf{lemma} \( \text{outpt-HInv6-2}: \)

\textbf{assumes} \( \text{outpt}: \text{outpt } s' = (\text{outpt } s) \ (p:= \text{NotAnInput}) \)

\textbf{and} \( \forall p. \ \text{outpt } s \ p \in \{\text{chosen } s, \text{NotAnInput}\} \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{and} \( \text{nextp}: \text{HNextPart } s \ s' \)

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

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

\textbf{theorem} \( \text{HFail-Inv6}: \)

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

\textbf{and} \( \text{inv}: \text{HInv6 } s \)

\textbf{and} \( \text{inv2c}: \text{Inv2c } s \)

\textbf{shows} \( \text{HInv6 } s' \)

\( \langle \text{proof} \rangle \)
\textbf{C.8 The Complete Invariant}

\textbf{definition} $HInv :: \text{state} \Rightarrow \text{bool}$
\textbf{where}
\begin{align*}
HInv s &= (HInv_1 s \\
&\quad \land HInv_2 s \\
&\quad \land HInv_3 s \\
&\quad \land HInv_4 s \\
&\quad \land HInv_5 s \\
&\quad \land HInv_6 s)
\end{align*}

\textbf{theorem} I1:\n$HInit s \Rightarrow HInv s$
\textbf{(proof)}

\textbf{theorem} I2:\n\textbf{assumes} inv: $HInv s$
\textbf{and} nxt: $HNext s s'$
\textbf{shows} $HInv s'$
\textbf{(proof)}
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

definition 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)
\textbf{theorem} \textit{R1}: \\
\[
\left[ \text{HInit } s; \ is = s2is \ s \right] \implies \text{HInit } is
\]
\langle \text{proof} \rangle

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

\textbf{end}