Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff    Stephan Merz

September 13, 2023

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 HInv1 and HInv3) 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 \( \text{input}[p] \) that will not be modified, except possibly when recovering from a failure.

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

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

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

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

In each block, processors maintain three values:

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

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

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

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

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

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

$$HDiskSyno dSp e c \triangleq HInit \land \Box [HNext](vars, chosen, allInput)$$

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

Analogously, we have the specification of the Inner module:

$$ISp e c \triangleq IInit \land \Box [INext](input, output, chosen, allInput)$$

We define $ivars = \langle input, output, chosen, allInput \rangle$. In order to prove that $HDiskSyno dSpec$ implies $ISp e c$, we follow the structure of the proof given by Gafni and Lamport. We must prove two theorems:

**THEOREM R1** \[ HInit \Rightarrow IInit \]

**THEOREM R2** \[ HInit \land \Box [HNext](vars, chosen, allInput) \Rightarrow \Box [INext]ivars \]

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

**THEOREM R2a** \[ HInit \land \Box [HNext](vars, chosen, allInput) \Rightarrow \Box HInv \]

**THEOREM R2b** \[ HInv \land HInv' \land HNext \Rightarrow INext \lor (UNCHANGED ivars) \]

A predicate satisfying $HInv$ is said to be an invariant of $HDiskSyno dSpec$. To prove $R2a$, we make $HInv$ strong enough to satisfy:
The translation from TLA\(^+\) to Isabelle/HOL is pretty straightforward as Isabelle/HOL has equivalent counterparts for most of the constructs in TLA\(^+\) (some representative examples are shown in table 1). Nevertheless, there are some semantic discrepancies. In the following, we discuss these differences, some of the options that one has when dealing with them, and the reasons for our choices\(^1\).

### 3.1 Typed vs. Untyped

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

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

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

Isabelle/HOL:

typedec \text{InputsOrNi}

def inputs :: \text{InputsOrNi set} \\
\text{NotAnInput} :: \text{InputsOrNi}

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

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

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

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

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

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

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

Isabelle/HOL:

\[ \text{Phase1or2Write} :: \ state \Rightarrow \ state \Rightarrow \ Proc \Rightarrow \ Disk \Rightarrow \ bool \]
\[ \text{Phase1or2Write} s\ s' \ p \ d \equiv \]
\[ \land \ disk\ s' = (disk\ s) \ (d := (disk\ s\ d) \ (p := \ block\ s\ p)) \]
\[ \land \ disks\text{Written}\ s' = (disks\text{Written}\ s) \ (p := (disks\text{Written}\ s\ p) \cup \{d\}) \]
\[ \land \ inpt\ s' = \ inpt\ s \land \ outpt\ s' = \ outpt\ s \]
\[ \land \ phase\ s' = \ phase\ s \land \ block\ s' = \ block\ s \]
\[ \land \ blocks\text{Read}\ s' = \ blocks\text{Read}\ s \]

Figure 3: Translation of an action

3.2 Primed Variables

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

3.3 Restructuring the specification

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

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

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

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

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

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

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

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

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

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

we write:

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

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

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

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

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

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

Now we can express that we want to obtain the fact

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

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

4 Structure of the Correctness Proof

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

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

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase1$. 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**

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

```plaintext
VARIABLES allInput, chosen
```

```plaintext
```

```plaintext
IChoose(p) ∆= output[p] = NotAnInput ∧ IF chosen = NotAnInput
  THEN ip ∈ allInput : ∧ chosen' = ip ∧ output' = [output except ![p] = ip]
  ELSE ∧ output' = [output except ![p] = chosen] ∧ UNCHANGED chosen ∧ UNCHANGED ⟨input, allInput⟩
```

```plaintext
IFail(p) ∆= output' = [output except ![p] = NotAnInput] ∧ ∃ ip ∈ Inputs : ∧ input' = [input except ![p] = ip] ∧ allInput' = allInput ∪ {ip}
```

```plaintext
INext ∆= ∃ p ∈ Proc : IChoose(p) ∨ IFail(p)
ISpec ∆= IInit ∧ □[INext]⟨input, output, chosen, allInput⟩
```

```plaintext
IS(chosen, allInput) ∆= INSTANCE Inner
SynodSpec ∆= ∃ chosen, allInput : IS(chosen, allInput)!ISpec
```

---

B Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedecl InputsOrNi
typedecl Disk
typedecl Proc

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

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

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

proof (rule ccontr)
  assume b: b ∈ Ballot p
  and contr: ¬ (0 < b)
  from Ballot-nzero
  have 0 ∉ Ballot p ..
  with b contr
  show False
    by auto
qed

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

proof (auto)
  from majorities-intersect
  have IsMajority(()) ∧ IsMajority(()) → {} ∩ {} ≠ {} by auto
  thus IsMajority {} ⇒ False
    by auto
qed

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 =

14
(\text{disksWritten } s') = (\text{disksWritten } s)(p := \{\})
& \text{blocksRead } s' = (\text{blocksRead } s)(p := (\lambda d. \{\}))

\textbf{definition} \ \text{StartBallot} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\textbf{where}
StartBallot s s' p =
(\text{phase } s \in \{1, 2\})
& \text{phase } s' = (\text{phase } s)(p := 1)
& (\exists b \in \text{Ballot } p.
    \text{mbal } (\text{dblock } s p) < b
    & \text{dblock } s' = (\text{dblock } s)(p := (\text{dblock } s p)(\text{mbal } = b )))
& \text{InitializePhase } s s' p
& \text{inpt } s' = \text{inpt } s & \text{outpt } s' = \text{outpt } s & \text{disk } s' = \text{disk } s)

\textbf{definition} \ \text{Phase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}
\textbf{where}
Phase1or2Write s s' p d =
(\text{phase } s \in \{1, 2\})
& \text{disk } s' = (\text{disk } s)(d := (\text{disk } s d)(p := \text{dblock } s p))
& \text{disksWritten } s' = (\text{disksWritten } s)(p := (\text{disksWritten } s p) \cup \{d\})
& \text{inpt } s' = \text{inpt } s & \text{outpt } s' = \text{outpt } s
& \text{phase } s' = \text{phase } s & \text{dblock } s' = \text{dblock } s
& \text{blocksRead } s' = \text{blocksRead } s)

\textbf{definition} \ \text{Phase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\textbf{where}
Phase1or2ReadThen s s' p d q =
(d \in \text{disksWritten } s p
& \text{mbal}(\text{disk } s d q) < \text{mbal}(\text{dblock } s p)
& \text{blocksRead } s' = (\text{blocksRead } s)(p := (\text{blocksRead } s p)(d :=
    (\text{blocksRead } s p d) \cup \{\ block = \text{disk } s d q,
    \text{proc } = q \}))
& \text{inpt } s' = \text{inpt } s & \text{outpt } s' = \text{outpt } s
& \text{disk } s' = \text{disk } s & \text{phase } s' = \text{phase } s
& \text{dblock } s' = \text{dblock } s & \text{disksWritten } s' = \text{disksWritten } s)

\textbf{definition} \ \text{Phase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\textbf{where}
Phase1or2ReadElse s s' p d q =
(d \in \text{disksWritten } s p
& \text{StartBallot } s s' p)

\textbf{definition} \ \text{Phase1or2Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\textbf{where}
Phase1or2Read s s' p d q =
(\text{Phase1or2ReadThen } s s' p d q
\lor \text{Phase1or2ReadElse } s s' p d q)

\textbf{definition} \ \text{blocksSeen} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set}

15
where \( \text{blocksSeen} \ s \ p = \text{allBlocksRead} \ s \ p \cup \{ \text{dblock} \ s \ p \} \)

**Definition** nonInitBlks :: state \( \Rightarrow \) Proc \( \Rightarrow \) DiskBlock set

where\( \text{nonInitBlks} \ s \ p = \{ \text{bs} . \ \text{bs} \in \text{blocksSeen} \ s \ p \land \ \text{inp} \ \text{bs} \in \text{Inputs} \} \)

**Definition** maxBlk :: state \( \Rightarrow \) Proc \( \Rightarrow \) DiskBlock

where

\[
\text{maxBlk} \ s \ p =
\begin{cases}
(\text{SOME} \ b. \ b \in \text{nonInitBlks} \ s \ p \land (\forall c \in \text{nonInitBlks} \ s \ p. \ \text{bal} \ c \leq \text{bal} \ b))
\end{cases}
\]

**Definition** EndPhase1 :: state \( \Rightarrow \) state \( \Rightarrow \) Proc \( \Rightarrow \) bool

where

\[
\text{EndPhase1} \ s \ s' \ p =
\begin{cases}
(\text{IsMajority} \ \{ d . \ d \in \text{disksWritten} \ s \ p \land (\forall q \in \text{UNIV} - \{ p \}. \ \text{hasRead} \ s \ p \ d \ q)\})
\land \ \text{phase} \ s \ p = 1
\land \ \text{dblock} \ s' = (\text{dblock} \ s) \ (p := \text{dblock} \ s \ p)
\{ \ \text{bal} := \text{mbal}(\text{dblock} \ s \ p),
\text{inp} :=
\begin{cases}
(\text{if} \ \text{nonInitBlks} \ s \ p = \{\}
\land \ \text{inp} \text{inp} (\text{maxBlk} \ s \ p))
\end{cases}
\}
\land \ \text{outpt} \ s' = \text{outpt} \ s
\land \ \text{phase} \ s' = (\text{phase} \ s) \ (p := \text{phase} \ s \ p + 1)
\land \ \text{InitializePhase} \ s \ s' \ p
\land \ \text{inpt} \ s' = \text{inpt} \ s \land \ \text{disk} \ s' = \text{disk} \ s
\end{cases}
\]

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

where

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

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

where

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

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

where

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

**Definition** Phase0Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool

**Where**

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

**Definition** EndPhase0 :: state ⇒ state ⇒ Proc ⇒ bool

**Where**

\(\text{EndPhase0} s s' p =\)
\(\land \text{phase} s p = 0\)
\(\land \text{IsMajority} (\{d. \text{hasRead} s p d p\})\)
\(\land (\exists b \in \text{Ballot} p.\)
\(\land (\forall r \in \text{allBlocksRead} s p. \text{mbal} r < b)\)
\(\land \text{dblock} s' = (\text{dblock} s) (p :=\)
\(\land (\exists \text{some} r. r \in \text{allBlocksRead} s p\)
\(\land (\forall s \in \text{allBlocksRead} s p. \text{bal} s \leq \text{bal} r)) (\{m \text{bal} := b \})\))
\(\land \text{InitializePhase} s s' p\)
\(\land \text{phase} s' = (\text{phase} s) (p := 1)\)
\(\land \text{inpt} s' = \text{inpt} s \land \text{outpt} s' = \text{outpt} s \land \text{disk} s' = \text{disk} s\)

**Definition** Next :: state ⇒ state ⇒ bool

**Where**

\(\text{Next} s s' = (\exists p.\)
\(\lor (\exists d. \text{Phase0Read} s s' p d)\)
\(\lor (\exists q. q \neq p \land \text{Phase1or2Read} s s' p d q))\)
\(\lor \text{EndPhase1or2} s s' p\)
\(\lor \text{Fail} s s' p\)
\(\lor \text{EndPhase0} s s' p)\)

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

**Definition** HInit :: state ⇒ bool

**Where**

\(\text{HInit} s =\)
\(\text{Init} s\)
\(\& \text{chosen} s = \text{NotAnInput}\)
\(\& \text{allInput} s = \text{range} (\text{inpt} s))\)
HNextPart is the part of the Next action that is concerned with history variables.

**Definition**

**HNextPart** :: state ⇒ state => bool

where

\[
H\text{NextPart} s s' = (\text{chosen} s' = \\
(\text{if chosen} s \neq \text{NotAnInput} \lor (\forall p. \text{outpt} s' p = \text{NotAnInput}) \\
\text{then chosen} s \\
\text{else outpt} s' (\text{SOME} p. \text{outpt} s' p \neq \text{NotAnInput})) \\
\land \text{allInput} s' = \text{allInput} s \cup (\text{range (inpt} s')))
\]

**Definition**

**HNext** :: state ⇒ state ⇒ bool

where

\[
H\text{Next} s s' = (\text{Next} s s' \\
\land H\text{NextPart} s s')
\]

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

**Definition**

**HPhase1or2ReadThen** :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool

where

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

**Definition**

**HEndPhase1** :: state ⇒ state ⇒ Proc ⇒ bool

where

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

**Definition**

**HStartBallot** :: state ⇒ state ⇒ Proc ⇒ bool

where

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

**Definition**

**HPhase1or2Write** :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool

where

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

**Definition**

**HPhase1or2ReadElse** :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool

where

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

**Definition**

**HEndPhase2** :: state ⇒ state ⇒ Proc ⇒ bool

where

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

**Definition**

**HFail** :: state ⇒ state ⇒ Proc ⇒ bool

where

\[
H\text{Fail} s s' p = (\text{Fail} s s' p \land H\text{NextPart} s s')
\]
definition

\( H_{\text{Phase0Read}} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \) where

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

definition

\( H_{\text{EndPhase0}} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \) where

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

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

declare \( H_{\text{Phase1or2ReadThen-def}} \) [simp]
declare \( H_{\text{Phase1or2ReadElse-def}} \) [simp]
declare \( H_{\text{EndPhase1-def}} \) [simp]
declare \( H_{\text{StartBallot-def}} \) [simp]
declare \( H_{\text{Phase1or2Write-def}} \) [simp]
declare \( H_{\text{EndPhase2-def}} \) [simp]
declare \( H_{\text{Fail-def}} \) [simp]
declare \( H_{\text{Phase0Read-def}} \) [simp]
declare \( H_{\text{EndPhase0-def}} \) [simp]

declare \( \text{end} \)

C Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1 Invariant 1

This is just a type Invariant.

definition \( \text{Inv1} :: \text{state} \Rightarrow \text{bool} \) where

\( \text{Inv1} s = (\forall p. \langle\text{inpt} s p \in \text{Inputs} \land \text{phase} s p \leq 3 \land \text{finite} (\text{allRdBlks} s p)\rangle) \)

definition \( \text{HInv1} :: \text{state} \Rightarrow \text{bool} \) where

\( \text{HInv1} s = (\text{Inv1} s \land \text{allInput} s \subseteq \text{Inputs}) \)

declare \( \text{HInv1-def [simp]} \)

We added the assertion that the set \( \text{allRdBlks} s p \) is finite for every process \( p \); one may therefore choose a block with a maximum ballot number in action \( \text{EndPhase1} \).
With the following the lemma, it will be enough to prove Inv1 s’ for every action, without taking the history variables in account.

**Lemma HNextPart-Inv1:** \[ \[ HInv1 s; HNextPart s s'; Inv1 s' \] \implies HInv1 s' \]

**Proof**

\begin{verbatim}
by(auto simp add: HNextPart-def Inv1-def)
\end{verbatim}

**Theorem HInit-HInv1:** \( HInit s \rightarrow HInv1 s \)

**Proof**

\begin{verbatim}
by(auto simp add: HInit-def Inv1-def Init-def allRdBlks-def)
\end{verbatim}

**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**

\begin{verbatim}
fix pp
from inv
have \( \forall p. \text{finite } (\text{allRdBlks } s p) \)
  by(auto simp add: Inv1-def)
  hence \( \text{finite } (\text{allRdBlks } s pp) \)
  by blast
with asm
show \( \text{finite } (\text{allRdBlks } s' pp) \)
  by(auto intro: finite-subset)
qed
\end{verbatim}

**Theorem HPhase1or2ReadThen-HInv1:**

**Assumes** inv1: \( HInv1 s \)

**And** act: \( HPhase1or2ReadThen s s' p d q \)

**Shows** \( HInv1 s' \)

**Proof**

\begin{verbatim}
−
  we focus on the last conjunct of Inv1
  from act
  have \( \forall p. \text{allRdBlks } s' p \subseteq \text{allRdBlks } s p \cup \{ (\text{block } = \text{disk } s d q, \text{proc } = q) \} \)
    by(auto simp add: Phase1or2ReadThen-def allRdBlks-def split: if-split-asm)
  with inv1
  have \( \forall p. \text{finite } (\text{allRdBlks } s' p) \)
    by(blast dest: allRdBlks-finite)
  − the others conjuncts are trivial
  with inv1 act
  show \( ?thesis \)
  by(auto simp add: Inv1-def Phase1or2ReadThen-def HNextPart-def)
qed
\end{verbatim}

**Theorem HEndPhase1-HInv1:**

**Assumes** inv1: \( HInv1 s \)

**And** act: \( HEndPhase1 s s' p \)

**Shows** \( HInv1 s' \)

**Proof**

\begin{verbatim}
−
  from inv1 act
\end{verbatim}

20
have Inv1 s'
  by (auto simp add: Inv1-def EndPhase1-def InitializePhase-def allRdBlks-def)
with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def dest: HNextPart-Inv1)
qed

theorem HStartBallot-HInv1:
  assumes inv1: HInv1 s
  and act: HStartBallot s s' p
  shows HInv1 s'
proof –
  from inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def StartBallot-def InitializePhase-def allRdBlks-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HPhase1or2Write-HInv1:
  assumes inv1: HInv1 s
  and act: HPhase1or2Write s s' p d
  shows HInv1 s'
proof –
  from inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def Phase1or2Write-def allRdBlks-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HPhase1or2ReadElse-HInv1:
  assumes act: HPhase1or2ReadElse s s' p d q
  and inv1: HInv1 s
  shows HInv1 s'
  using HStartBallot-HInv1[OF inv1] act
  by (auto simp add: Phase1or2ReadElse-def)

theorem HEndPhase2-HInv1:
  assumes inv1: HInv1 s
  and act: HEndPhase2 s s' p
  shows HInv1 s'
proof –
  from inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def EndPhase2-def InitializePhase-def allRdBlks-def)
  with inv1 act
show ?thesis
  by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HFail-HInv1:
  assumes inv1: HInv1 s
  and act: HFail s s' p
  shows HInv1 s'
proof –
  from inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def Fail-def InitializePhase-def allRdBlks-def)
  with inv1 act show ?thesis
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HPhase0Read-HInv1:
  assumes inv1: HInv1 s
  and act: HPhase0Read s s' p d
  shows HInv1 s'
proof –
  — we focus on the last conjunct of Inv1
  from act
  have ∀ pp. allRdBlks s' pp ⊆ allRdBlks s pp ∪ {(|block = disk s d p, proc = p|)}
    by (auto simp add: Phase0Read-def allRdBlks-def
      split: if-split-asm)
  with inv1
  have ∀ p. finite (allRdBlks s' p)
    by (blast dest: allRdBlks-finite)
  — the others conjuncts are trivial
  with inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def Phase0Read-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HEndPhase0-HInv1:
  assumes inv1: HInv1 s
  and act: HEndPhase0 s s' p
  shows HInv1 s'
proof –
  from inv1 act
  have Inv1 s'
    by (auto simp add: Inv1-def EndPhase0-def allRdBlks-def InitializePhase-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
declare \( HInv1 \)-def [simp del]

\( HInv1 \) is an invariant of \( HNext \)

lemma I2a:
assumes \( \text{nxt: } HNext \ s s' \)
and \( \text{inv: } HInv1 \ s \)
shows \( HInv1 \ s' \)
using assms
by (auto
  simp add: HNext-def Next-def,
  auto intro: HStartBallot-HInv1,
  auto intro: HPhase0Read-HInv1,
  auto intro: HPhase1or2Write-HInv1,
  auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-HInv1
    HPhase1or2ReadElse-HInv1,
  auto simp add: EndPhase1or2-def
    intro: HEndPhase1-HInv1
    HEndPhase2-HInv1,
  auto intro: HFail-HInv1,
  auto intro: HEndPhase0-HInv1)

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 :: \( \text{state \Rightarrow Proc \Rightarrow Proc \Rightarrow Disk \Rightarrow BlockProc set} \)
where
rdBy s p q d = 
  \{ br . \ br \in \text{blocksRead } s q d \land \text{proc } br = p \}

definition blocksOf :: \( \text{state \Rightarrow Proc \Rightarrow DiskBlock set} \)
where
blocksOf s p =
  \{ \text{dblock } s p \}
  \cup \{ \text{disk } s d p \mid d . \ d \in \text{UNIV} \}
  \cup \{ \text{block } br \mid br . \ br \in (\text{UN } q d . \ \text{rdBy } s p q d) \} \}

definition allBlocks :: \( \text{state \Rightarrow DiskBlock set} \)
where allBlocks s = (UN p. blocksOf s p)

**definition Inv2a-innermost :: state ⇒ Proc ⇒ DiskBlock ⇒ bool**
where
Inv2a-innermost s p bk =
  (mbal bk ∈ (Ballot p) ∪ {0})
∧ (bal bk ∈ (Ballot p) ∪ {0})
∧ (bal bk = 0) = (inp bk = NotAnInput)
∧ bal bk ≤ mbal bk
∧ inp bk ∈ (allInput s) ∪ {NotAnInput})

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

**definition Inv2b :: state ⇒ bool**
where
Inv2b s = (∀ p. 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 p))
∧ (phase s p ≠ 0 →
   (mbal(dblock s p) ∈ Ballot p
∧ bal(dblock s p) ∈ Ballot p ∪ {0})
∧ (∀ d, ∀ br ∈ blocksRead s p d.
mbal(block br) < mbal(dblock s p)))
∧ (phase s p ∈ {2, 3} → bal(dblock s p) = mbal(dblock s p))
∧ outpt s p = (if phase s p = 3 then inp(dblock s p) else NotAnInput)
∧ chosen s ∈ allInput s ∪ {NotAnInput}
∧ (∀ p. inpt s p ∈ allInput s
∧ (chosen s = NotAnInput → outpt s p = NotAnInput))

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

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

C.2.1 Proofs of Invariant 2 a

theorem \( \text{HInit-Inv2a} \): \( \text{HInit } s \Rightarrow \text{Inv2a } s \)
by (auto simp add: \( \text{HInit-def}\) Init-def Inv2a-def Inv2a-inner-def Inv2a-innermost-def rdBy-def blocksOf-def InitDB-def)

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

lemma \( \text{HPhase1or2ReadThen-blocksOf} \):
\( [HPhase1or2ReadThen \ s \ s' \ p \ d \ q] \Rightarrow \text{blocksOf } s' \ r \subseteq \text{blocksOf } s \ r \)
by (auto simp add: Phase1or2ReadThen-def blocksOf-def rdBy-def)

theorem \( \text{HPhase1or2ReadThen-Inv2a} \):
assumes \( \text{inv} : \text{Inv2a } s \) and \( \text{act} : \text{HPhase1or2ReadThen } s \ s' \ p \ d \ q \)
shows \( \text{Inv2a } s' \)
proof (clarsimp simp add: Inv2a-def)
fix \( pp \ bk \)
assume \( bk : bk \in \text{blocksOf } s' \ pp \)
with \( \text{inv} \) \( \text{HPhase1or2ReadThen-blocksOf[OF act]} \)
have \( \text{Inv2a-innermost } s \ pp \ bk \)
  by (auto simp add: Inv2a-def)
with \( \text{act} \)
show \( \text{Inv2a-innermost } s' \ pp \ bk \)
  by (auto simp add: Inv2a-innermost-def HNextPart-def)
qed

lemma \( \text{InitializePhase-rdBy} \):
\( \text{InitializePhase } s \ s' \ p \Rightarrow \text{rdBy } s' \ pp \ qq \ dd \subseteq \text{rdBy } s \ pp \ qq \ dd \)
by (auto simp add: InitializePhase-def rdBy-def)

lemma \( \text{HStartBallot-blocksOf} \):
\( \text{HStartBallot } s \ s' \ p \Rightarrow \text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \cup \{\text{dblock } s' \ q\} \)
by (auto simp add: StartBallot-def blocksOf-def dest: subsetD[OF InitializePhase-rdBy])

lemma \( \text{HStartBallot-Inv2a-dbloc} \):
assumes \( \text{act} : \text{HStartBallot } s \ s' \ p \)
and \( \text{inv2a} : \text{Inv2a-innermost } s \ p \ (\text{dblock } s \ p) \)
shows \textit{Inv2a-innermost} \(s'\ p\ (d\text{block} \ s' \ p)\)

\textbf{proof} –

\textbf{from} act
\textbf{have} \(mbal':\ mbal\ (d\text{block} \ s' \ p) \in \text{Ballot} \ p\)
\hspace{1em} by (auto simp add: \text{StartBallot-def})

\textbf{from} act
\textbf{have} \(bal':\ bal\ (d\text{block} \ s' \ p) = bal\ (d\text{block} \ s \ p)\)
\hspace{1em} by (auto simp add: \text{StartBallot-def})

\textbf{with} act
\textbf{have} \(inp':\ inp\ (d\text{block} \ s' \ p) = inp\ (d\text{block} \ s \ p)\)
\hspace{1em} by (auto simp add: \text{StartBallot-def})

\textbf{from} act
\textbf{have} \(mbal\ (d\text{block} \ s \ p) \leq mbal\ (d\text{block} \ s' \ p)\)
\hspace{1em} by (auto simp add: \text{StartBallot-def})

\textbf{with} act
\textbf{have} \(bal'\ \text{inv2a}\)
\hspace{1em} by (auto simp add: \text{StartBallot-def})

\textbf{from} act
\textbf{have} \(allInput \ s \subseteq allInput \ s'\)
\hspace{1em} by (auto simp add: \text{HNextPart-def})

\textbf{with} mbal' bal' inp' mbal \text{ inv2a}
\textbf{show} ?thesis
\hspace{1em} by (auto simp add: \text{Inv2a-innermost-def})

\\qed

\textbf{lemma} \textit{HStartBallot-Inv2a-dblock-q}:
\begin{itemize}
\item \textbf{assumes} act: \textit{HStartBallot} \(s\ s'\ p\)
\item \textbf{and} inv2a: \textit{Inv2a-innermost} \(s\ q\ (d\text{block} \ s \ q)\)
\item \textbf{shows} \textit{Inv2a-innermost} \(s'\ q\ (d\text{block} \ s' \ q)\)
\end{itemize}

\textbf{proof}(cases \(p=q\))
\begin{itemize}
\item \textbf{case} True
\hspace{1em} \textbf{with} act inv2a
\hspace{1em} \textbf{show} ?thesis
\hspace{1em} \hspace{1em} by (blast dest: \textit{HStartBallot-Inv2a-dblock})
\end{itemize}

\textbf{next}
\begin{itemize}
\item \textbf{case} False
\hspace{1em} \textbf{with} act inv2a
\hspace{1em} \textbf{show} ?thesis
\hspace{1em} \hspace{1em} by (clarsimp simp add: \text{StartBallot-def} \text{HNextPart-def} \text{InitializePhase-def} \text{Inv2a-innermost-def})
\end{itemize}

\\qed

\textbf{theorem} \textit{HStartBallot-Inv2a}:
\begin{itemize}
\item \textbf{assumes} inv: \textit{Inv2a} \(s\)
\item \textbf{and} act: \textit{HStartBallot} \(s\ s'\ p\)
\item \textbf{shows} \textit{Inv2a} \(s'\)
\end{itemize}

\textbf{proof} (clarsimp simp add: \textit{Inv2a-def} \textit{Inv2a-inner-def})
\begin{itemize}
\item \textbf{fix} \(q\ bk\)
\item \textbf{assume} \(bk: bk \in \text{blocksOf} \ s' \ q\)
\end{itemize}
with \( \text{inv} \) have \( \text{oldBlks} \): \( bk \in \text{blocksOf} \ s \ q \longrightarrow \text{Inv2a-innermost} \ s \ q \ bk \) by (auto simp add: Inv2a-def Inv2a-inner-def)

from \( bk \) \( \text{HStartBallot-blocksOf[OF act]} \)

have \( bk \in \{ \text{dblock} \ s' \ q \} \cup \text{blocksOf} \ s \ q \)
  by blast

thus \( \text{Inv2a-innermost} \ s' \ q \ bk \)

proof
  assume \( bk \)-\( \text{dblock} \): \( bk \in \{ \text{dblock} \ s' \ q \} \)
  from \( \text{inv} \)
  have \( \text{inv-q-\text{dblock}} \): \( \text{Inv2a-innermost} \ s \ q \ \text{dblock} \ s \ q \)
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)

  with \( \text{act} \) \( \text{inv} \) \( bk \)-\( \text{dblock} \)
  show \(?\text{thesis}\)
    by (blast dest: HStartBallot-Inv2a-\text{dblock-q})

next
  assume \( bk \)-\( \text{in-blocks} \): \( bk \in \text{blocksOf} \ s \ q \)
  with \( \text{oldBlks} \)
  have \( \text{Inv2a-innermost} \ s \ q \ bk \)
  with \( \text{act} \)
  show \(?\text{thesis}\)
    by (auto simp add: StartBallot-def HNextPart-def InitializePhase-def Inv2a-innermost-def)

qed

qed

lemma \( \text{HPhase1or2Write-blocksOf} \):
  \( [ \text{HPhase1or2Write} \ s \ s' \ p \ d ] \Longrightarrow \text{blocksOf} \ s' \ r \subseteq \text{blocksOf} \ s \ r \)
  by (auto simp add: Phase1or2Write-def blocksOf-def rdBy-def)

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

proof(clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix \( q \) \( bk \)
  assume \( bk \): \( bk \in \text{blocksOf} \ s' \ q \)
  from \( \text{inv} \) \( bk \) \( \text{HPhase1or2Write-blocksOf[OF act]} \)
  have \( \text{inp-q-bk} \): \( \text{Inv2a-innermost} \ s \ q \ bk \)
    by (auto simp add: Inv2a-def Inv2a-inner-def)
  with \( \text{act} \)
  show \( \text{Inv2a-innermost} \ s' \ q \ bk \)
    by (auto simp add: Inv2a-innermost-def HNextPart-def)

qed

theorem \( \text{HPhase1or2ReadElse-Inv2a} \):
  assumes \( \text{inv} \): \( \text{Inv2a} \ s \) and \( \text{act} \): \( \text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ q \)
  shows \( \text{Inv2a} \ s' \)
proof –
from act
have HStartBallot s s' p
  by (simp add: Phase1or2ReadElse-def)
with inv
show ?thesis
  by (auto elim: HStartBallot-Inv2a)
qed

lemma HEndPhase2-blocksOf:
  [ HEndPhase2 s s' p ] \implies blocksOf s' q \subseteq blocksOf s q
by (auto simp add: EndPhase2-def blocksOf-def
dest: subsetD[OF InitializePhase-rdBy])

theorem HEndPhase2-Inv2a:
  assumes inv: Inv2a s
  and act: HEndPhase2 s s' p
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk \in blocksOf s' q
  from inv bk HEndPhase2-blocksOf[OF act]
  have inp-q-bk: Inv2a-innermost s q bk
    by (auto simp add: Inv2a-def Inv2a-inner-def)
  with act
  show Inv2a-innermost s' q bk
    by (auto simp add: Inv2a-innermost-def HNextPart-def)
qed

lemma HFail-blocksOf:
  HFail s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
by (auto simp add: Fail-def blocksOf-def
dest: subsetD[OF InitializePhase-rdBy])

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 (cases p=q)
  case True
  with act
  have dblock s' q = InitDB
    by (simp add: Fail-def)
  with True
  show ?thesis
    by (auto simp add: InitDB-def Inv2a-innermost-def)
next
  case False
  with inv act
show \( ?\text{thesis} \)
  by (auto simp add: \text{Fail-def} \text{HNextPart-def} 
    InitializePhase-def Inv2a-innermost-def)
qed

\textbf{theorem} \text{HFail-Inv2a:}
\begin{align*}
\text{assumes inv: Inv2a s} \\
\text{and act: HFail s s' p} \\
\text{shows Inv2a s'}
\end{align*}
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk 
  assume bk: \( bk \in \text{blocksOf s'} q \) 
  with \text{HFail-blocksOf[OF act]} 
  have dblock-blocks: \( bk \in \{\text{dblock s'} q\} \cup \text{blocksOf s} q \) 
  by blast 
  thus Inv2a-innermost s' q bk 
proof 
  assume bk-dblock: \( bk \in \{\text{dblock s'} q\} \) 
  from inv have inv-q-dblock: Inv2a-innermost s q (dblock s q) 
  by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def) 
  with act bk-dblock 
  show \( ?\text{thesis} \) 
  by (blast dest: HFail-Inv2a-dblock-q)
next 
  assume bk-in-blocks: \( bk \in \text{blocksOf s} q \) 
  with inv 
  have Inv2a-innermost s q bk 
  by (auto simp add: Inv2a-def Inv2a-inner-def) 
  with act 
  show \( ?\text{thesis} \) 
  by (auto simp add: \text{Fail-def} \text{HNextPart-def} 
    InitializePhase-def Inv2a-innermost-def)
qed

\textbf{lemma} \text{HPhase0Read-blocksOf:}
\( \text{HPhase0Read s s' p d} \implies \text{blocksOf s'} q \subseteq \text{blocksOf s} q \)
by (auto simp add: Phase0Read-def InitializePhase-def blocksOf-def rdBy-def)

\textbf{theorem} \text{HPhase0Read-Inv2a:}
\begin{align*}
\text{assumes inv: Inv2a s} \\
\text{and act: HPhase0Read s s' p d} \\
\text{shows Inv2a s'}
\end{align*}
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk 
  assume bk: \( bk \in \text{blocksOf s'} q \) 
  from inv bk HPhase0Read-blocksOf[OF act]
have inp-q-bk: Inv2a-innermost s q bk
  by (auto simp add: Inv2a-def Inv2a-inner-def)
with act
show Inv2a-innermost s' q bk
  by (auto simp add: Inv2a-innermost-def HNextPart-def)
qed

lemma HEndPhase0-blocksOf:
HEndPhase0 s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
by (auto simp add: EndPhase0-def blocksOf-def
dest: subsetD[OF InitializePhase-rdBy])

lemma HEndPhase0-blocksRead:
assumes act: HEndPhase0 s s' p
shows \exists d. blocksRead s p d \neq \{\}
proof
  from act have IsMajority(\{d. hasRead s p d p\}) by (simp add: EndPhase0-def)
hence \{d. hasRead s p d p\} \neq {} by (rule majority-nonempty)
  thus \?thesis
    by (auto simp add: hasRead-def)
qed

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 (SOME b. b \in allBlocksRead s p
  \land (\forall t \in allBlocksRead s p. bal t \leq bal b)
  \land (\forall t \in allBlocksRead s p.
    bal t \leq bal (SOME b. b \in allBlocksRead s p
    \land (\forall t \in allBlocksRead s p. bal t \leq bal b))))
proof
  from inv1 have finite (bal \cdot allBlocksRead s p) (is finite \?S)
    by (simp add: Inv1-def allBlocksRead-def)
  moreover from HEndPhase0-blocksRead[OF act]
  have \?S \neq {}
    by (auto simp add: allBlocksRead-def allRdBlks-def)
  ultimately have Max \?S \in \?S and \forall t \in \?S. t \leq Max \?S by auto
  hence \exists r \in \?S, \forall t \in \?S. t \leq r ..
  then obtain mblk
  where mblk \in allBlocksRead s p
    \land (\forall t \in allBlocksRead s p. bal t \leq bal mblk) (is \ ?P mblk)

30
by auto
thus \(\text{thesis}\)
by (rule someI)
qed

lemma HEndPhase0-dblock-allBlocksRead:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
shows \(\text{dblock s'} p \in (\lambda x. x \ (mbal:= \text{mbal(dblock s'} p)))' \text{ allBlocksRead s p}\)
using act HEndPhase0-some[OF act inv1]
by (auto simp add: EndPhase0-def)

lemma HNextPart-allInput-or-NotAnInput:
assumes act: HNextPart s s' p
and inv2a: Inv2a-innermost s p (dblock s' p)
shows \(\text{inp (dblock s'} p) \in \text{allInput s'} \cup \{\text{NotAnInput}\}\)
proof -
from act have allInput s' = allInput s \cup (\text{range (inpt s')})
  by (simp add: HNextPart-def)
moreover from inv2a have \(\text{inp (dblock s'} p) \in \text{allInput s'} \cup \{\text{NotAnInput}\}\)
  by (simp add: Inv2a-innermost-def)
ultimately show \(\text{thesis}\)
by blast
qed

lemma HEndPhase0-Inv2a-allBlocksRead:
assumes act: HEndPhase0 s s' p
and inv2a: Inv2a-inner s p
and inv2c: Inv2c-inner s p
shows \(\forall t \in (\lambda x. x \ (mbal:= \text{mbal(dblock s'} p)))' \text{ allBlocksRead s p.}\)
\(\text{Inv2a-innermost s p t}\)
proof -
from act have \(\text{mbal'}: \text{mbal(dblock s'} p) \in \text{Ballot p}\)
  by (auto simp add: EndPhase0-def)
from inv2c act have allproc-p: \(\forall d. \forall br \in \text{blocksRead s p d}. \text{proc br = p}\)
  by (simp add: Inv2c-inner-def EndPhase0-def)
with inv2a have allBlocks-inv2a: \(\forall t \in \text{allBlocksRead s p.} \text{Inv2a-innermost s p t}\)
proof (auto simp add: Inv2a-inner-def allBlocksRead-def
  allRdBlks-def blocksOf-def rdBy-def)
fix d bk
assume bk-in-blocksRead: \(bk \in \text{blocksRead s p d}\)
and inv2a-bk: \(\forall x \in \{u. \exists d. u = \text{disk s d p}\}
  \cup \{\text{block br | br.} \exists q d. br \in \text{blocksRead s q d} \)
with allproc-p have proc bk = p by auto
with bk-in-blocksRead inv2a-bk
show Inv2a-innermost s p (block bk) by blast
qed

from act
have mbal'-gt: \forall bk \in allBlocksRead s p. mbal bk \leq mbal (dblock s' p) 
  by(auto simp add: EndPhase0-def)
with mbal' allBlocks-inv2a
show \?thesis
proof (auto simp add: Inv2a-innermost-def)
fix t
assume t-blocksRead: t \in allBlocksRead s p
with allBlocks-inv2a
have bal t \leq mbal t by (auto simp add: Inv2a-innermost-def)
moreover
from t-blocksRead mbal'-gt
have mbal t \leq mbal (dblock s' p) by blast
ultimately show bal t \leq mbal (dblock s' p)
  by auto
qed

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 –
from act inv2a inv2c
have t1: \forall t \in (\lambda x. (\{mbal:= mbal (dblock s' p)\}) \cdot allBlocksRead s p.
  Inv2a-innermost s p t 
  by(blast dest: HEndPhase0-Inv2a-allBlocksRead)
from act inv1
have dblock s' p \in (\lambda x. (\{mbal:= mbal(dblock s' p)\}) \cdot allBlocksRead s p
  by(simp, blast dest: HEndPhase0-dblock-allBlocksRead)
with t1
have inv2-dblock: Inv2a-innermost s p (dblock s' p) by auto
with act
have inp (dblock s' p) \in allInput s' \cup \{NotAnInput\}
  by(auto dest: HNextPart-allInput-or-NotAnInput)
with inv2-dblock
show \?thesis
  by(auto simp add: Inv2a-innermost-def)
qed

lemma HEndPhase0-Inv2a-dblock-q:
  assumes act: HEndPhase0 s s' p
and \( \text{inv1} : \text{Inv1} s \)
and \( \text{inv2a} : \text{Inv2a-inner} s q \)
and \( \text{inv2c} : \text{Inv2c-inner} s p \)
shows \( \text{Inv2a-innermost} s' q (\text{dblock} s' q) \)
proof (cases \( p = q \))
  case True
  with act inv2a inv2c inv1
  show ?thesis
    by (blast dest: \text{HEndPhase0-Inv2a-dblock})
next
  case False
  from inv2a
  have inv-q-dblock: \( \text{Inv2a-innermost} s q (\text{dblock} s q) \)
    by (auto simp add: \text{Inv2a-inner-def} \text{blocksOf-def})
  with False act
  show ?thesis
    by (clarsimp simp add: \text{EndPhase0-def} \text{HNextPart-def} \text{InitializePhase-def} \text{Inv2a-innermost-def})
qed

theorem \text{HEndPhase0-Inv2a}: assumes \( \text{inv} : \text{Inv2a} s \)
and \( \text{act} : \text{HEndPhase0} s s' p \)
and \( \text{inv1} : \text{Inv1} s \)
and \( \text{inv2c} : \text{Inv2c-inner} s p \)
shows \( \text{Inv2a} s' \)
proof (clarsimp simp add: \text{Inv2a-def} \text{Inv2a-inner-def})
  fix \( q \) \( bk \)
  assume bk: \( bk \in \text{blocksOf} s' q \)
  with \text{HEndPhase0-blocksOf}\[\text{OF} \text{act}]\n  have dblock-blocks: \( bk \in \{\text{dblock} s' q\} \cup \text{blocksOf} s q \)
    by blast
  thus \( \text{Inv2a-innermost} s' q bk \)
proof
  from inv
  have inv-q: \( \text{Inv2a-inner} s q \)
    by (auto simp add: \text{Inv2a-def})
  assume bk \in \{\text{dblock} s' q\}
  with act inv1 inv2c inv-q
  show ?thesis
    by (blast dest: \text{HEndPhase0-Inv2a-dblock-q})
next
  assume bk-in-blocks: \( bk \in \text{blocksOf} s q \)
  with inv
  have Inv2a-innermost s q bk
    by (auto simp add: \text{Inv2a-def} \text{Inv2a-inner-def})
  with act show ?thesis
    by (auto simp add: \text{EndPhase0-def} \text{HNextPart-def} \text{InitializePhase-def} \text{Inv2a-innermost-def})
lemma \textit{HEndPhase1-blocksOf}:
\textit{HEndPhase1} \ s \ s' \ p \implies \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \cup \{ \text{dblock} \ s' \ q \}
by (\text{auto simp add: EndPhase1-def blocksOf-def dest: subsetD[OF InitializePhase-rdBy]})

lemma \textit{maxBlk-in-nonInitBlks}:
assumes \textit{b}: \ b \in \text{nonInitBlks} \ s \ p
and \textit{inv1}: Inv1 \ s
shows \textit{maxBlk} \ s \ p \in \text{nonInitBlks} \ s \ p
\land (\forall \ c \in \text{nonInitBlks} \ s \ p. \ \text{bal} \ c \leq \text{bal} \ (\text{maxBlk} \ s \ p))
proof –
have \textit{nibals-finite}: finite (bal ' (\text{nonInitBlks} \ s \ p)) (is finite ?S)
proof (rule finite-imageI)
from \textit{inv1}
have finite (\text{allRdBlks} \ s \ p) by (auto simp add: Inv1-def)
hence finite (\text{allBlocksRead} \ s \ p) by (auto simp add: allBlocksRead-def)
hence finite (\text{blocksSeen} \ s \ p) by (simp add: blocksSeen-def)
thus finite (\text{nonInitBlks} \ s \ p) by (auto simp add: nonInitBlks-def intro: finite-subset)
from \textit{b} have bal ' \text{nonInitBlks} \ s \ p \neq \{} by auto
with \textit{nibals-finite}
have \textit{Max} \ ?S \in \ ?S \land \forall \ bb \in \ ?S. \ \text{bb} \leq \textit{Max} \ ?S by auto
hence \exists \textit{mb} \in \ ?S. \ \forall \ bb \in \ ?S. \ \text{bb} \leq \textit{mb} ..
then obtain \textit{mblk}
where \textit{mblk} \in \text{nonInitBlks} \ s \ p
\land (\forall \ c \in \text{nonInitBlks} \ s \ p. \ \text{bal} \ c \leq \text{bal} \ \textit{mblk})
(is \ ?P \ \textit{mblk})
by auto
hence \ ?P \ (SOME \ b. \ ?P \ b) by (rule someI)
thus \textit{thesis}
by (simp add: maxBlk-def)
qed

lemma \textit{blocksOf-nonInitBlks}:
(\forall \ p \ bk. \ bk \in \text{blocksOf} \ s \ p \implies \text{P} \ bk)
\implies \text{blk} \in \text{nonInitBlks} \ s \ p \implies \text{P} \ bk
by (auto simp add: allRdBlks-def blocksOf-def nonInitBlks-def blocksSeen-def allBlocksRead-def rdBy-def, blast)
lemma \(\text{maxBlk-allInput}:\)
\[\text{assumes inv: Inv2a } s\]
\[
\text{and mblk: maxBlk } s \ p \in \text{nonInitBlks } s \ p\]
\[\text{shows inp } (\text{maxBlk } s \ p) \in \text{allInput } s\]
proof
\[
\text{from inv have blocks: } \forall \ p \ bk. \ bk \in \text{blocksOf } s \ p
\]
\[
\rightarrow \text{inp bk } \in (\text{allInput } s) \cup \{\text{NotAnInput}\}
\]
by
\[\text{auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def}\]
\[
\text{from mblk NotAnInput have } \text{inp } (\text{maxBlk } s \ p) \neq \text{NotAnInput}
\]
by
\[\text{auto simp add: nonInitBlks-def}\]
\[
\text{with mblk blocksOf-nonInitBlks[OF blocks]}
\]
\[\text{show } ?\text{thesis}\]
by
\[\text{auto}\]
qed

lemma \(\text{HEndPhase1-dblock-allInput}:\)
\[\text{assumes act: HEndPhase1 } s \ s' \ p\]
\[
\text{and inv1: HInv1 } s\]
\[
\text{and inv2: Inv2a } s\]
\[\text{shows inp'}: \text{inp } (\text{dblock } s' \ p) \in \text{allInput } s'\]
proof
\[
\text{from act have inpt: inp } s \ p \in \text{allInput } s'
\]
by
\[\text{auto simp add: HNextPart-def EndPhase1-def}\]
\[
\text{have nonInitBlks } s \ p \neq \{\} \rightarrow \text{inp } (\text{maxBlk } s \ p) \in \text{allInput } s
\]
proof
assume ni: nonInitBlks } s \ p \neq \{\}
with inv1
\[
\text{have maxBlk } s \ p \in \text{nonInitBlks } s \ p
\]
by
\[\text{auto simp add: HInv1-def maxBlk-in-nonInitBlks}\]
with inv2
\[
\text{show } \text{inp } (\text{maxBlk } s \ p) \in \text{allInput } s
\]
by
\[\text{blast dest: maxBlk-allInput}\]
qed
with act inpt
\[\text{show } ?\text{thesis}\]
by
\[\text{auto simp add: EndPhase1-def HNextPart-def}\]
qed

lemma \(\text{HEndPhase1-Inv2a-dblock}:\)
\[\text{assumes act: HEndPhase1 } s \ s' \ p\]
\[
\text{and inv1: HInv1 } s\]
\[
\text{and inv2: Inv2a } s\]
\[
\text{and inv2c: Inv2c-inner } s \ p\]
\[\text{shows } \text{Inv2a-innermost } s' \ p \ (\text{dblock } s' \ p)\]
proof
\[
\text{from inv1 act have inv1': HInv1 } s'
\]
by (blast dest: HEndPhase1-HInv1)

from inv2
have inv2a: Inv2a-innermost s p (dblock s p)
  by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)

from act inv2c
have mbal': mbal (dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)

moreover
from act
have bal': bal (dblock s' p) = mbal (dblock s p)
  by (auto simp add: EndPhase1-def)

moreover
from act inv2c
have inp': inp (dblock s' p) ∈ allInput s'
  by (blast dest: HEndPhase1-dblock-allInput)

moreover
with inv1' NotAnInput
have inp (dblock s' p) ≠ NotAnInput
  by (auto simp add: HInv1-def)

ultimately show ?thesis
  using act inv2a
  by (auto simp add: Inv2a-innermost-def EndPhase1-def)

qed

lemma HEndPhase1-Inv2a-dblock-q:
  assumes act: HEndPhase1 s s' p
  and inv1: HInv1 s
  and inv: Inv2a s
  and inv2c: Inv2c-inner s p
  shows Inv2a-innermost s' q (dblock s' q)
proof (cases p=q)
  case True
  with act inv inv2c inv1
  show ?thesis
    by (blast dest: HEndPhase1-Inv2a-dblock)

next
  case False
  from act
  have inv-q-dblock: Inv2a-innermost s q (dblock s q)
    by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
  with False act
  show ?thesis
    by (clarsimp simp add: EndPhase1-def HNextPart-def InitializePhase-def Inv2a-innermost-def)

qed

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

**proof** (clarsimp simp add: Inv2a-def Inv2a-inner-def)

fix \( q \) \( bk \)
assume \( \text{bk-in-bks}: bk \in \text{blocksOf} s' q \)
with \( \text{HEndPhase1-blocksOf[OF act]} \)
have \( \text{dblock-blocks}: bk \in \{ \text{dblock s' q} \} \cup \text{blocksOf} s q \)
by blast
thus \( \text{Inv2a-innermost} s' q bk \)

**proof**

assume \( bk \in \{ \text{dblock s' q} \} \)
with \( \text{act inv1 inv2c inv} \)
show \( \text{?thesis} \)
by (blast dest: HEndPhase1-Inv2a-dblock-q)

next

assume \( \text{bk-in-blocks}: bk \in \text{blocksOf} s q \)
with \( \text{inv} \)
have \( \text{Inv2a-innermost} s q bk \)
by (auto simp add: Inv2a-def Inv2a-inner-def)
with \( \text{act} \)
show \( \text{?thesis} \)
by (auto simp add: EndPhase1-def HNextPart-def InitializePhase-def Inv2a-innermost-def)

qed

**C.2.2 Proofs of Invariant 2 b**

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

**theorem** \( \text{HInit-Inv2b}: \text{HInit s} \rightarrow \text{Inv2b} s \)
**by** (auto simp add: HInit-def Init-def Inv2b-def
Inv2b-inner-def InitDB-def)

**theorem** \( \text{HPhase1or2ReadThen-Inv2b}: \)
\([ \text{Inv2b s}; \text{HPhase1or2ReadThen s s' p d q }] \)
\( \rightarrow \text{Inv2b} s' \)
**by** (auto simp add: Phase1or2ReadThen-def Inv2b-def
Inv2b-inner-def hasRead-def)

**theorem** \( \text{HStartBallot-Inv2b}: \)
\([ \text{Inv2b s}; \text{HStartBallot s s' p }] \)
\( \rightarrow \text{Inv2b} s' \)
**by** (auto simp add: StartBallot-def InitializePhase-def
Inv2b-def Inv2b-inner-def hasRead-def)

**theorem** \( \text{HPhase1or2Write-Inv2b}: \)
\([ \text{Inv2b s}; \text{HPhase1or2Write s s' p d }] \)
\( \rightarrow \text{Inv2b} s' \)
by (auto simp add: Phase1or2Write-def Inv2b-def
    Inv2b-inner-def hasRead-def)

theorem HPhase1or2ReadElse-Inv2b:
[ Inv2b s; HPhase1or2ReadElse s s' p d q ]
⇒ Inv2b s'
by (auto simp add: Phase1or2ReadElse-def StartBallot-def hasRead-def
    InitializePhase-def Inv2b-def Inv2b-inner-def)

theorem HEndPhase1-Inv2b:
[ Inv2b s; HEndPhase1 s s' p ] ⇒ Inv2b s'
by (auto simp add: EndPhase1-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

theorem HFail-Inv2b:
[ Inv2b s; HFail s s' p ]
⇒ Inv2b s'
by (auto simp add: Fail-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

theorem HEndPhase2-Inv2b:
[ Inv2b s; HEndPhase2 s s' p ] ⇒ Inv2b s'
by (auto simp add: EndPhase2-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

theorem HPhase0Read-Inv2b:
[ Inv2b s; HPhase0Read s s' p d ] ⇒ Inv2b s'
by (auto simp add: Phase0Read-def Inv2b-def
    Inv2b-inner-def hasRead-def)

theorem HEndPhase0-Inv2b:
[ Inv2b s; HEndPhase0 s s' p ] ⇒ Inv2b s'
by (auto simp add: EndPhase0-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

C.2.3 Proofs of Invariant 2 c

theorem HInit-Inv2c: HInit s ⇒ Inv2c s
by (auto simp add: HInit-def Init-def Inv2c-def Inv2c-inner-def)

lemma HNextPart-Inv2c-chosen:
assumes hnp: HNextPart s s'
and inv2c: Inv2c s
and outpt': ∀ p. outpt s' p = (if phase s' p = 3 then inp(dblock s' p)
 else NotAnInput)
and inp-dblk: ∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
shows chosen s' ∈ allInput s' ∪ {NotAnInput}
using hnp outpt' inp-dblk inv2c

proof(auto simp add: HNextPart-def Inv2c-def Inv2c-inner-def
   split: if-split_asm)

qed

lemma HNextPart-chosen:
  assumes hnp: HNextPart s s'
  shows chosen s' = NotAnInput --> (∀ p. outpt s' p = NotAnInput)
using hnp
proof(auto simp add: HNextPart-def split: if-split_asm)
  fix p pa
  assume o1: outpt s' p ≠ NotAnInput
  and o2: outpt s' (SOME p. outpt s' p ≠ NotAnInput) = NotAnInput
  from o1
  have ∃ p. outpt s' p ≠ NotAnInput
    by auto
  hence outpt s' (SOME p. outpt s' p ≠ NotAnInput) ≠ NotAnInput
    by(rule someI-ex)
  with o2
  show outpt s' pa = NotAnInput
    by simp
qed

lemma HNextPart-allInput:
  [ HNextPart s s'; Inv2c s ] --> (∀ p. inpt p s' ∈ allInput s')
by(auto simp add: HNextPart-def Inv2c-def Inv2c-inner-def)

theorem HPhase1or2ReadThen-Inv2c:
  assumes inv: Inv2c s
  and act: HPhase1or2ReadThen s s' p d q
  and inv2a: Inv2a s
  shows Inv2c s'
proof --
  from inv2a act
  have inv2a': Inv2a s'
    by(blast dest: HPhase1or2ReadThen-Inv2a)
  from act inv
  have outpt': (∀ p. outpt s' p = (if phase s' p = 3
    then inp(dblock s' p)
    else NotAnInput))
    by(auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: (∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput})
    by(auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
    by(auto dest: HNextPart-Inv2c-chosen)
  from act inv

39
have $\forall p. \inpt s' p \in allInput s'$
\wedge (\text{chosen } s' = \text{NotAnInput} \rightarrow \outpt s' p = \text{NotAnInput})

by(auto dest: HNextPart-chosen HNextPart-allInput)

with $\outpt' \text{ chosen'}$ act inv
show $?\text{thesis}$
by(auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)

qed

theorem $H\text{StartBallot-Inv2c}$:
assumes inv: $\text{Inv2c } s$
and act: $H\text{StartBallot } s \ s' p$
and inv2a: $\text{Inv2a } s$
shows $\text{Inv2c } s'$

proof –
from act
have phase': phase $s' p = 1$
by(simp add: StartBallot-def)

from act
have phase: phase $s p \in \{1,2\}$
by(simp add: StartBallot-def)

from act inv
have mbal': $\text{mbal}(\text{dblock } s' p) \in \text{Ballot } p$
by(auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)

from inv phase
have bal(\text{dblock } s p) \in \text{Ballot } p \cup \{0\}$
by(auto simp add: Inv2c-def Inv2c-inner-def)

with act
have bal': bal(\text{dblock } s' p) \in \text{Ballot } p \cup \{0\}$
by(auto simp add: StartBallot-def)

from act inv phase phase'
have blks': ($\forall d. \forall br \in \text{blocksRead } s' p d.$
mbal(\text{block } br) < \text{mbal}(\text{dblock } s' p))
by(auto simp add: StartBallot-def InitializePhase-def
Inv2c-def Inv2c-inner-def)

from inv2a act
have inv2a': $\text{Inv2a } s'$
by(blast dest: HStartBallot-Inv2a)

from act inv
have outpt': $\forall p. \outpt s' p = (\text{if phase } s' p = 3$
then $\text{inp}(\text{dblock } s' p)$
else $\text{NotAnInput})$
by(auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)

from inv2a'
have dblk: $\forall p. \text{inp } (\text{dblock } s' p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}$
by(auto simp add: Inv2a-def Inv2a-inner-def
Inv2a-innermost-def blocksOf-def)

with act inv outpt'
have chosen': $\text{chosen } s' \in \text{allInput } s' \cup \{\text{NotAnInput}\}$
by(auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \(\forall p. \text{inpt } s' \ p \in \text{allInput } s'\)
\(\land (\text{chosen } s' = \text{NotAnInput} \rightarrow \text{outpt } s' \ p = \text{NotAnInput})\)
by(auto dest: HNextPart-chosen HNextPart-allInput)
with phase' mbal' bal' outpt' chosen' act inv blks'
show ?thesis
by(auto simp add: StartBallot-def InitializePhase-def Inv2c-def Inv2c-inner-def)
qed

theorem HPhase1or2Write-Inv2c:
assumes inv: Inv2c s
and act: HPhase1or2Write s s' p d
and inv2a: Inv2a s
shows Inv2a s'
proof -
from inv2a act
have inv2a': Inv2a s'
  by(blast dest: HPhase1or2Write-Inv2a)
from act inv
have outpt': \(\forall p. \text{outpt } s' \ p = (\text{if phase } s' \ p = 3 \then \text{inp}(\text{dblock } s' \ p) \else \text{NotAnInput})\)
  by(auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: \(\forall p. \text{inp} (\text{dblock } s' \ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}\)
  by(auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen' s' \in \text{allInput } s' \cup \{\text{NotAnInput}\}
  by(auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \(\forall p. \text{inpt } s' p \in \text{allInput } s' \land (\text{chosen } s' = \text{NotAnInput} \rightarrow \text{outpt } s' \ p = \text{NotAnInput})\)
  by(auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
  by(auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
qed

theorem HPhase1or2ReadElse-Inv2c:
\[\text{[ } \text{Inv2c } s; \text{HPhase1or2ReadElse } s \ s' p \ d \ q; \text{Inv2a } s \text{ ] } \implies \text{Inv2c } s'\]
by(auto simp add: Phase1or2ReadElse-def dest: HStartBallot-Inv2c)

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 –
from inv
have Inv2c-inner s p by (auto simp add: Inv2c-def)
with inv2a act inv1
have inv2a': Inv2a s'
  by (blast dest: HEndPhase1-Inv2a)
from act inv
have mbal': mbal(dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
from act
have bal': bal(dblock s' p) = mbal(dblock s' p)
  by (auto simp add: EndPhase1-def)
from act inv
have blks': (∀ d. ∀ br ∈ blocksRead s' p d.
  mbal(block br) < mbal(dblock s' p))
  by (auto simp add: EndPhase1-def InitializePhase-def
  Inv2c-def Inv2c-inner-def)
from act inv
have outpt': ∀ p. outpt s' p = (if phase s' p = 3
  then inp(dblock s' p)
  else NotAnInput)
  by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: ∀ p. inp(dblock s' p) ∈ allInput s' ∪ {NotAnInput}
  by (auto simp add: Inv2a-def Inv2a-inner-def
  Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
  by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: ∀ p. inp s' p ∈ allInput s'
  ∧ (chosen s' = NotAnInput
  → outpt s' p = NotAnInput)
  by (auto dest: HNextPart-chosen HNextPart-allInput)
with mbal' bal' blks' outpt' chosen' act inv
show ?thesis
  by (auto simp add: EndPhase1-def InitializePhase-def
  Inv2c-def Inv2c-inner-def)
qed

theorem HEndPhase2-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase2 s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
proof –
from inv2a act
have \(\text{inv2a}'\): Inv2a s'
by (blast dest: HEndPhase2-Inv2a)
from act inv
have \(\text{outpt}'\): \(\forall p. \text{outpt} s' p = (\text{if phase} s' p = 3 \quad \text{then} \text{inp}(\text{dblock} s' p) \quad \text{else} \text{NotAnInput})\)
by (auto simp add: EndPhase2-def Inv2c-def Inv2c-inner-def)
from inv2a'
have \(\text{dblk}\): \(\forall p. \text{inp}(\text{dblock} s' p) \in \text{allInput} s' \cup \{\text{NotAnInput}\}\)
by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have \(\text{chosen}'\): chosen s' \(\in\) allInput s' \(\cup\) \{NotAnInput\}
by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \(\forall p. \text{inpt} s' p \in \text{allInput} s'\)
\& (chosen s' = NotAnInput
\quad \rightarrow \text{outpt} s' p = \text{NotAnInput})
by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show \(?\text{thesis}\)
by (auto simp add: EndPhase2-def InitializePhase-def Inv2c-def Inv2c-inner-def)
qed

theorem \(\text{HFail-Inv2c}\):
assumes inv: Inv2c s
and act: HFail s s' p
and inv2a: Inv2a s
shows Inv2c s'
proof --
from inv2a act
have inv2a': Inv2a s'
by (blast dest: HFail-Inv2a)
from act inv
have \(\text{outpt}'\): \(\forall p. \text{outpt} s' p = (\text{if phase} s' p = 3 \quad \text{then} \text{inp}(\text{dblock} s' p) \quad \text{else} \text{NotAnInput})\)
by (auto simp add: Fail-def Inv2c-def Inv2c-inner-def)
from inv2a'
have \(\text{dblk}\): \(\forall p. \text{inp}(\text{dblock} s' p) \in \text{allInput} s' \cup \{\text{NotAnInput}\}\)
by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have \(\text{chosen}'\): chosen s' \(\in\) allInput s' \(\cup\) \{NotAnInput\}
by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \(\forall p. \text{inpt} s' p \in \text{allInput} s'\)
\& (chosen s' = NotAnInput
\quad \rightarrow \text{outpt} s' p = \text{NotAnInput})
43
by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
  by (auto simp add: Fail-def InitializePhase-def
       Inv2c-def Inv2c-inner-def)
qed

theorem HPhase0Read-Inv2c:
  assumes inv: Inv2c s
  and act: HPhase0Read s s' p d
  and inv2a: Inv2a s
  shows Inv2c s'
proof
  from inv2a act
  have inv2a': Inv2a s'
    by (blast dest: HPhase0Read-Inv2a)
  from act inv
  have outpt': \( \forall \ p. \ output s' p = (\text{if } phase s' p = 3\)
    then inp(dblock s' p)
    else NotAnInput\)
    by (auto simp add: Phase0Read-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: \( \forall \ p. \ inp(dblock s' p) \in allInput s' \cup \{\text{NotAnInput}\}\)
    by (auto simp add: Inv2a-def Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen s' \in allInput s' \cup \{\text{NotAnInput}\}
    by (auto dest: HNextPart-Inv2c-chosen)
  from act inv
  have allinp: \( \forall \ p. \ inp(s' p) \in allInput s'\)
    \( \land (\text{chosen } s' = \text{NotAnInput} \)
    \( \implies output s' p = \text{NotAnInput}\)
    by (auto dest: HNextPart-chosen HNextPart-allInput)
  with outpt' chosen' act inv
  show ?thesis
    by (auto simp add: Phase0Read-def
       Inv2c-def Inv2c-inner-def)
qed

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
  from inv
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  with inv2a act inv1
have inv2a': Inv2a s'
  by (blast dest: HEndPhase0-Inv2a)

hence bal': bal(dblock s' p) ∈ Ballot p ∪ {0}
  by (auto simp add: Inv2a-def Inv2a-inner-def
      Inv2a-innermost-def blocksOf-def)

from act inv
have mbal': mbal(dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase0-def Inv2c-def Inv2c-inner-def)

from act inv
have allinp: ∀ p. inp d p ∈ allInput s' ∨ NotAnInput
  by (auto dest: HNextPart-allInput)

with mbal' bal' blks' outpt' chosen' act inv
have ?thesis
  by (auto simp add: EndPhase0-def InitializePhase-def
       Inv2c-def Inv2c-inner-def)

qed

theorem HInit-HInv2:
  HInit s ⇒ HInv2 s
using HInit-Inv2a HInit-Inv2b HInit-Inv2c
by (auto simp add: HInv2-def)

HInv1 ∧ HInv2 is an invariant of HNext.

lemma I2b:
  assumes nxt: HNext s s'
  and inv: HInv1 s ∧ HInv2 s
  shows HInv2 s'
proof (auto simp add: HInv2-def)
show Inv2a s' using assms
  by (auto simp add: HInv2-def HNext-def Next-def)

  (auto intro: HStartBallot-Inv2a,
auto intro: HPhase1or2Write-Inv2a,
auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2a
    HPhase1or2ReadElse-Inv2a,
auto intro: HPhase0Read-Inv2a,
auto simp add: EndPhase1or2-def Inv2c-def
    intro: HEndPhase1-Inv2a
    HEndPhase2-Inv2a,
auto intro: HFail-Inv2a,
auto simp add: HInv1-def
    intro: HEndPhase0-Inv2a)

show Inv2b s' using assms by(auto simp add: HInv2-def HNext-def Next-def,  
auto intro: HStartBallot-Inv2b,  
auto intro: HPhase0Read-Inv2b,  
auto intro: HPhase1or2Write-Inv2b,  
auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2b  
    HPhase1or2ReadElse-Inv2b,
auto simp add: EndPhase1or2-def
    intro: HEndPhase1-Inv2b  
    HEndPhase2-Inv2b,
auto intro: HFail-Inv2b  
    HEndPhase0-Inv2b)

show Inv2c s' using assms by(auto simp add: HInv2-def HNext-def Next-def,  
auto intro: HStartBallot-Inv2c,  
auto intro: HPhase0Read-Inv2c,  
auto intro: HPhase1or2Write-Inv2c,  
auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2c  
    HPhase1or2ReadElse-Inv2c,
auto simp add: EndPhase1or2-def
    intro: HEndPhase1-Inv2c  
    HEndPhase2-Inv2c,
auto intro: HFail-Inv2c,
auto simp add: HInv1-def intro: HEndPhase0-Inv2c)

qed

end

theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

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

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

definition \( HInv3-R \) :: state \( \Rightarrow \) Proc \( \Rightarrow \) Proc \( \Rightarrow \) Disk \( \Rightarrow \) bool where
\[ HInv3-R \ s \ p \ q \ d = (\langle block= \ dblock \ s \ q, proc= \ q \rangle \in \text{blocksRead} \ s \ p \ d \lor \langle block= \ dblock \ s \ p, proc= \ p \rangle \in \text{blocksRead} \ s \ q \ d) \]

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

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

### C.3.1 Proofs of Invariant 3

**theorem** \( \text{HInit-HInv3} \): \( \text{HInit} \ s \implies \ HInv3 \ s \)
by (simp add: HInit-def Init-def HInv3-def
\( HInv3-inner-def \) HInv3-L-def HInv3-R-def)

**lemma** \( \text{InitPhase-HInv3-p} \):
\[ \langle \text{InitializePhase} \ s \ s' \ p; HInv3-L \ s' \ p \ q \ d \rangle \implies HInv3-R \ s' \ p \ q \ d \]
by (auto simp add: InitializePhase-def HInv3-inner-def
\( \text{hasRead-def} \) HInv3-L-def HInv3-R-def)

**lemma** \( \text{InitPhase-HInv3-q} \):
\[ \langle \text{InitializePhase} \ s \ s' \ q; HInv3-L \ s' \ p \ q \ d \rangle \implies HInv3-R \ s' \ p \ q \ d \]
by (auto simp add: InitializePhase-def HInv3-inner-def
\( \text{hasRead-def} \) HInv3-L-def HInv3-R-def)

**lemma** \( \text{HInv3-L-sym} \): \( \text{HInv3-L} \ s \ p \ q \ d \implies \text{HInv3-L} \ s \ q \ p \ d \)
by (auto simp add: HInv3-L-def)

**lemma** \( \text{HInv3-R-sym} \): \( \text{HInv3-R} \ s \ p \ q \ d \implies \text{HInv3-R} \ s \ q \ p \ d \)
by (auto simp add: HInv3-R-def)

**lemma** \( \text{Phase1or2ReadThen-HInv3-pq} \):
assumes \( \text{act} \in \text{Phase1or2ReadThen} \)
and \( \text{inv-L'}: \text{HInv3-L} \ s' \ p \ q \ d \)
and \( \text{pq} \neq q \)
and \( \text{inv2b: Inv2b} \)
shows \( \text{HInv3-R} \ s' \ p \ q \ d \)
proof –
from \( \text{inv-L'} \) \( \text{act} \ pq \)
have \( phase \ s \ q \in \{1,2\} \land hasRead \ s \ q \ d \ p \)
by (auto simp add: Phase1or2ReadThen-def HInv3-L-def
\( \text{hasRead-def} \) split: if-split-asm)
with inv2b
have disk s d q = dblock s q
  by (auto simp add: Inv2b-def Inv2b-inner-def
       hasRead-def)
with act
show thesis
  by (auto simp add: Phase1or2ReadThen-def HInv3-def
       HInv3-inner-def HInv3-R-def)
qed

lemma Phase1or2ReadThen-HInv3-hasRead:
  [-
    ~hasRead s pp dd qq;
    Phase1or2ReadThen s s' p d q;
    pp\neq p \lor qq\neq q \lor dd\neq d]
  \Rightarrow \neg hasRead s' pp dd qq
by (auto simp add: hasRead-def Phase1or2ReadThen-def)

theorem HPhase1or2ReadThen-HInv3:
  assumes act: HPhase1or2ReadThen s s' p d q
  and inv: HInv3 s
  and pq: p\neq q
  and inv2b: Inv2b s
  shows HInv3 s'
proof (clarsimp simp add: HInv3-def HInv3-inner-def)
fix pp qq dd
assume h3l': HInv3-L s' pp qq dd
show HInv3-R s' pp qq dd
proof (cases HInv3-L s pp qq dd)
  case True
  with inv
  have HInv3-R s pp qq dd
    by (auto simp add: HInv3-def HInv3-inner-def)
  with act h3l'
  show thesis
    by (auto simp add: HInv3-R-def HInv3-L-def
                      Phase1or2ReadThen-def)
next
  case False
  assume nh3l: \neg HInv3-L s pp qq dd
  show HInv3-R s' pp qq dd
  proof (cases ((pp=p \land qq=q) \lor (pp=q \land qq=p)) \land dd=d)
    case True
    with act pq inv2b h3l' HInv3-L-sym[OF h3l']
    show thesis
      by (auto dest: Phase1or2ReadThen-HInv3-pq HInv3-R-sym)
  next
    case False
  from nh3l h3l' act
  have (~hasRead s pp dd \lor \neg hasRead s qq dd pp)
∧ hasRead s' pp dd qq ∧ hasRead s' qq dd pp
by(auto simp add: HInv3-L-def Phase1or2ReadThen-def)
with act False
show ?thesis
  by(auto dest: Phase1or2ReadThen-HInv3-hasRead)
qed
qed
qed

lemma StartBallot-HInv3-p:
  [ StartBallot s s' p; HInv3-L s' p q d ]
  \implies HInv3-R s' p q d
by(auto simp add: StartBallot-def dest: InitPhase-HInv3-p)

lemma StartBallot-HInv3-q:
  [ StartBallot s s' q; HInv3-L s' p q d ]
  \implies HInv3-R s' p q d
by(auto simp add: StartBallot-def dest: InitPhase-HInv3-q)

lemma StartBallot-HInv3-nL:
  [ StartBallot s s' t; \neg HInv3-L s p q d; t\neq p; t\neq q ]
  \implies \neg HInv3-L s' p q d
by(auto simp add: StartBallot-def InitializePhase-def
        HInv3-L-def hasRead-def)

lemma StartBallot-HInv3-R:
  [ StartBallot s s' t; HInv3-R s p q d; t\neq p; t\neq q ]
  \implies HInv3-R s' p q d
by(auto simp add: StartBallot-def InitializePhase-def
        HInv3-R-def hasRead-def)

lemma StartBallot-HInv3-t:
  [ StartBallot s s' t; HInv3-inner s p q d; t\neq p; t\neq q ]
  \implies HInv3-inner s' p q d
by(auto simp add: HInv3-inner-def
        dest: StartBallot-HInv3-nL StartBallot-HInv3-R)

lemma StartBallot-HInv3:
  assumes act: StartBallot s s' t
  and inv: HInv3-inner s p q d
  shows HInv3-inner s' p q d
proof(cases t=p \lor t=q)
  case True
  with act inv
  show ?thesis
    by(auto simp add: HInv3-inner-def
           dest: StartBallot-HInv3-p StartBallot-HInv3-q)
next
  case False
with inv act
show ?thesis
  by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-t)
qed

theorem HStartBallot-HInv3:
  \[ \text{HStartBallot } s \; s' ; \; \text{HInv3 } s \] \implies \text{HInv3 } s'
  by (auto simp add: HInv3-def dest: StartBallot-HInv3)

theorem HPhase1or2ReadElse-HInv3:
  \[ \text{HPhase1or2ReadElse } s \; s' \; p \; d \; q ; \; \text{HInv3 } s \] \implies \text{HInv3 } s'
  by (auto simp add: Phase1or2ReadElse-def HInv3-def dest: StartBallot-HInv3)

theorem HPhase1or2Write-HInv3:
  assumes act: \text{HPhase1or2Write } s \; s' \; p \; d
and inv: \text{HInv3 } s
  shows \text{HInv3 } s'
proof (auto simp add: HInv3-def)
  fix pp qq dd
  show \text{HInv3-inner } s' \; pp \; qq \; dd
proof (cases \text{HInv3-L } s \; pp \; qq \; dd)
  case True
  with inv
  have \text{HInv3-R } s \; pp \; qq \; dd
    by (simp add: HInv3-def HInv3-inner-def)
  with act
  show ?thesis
    by (auto simp add: HInv3-inner-def HInv3-R-def)
next
  case False
  with act
  have \lnot \text{HInv3-L } s' \; pp \; qq \; dd
    by (auto simp add: HInv3-L-def hasRead-def Phase1or2Write-def)
  thus ?thesis
    by (simp add: HInv3-inner-def)
qed

lemma EndPhase1-HInv3-p:
  \[ \text{EndPhase1 } s \; s' ; \; \text{HInv3-L } s' \; p \; q \; d \] \implies \text{HInv3-R } s' \; p \; q \; d
  by (auto simp add: EndPhase1-def dest: InitPhase-HInv3-p)

lemma EndPhase1-HInv3-q:
  \[ \text{EndPhase1 } s \; s' \; q ; \; \text{HInv3-L } s' \; p \; q \; d \] \implies \text{HInv3-R } s' \; p \; q \; d
  by (auto simp add: EndPhase1-def dest: InitPhase-HInv3-q)

lemma EndPhase1-HInv3-nL:

50
[EndPhase1 \( s s' t ; \neg HInv3-L \ s \ p \ q \ d ; \ t \neq p ; \ t \neq q \) ]

\[ \Rightarrow \neg HInv3-L \ s' p \ q \ d \]

by(auto simp add: EndPhase1-def InitializePhase-def HInv3-L-def hasRead-def)

**lemma** EndPhase1-HInv3-R:

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

\[ \Rightarrow HInv3-R \ s' p \ q \ d \]

by(auto simp add: EndPhase1-def InitializePhase-def HInv3-R-def hasRead-def)

**lemma** EndPhase1-HInv3-t:

[EndPhase1 \( s s' t ; HInv3-inner \ s \ p \ q \ d ; \ t \neq p ; \ t \neq q \) ]

\[ \Rightarrow HInv3-inner \ s' p \ q \ d \]

by(auto simp add: HInv3-inner-def dest EndPhase1-HInv3-nL EndPhase1-HInv3-R)

**lemma** EndPhase1-HInv3:

assumes act: EndPhase1 \( s s' t \)

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

shows \( \ HInv3-inner \ s' p \ q \ d \)

proof(cases \( t = p \lor t = q \))

  case True

  with \( \text{act inv} \)

  show \( \ ? \text{thesis} \)

  by(auto simp add: HInv3-inner-def dest EndPhase1-HInv3-p EndPhase1-HInv3-q)

next

  case False

  with \( \text{inv act} \)

  show \( \ ? \text{thesis} \)

  by(auto simp add: HInv3-inner-def dest EndPhase1-HInv3-t)

qed

**theorem** HEndPhase1-HInv3:

[EndPhase1 \( s s' t ; HInv3-L \ s' p \ q \ d \) ]

\[ \Rightarrow HInv3 s' \]

by(auto simp add: HInv3-def dest: EndPhase1-HInv3)

**lemma** EndPhase2-HInv3-p:

[EndPhase2 \( s s' p ; HInv3-L \ s' p \ q \ d \) ]

\[ \Rightarrow HInv3-R \ s' p \ q \ d \]

by(auto simp add: EndPhase2-def dest: InitPhase-HInv3-p)

**lemma** EndPhase2-HInv3-q:

[EndPhase2 \( s s' q ; HInv3-L \ s' p \ q \ d \) ]

\[ \Rightarrow HInv3-R \ s' p \ q \ d \]

by(auto simp add: EndPhase2-def dest: InitPhase-HInv3-q)

**lemma** EndPhase2-HInv3-nL:

[EndPhase2 \( s s' t ; \neg HInv3-L \ s \ p \ q \ d ; \ t \neq p ; \ t \neq q \) ]

\[ \Rightarrow \neg HInv3-L \ s' p \ q \ d \]
by(auto simp add: EndPhase2-def InitializePhase-def
Hinv3-L-def hasRead-def)

lemma EndPhase2-Hinv3-R:
[ EndPhase2 s s' t; Hinv3-R s p q d; t\neq p; t\neq q ]
\implies Hinv3-R s' p q d
by(auto simp add: EndPhase2-def InitializePhase-def
Hinv3-R-def hasRead-def)

lemma EndPhase2-Hinv3-t:
[ EndPhase2 s s' t; Hinv3-inner s p q d; t\neq p; t\neq q ]
\implies Hinv3-inner s' p q d
by(auto simp add: Hinv3-inner-def
dest: EndPhase2-Hinv3-nL EndPhase2-Hinv3-R)

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(cases t\neq p \lor t\neq q)
  case True with act inv
  show ?thesis
  by(auto simp add: Hinv3-inner-def
  dest: EndPhase2-Hinv3-p EndPhase2-Hinv3-q)
next
  case False with inv act
  show ?thesis
  by(auto simp add: Hinv3-inner-def dest: EndPhase2-Hinv3-t)
qed

theorem HEndPhase2-Hinv3:
[ HEndPhase2 s s' p; Hinv3 s ] \implies Hinv3 s'
by(auto simp add: Hinv3-def dest: EndPhase2-Hinv3)

lemma Fail-Hinv3-p:
[ Fail s s' p; Hinv3-L s' p q d ] \implies Hinv3-R s' p q d
by(auto simp add: Fail-def dest: InitPhase-Hinv3-p)

lemma Fail-Hinv3-q:
[ Fail s s' q; Hinv3-L s' p q d ] \implies Hinv3-R s' p q d
by(auto simp add: Fail-def dest: InitPhase-Hinv3-q)

lemma Fail-Hinv3-nL:
[ Fail s s' t; \neg Hinv3-L s p q d; t\neq p; t\neq q ]
\implies \neg Hinv3-L s' p q d
by(auto simp add: Fail-def InitializePhase-def
Hinv3-L-def hasRead-def)
lemma Fail-HInv3-R:
\[
[ \text{Fail} \ s \ s' \ t; \ HInv3-R \ s \ p \ q \ d; \ t \neq p; \ t \neq q ] \implies HInv3-R \ s' \ p \ q \ d
\]
by(auto simp add: Fail-def InitializePhase-def HInv3-R-def hasRead-def)

lemma Fail-HInv3-t:
\[
[ \text{Fail} \ s \ s' \ t; \ HInv3-inner \ s \ p \ q \ d; \ t \neq p; \ t \neq q ] \implies HInv3-inner \ s' \ p \ q \ d
\]
by(auto simp add: HInv3-inner-def dest: Fail-HInv3-nL Fail-HInv3-R)

lemma Fail-HInv3:
assumes act: Fail \ s \ s' \ t
and inv: HInv3-inner \ s \ p \ q \ d
shows HInv3-inner \ s' \ p \ q \ d
proof(cases t=p \lor t=q)
  case True
  with inv act
  show \ ?thesis
    by(auto simp add: HInv3-inner-def dest: Fail-HInv3-p Fail-HInv3-q)
next
  case False
  with inv act
  show \ ?thesis
    by(auto simp add: HInv3-inner-def dest: Fail-HInv3-t)
qed

theorem HFail-HInv3:
\[
[ \text{HFail} \ s \ s' \ p; \ HInv3 \ s ] \implies HInv3 \ s'
\]
by(auto simp add: HInv3-def dest: Fail-HInv3)

theorem HPhase0Read-HInv3:
assumes act: HPhase0Read \ s \ s' \ p \ d
and inv: HInv3 \ s
shows HInv3 \ s'
proof(auto simp add: HInv3-def)
  fix \ pp \ qq \ dd
  proof(cases HInv3-inner \ s' \ pp \ qq \ dd)
    case True
    with inv
    have HInv3-R \ s \ pp \ qq \ dd
      by(simp add: HInv3-def HInv3-inner-def)
    with act
    show \ ?thesis
      by(auto simp add: HInv3-inner-def HInv3-R-def Phase0Read-def)
next
    case False
    with act
    have ¬HInv3-L s' pp qq dd
      by(auto simp add: HInv3-L-def hasRead-def Phase0Read-def)
    thus ?thesis
      by(simp add: HInv3-inner-def)
qed

theorem EndPhase0-HInv3-p: [ [ EndPhase0 s s' p; HInv3-L s' p q d ] ]
  ⇒ HInv3-R s' p q d
by(auto simp add: EndPhase0-def dest: InitPhase-HInv3-p)

theorem EndPhase0-HInv3-q: [ [ EndPhase0 s s' q; HInv3-L s' p q d ] ]
  ⇒ HInv3-R s' p q d
by(auto simp add: EndPhase0-def dest: InitPhase-HInv3-q)

theorem EndPhase0-HInv3-nL: [ [ EndPhase0 s s' t; ¬HInv3-L s p q d; t≠p; t≠q ] ]
  ⇒ ¬HInv3-L s' p q d
by(auto simp add: EndPhase0-def InitializePhase-def HInv3-L-def hasRead-def)

theorem EndPhase0-HInv3-R: [ [ EndPhase0 s s' t; HInv3-R s p q d; t≠p; t≠q ] ]
  ⇒ HInv3-R s' p q d
by(auto simp add: EndPhase0-def InitializePhase-def HInv3-R-def hasRead-def)

theorem EndPhase0-HInv3-t: [ [ EndPhase0 s s' t; HInv3-inner s p q d; t≠p; t≠q ] ]
  ⇒ HInv3-inner s' p q d
by(auto simp add: HInv3-inner-def
       dest: EndPhase0-HInv3-nL EndPhase0-HInv3-R)

theorem EndPhase0-HInv3: assumes act: EndPhase0 s s' t
      and inv: HInv3-inner s p q d
    shows HInv3-inner s' p q d
proof(cases t=p ∨ t=q)
  case True
  with act inv
  show ?thesis
    by(auto simp add: HInv3-inner-def
       dest: EndPhase0-HInv3-p EndPhase0-HInv3-q)
case False
with inv act
show \( ? \text{thesis} \)
  by (auto simp add: HInv3-inner-def dest: EndPhase0-HInv3-t)
qed

theorem HEndPhase0-HInv3:
\[
[\begin{array}{l}
  \text{HEndPhase0 } s \ s' \ p; \text{ HInv3 } s \n\end{array}] \implies \text{HInv3 } s'
\]
by (auto simp add: HInv3-def dest: EndPhase0-HInv3)

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

lemma I2c:
assumes \text{next: HNext } s \ s'
and \text{inv: HInv1 } s \land \text{HInv2 } s \land \text{HInv3 } s
shows \text{HInv3 } s' \text{ using assms}
by (auto simp add: HNext-def Next-def,
  auto intro: HStartBallot-HInv3,
  auto intro: HPhase0Read-HInv3,
  auto intro: HPhase1or2Write-HInv3,
  auto simp add: Phase1or2Read-def HInv2-def
    intro: HPhase1or2ReadThen-HInv3
    HPhase1or2ReadElse-HInv3,
  auto simp add: EndPhase1or2-def
    intro: HEndPhase1-HInv3
    HEndPhase2-HInv3,
  auto intro: HFail-HInv3,
  auto intro: HEndPhase0-HInv3)
end

theory DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

C.4 Invariant 4

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

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

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

**definition** \( HInv4a2 :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4a2 \ s \ p = (\forall D \in \text{MajoritySet}.(\exists d \in D. \ \text{mbal}(\text{disk } s \ d \ p) \leq \text{mbal}(\text{dblock } s \ p)) \\
\land \ bal(\text{disk } s \ d \ p) \leq \text{bal}(\text{dblock } s \ p))
\]

**definition** \( HInv4a :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4a \ s \ p = (\forall D \in \text{MajoritySet}.(\exists d \in D. \ \text{mbal}(\text{disk } s \ d \ p) \leq \text{mbal}(\text{dblock } s \ p)) \\
\land \ bal(\text{disk } s \ d \ p) \leq \text{bal}(\text{dblock } s \ p))
\]

**definition** \( HInv4b :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4b \ s \ p = (\forall bk \in \text{blocksOf } s \ p. \ bal bk < \text{mbal}(\text{dblock } s \ p))
\]

**definition** \( HInv4c :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4c \ s \ p = (\forall bk \in \text{blocksOf } s \ p. \ \exists D \in \text{MajoritySet}. \ (\forall d \in D. \ \text{mbal}(\text{disk } s \ d \ p)) = \text{bal}(\text{dblock } s \ p))
\]

**definition** \( HInv4d :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4d \ s \ p = (\forall bk \in \text{blocksOf } s \ p. \ \exists D \in \text{MajoritySet}. \ (\forall d \in D. \ \text{mbal}(\text{disk } s \ d \ p) \leq \text{mbal}(\text{dblock } s \ p))
\]

**definition** \( HInv4 :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

where
\[
HInv4 \ s = (\forall p. \ HInv4a \ s \ p \land HInv4b \ s \ p \land HInv4c \ s \ p \land HInv4d \ s \ p)
\]

The initial state implies Invariant 4.

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

using \( \text{Disk-isMajority} \)

by \( (\text{auto simp add: HInit-def Init-def HInv4-def HInv4a-def HInv4a1-def} \)

\( HInv4a2-def HInv4b-def HInv4c-def HInv4d-def \)

\( \text{MajoritySet-def blocksOf-def InitDB-def rdBy-def}) \)

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 \( HInv4x's'p \), we prove two lemmas. The first lemma \( \text{action-HInv4x-p} \) proves the case of \( p = q \), while lemma \( \text{action-HInv4x-q} \) proves the other case.

### C.4.1 Proofs of Invariant 4a

**lemma** \( HStartBallot-HInv4a1 :: \)

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

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

and \( \text{inv2a:} \ Inv2a-inner \ s' \ p \)

shows \( HInv4a1 \ s' \ p \)

proof \( (\text{auto simp add: HInv4a1-def}) \)

fix \( bk \)
assume \( bk \in \text{blocksOf} \, s' \, p \) with \( \text{HStartBallot-blocksOf[OF act]} \)
have \( bk \in \{ \text{dblock} \, s' \, p \} \cup \text{blocksOf} \, s \, p \)
  by blast
thus \( \text{bal} \, bk \leq \text{mbal} \, (\text{dblock} \, s' \, p) \)
proof
  assume \( bk \in \{ \text{dblock} \, s' \, p \} \)
  with \( \text{inv2a} \)
  show \( ? \text{thesis} \)
    by (auto simp add: Inv2a-innermost-def Inv2a-inner-def blocksOf-def)
next
  assume \( bk \in \text{blocksOf} \, s \, p \)
  with \( \text{inv \, act} \)
  show \( ? \text{thesis} \)
    by (auto simp add: StartBallot-def HInv4a1-def)
qed
qed

lemma \( \text{HStartBallot-HInv4a2} \):
assumes \( \text{act: HStartBallot \, s \, s' \, p} \)
and \( \text{inv \, HInv4a2 \, s \, p} \)
shows \( \text{HInv4a2 \, s' \, p} \)
proof (auto simp add: HInv4a2-def)
fix \( D \)
assume \( \text{Dmaj: \, D} \in \text{MajoritySet} \)
from \( \text{inv \, Dmaj} \)
have \( \exists \, d \in D. \quad \text{mbal} \, (\text{disk} \, s \, d \, p) \leq \text{mbal} \, (\text{dblock} \, s \, p) \)
  \land \ \text{bal} \, (\text{disk} \, s \, d \, p) \leq \text{bal} \, (\text{dblock} \, s \, p) \)
  by (auto simp add: HInv4a2-def)
then obtain \( d \)
  where \( \quad d \in D \)
  \land \ \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) \)
  by auto
with \( \text{act} \)
have \( \quad d \in D \)
  \land \ \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) \)
  by (auto simp add: StartBallot-def)
with \( \text{Dmaj} \)
show \( \exists \, d \in D. \quad \text{mbal} \, (\text{disk} \, s' \, d \, p) \leq \text{mbal} \, (\text{dblock} \, s' \, p) \)
  \land \ \text{bal} \, (\text{disk} \, s' \, d \, p) \leq \text{bal} \, (\text{dblock} \, s' \, p) \)
  by auto
qed

lemma \( \text{HStartBallot-HInv4a-p} \):
assumes \( \text{act: HStartBallot \, s \, s' \, p} \)
and \( \text{inv \, HInv4a \, s \, p} \)
and \( \text{inv2a: Inv2a-inner \, s' \, p} \)
shows \( H_{\text{Inv4a}} \) \( s' \) \( p \) using \( \text{act inv inv2a} \)

proof –
from \( \text{act} \) have phase: \( 0 < \text{phase} \ s \ p \)
  by(auto simp add: StartBallot-def)
from \( \text{act inv inv2a} \) show ?thesis
  by(auto simp del: HStartBallot-def simp add: HInv4a-def phase elim: HStartBallot-HInv4a1 HStartBallot-HInv4a2)
qed

lemma HStartBallot-HInv4a-q:
assumes \( \text{act: HStartBallot} \ s \ s' \ p \) and \( \text{inv: HInv4a} \ s \ q \) and \( \text{pnq:} \ p \neq q \)
shows \( H_{\text{Inv4a}} \) \( s' \) \( q \)
proof –
from \( \text{act pnq} \) have blocksOf \( s' \) \( q \) \( \subseteq \) blocksOf \( s \) \( q \)
  by(auto simp add: StartBallot-def InitializePhase-def blocksOf-def rdBy-def)
with \( \text{act inv pnq} \) show ?thesis
  by(auto simp add: StartBallot-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

theorem HStartBallot-HInv4a:
assumes \( \text{act: HStartBallot} \ s \ s' \ p \) and \( \text{inv: HInv4a} \ s \ q \) and \( \text{inv2a: Inv2a} \ s' \)
shows \( H_{\text{Inv4a}} \) \( s' \) \( q \)
proof(cases \( p = q \))
case True
from \( \text{inv2a} \) have \( \text{Inv2a-inner} \) \( s' \) \( p \)
  by(auto simp add: Inv2a-def)
with \( \text{act inv True} \) show ?thesis
  by(blast dest: HStartBallot-HInv4a-p)
next
case False
with \( \text{act inv} \) show ?thesis
  by(blast dest: HStartBallot-HInv4a-q)
qed

lemma Phase1or2Write-HInv4a1:
lemma Phase1or2Write-HInv4a1:
\[
[\text{Phase1or2Write } s \; s' \; p \; d; \; H\text{Inv4a1} \; s \; q ] \implies H\text{Inv4a1} \; s' \; q
\]
by (auto simp add: Phase1or2Write-def HInv4a1-def blocksOf-def rdBy-def)

lemma Phase1or2Write-HInv4a2:
\[
[\text{Phase1or2Write } s \; s' \; p \; d; \; H\text{Inv4a2} \; s \; q ] \implies H\text{Inv4a2} \; s' \; q
\]
by (auto simp add: Phase1or2Write-def HInv4a2-def)

theorem HPhase1or2Write-HInv4a:
assumes act: \[\text{HPhase1or2Write } s \; s' \; p \; d\]
and inv: \[H\text{Inv4a} \; s \; q\]
shows \[H\text{Inv4a} \; s' \; q\]
proof –
  from act
  have phase’: phase \( s = phase \) \( s' \)
    by (simp add: Phase1or2Write-def)
  show \(?thesis\)
    proof (cases phase \( s = 0\))
      case True
      with phase’ act
      show \(?thesis\)
        by (auto simp add: HInv4a-def)
    next
      case False
      with phase’ act inv
      show \(?thesis\)
        by (auto simp add: HInv4a-def dest: Phase1or2Write-HInv4a1 Phase1or2Write-HInv4a2)
    qed
next
lemma HPhase1or2ReadThen-HInv4a1-p:
assumes act: \[\text{HPhase1or2ReadThen } s \; s' \; p \; d \; q\]
and inv: \[H\text{Inv4a1} \; s \; p\]
shows \[H\text{Inv4a1} \; s' \; p\]
proof (auto simp: HInv4a1-def)
  fix \( bk \)
  assume \( bk \): \( bk \in \text{blocksOf } s' \; p \)
  with \[\text{HPhase1or2ReadThen-blocksOf[OF act]}\]
  have \( bk \in \text{blocksOf } s \; p \) by auto
  with inv act
  show bal \( bk \) \( \leq \) mbal (dblock \( s' \; p \))
    by (auto simp add: HInv4a1-def Phase1or2ReadThen-def)
qed

lemma HPhase1or2ReadThen-HInv4a2:
\[
[\text{HPhase1or2ReadThen } s \; s' \; p \; d \; r; \; H\text{Inv4a2} \; s \; q ] \implies H\text{Inv4a2} \; s' \; q
\]
by (auto simp add: Phase1or2ReadThen-def HInv4a2-def)
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 –
  from act inv2b
  have phase s p ∈ {1, 2}
    by(auto simp add: Phase1or2ReadThen-def Inv2b-def Inv2b-inner-def)
  with act inv
  show ?thesis
    by(auto simp del: HPhase1or2ReadThen-def simp add: HInv4a-def
        elim: HPhase1or2ReadThen-HInv4a1-p HPhase1or2ReadThen-HInv4a2)
qed

lemma HPhase1or2ReadThen-HInv4a-q:
assumes act: HPhase1or2ReadThen s s’ p d r
and inv: HInv4a s q
and pnq: p ≠ q
shows HInv4a s’ q
proof –
  from act pnq
  have blocksOf s’ q ⊆ blocksOf s q
    by(auto simp add: Phase1or2ReadThen-def InitializePhase-def blocksOf-def rdBy-def)
  with act inv pnq
  show ?thesis
    by(auto simp add: Phase1or2ReadThen-def HInv4a-def
        HInv4a1-def HInv4a2-def)
qed

theorem HPhase1or2ReadThen-HInv4a:
[ HPhase1or2ReadThen s s’ p d r; HInv4a s q; Inv2b s ] ⇒ HInv4a s’ q
by(blast dest: HPhase1or2ReadThen-HInv4a-p HPhase1or2ReadThen-HInv4a-q)

theorem HPhase1or2ReadElse-HInv4a:
assumes act: HPhase1or2ReadElse s s’ p d r
and inv: HInv4a s q and inv2a: Inv2a s’
shows HInv4a s’ q
proof –
  from act have HStartBallot s s’ p
    by(simp add: Phase1or2ReadElse-def)
  with inv inv2a show ?thesis
    by(blast dest: HStartBallot-HInv4a )
qed

lemma HEndPhase1-HInv4a1:
assumes act: HEndPhase1 s s’ p
and inv: HInv4a1 s p
shows $H_{Inv4a1} s' p$

proof (auto simp add: $H_{Inv4a1-def}$)
  fix bk
  assume bk: $bk \in \text{blocksOf } s' p$
  from bk $H_{EndPhase1-blocksOf[OF act]}
  have bk \in \{ \text{dblock } s' p \} \cup \text{blocksOf } s p$
    by blast
  with act inv
  show $bal bk \leq mbal (\text{dblock } s' p)$
    by (auto simp add: $H_{Inv4a-def}$ $H_{Inv4a1-def}$ $EndPhase1-def$)
qed

lemma $H_{EndPhase1-HInv4a2}$:
  assumes act: $H_{EndPhase1} s s' p$
  and inv: $H_{Inv4a2} s p$
  and inv2a: $Inv2a s$
  shows $H_{Inv4a2} s' p$
proof (auto simp add: $H_{Inv4a2-def}$)
  fix D
  assume Dmaj: $D \in \text{MajoritySet}$
  from inv Dmaj
  have $\exists d \in D. \ mbal (disk s d p) \leq mbal (\text{dblock } s p)$
    $\land bal (disk s d p) \leq bal (\text{dblock } s p)$
    by (auto simp add: $H_{Inv4a2-def}$)
  then obtain d
    where d-cond: $d \in D$
      $\land mbal (disk s d p) \leq mbal (\text{dblock } s p)$
      $\land bal (disk s d p) \leq bal (\text{dblock } s p)$
      by auto
  have disk s d p \in blocksOf s p
    by (auto simp add: blocksOf-def)
  with inv2a
  have $bal (disk s d p) \leq mbal (disk s d p)$
    by (auto simp add: $Inv2a-def$ $Inv2a-inner-def$ $Inv2a-innermost-def$)
  with act d-cond
  have $d \in D$
    $\land mbal (disk s' d p) \leq mbal (\text{dblock } s' p)$
    $\land bal (disk s' d p) \leq bal (\text{dblock } s' p)$
    by (auto simp add: $EndPhase1-def$)
  with Dmaj
  show $\exists d \in D. \ mbal (disk s' d p) \leq mbal (\text{dblock } s' p)$
    $\land bal (disk s' d p) \leq bal (\text{dblock } s' p)$
    by auto
qed

lemma $H_{EndPhase1-HInv4a-p}$:
  assumes act: $H_{EndPhase1} s s' p$
  and inv: $H_{Inv4a} s p$
  and inv2a: $Inv2a s$

61
shows $HInv4a\ s'\ p$

proof –

from act

have phase: $0 < phase\ s\ p$
  by (auto simp add: EndPhase1-def)

with act inv inv2a

show ?thesis
  by (auto simp del: HEndPhase1-def simp add: HInv4a-def)

qed

lemma HEndPhase1-HInv4a-q:
  assumes act: $HEndPhase1\ s\ s'\ p$
  and inv: $HInv4a\ s\ q$
  and pnq: $p\neq q$

shows $HInv4a\ s'\ q$

proof –

from act pnq

have $dblock\ s'\ q = dblock\ s\ q \land disk\ s' = disk\ s$
  by (auto simp add: EndPhase1-def)

moreover from act pnq

have $\forall\ p\ d.\ rdBy\ s'\ q\ p\ d \subseteq rdBy\ s\ q\ p\ d$
  by (auto simp add: EndPhase1-def InitializePhase-def rdBy-def)

hence $(\bigcup\ p\ d.\ rdBy\ s'\ q\ p\ d) \subseteq (\bigcup\ p\ d.\ rdBy\ s\ q\ p\ d)\
  by (auto, blast)

ultimately

have $blocksOf\ s'\ q \subseteq blocksOf\ s\ q$
  by (auto simp add: blocksOf-def, blast)

with act inv pnq

show ?thesis
  by (auto simp add: EndPhase1-def HInv4a-def HInv4a1-def HInv4a2-def)

qed

theorem HEndPhase1-HInv4a:
  [$HEndPhase1\ s\ s'\ p;\ HInv4a\ s\ q$] $\Rightarrow$ $HInv4a\ s'\ q$
  by (blast dest: HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q)

theorem HFail-HInv4a:
  [$HFail\ s\ s'\ p;\ HInv4a\ s\ q$] $\Rightarrow$ $HInv4a\ s'\ q$
  by (auto simp add: Fail-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)

theorem HPhase0Read-HInv4a:
  [$HPhase0Read\ s\ s'\ p\ d;\ HInv4a\ s\ q$] $\Rightarrow$ $HInv4a\ s'\ q$
  by (auto simp add: Phase0Read-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)
theorem **HEndPhase2-HInv4a**:
\[
[ \text{HEndPhase2 } s s' p; \text{HInv4a } s q ] \implies \text{HInv4a } s' q
\]
by (auto simp add: EndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)

lemma **allSet**:
assumes aPQ: \( \forall a. \forall r \in P a. Q r \) and \( rb \in P d \)
shows \( Q rb \)
proof –
from aPQ have \( \forall r \in P d. Q r \) by auto
with \( rb \)
show \(?thesis\) by auto
qed

lemma **EndPhase0-44**:
assumes act: \( \text{EndPhase0 } s s' p \)
and \( bk \): \( bk \in \text{blocksOf } s p \)
and \( \text{inv4d} \): \( \text{HInv4d } s p \)
and \( \text{inv2c} \): \( \text{Inv2c-inner } s p \)
shows \( \exists d. \exists rb \in \text{blocksRead } s p d. \text{bal } bk \leq \text{mbal } (\text{block } rb) \)
proof –
from \( bk \) \( \text{inv4d} \)
have \( \exists D1 \in \text{MajoritySet}. \forall d \in D1. \text{bal } bk \leq \text{mbal } (\text{disk } s d p) \) — 4.2
by (auto simp add: HInv4d-def)
with \( \text{majorities-intersect} \)
have p43: \( \forall D \in \text{MajoritySet}. \exists d \in D. \text{bal } bk \leq \text{mbal } (\text{disk } s d p) \)
by (simp add: MajoritySet-def, blast)
from \( \text{act} \)
have phase \( s p = 0 \) by (simp add: EndPhase0-def)
with \( \text{inv2c} \)
have \( \forall d. \forall rb \in \text{blocksRead } s p d. \text{block } rb = \text{disk } s d p \) — 5.1
by (simp add: Inv2c-inner-def)
hence \( \forall d. \text{hasRead } s p d \)
\( \implies (\exists rb \in \text{blocksRead } s p d. \text{block } rb = \text{disk } s d p) \) — 5.2
(is \( \forall d. \ ?H d \implies \ ?P d \))
by (auto simp add: hasRead-def)
with \( \text{act} \)
have p53: \( \exists D \in \text{MajoritySet}. \forall d \in D. \ ?P d \)
by (auto simp add: MajoritySet-def EndPhase0-def)
show \(?thesis\)
proof –
from \( p43 \) \( p53 \)
have \( \exists D \in \text{MajoritySet}. (\exists d \in D. \text{bal } bk \leq \text{mbal } (\text{disk } s d p)) \)
\( \land (\forall d \in D. \ ?P d) \)
by auto
thus \(?thesis\)
by force
qed
lemma \textit{HEndPhase0-HInv4a1-p}:
assumes \text{act}: \text{HEndPhase0 \ s \ s'} \ p
and \ inv2a': \text{Inv2a \ s'}
and \ inv2c: \text{Inv2c-inner \ s \ p}
and \ inv4d: \text{HInv4d \ s \ p}
shows \text{HInv4a1 \ s' \ p}
proof (auto simp add: \text{HInv4a1-def})
fix \ bk
assume \ bk: \ bk \in \text{blocksOf \ s' \ p}
with \text{HEndPhase0-blocksOf[OF \ act]}
have \ bk \in \{\text{dblock \ s' \ p}\} \cup \text{blocksOf \ s \ p} \ by \ auto
thus bal \ bk \leq \text{mbal (dblock \ s' \ p)}
proof
assume \ bk: \ bk \in \{\text{dblock \ s' \ p}\}
with \ inv2a'
have \text{Inv2a-innermost \ s' \ p \ bk}
by (auto simp add: \text{Inv2a-def \ Inv2a-inner-def \ blocksOf-def})
with \ bk show \ ?thesis
by (auto simp add: \text{Inv2a-innermost-def})
next
assume \ bk: \ bk \in \text{blocksOf \ s \ p}
from \ act
have \ f1: \forall \ r \in \text{allBlocksRead \ s \ p}. \text{mbal \ r} < \text{mbal (dblock \ s' \ p)}
by (auto simp add: \text{EndPhase0-def})
with \ act \ inv4d \ inv2c \ bk
have \ \exists \ d. \exists \ rb \in \text{blocksRead \ s \ p \ d}. \text{bal \ bk} \leq \text{mbal (block \ rb)}
by (auto dest: \text{EndPhase0-44})
with \ f1
show \ ?thesis
by (auto simp add: \text{EndPhase0-def \ allBlocksRead-def \ allRdBlks-def \ dest: \ allSet})
qed

lemma \textit{hasRead-allBlks}:
assumes \text{inv2c}: \text{Inv2c-inner \ s \ p}
and \ phase: \text{phase \ s \ p} = 0
shows \((\forall d \in \{d. \text{hasRead \ s \ p \ d \ p}\}. \text{disk \ s \ d \ p} \in \text{allBlocksRead \ s \ p})\)
proof
fix \ d
assume \ d: \ d \in \{d. \text{hasRead \ s \ p \ d \ p}\} \ (\text{is \ d \in \ ?D})
hence \ br-ne: \text{blocksRead \ s \ p \ d} \neq \{\}
by (auto simp add: \text{hasRead-def})
from \ inv2c \ phase
have \ \forall \ br \in \text{blocksRead \ s \ p \ d}. \text{block \ br} = \text{disk \ s \ d \ p}
by (auto simp add: \text{Inv2c-inner-def})
with \ br-ne

qed
have disk s d p ∈ block \ ' blocksRead s p d
  by force.
thus disk s d p ∈ allBlocksRead s p
  by (auto simp add: allBlocksRead-def allRdBlks-def)
qed

lemma HEndPhase0-41:
  assumes act: HEndPhase0 s s' p
  and \ inv1: Inv1 s
  and \ inv2c: Inv2c-inner s p
  shows \exists D ∈ MajoritySet. \forall d ∈ D. mbal(disk s d p) ≤ mbal(dblock s' p)
  ∧ bal(disk s d p) ≤ bal(dblock s' p)
proof –
  from act HEndPhase0-some[OF act inv1]
  have p51: ∀ br ∈ allBlocksRead s p. mbal br < mbal(dblock s' p)
  ∧ bal br ≤ bal(dblock s' p)
  and a: IsMajority({d. hasRead s p d p})
  and phase: phase s p = 0
  by (auto simp add: EndPhase0-def)+
  from inv2c phase
  have (∀ d ∈ {d. hasRead s p d p}. disk s d p ∈ allBlocksRead s p)
  by (auto dest: hasRead-allBlks)
  with p51
  have (∀ d ∈ {d. hasRead s p d p}. mbal(disk s d p) ≤ mbal(dblock s' p)
  ∧ bal(disk s d p) ≤ bal(dblock s' p))
  by force
  with a show ?thesis
  by (auto simp add: MajoritySet-def)
qed

lemma Majority-exQ:
  assumes asm1: \exists D ∈ MajoritySet. \forall d ∈ D. P d
  shows \forall D ∈ MajoritySet. \exists d ∈ D. P d
using asm1
proof (auto simp add: MajoritySet-def)
  fix D1 D2
  assume D1: IsMajority D1 and D2: IsMajority D2
  and Px: \forall x ∈ D1. P x
    from D1 D2 majorities-intersect
    have \exists d ∈ D1. d ∈ D2 by auto
    with Px
    show \exists x ∈ D2. P x
      by auto
qed

lemma HEndPhase0-HInv4a2-p:
  assumes act: HEndPhase0 s s' p
  and \ inv1: Inv1 s

65
and \( inv2c: Inv2c-inner s p \)
shows \( HInv4a2 s' p \)
proof (simp add: HInv4a2-def)
  from act
  have disk': disk s' = disk s
    by (simp add: EndPhase0-def)
  from act inv inv2c
  have \( \exists D \in \text{MajoritySet}. \forall d \in D. \ mbal(disk s d p) \leq \mbal(dblock s' p) \land bal(disk s d p) \leq bal(dblock s' p) \)
    by (blast dest: HEndPhase0-41)
  from Majority-exQ \[ OF this \]
  have \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal(disk s d p) \leq \mbal(dblock s' p) \land bal(disk s d p) \leq bal(dblock s' p) \)
    (is ?P (disk s)) .
  from subst [OF disk', of ?P, OF this]
  show \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal(disk s' d p) \leq \mbal(dblock s' p) \land bal(disk s' d p) \leq bal(dblock s' p) \).
qed

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 –
  from inv2
  have \( inv2c: Inv2c-inner s p \)
    by (auto simp add: Inv2c-def)
  with inv1 inv2a act
  have \( inv2a': Inv2a s' \)
    by (blast dest: HEndPhase0-Inv2a)
  from act
  have phase s' p = 1
    by (auto simp add: EndPhase0-def)
  with act inv inv2c inv4d inv2a' inv1
  show \( ?thesis \)
    by (auto simp add: HInv4a-def simp del: HEndPhase0-def elim: HEndPhase0-HInv4a1-p HEndPhase0-HInv4a2-p)
qed

lemma HEndPhase0-HInv4a-q:
assumes act: HEndPhase0 s s' p
and \( HInv4a s' q \)
and \( \text{pnq}: p \neq q \)
shows \( HInv4a s' q \)
proof –

from act pnq
have \( \text{dblock } s' \ q = \text{dblock } s \ q \land \text{disk } s' = \text{disk } s \)
  by (auto simp add: EndPhase0-def)
moreover
from act pnq
have \( \forall p \ d. \ \text{rdBy } s' \ q \ p \ d \subseteq \text{rdBy } s \ q \ p \ d \)
  by (auto simp add: EndPhase0-def InitializePhase-def rdBy-def)
hence \((\text{UN } p \ d. \ \text{rdBy } s' \ q \ p \ d) \subseteq (\text{UN } p \ d. \ \text{rdBy } s \ q \ p \ d)\)
  by (auto, blast)
ultimately
have \( \text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \)
  by (auto simp add: blocksOf-def, blast)
hence \((\text{UN } p \ d. \ \text{blocksOf } s' \ q \ p \ d)) \subseteq (\text{UN } p \ d. \ \text{blocksOf } s \ q \ p \ d))\)
  by (auto, blast)
ultimately
have \( \text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \)
  by (auto simp add: EndPhase0-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

theorem HEndPhase0-HInv4a:
\[
[ \begin{array}{c}
\text{HEndPhase0 } s \ s' \ p; \\
\text{HInv4a } s \ q; \\
\text{HInv4d } s \ p; \\
\text{Inv2a } s; \\
\text{Inv1 } s; \\
\text{Inv2a } s; \\
\text{Inv2c } s
\end{array} ] \Rightarrow \text{HInv4a } s' \ q
\]
  by (blast dest: HEndPhase0-HInv4a-p HEndPhase0-HInv4a-q)

C.4.2 Proofs of Invariant 4b

lemma blocksRead-allBlocksRead:
\( \text{rb} \in \text{blocksRead } s \ p \ d \Rightarrow \text{block } \text{rb} \in \text{allBlocksRead } s \ p \)
  by (auto simp add: allBlocksRead-def allRdBlks-def)

lemma HEndPhase0-dblock-mbal:
\[
[ \begin{array}{c}
\text{HEndPhase0 } s \ s' \ p \\
\Rightarrow \forall \text{br} \in \text{blocksRead } s \ p \ d. \ \text{mbal } \text{br} < \text{mbal}(\text{dblock } s' \ p)
\end{array} ]
\]
  by (auto simp add: EndPhase0-def)

lemma HEndPhase0-HInv4b-p-dblock:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows \( \text{bal}(\text{dblock } s' \ p) < \text{mbal}(\text{dblock } s' \ p) \)
proof
  from act have \( \text{phase } s \ p = 0 \)
    by (auto simp add: EndPhase0-def)
  with inv2c
  have \( \forall d. \forall \text{br} \in \text{blocksRead } s \ p \ d. \ \text{proc } \text{br} = p \land \text{block } \text{br} = \text{disk } s \ d \ p \)
    by (auto simp add: Inv2c-inner-def)
hence \( \text{allBlks-in-blocksOf } s \ p \subseteq \text{blocksOf } s \ p \)
    by (auto simp add: allBlocksRead-def allRdBlks-def blocksOf-def)
  from act HEndPhase0-some[OF act inv1]
end
have \( p53 \): \( \exists br \in \text{allBlocksRead } s \ p, \ \text{bal}((\text{dblock } s \ p)) = \text{bal } br \)
by (auto simp add: EndPhase0-def)
from inv2a
have \( i2 \): \( \forall \ p. \ \forall bk \in \text{blocksOf } s \ p, \ \text{bal } bk \leq \text{mbal } bk \)
by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
with allBlks-in-blocksOf
have \( \forall bk \in \text{allBlocksRead } s \ p, \ \text{bal } bk \leq \text{mbal } bk \)
by auto
with p53
have \( \exists br \in \text{allBlocksRead } s \ p, \ \text{bal}((\text{dblock } s \ p)) \leq \text{mbal } br \)
by force
with HEndPhase0-dblock-mbal[OF act]
show \( \text{?thesis} \)
by auto
qed

lemma \( HEndPhase0-HInv4b-p-blocksOf \):
assumes act: \( HEndPhase0 \ s \ s' \ p \)
and inv4d: \( HInv4d \ s \ p \)
and inv2c: \( \text{Inv2c-inner } s \ p \)
and bk: \( bk \in \text{blocksOf } s \ p \)
shows \( \text{bal } bk < \text{mbal}((\text{dblock } s \ p)) \)
proof –
from inv4d
have \( p43 \): \( \forall D \in \text{MajoritySet}. \exists d \in D. \ \text{bal } bk \leq \text{mbal}((\text{disk } s \ d \ p)) \)
by (auto simp add: HInv4d-def MajoritySet-def Majority-exQ)
with p53
have \( \exists br \in \text{allBlocksRead } s \ p, \ \text{bal}((\text{dblock } s \ p)) \leq \text{mbal } br \)
proof –
from act
have \( \text{maj} \): \( \text{IsMajority}((\{ d. \ \text{hasRead } s \ p \ d \}) \ (\text{IsMajority}(?D)) \)
and phase: \( \text{phase } s \ p = 0 \)
by (simp add: EndPhase0-def)+
have \( \text{br-ne} \): \( \forall d \in ?D. \ \text{blocksRead } s \ p \ d \neq \{ \} \)
by (auto simp add: hasRead-def)
from phase
have \( \forall d \in ?D. \exists br \in \text{blocksRead } s \ p \ d. \ \text{block } br = \text{disk } s \ d \ p \)
by (auto simp add: Inv2c-inner-def)
with \( \text{br-ne} \)
have \( \forall d \in ?D. \ \exists br \in \text{allBlocksRead } s \ p, \ \text{br} = \text{disk } s \ d \ p \)
by (blast dest: blocksRead-allBlocksRead)
with \( p43 \ \text{maj} \)
show \( \text{?thesis} \)
by (auto simp add: MajoritySet-def)
qed
with HEndPhase0-dblock-mbal[OF act]
show \( \text{?thesis} \)
by auto
qed
lemma HEndPhase0-HInv4b-p:
  assumes act: HEndPhase0 s s' p
  and inv4d: HInv4d s p
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2c: Inv2c-inner s p
  shows HInv4b s' p
proof (clarsimp simp add: HInv4b-def)
  from act have phase: phase s p = 0
    by (auto simp add: EndPhase0-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
  with HEndPhase0-blocksOf[OF act]
  have bk∈{dblock s' p} ∨ bk∈blocksOf s p
    by blast
  thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk ∈ {dblock s' p}
  with act inv1 inv2a inv2c
  show ?thesis
    by (auto simp del: HEndPhase0-def dest: HEndPhase0-HInv4b-p-dblock)
next
  assume bk: bk ∈ blocksOf s p
  with act inv2c inv4d
  show ?thesis
    by (blast dest: HEndPhase0-HInv4b-p-blocksOf)
qed
qed

lemma HEndPhase0-HInv4b-q:
  assumes act: HEndPhase0 s s' p
  and pnq: p≠q
  and inv: HInv4b s q
  shows HInv4b s' q
proof –
  from act pnq
  have disk': disk' s' = disk s
    and dblock': dblock s' q = dblock s q
    and phase': phase' s' q = phase s q
    by (auto simp add: EndPhase0-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by (auto simp add: EndPhase0-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q ⊆ blocksOf s q
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
theorem HEndPhase0-HInv4b:
assumes act: HEndPhase0 s s' p
and inv4d: HInv4d s q
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows HInv4b s' q
proof (cases p=q)
  case True
  with HEndPhase0-HInv4b-p[OF act inv4d inv1 inv2a inv2c]
  show ?thesis by (auto simp)
  next
  case False
  from HEndPhase0-HInv4b-q[OF act False inv] shows ?thesis .
qed

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 (clarsimp simp add: HInv4b-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
  from act have phase'': phase s' p = 1
    and phase: phase s p ∈ {1,2}
    by (auto simp add: StartBallot-def)
  from act have p42: mbal (dblock s p) < mbal (dblock s' p)
    ∧ bal(dblock s p) = bal(dblock s' p)
    by (auto simp add: StartBallot-def)
  from HStartBallot-blocksOf[OF act] bk
  have bk ∈ {dblock s' p} ∪ blocksOf s p
    by blast
  thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk ∈ {dblock s' p}
  from inv2a
  have bal (dblock s p) ≤ mbal (dblock s p)
    by (auto simp add: Inv2a-innermost-def)
  with p42 bk
show \( ? \)thesis by auto

next
assume \( bk \) : \( bk \in \text{blocksOf}\ s\ p \)
from phase inv4a
have \( p41 : HInv4a1\ s\ p \)
by(auto simp add: HInv4a-def)
with \( p42 \) \( bk \)
show \( ? \)thesis
by(auto simp add: HInv4a1-def)
qed

lemma \( HStartBallot-HInv4b-q \):
assumes act : \( HStartBallot\ s\ s'\ p \)
and \( \text{png} ; p \neq q \)
and \( \text{inv} : HInv4b\ s\ q \)
shows \( HInv4b\ s\ q \)
proof –
from act \( \text{png} \)
have \( \text{disk}' : \text{disk}\ s' = \text{disk}\ s \)
and \( \text{dblock}' : \text{dblock}\ s' = \text{dblock}\ s\ q \)
and \( \text{phase}' : \text{phase}\ s' = \text{phase}\ s\ q \)
by(auto simp add: StartBallot-def)
from act \( \text{png} \)
have \( \text{blocksRead}' : \forall \ q . \ \text{allRdBlks}\ s'\ q \subseteq \text{allRdBlks}\ s\ q \)
by(auto simp add: StartBallot-def InitializePhase-def allRdBlks-def)
with \( \text{disk}' \) \( \text{dblock}' \)
have \( \text{blocksOf}\ s'\ q \subseteq \text{blocksOf}\ s\ q \)
by(auto simp add: blocksOf-def rdBy-def, blast)
with inv \( \text{phase}' \) \( \text{dblock}' \)
show \( ? \)thesis
by(auto simp add: HInv4b-def)
qed

theorem \( HStartBallot-HInv4b \):
assumes act : \( HStartBallot\ s\ s'\ p \)
and \( \text{inv2a} : HInv2a\ s \)
and \( \text{inv4b} : HInv4b\ s\ q \)
and \( \text{inv4a} : HInv4a\ s\ p \)
shows \( HInv4b\ s'\ q \)
using act \( \text{inv2a} \) \( \text{inv4b} \) \( \text{inv4a} \)
proof (cases \( p=q \))
case True
from \( \text{inv2a} \)
have \( \text{inv2a-innermost}\ s\ p\ (\text{dblock}\ s\ p) \)
by(auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
with act \( \text{True} \) \( \text{inv4b} \) \( \text{inv4a} \)
show \( ? \)thesis
by(blast dest: HStartBallot-HInv4b-p)
next  
  case False  
  with act inv4b  
  show \( ?\text{thesis} \)  
  by(\text{blast dest: } H\text{StartBallot-Hinv4b-q})

qed

\textbf{theorem} HPhase1or2Write-HInv4b:
\[ [ \text{HPhase1or2Write} \ s \ s' \ p \ d \ ; \ \text{Hinv4b} \ s \ q ] \implies \text{Hinv4b} \ s' \ q \]
\textbf{by}(\text{auto simp add: Phase1or2Write-def Hinv4b-def } 
\text{blocksOf-def rdBy-def})

\textbf{lemma} HPhase1or2ReadThen-HInv4b-p:
\textbf{assumes} act: HPhase1or2ReadThen \ s \ s' \ p \ d \ q
\textbf{and} inv: HInv4b \ s \ p
\textbf{shows} HInv4b \ s' \ p
\textbf{proof} –
\textbf{from} HPhase1or2ReadThen-blocksOf[\text{OF} \ act] \ \text{inv act}
\textbf{show} \ ?\text{thesis}  
\textbf{by}(\text{auto simp add: HInv4b-def Phase1or2ReadThen-def})

\textbf{lemma} HPhase1or2ReadThen-HInv4b-q:
\textbf{assumes} act: HPhase1or2ReadThen \ s \ s' \ p \ d \ r
\textbf{and} inv: HInv4b \ s \ q
\textbf{and} pnq: \ p \neq \ q
\textbf{shows} HInv4b \ s' \ q
\textbf{using} \text{assms HPhase1or2ReadThen-blocksOf[\text{OF} \ act]}  
\textbf{by}(\text{auto simp add: Phase1or2ReadThen-def HInv4b-def})

\textbf{theorem} HPhase1or2ReadThen-HInv4b:
\[ [ \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q \ ; \ \text{Hinv4b} \ s \ r ] \implies \text{Hinv4b} \ s' \ r \]
\textbf{by}(\text{blast dest: } H\text{Phase1or2ReadThen-Hinv4b-p} 
H\text{Phase1or2ReadThen-Hinv4b-q})

\textbf{theorem} HPhase1or2ReadElse-HInv4b:
\[ [ \text{HPhase1or2ReadElse} \ s \ s' \ p \ d \ q \ ; \ \text{Hinv4b} \ s \ r \ ; \ 
\text{Inv2a} \ s \ ; \ \text{Hinv4a} \ s \ p ] \implies \text{Hinv4b} \ s' \ r \]
\textbf{using} \text{HStartBallot-Hinv4b}  
\textbf{by}(\text{auto simp add: Phase1or2ReadElse-def})

\textbf{lemma} HEndPhase1-HInv4b-p:
\textbf{HEndPhase1} \ s \ s' \ p \implies \text{Hinv4b} \ s' \ p
\textbf{by}(\text{auto simp add: EndPhase1-def Hinv4b-def})

\textbf{lemma} HEndPhase1-HInv4b-q:
\textbf{assumes} act: HEndPhase1 \ s \ s' \ p
\textbf{and} pnq: \ p \neq \ q
and \( \text{inv: } HInv4b \ s \ q \) shows \( HInv4b \ s' \ q \)

proof –

from act pnq

have disk': disk s' = disk s

and dblock': dblock s' q = dblock s q

and phase': phase s' q = phase s q

by (auto simp add: EndPhase1-def)

from act pnq

have blocksRead': \( \forall q. \\text{allRdBlks} \ s' q \subseteq \text{allRdBlks} \ s q \)

by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)

with disk' dblock'

have blocksOf s' q \( \subseteq \) blocksOf s q

by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)

with inv phase' dblock'

show ?thesis

by (auto simp add: HInv4b-def)

qed

theorem HEndPhase1-HInv4b:

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

and \( \text{inv: } HInv4b \ s \ q \)

shows \( HInv4b \ s' \ q \)

proof (cases \( p=q \))

case True

with \( \text{HEndPhase1-HInv4b-p[OF act]} \)

show ?thesis by simp

next

case False

from \( \text{HEndPhase1-HInv4b-q[OF act False inv]} \)

show ?thesis .

qed

lemma HEndPhase2-HInv4b-p:

\( \text{HEndPhase2} \ s \ s' \ p \Longrightarrow HInv4b \ s' \ p \)

by (auto simp add: EndPhase2-def HInv4b-def)

lemma HEndPhase2-HInv4b-q:

assumes act: \( \text{HEndPhase2} \ s \ s' \ p \)

and pnq: \( p \neq q \)

and \( \text{inv: } HInv4b \ s \ q \)

shows \( HInv4b \ s' \ q \)

proof –

from act pnq

have disk': disk s' = disk s

and dblock': dblock s' q = dblock s q

and phase': phase s' q = phase s q

by (auto simp add: EndPhase2-def)

from act pnq
have \( \forall q. \) allRdBlks \( s \to s' \subseteq \text{allRdBlks} \ s \ q \)
by (auto simp add: EndPhase2-def InitializePhase-def allRdBlks-def)

with \( \text{disk}' \ \text{dblock}' \)
have blocksOf \( s \to s' \subseteq \text{blocksOf} \ s \ q \)
  by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)

with \( \text{inv phase}' \ \text{dblock}' \)
show \(?thesis\)
  by (auto simp add: HInv4b-def)
qed

theorem \( \text{HEndPhase2-HInv4b}: \)
assumes \( \text{act: } \text{HEndPhase2} \ s \ s' \ p \) and \( \text{inv: } \text{HInv4b} \ s \ q \)
shows \( \text{HInv4b} \ s \ s' \ q \)
proof (cases \( p=q \))
  case True
  with \( \text{HEndPhase2-HInv4b-p[OF act]} \)
  show \(?thesis by simp\)
next
  case False
  from \( \text{HEndPhase2-HInv4b-q[OF act False inv]} \)
  show \(?thesis .\)
qed

lemma \( \text{HFail-HInv4b-p}: \)
\( \text{HFail} \ s \ s' \ p \imp \text{HInv4b} \ s \ s' \ p \)
by (auto simp add: Fail-def HInv4b-def)

lemma \( \text{HFail-HInv4b-q}: \)
assumes \( \text{act: } \text{HFail} \ s \ s' \ p \)
and \( \text{pnq: } p \neq q \)
and \( \text{inv: } \text{HInv4b} \ s \ q \)
shows \( \text{HInv4b} \ s \ s' \ q \)
proof (cases)
  from \( \text{act pnq} \)
  have \( \text{disk': disk'} \ s' = \text{disk} \ s \)
    and \( \text{dblock': dblock} \ s' \to q = \text{dblock} \ s \ q \)
    and \( \text{phase': phase} \ s' \ q = \text{phase} \ s \ q \)
    by (auto simp add: Fail-def)
  from \( \text{act pnq} \)
  have \( \text{blocksRead': \forall q. allRdBlks} \ s' \ q \subseteq \text{allRdBlks} \ s \ q \)
    by (auto simp add: Fail-def InitializePhase-def allRdBlks-def)

  with \( \text{disk'} \ \text{dblock}' \)
  have \( \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \)
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)

  with \( \text{inv phase'} \ \text{dblock'} \)
  show \(?thesis\)
    by (auto simp add: HInv4b-def)
qed
Theorem HFail-HInv4b:
assumes act: HFail s s' p
and inv: HInv4b s q
shows HInv4b s' q
proof (cases p=q)
case True
with HFail-HInv4b-p[OF act]
show ?thesis by simp
next
case False
from HFail-HInv4b-q[OF act False inv]
show ?thesis.
qed

Lemma HPhase0Read-HInv4b-p:
HPhase0Read s s' p d ⇒ HInv4b s' p
by (auto simp add: Phase0Read-def HInv4b-def)

Lemma HPhase0Read-HInv4b-q:
assumes act: HPhase0Read s s' p d
and pnq: p̸=q
and inv: HInv4b s q
shows HInv4b s' q
proof
from act pnq
have disk': disk s'=disk s
and dblock': dblock s' q=dblock s q
and phase': phase s' q =phase s q
by (auto simp add: Phase0Read-def)
from HPhase0Read-blocksOf[OF act] inv phase' dblock'
show ?thesis
by (auto simp add: HInv4b-def)
qed

Theorem HPhase0Read-HInv4b:
assumes act: HPhase0Read s s' p d
and inv: HInv4b s q
shows HInv4b s' q
proof (cases p=q)
case True
with HPhase0Read-HInv4b-p[OF act]
show ?thesis by simp
next
case False
from HPhase0Read-HInv4b-q[OF act False inv]
show ?thesis.
qed
C.4.3 Proofs of Invariant 4c

lemma HStartBallot-HInv4c-p:
  \[ [\text{HStartBallot} \, s \, s' \, p; \text{HInv4c} \, s \, p] \implies \text{HInv4c} \, s' \, p \]
  by (auto simp add: StartBallot-def HInv4c-def)

lemma HStartBallot-HInv4c-q:
  assumes act: HStartBallot \, s \, s' \, p 
  and inv: HInv4c \, s \, q 
  and \, pnq: \, p \neq q 
  shows HInv4c \, s' \, q 
  proof
    from act \, pnq 
    have phase: \, \text{phase} \, s' \, q = \, \text{phase} \, s \, q 
    and \, dblock: \, \text{dblock} \, s \, q = \, \text{dblock} \, s' \, q 
    and \, disk: \, \text{disk} \, s' = \, \text{disk} \, s 
    by (auto simp add: StartBallot-def) 
    with inv 
    show ?thesis 
    by (auto simp add: HInv4c-def) 
  qed 

theorem HStartBallot-HInv4c:
  \[ [\text{HStartBallot} \, s \, s' \, p; \text{HInv4c} \, s \, q] \implies \text{HInv4c} \, s' \, q \]
  by (blast dest: HStartBallot-HInv4c-p HStartBallot-HInv4c-q)

lemma HPhase1or2Write-HInv4c-p:
  assumes act: HPhase1or2Write \, s \, s' \, p \, d 
  and inv: HInv4c \, s \, p 
  and inv2c: Inv2c \, s 
  shows HInv4c \, s' \, p 
  proof (cases phase \, s' \, p = 2) 
    assume phase': \, \text{phase}' \, \text{phase} \, s' \, p = 2 
    show ?thesis 
    proof (auto simp add: HInv4c-def phase'-MajoritySet-def) 
      from act phase' 
      have bal: \, \text{bal} \, (\text{dblock} \, s' \, p) = \text{bal} \, (\text{dblock} \, s \, p) 
      and phase: \, \text{phase} \, s \, p = 2 
      by (auto simp add: Phase1or2Write-def) 
      from phase' inv2c act 
      have mbal: \, \text{mbal} \, (\text{disk} \, s' \, d \, p) = \text{bal} \, (\text{dblock} \, s \, p) 
      by (auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def) 
      with bal 
      have bal (dblock s' p) = mbal (disk s' d p) 
      by auto 
      with inv phase act 
      show \, \exists \, D. \, \text{IsMajority} \, D 
      \land \, (\forall \, d \in D. \, \text{mbal} \, (\text{disk} \, s' \, d \, p) = \text{bal} \, (\text{dblock} \, s' \, p)) 
      by (auto simp add: HInv4c-def Phase1or2Write-def MajoritySet-def) 
    qed 
  qed
next
case False
with act
show ?thesis
  by (auto simp add: HInv4c-def Phase1or2Write-def)
qed

lemma HPhase1or2Write-HInv4c-q:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv4c s q
  and pnq: p ≠ q
  shows HInv4c s' q
proof –
  from act pnq
  have phase: phase s' q = phase s q
  and dblock: dblock s q = dblock s' q
  and disk: ∀ d. disk s' d q = disk s d q
  by (auto simp add: Phase1or2Write-def)
  with inv
  show ?thesis
  by (auto simp add: HInv4c-def)
qed

theorem HPhase1or2Write-HInv4c:
  [ [ HPhase1or2Write s s' p d; HInv4c s q; Inv2c s ] ]
  ⇒ HInv4c s' q
by (blast dest: HPhase1or2Write-HInv4c-p
      HPhase1or2Write-HInv4c-q)

lemma HPhase1or2ReadThen-HInv4c-p:
  [ [ HPhase1or2ReadThen s s' p d q; HInv4c s p ] ] ⇒ HInv4c s' p
by (auto simp add: Phase1or2ReadThen-def HInv4c-def)

lemma HPhase1or2ReadThen-HInv4c-q:
  assumes act: HPhase1or2ReadThen s s' p d r
  and inv: HInv4c s q
  and pnq: p ≠ q
  shows HInv4c s' q
proof –
  from act pnq
  have phase: phase s' q = phase s q
  and dblock: dblock s q = dblock s' q
  and disk: disk s' = disk s
  by (auto simp add: Phase1or2ReadThen-def)
  with inv
  show ?thesis
  by (auto simp add: HInv4c-def)
qed
theorem HPhase1or2ReadThen-HInv4c:
[ HPhase1or2ReadThen s s' p d r; HInv4c s q ]
⇒ HInv4c s' q
by (blast dest: HPhase1or2ReadThen-HInv4c-p HPhase1or2ReadThen-HInv4c-q)

theorem HPhase1or2ReadElse-HInv4c:
[ HPhase1or2ReadElse s s' p d r; HInv4c s q ] ⇒ HInv4c s' q
using HStartBallot-HInv4c
by (auto simp add: Phase1or2ReadElse-def)

lemma HEndPhase1-HInv4c-p:
assumes act: HEndPhase1 s s' p
and inv2b: Inv2b s
shows HInv4c s' p
proof -
from act
have maj: IsMajority {d. d ∈ disksWritten s p
 ∧ (∀ q ∈ (UNIV - {p}). hasRead s p d q)}
   (is IsMajority ?M)
   by (simp add: EndPhase1-def)
from inv2b
have ∀ d ∈ ?M. disk s d p = dblock s p
   by (auto simp add: Inv2b-def Inv2b-inner-def)
with act maj
show ?thesis
   by (auto simp add: HInv4c-def EndPhase1-def MajoritySet-def)
qed

lemma HEndPhase1-HInv4c-q:
assumes act: HEndPhase1 s s' p
and inv: HInv4c s q
and pnq: p ≠ q
shows HInv4c s' q
proof -
from act pnq
have phase: phase s' q = phase s q
   and dblock: dblock s q = dblock s' q
   and disk: disk s' = disk s
   by (auto simp add: EndPhase1-def)
with inv
show ?thesis
   by (auto simp add: HInv4c-def)
qed

theorem HEndPhase1-HInv4c:
[ HEndPhase1 s s' p; HInv4c s q; Inv2b s ] ⇒ HInv4c s' q
by (blast dest: HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q)
lemma \textit{HEndPhase2-HInv4c-p}:
\begin{align*}
[ & \text{HEndPhase2 } s \; s' \; p; \; HInv4c \; s \; p ] \implies HInv4c \; s' \; p \\
\text{by (auto simp add: EndPhase2-def HInv4c-def)}
\end{align*}

lemma \textit{HEndPhase2-HInv4c-q}:
\begin{align*}
\text{assumes } act & : \text{HEndPhase2 } s \; s' \; p \\
and \; inv & : HInv4c \; s \; q \\
and \; pnq & : p \neq q \\
\text{shows } HInv4c \; s' \; q \\
\text{proof} & - \\
\text{from } act \; pnq \\
\text{have } phase & : \text{phase } s' \; q = \text{phase } s \; q \\
\text{and } dblock & : \text{dblock } s \; q = \text{dblock } s' \; q \\
\text{and } disk & : \text{disk } s' = \text{disk } s \\
\text{by (auto simp add: EndPhase2-def)} \\
\text{with } inv \\
\text{show } \text{thesis} & \text{by (auto simp add: HInv4c-def)} \\
\text{qed}
\end{align*}

theorem \textit{HEndPhase2-HInv4c}:
\begin{align*}
[ & \text{HEndPhase2 } s \; s' \; p; \; HInv4c \; s \; q ] \implies HInv4c \; s' \; q \\
\text{by (blast dest: HEndPhase2-HInv4c-p HEndPhase2-HInv4c-q)}
\end{align*}

lemma \textit{HFail-HInv4c-p}:
\begin{align*}
[ & \text{HFail } s \; s' \; p; \; HInv4c \; s \; p ] \implies HInv4c \; s' \; p \\
\text{by (auto simp add: Fail-def HInv4c-def)}
\end{align*}

lemma \textit{HFail-HInv4c-q}:
\begin{align*}
\text{assumes } act & : \text{HFail } s \; s' \; p \\
and \; inv & : HInv4c \; s \; q \\
and \; pnq & : p \neq q \\
\text{shows } HInv4c \; s' \; q \\
\text{proof} & - \\
\text{from } act \; pnq \\
\text{have } phase & : \text{phase } s' \; q = \text{phase } s \; q \\
\text{and } dblock & : \text{dblock } s \; q = \text{dblock } s' \; q \\
\text{and } disk & : \text{disk } s' = \text{disk } s \\
\text{by (auto simp add: Fail-def)} \\
\text{with } inv \\
\text{show } \text{thesis} & \text{by (auto simp add: HInv4c-def)} \\
\text{qed}
\end{align*}

theorem \textit{HFail-HInv4c}:
\begin{align*}
[ & \text{HFail } s \; s' \; p; \; HInv4c \; s \; q ] \implies HInv4c \; s' \; q \\
\text{by (blast dest: HFail-HInv4c-p HFail-HInv4c-q)}
\end{align*}

lemma \textit{HPhase0Read-HInv4c-p}:
lemma HPhase0Read-HInv4c-q:
assumes act: HPhase0Read s s' p d
and inv: HInv4c s q
and pnq: p ≠ q
shows HInv4c s' q
proof –
  from act pnq
  have phase: phase s' q = phase s q
    and dblock: dblock s q = dblock s' q
    and disk: disk s' = disk s
  by(auto simp add: Phase0Read-def)
with inv
  show ?thesis
  by(auto simp add: HInv4c-def)
qed

theorem HPhase0Read-HInv4c:
[ HPhase0Read s s' p d; HInv4c s q ] ⇒ HInv4c s' q
by(blast dest: HPhase0Read-HInv4c-p HPhase0Read-HInv4c-q)

lemma HEndPhase0-HInv4c-p:
[ HEndPhase0 s s' p; HInv4c s q ] ⇒ HInv4c s' p
by(auto simp add: EndPhase0-def HInv4c-def)

lemma HEndPhase0-HInv4c-q:
assumes act: HEndPhase0 s s' p
and inv: HInv4c s q
and pnq: p ≠ q
shows HInv4c s' q
proof –
  from act pnq
  have phase: phase s' q = phase s q
    and dblock: dblock s q = dblock s' q
    and disk: disk s' = disk s
  by(auto simp add: EndPhase0-def)
with inv
  show ?thesis
  by(auto simp add: HInv4c-def)
qed

theorem HEndPhase0-HInv4c:
[ HEndPhase0 s s' p; HInv4c s q ] ⇒ HInv4c s' q
by(blast dest: HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q)
C.4.4 Proofs of Invariant 4d

lemma HStartBallot-HInv4d-p:
  assumes act: HStartBallot s s′ p
  and inv: HInv4d s p
  shows HInv4d s′ p
proof (clarsimp simp add: HInv4d-def)
  fix bk
  assume bk: bk \in blocksOf s′ p
  from act
  have bal′: bal (dblock s′ p) = bal (dblock s p)
    by (auto simp add: StartBallot-def)
  from subsetD[OF HStartBallot-blocksOf[OF act] bk]
  have \exists D \in MajoritySet. \forall d \in D. bal bk \leq mbal (disk s d p)
proof
  assume bk: bk \in blocksOf s p
  with inv
  show ?thesis
    by (auto simp add: HInv4d-def)
next
  assume bk: bk \in {dblock s′ p}
  with bal′ inv
  show ?thesis
    by (auto simp add: HInv4d-def blocksOf-def)
qed
with act
show \exists D \in MajoritySet. \forall d \in D. bal bk \leq mbal (disk s′ d p)
  by (auto simp add: StartBallot-def)
qed

lemma HStartBallot-HInv4d-q:
  assumes act: HStartBallot s s′ p
  and inv: HInv4d s q
  and pnq: p \neq q
  shows HInv4d s′ q
proof
  from act pnq
  have disk′: disk s′ = disk s
    and dblock′: dblock s′ q = dblock s q
      by (auto simp add: StartBallot-def)
  from act pnq
  have blocksRead′: \forall q. allRdBlks s′ q \subseteq allRdBlks s q
    by (auto simp add: StartBallot-def InitializePhase-def allRdBlks-def)
  with disk′ dblock′
  have blocksOf s′ q \subseteq blocksOf s q
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  from subsetD[OF this] inv
  have \forall bk \in blocksOf s′ q.
    \exists D \in MajoritySet. \forall d \in D. bal bk \leq mbal (disk s d q)
  by (auto simp add: HInv4d-def)
with disk′
show ?thesis
by (auto simp add: HInv4d-def)
qed

theorem HStartBallot-HInv4d:
[ HStartBallot s s′ p; HInv4d s q ] ⇒ HInv4d s′ q
by (blast dest: HStartBallot-HInv4d-p HStartBallot-HInv4d-q)

lemma HPhase1or2Write-HInv4d-p:
assumes act: HPhase1or2Write s s′ p d
and inv: HInv4d s p
and inv4a: HInv4a s p
shows HInv4d s′ p
proof (clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk ∈ blocksOf s′ p
from act have ddisk: ∀ d. disk s d p = (if d = dd then dblock s p else disk s dd p)
  and phase: phase s p ≠ 0
by (auto simp add: Phase1or2Write-def)
from inv subsetD[OF HPhase1or2Write-blocksOf[OF OF act] bk]
have asm3: ∃ D ∈ MajoritySet. ∀ dd ∈ D. bal bk ≤ mbal (disk s dd p)
  by (auto simp add: HInv4d-def)
from phase inv4a subsetD[OF HPhase1or2Write-blocksOf[OF OF act] bk] ddisk
have p41: bal bk ≤ mbal (disk s′ d p)
  by (auto simp add: HInv4a-def HInv4a1-def)
with ddisk asm3
show ∃ D ∈ MajoritySet. ∀ dd ∈ D. bal bk ≤ mbal (disk s′ dd p)
  by (auto simp add: MajoritySet-def split: if-split-asm)
qed

lemma HPhase1or2Write-HInv4d-q:
assumes act: HPhase1or2Write s s′ p d
and inv: HInv4d s q
and pnq: p ≠ q
shows HInv4d s′ q
proof –
from act pnq have disk: ∀ d. disk s d q = disk s d q
  by (auto simp add: Phase1or2Write-def)
from act pnq have blocksRead: ∀ q. allRdBlks s′ q ⊆ allRdBlks s q
  by (auto simp add: Phase1or2Write-def
      InitializePhase-def allRdBlks-def)
with act pnq have blocksOf s′ q ⊆ blocksOf s q
by(auto simp add: Phase1or2Write-def allRdBlks-def blocksOf-def rdBy-def)
from subsetD[OF this] inv
have \( \forall bk \in \text{blocksOf } s' \ q. \)
\( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal}(disk \ s \ d \ q) \)
by(auto simp add: HInv4d-def)
with disk'
show \( ?\text{thesis} \)
by(auto simp add: HInv4d-def)
qed

theorem HPhase1or2Write-HInv4d:
[ HPhase1or2Write \( s \ s' \ p \ d \); HInv4d \( s \ q \); HInv4a \( s \ p \) ] \( \Rightarrow \) HInv4d \( s' \ q \)
by(blast dest: HPhase1or2Write-HInv4d-p HPhase1or2Write-HInv4d-q)

lemma HPhase1or2ReadThen-HInv4d-p:
assumes act: HPhase1or2ReadThen \( s \ s' \ p \ d \) q
and inv: HInv4d \( s \ p \)
shows HInv4d \( s' \ p \)
proof(clarsimp simp add: HInv4d-def)
fix bk
assume bk: \( bk \in \text{blocksOf } s' \ p \)
from act
have bal': \( \text{bal} (\text{dblock } s' \ p) = \text{bal} (\text{dblock } s \ p) \)
by(auto simp add: Phase1or2ReadThen-def)
from subsetD[OF HPhase1or2ReadThen-blocksOf[OF act] bk] inv
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal} (\text{disk } s \ d \ p) \)
by(auto simp add: HInv4d-def)
with act
show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal} (\text{disk } s' \ d \ p) \)
by(auto simp add: Phase1or2ReadThen-def)
qed

lemma HPhase1or2ReadThen-HInv4d-q:
assumes act: HPhase1or2ReadThen \( s \ s' \ p \ d \) r
and inv: HInv4d \( s \ q \)
and pnq: \( p \neq q \)
shows HInv4d \( s' \ q \)
proof
from act pnq
have disk': \( \text{disk } s' = \text{disk } s \)
by(auto simp add: Phase1or2ReadThen-def)
from act pnq
have blocksOf \( s' \ q \subseteq \text{blocksOf } s \ q \)
by(auto simp add: Phase1or2ReadThen-def allRdBlks-def blocksOf-def rdBy-def)
from subsetD[OF this] inv
have \( \forall bk \in \text{blocksOf } s' \ q. \)
\( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal}(\text{disk } s \ d \ q) \)
by (auto simp add: HInv4d-def)
with disk'
show ?thesis
by (auto simp add: HInv4d-def)
qed

theorem HPhase1or2ReadThen-HInv4d:
[ HPhase1or2ReadThen s s' p d r; HInv4d s q ] \Rightarrow HInv4d s' q
by (blast dest: HPhase1or2ReadThen-HInv4d-p
     HPhase1or2ReadThen-HInv4d-q)

theorem HPhase1or2ReadElse-HInv4d:
[ HPhase1or2ReadElse s s' p d r; HInv4d s q ] \Rightarrow HInv4d s' q
using HStartBallot-HInv4d
by (auto simp add: Phase1or2ReadElse-def)

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 (clarsimp simp add: HInv4d-def)
  fix bk
  assume bk: bk \in blocksOf s' p
  from HEndPhase1-HInv4c[OF act inv4c inv2b]
  have HInv4c s' p .
  with act
  have p31: \exists D \in MajoritySet.
    \forall d \in D. mbal (disk s' d p) = bal (dblock s' d p)
    and disk': disk s' = disk s
    by (auto simp add: EndPhase1-def HInv4c-def)
  from subsetD[OF HEndPhase1-blocksOf[OF act] bk]
  show \exists D \in MajoritySet. \forall d \in D. bal bk \leq mbal (disk s' d p)
proof
  assume bk: bk \in blocksOf s p
  with inv disk'
  show ?thesis
    by (auto simp add: HInv4d-def)
next
  assume bk: bk \in \{ dblock s' p \}
  with p31
  show ?thesis
    by force
qed

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

proof –
from act pnq
have disk': disk s' = disk s
and dblock': dblock s' = dblock s q
by (auto simp add: EndPhase1-def)
from act pnq
have blocksRead': \( \forall q. \ \text{allRdBlks} \ s' q \subseteq \text{allRdBlks} \ s \ q \)
by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
with disk' dblock'
have blocksOf s' q \subseteq blocksOf s q
by (auto simp add: blocksOf-def rdBy-def, blast)
from subsetD[OF this] inv
have \( \forall b \in \text{blocksOf} \ s'. \ q. \ \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ q) \)
by (auto simp add: HInv4d-def)
with disk'
show ?thesis
by (auto simp add: HInv4d-def)
qed

theorem HEndPhase1-HInv4d:
[ \[ \text{HEndPhase1} \ s \ s'; \text{HInv4d} \ s \ q; \text{Inv2b} \ s; \text{HInv4c} \ s \ p \] \Rightarrow \text{HInv4d} \ s' q \]
by (blast dest: HEndPhase1-HInv4d-p HEndPhase1-HInv4d-q)

lemma HEndPhase2-HInv4d-p:
assumes act: \( \text{HEndPhase2} \ s \ s' \ p \)
and inv: \( \text{HInv4d} \ s \ p \)
shows \( \text{HInv4d} \ s' \ p \)
proof (clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk \in \text{blocksOf} \ s' \ p
from act
have bal': bal (dblock s' p) = bal (dblock s p)
by (auto simp add: EndPhase2-def)
from subsetD[OF HEndPhase2-blocksOf[OF act] bk] inv
have \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ p) \)
by (auto simp add: HInv4d-def)
with act
show \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s' \ d \ p) \)
by (auto simp add: EndPhase2-def)
qed

lemma HEndPhase2-HInv4d-q:
assumes act: \( \text{HEndPhase2} \ s \ s' \ p \)
and \( \text{inv} : HInv4d \ s \ q \)
and \( \text{pnq} : p \neq q \)
shows \( HInv4d \ s' \ q \)

**proof** –

from act pnq
have disk': disk s' = disk s
  by (auto simp add: EndPhase2-def)
from act pnq
have blocksOf s' \( q \) \( \subseteq \) blocksOf s \( q \)
  by (auto simp add: EndPhase2-def InitializePhase-def allRdBlks-def blocksOf-def rdBy-def)
from subsetD[OF this] \( \text{inv} \)
have \( \forall \ bk \in \text{blocksOf s' q} \).
  \( \exists \ D \in \text{MajoritySet} \.
  \forall d \in D. \text{bal bk} \leq \text{mbal} (\text{disk s d q}) \)
  by (auto simp add: HInv4d-def)
with disk'
show \(?\text{thesis}\)
by (auto simp add: HInv4d-def)
qed

**theorem** \( \text{HEndPhase2-HInv4d}: \)
[ \( \text{HEndPhase2} \ s \ s' \ p; \ HInv4d \ s \ q \] \( \implies \) \( HInv4d \ s' \ q \)
by (blast dest: HEndPhase2-HInv4d-p HEndPhase2-HInv4d-q)

**lemma** \( \text{HFail-HInv4d-p}: \)
assumes act: \( \text{HFail} \ s \ s' \ p \)
and \( \text{inv} : \text{HInv4d} \ s \ p \)
shows \( \text{HInv4d} \ s' \ p \)
proof (clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk \( \in \) blocksOf s' \( p \)
from act
have disk': disk s' = disk s
  by (auto simp add: Fail-def)
from subsetD[OF \( \text{HFail-blocksOf} \) \[OF act\] \( bk \)]
show \( \exists \ D \in \text{MajoritySet} \.
  \forall d \in D. \text{bal bk} \leq \text{mbal} (\text{disk s d q}) \)
proof
assume bk: bk \( \in \) \{dblock s' \( p\}\}
with \( \text{inv} \) disk'
show \(?\text{thesis}\)
  by (auto simp add: HInv4d-def)
next
assume bk: bk \( \in \) \{dblock s' \( p\}\}
with act
have \( \text{bal bk} = 0 \)
  by (auto simp add: Fail-def InitDB-def)
with Disk-isMajority
show \(?\text{thesis}\)
  by (auto simp add: MajoritySet-def)
lemma HFail-HInv4d-q:
assumes act: HFail s s' p
and inv: HInv4d s q
and pnq: p≠q
shows HInv4d s' q
proof –
from act pnq
have disk': disk s'=disk s
  and dblock': dblock s' q=dblock s q
  by(auto simp add: Fail-def)
from act pnq
have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
  by(auto simp add: Fail-def)
with disk' dblock'
have blocksOf s' q ⊆ blocksOf s q
  by(auto simp add: allRdBlks-def rdBy-def, blast)
from subsetD[OF this] inv
have ∀ bk∈blocksOf s' q.
  ∃ D∈MajoritySet. ∀ d∈D. bal bk ≤ mbal (disk s d q)
  by(auto simp add: HInv4d-def)
with disk'
show ?thesis
by(auto simp add: HInv4d-def)
qed

theorem HFail-HInv4d:
[ HFail s s' p; HInv4d s q ] ⇒ HInv4d s' q
by(blast dest: HFail-HInv4d-p HFail-HInv4d-q)

lemma HPhase0Read-HInv4d-p:
assumes act: HPhase0Read s s' p d
and inv: HInv4d s p
shows HInv4d s' p
proof(clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk ∈ blocksOf s' p
from act
have bal': bal (dblock s' p) = bal (dblock s p)
  by(auto simp add: Phase0Read-def)
from subsetD[OF HPhase0Read-blocksOf[OF act] bk] inv
have ∃ D∈MajoritySet. ∀ d∈D. bal bk ≤ mbal (disk s d p)
  by(auto simp add: HInv4d-def)
with act
show ∃ D∈MajoritySet. ∀ d∈D. bal bk ≤ mbal (disk s' d p)
  by(auto simp add: Phase0Read-def)
qed
lemma \( \text{HPhase0Read-HInv4d-q} \):
assumes \( \text{act: HPhase0Read s s' p d} \)
and \( \text{inv: HInv4d s q} \)
and \( \text{pq: p\neq q} \)
shows \( \text{HInv4d s' q} \)
proof -
from \( \text{act pq} \)
have \( \text{disk': disk s'=disk s} \)
  by (auto simp add: Phase0Read-def)
from \( \text{act pq} \)
have \( \text{blocksOf s' q \subseteq blocksOf s q} \)
  by (auto simp add: Phase0Read-def allRdBlks-def
blocksOf-def rdBy-def)
from \( \text{subsetD[OF this]} \)
have \( \forall \text{tk@blocksOf s' q} \).
  \( \exists D\in\text{MajoritySet} \). \( \forall d\in D \). \( \text{bal tk} \leq \text{mbal(disk s d q)} \)
  by (auto simp add: HInv4d-def)
with \( \text{disk'} \)
show \( \text{?thesis} \)
by (auto simp add: HInv4d-def)
qed

theorem \( \text{HPhase0Read-HInv4d} \):
[ \( \text{HPhase0Read s s' p d; HInv4d s q} \) \( \implies \) \( \text{HInv4d s' q} \) ]
by (blast dest: \( \text{HPhase0Read-HInv4d-p HPhase0Read-HInv4d-q} \))

lemma \( \text{HEndPhase0-blocksOf2} \):
assumes \( \text{act: HEndPhase0 s s' p} \)
and \( \text{inv2c: Inv2c-inner s p} \)
shows \( \text{allBlocksRead s p \subseteq blocksOf s p} \)
proof -
from \( \text{act inv2c} \)
have \( \forall d.\forall \text{br} \in \text{blocksRead s p d} \). \( \text{proc br =p} \)
  \( \wedge \) \( \text{block br = disk s d p} \)
  by (auto simp add: EndPhase0-def Inv2c-inner-def)
thus \( \text{?thesis} \)
by (auto simp add: allBlocksRead-def allRdBlks-def
blocksOf-def)
qed

lemma \( \text{HEndPhase0-HInv4d-p} \):
assumes \( \text{act: HEndPhase0 s s' p} \)
and \( \text{inv: HInv4d s p} \)
and \( \text{inv2c: Inv2c s} \)
and \( \text{inv1: Inv1 s} \)
shows \( \text{HInv4d s' p} \)
proof (clarsimp simp add: HInv4d-def)
  fix \( \text{bk} \)
assume \( bk: bk \in \text{blocksOf} \ s' \ p \)
from \( \text{subsetD[OF } \text{HEndPhase0-blocksOf[OF } \text{act}] \ bk \) have \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ p) \)
proof
assume \( bk: bk \in \text{blocksOf} \ s \ p \)
with \( \text{inv} \)
show \( \text{?thesis} \)
by (auto simp add: \( \text{HInv4d-def} \))
next
assume \( bk: bk \in \{ \text{dblock} \ s' \ p \} \)
from \( \text{inv2c} \)
have \( \text{inv2c-inner} : \text{inv2c-inner} \ s \ p \)
by (auto simp add: \( \text{Inv2c-def} \))
from \( bk \ HEndPhase0-some[OF } \text{act inv1] \)
\( HEndPhase0-blocksOf2[OF } \text{act inv2c-inner] } \text{act} \)
have \( \text{bal} \ bk \in \text{bal}(\text{blocksOf} \ s \ p) \)
by (auto simp add: \( \text{EndPhase0-def} \))
with \( \text{inv} \)
show \( \text{?thesis} \)
by (auto simp add: \( \text{HInv4d-def} \))
qed
with \( \text{act} \)
show \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s' \ d \ p) \)
by (auto simp add: \( \text{EndPhase0-def} \))
qed

lemma \( \text{HEndPhase0-HInv4d-q} \):
assumes \( \text{act}: \text{HEndPhase0} \ s \ s' \ p \)
and \( \text{inv}: \text{HInv4d} \ s \ q \)
and \( \text{pnq} : p \neq q \)
shows \( \text{HInv4d} \ s' \ q \)
proof
from \( \text{act pnq} \)
have \( \text{dblock} \ s' \ q = \text{dblock} \ s \ q \land \text{disk} \ s' = \text{disk} \ s \)
by (auto simp add: \( \text{EndPhase0-def} \))
moreover
from \( \text{act pnq} \)
have \( \forall p \ d. \ \text{rdBy} \ s' \ q \ p \ d \subseteq \text{rdBy} \ s \ q \ p \ d \)
by (auto simp add: \( \text{EndPhase0-def InitializePhase-def rdBy-def} \))
hence \( \text{(UN p d. } \text{rdBy} \ s' \ q \ p \ d) \subseteq (\text{UN p d. } \text{rdBy} \ s \ q \ p \ d) \)
by (auto, blast)
ultimately
have \( \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \)
by (auto simp add: \( \text{blocksOf-def, blast} \))
from \( \text{subsetD[OF this]} \) \( \text{inv} \)
have \( \forall bk \in \text{blocksOf} \ s' \ q. \)
\( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ q) \)
by (auto simp add: \( \text{HInv4d-def} \))
with \( \text{act} \)

89
show thesis
by(auto simp add: EndPhase0-def HInv4d-def)
qed

theorem HEndPhase0-HInv4d:
  [ HEndPhase0 s s' p; Hinv4d s q;
    Inv2c s; Inv1 s] \implies Hinv4d s' q
by(blast dest: HEndPhase0-HInv4d-p HEndPhase0-HInv4d-q)

Since we have already proved HInv2 is an invariant of HNext, Hinv1 \land HInv2 \land HInv4 is also an invariant of HNext.

lemma I2d:
  assumes nxt: HNext s s'
  and inv: HInv1 s \land HInv2 s \land HInv2 s' \land HInv4 s
  shows HInv4 s'
proof(auto simp add: HInv4-def)
  fix p
  show Hinv4a s' p using assms
    by(auto simp add: HInv4-def HNext-def Next-def,
    auto simp add: HInv2-def intro: HStartBallot-HInv4a,
    auto intro: HPhase0Read-HInv4a,
    auto intro: HPhase1or2Write-HInv4a,
    auto simp add: Phase1or2Read-def
      intro: HPhase1or2ReadThen-HInv4a
      HPhase1or2ReadElse-HInv4a,
    auto simp add: EndPhase1or2-def
      intro: HEndPhase1-HInv4a
      HEndPhase2-HInv4a,
    auto intro: HFail-HInv4a,
    auto intro: HEndPhase0-HInv4a simp add: HInv1-def)

  show HInv4b s' p using assms
    by(auto simp add: HInv4-def HNext-def Next-def,
    auto simp add: HInv2-def
      intro: HStartBallot-HInv4b,
    auto intro: HPhase0Read-HInv4b,
    auto intro: HPhase1or2Write-HInv4b,
    auto simp add: Phase1or2Read-def
      intro: HPhase1or2ReadThen-HInv4b
      HPhase1or2ReadElse-HInv4b,
    auto simp add: EndPhase1or2-def
      intro: HEndPhase1-HInv4b
      HEndPhase2-HInv4b,
    auto intro: HFail-HInv4b,
    auto intro: HEndPhase0-HInv4b simp add: HInv1-def Inv2c-def)

  show HInv4c s' p using assms
    by(auto simp add: HInv4-def HNext-def Next-def,
    auto simp add: HInv2-def
      intro: HStartBallot-HInv4c,
    auto intro: HPhase0Read-HInv4c,
auto intro: HPhase1or2Write-HInv4c,
auto simp add: Phase1or2Read-def
  intro: HPhase1or2ReadThen-HInv4c
  HPhase1or2ReadElse-HInv4c,
auto simp add: EndPhase1or2-def
  intro: HEndPhase1-HInv4c
  HEndPhase2-HInv4c,
auto intro: HFail-HInv4c,
auto intro: HEndPhase0-HInv4c simp add: HInv1-def)

show HInv4d s' p using assms
  by(auto simp add: HInv4-def HNext-def Next-def,
      auto simp add: HInv2-def
        intro: HStartBallot-HInv4d,
      auto intro: HPhase0Read-HInv4d,
      auto simp add: Phase1or2Read-def
        intro: HPhase1or2ReadThen-HInv4d
        HPhase1or2ReadElse-HInv4d,
      auto simp add: EndPhase1or2-def
        intro: HEndPhase1-HInv4d
        HEndPhase2-HInv4d,
      auto intro: HFail-HInv4d,
      auto intro: HEndPhase0-HInv4d simp add: HInv1-def)

qed

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 ⇒ nat ⇒ InputsOrNi ⇒ bool
  where maxBalInp s b v = (∀ bk∈ allBlocks s. b ≤ bal bk → inp bk = v)

definition HInv5-inner-R :: state ⇒ Proc ⇒ bool
  where
    HInv5-inner-R s p =
      (maxBalInp s (bal(dblock s p)) (inp(dblock s p))
      ∨ (∃ D∈ MajoritySet. ∃ q. (∀ d∈ D. bal(dblock s p) < mbal(disk s d q)
        ∧ ¬hasRead s p d q)))

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

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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

theorem $\text{HInit-HInv5}: \text{HInit s \Rightarrow HInv5 s}$
using $\text{Disk-isMajority}$
by (auto simp add: $\text{HInit-def Init-def HInv5-def HInv5-inner-def}$)

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 $\text{HInv2}$. 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
dblock.

lemma $\text{HStartBallot-HInv5-p}$:
assumes act: $\text{HStartBallot s s' p}$
and inv: $\text{HInv5-inner s p}$
shows $\text{HInv5-inner s' p}$ using assms
by (auto simp add: $\text{StartBallot-def HInv5-inner-def}$)

lemma $\text{HStartBallot-blocksOf-q}$:
assumes act: $\text{HStartBallot s s' p}$
and $p \neq q$
shows $\text{blocksOf s' q \subseteq blocksOf s q}$ using assms
by (auto simp add: $\text{StartBallot-def InitializePhase-def blocksOf-def rdBy-def}$)

lemma $\text{HStartBallot-allBlocks}$:
assumes act: $\text{HStartBallot s s' p}$
shows $\text{allBlocks s' \subseteq allBlocks s \cup \{dblock s' p\}}$
proof (auto simp del: $\text{HStartBallot-def simp add: allBlocks-def}$
dest: $\text{HStartBallot-blocksOf-q[OF act]}$)
fix $x \, \text{pa}$
assume x-pa: $x \in \text{blocksOf s' pa}$ and
x-nblks: $\forall \text{xa. x} \notin \text{blocksOf s xa}$
show $x = \text{dblock s' p}$
proof (cases $p = \text{pa}$)
case True
from x-nblks
have $x \notin \text{blocksOf s p}$
by auto
with True subsetD[OF $\text{HStartBallot-blocksOf[of act]}$ x-pa]
show ?thesis
  by auto
next
case False
  from x-nblks subsetD[OF HStartBallot-blocksOf-q[OF act False] x-pa]
  show ?thesis
  by auto
qed
qed

lemma HStartBallot-HInv5-q1:
  assumes act: HStartBallot s s' p
  and pnq: p≠q
  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(auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk ∈ allBlocks s'
    and bal: bal (dblock s' q) ≤ bal bk
  from act pnq
  have dblock': dblock s' q = dblock s q by(auto simp add: StartBallot-def)
  from subsetD[OF HStartBallot-allBlocks[OF act] bk]
  show inp bk = inp (dblock s' q)
proof
  assume bk: bk ∈ allBlocks s
  with inv5-1 dblock' bal
  show ?thesis
    by(auto simp add: maxBalInp-def)
next
  assume bk: bk ∈ {dblock s' p}
  have dblock s p ∈ allBlocks s
    by(auto simp add: allBlocks-def blocksOf-def)
  with bal act bk dblock' inv5-1
  show ?thesis
    by(auto simp add: maxBalInp-def StartBallot-def)
qed
qed

lemma HStartBallot-HInv5-q2:
  assumes act: HStartBallot s s' p
  and pnq: p≠q
  and inv5-2: ∃D∈MajoritySet. ∃qq. (∀d∈D. bal(dblock s q) < mbal(disk s d qq)
    ∧ ¬hasRead s q d qq)
  shows ∃D∈MajoritySet. ∃qq. (∀d∈D. bal(dblock s' q) < mbal(disk s' d qq)
    ∧ ¬hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
and \( \text{blocksRead}: \forall d. \text{blocksRead} \ s \ s' q d = \text{blocksRead} \ s \ q \ d \)
and \( \text{dblock}: \dblock \ s' q = \dblock \ s \ q \)
by(auto simp add: StartBallot-def InitializePhase-def)

with inv5-2
show \(?thesis\)
by(auto simp add: hasRead-def)
qed

lemma \( \text{HStartBallot-HInv5-q} \):
assumes act: \( \text{HStartBallot} \ s \ s' p \)
and inv: \( \text{HInv5-inner} \ s \ q \)
and pnq: \( p \neq q \)
shows \( \text{HInv5-inner} \ s' q \)
using assms and HStartBallot-HInv5-q1[OF act pnq]
HStartBallot-HInv5-q2[OF act pnq]
by(auto simp add: HInv5-inner-def HInv5-inner-R-def StartBallot-def)

theorem \( \text{HStartBallot-HInv5} \):
\[
[ \text{HStartBallot} \ s \ s' p ; \text{HInv5-inner} \ s \ q ] \implies \text{HInv5-inner} \ s' q
\]
by(blast dest: HStartBallot-HInv5-q HStartBallot-HInv5-p)

lemma \( \text{HPhase1or2Write-HInv5-1} \):
assumes act: \( \text{HPhase1or2Write} \ s \ s' p d \)
and inv5-1: \( \text{maxBalInp} \ s \ (\text{bal}(\dblock \ s \ q)) \ (\text{inp}(\dblock \ s \ q)) \)
shows \( \text{maxBalInp} \ s' \ (\text{bal}(\dblock \ s' q)) \ (\text{inp}(\dblock \ s' q)) \)
using assms and HPhase1or2Write-blocksOf[OF act]
by(auto simp add: Phase1or2Write-def maxBalInp-def allBlocks-def)

lemma \( \text{HPhase1or2Write-HInv5-p2} \):
assumes act: \( \text{HPhase1or2Write} \ s \ s' p d \)
and inv4c: \( \text{HInv4c} \ s \ p \)
and phase: \( \text{phase} \ s \ p = 2 \)
and inv5-2: \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\dblock \ s \ p) < \text{mbal}(\text{disk} \ s \ d \ q) \wedge \neg \text{hasRead} \ s \ p \ d \ q) \)
shows \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\dblock \ s' p) < \text{mbal}(\text{disk} \ s' \ d \ q) \wedge \neg \text{hasRead} \ s' \ p \ d \ q) \)
proof
from inv5-2
obtain D q
where i1: IsMajority D
and i2: \( \forall d \in D. \ \text{bal}(\dblock \ s \ p) < \text{mbal}(\text{disk} \ s \ d \ q) \)
and i3: \( \forall d \in D. \ \neg \text{hasRead} \ s \ p \ d \ q \)
by(auto simp add: MajoritySet-def)
have pnq: \( p \neq q \)
proof
from inv4c phase
obtain D1 where r1: IsMajority D1 \( \wedge (\forall d \in D1. \ \text{mbal}(\text{disk} \ s \ d \ p) = \text{bal}(\dblock \ s \ p)) \)
by (auto simp add: HInv4c-def MajoritySet-def)
with i1 majorities-intersect
have \( D \cap D1 \neq \{\} \) by auto
then obtain dd where dd \( \in D \cap D1 \)
by auto
with i1 i2 r1
have \( \text{bal}(\text{dblock} s p) < \text{mbal}(\text{disk} s dd q) \land \text{mbal}(\text{disk} s dd p) = \text{bal} (\text{dblock} s p) \)
by auto
thus \(?\)thesis by auto
qed
from act pnq
— \( \text{dblock} \) and \( \text{hasRead} \) do not change
have \( \text{dblock} s' = \text{dblock} s \)
and \( \forall d. \text{hasRead} s' p d q = \text{hasRead} s p d q \)
— In all disks \( q \) blocks don’t change
and \( \forall d. \text{disk} s' d q = \text{disk} s d q \)
by (auto simp add: Phase1or2Write-def hasRead-def)
with i2 i1 i3 majority-nonempty
have \( \forall d \in D. \text{bal} (\text{dblock} s' p) < \text{mbal} (\text{disk} s' d q) \land \neg \text{hasRead} s' p d q \)
by auto
with i1
show \(?\)thesis
by (auto simp add: MajoritySet-def)
qed

lemma HPhase1or2Write-HInv5-p:
assumes act: \( \text{HPhase1or2Write} s s' p d \)
and inv: \( \text{HInv5-inner} s p \)
and inv4: \( \text{HInv4c} s p \)
shows \( \text{HInv5-inner} s' p \)
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase’: phase s’ = 2
and i2: \( \forall D \in \text{MajoritySet}. \forall q. \exists d \in D. \text{bal} (\text{dblock} s' p) < \text{mbal} (\text{disk} s' d q) \rightarrow \text{hasRead} s' p d q \)
with act have phase: phase s p = 2
by (auto simp add: Phase1or2Write-def)
show maxBalInp s’ (bal (dblock s’ p)) (inp (dblock s’ p))
proof (rule HPhase1or2Write-HInv5-1[OF act, of p])
from HPhase1or2Write-HInv5-p2[OF act inv4 phase] inv i2 phase
show maxBalInp s (bal (dblock s p)) (inp (dblock s p))
by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

lemma HPhase1or2Write-allBlocks:
assumes act: \( \text{HPhase1or2Write} s s' p d \)
shows allBlocks s' \( \subseteq \) allBlocks s
using HPhase1or2Write-blocksOf[OF act]
by (auto simp add: allBlocks-def)

95
lemma \( H_\text{Phase1or2Write-HInv5-q2} \):
assumes act: \( H_\text{Phase1or2Write} s s' p d \)
and \( pnq : p\neq q \)
and \( \text{inv4a}: H_\text{Inv4a} s p \)
and \( \text{inv-2}: \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \; \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d q q) \)
\( \land \lnot \text{hasRead} s q d q q \)
shows \( \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \; \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s' d q q) \)
\( \land \lnot \text{hasRead} s' q d q q \)
proof –
from \( \text{inv-2} \)
obtain \( D q q \)
where \( i1: \text{IsMajority} D \)
and \( i2: \forall d \in D. \; \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d q q) \)
and \( i3: \forall d \in D. \; \lnot \text{hasRead} s q d q q \)
by(\( \text{auto simp add: MajoritySet-def} \))
from \( \text{act} pnq \)
— \( \text{dblock} \) and \( \text{hasRead} \) do not change
have \( \text{dblock'}: \text{dblock} s' = \text{dblock} s \)
and \( \text{hasread}: \forall d, \; \text{hasRead} s' q d q q = \text{hasRead} s q d q q \)
by(\( \text{auto simp add: Phase1or2Write-def hasRead-def} \))
have \( \forall d \in D. \; \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s' d q q) \land \lnot \text{hasRead} s' q d q q \)
proof(\( \text{cases } q q=p \))
case True
have \( \text{bal}(\text{dblock} s q) < \text{mbal}(\text{dblock} s p) \)
proof –
from \( \text{inv4a} \) \( \text{act} \) \( i1 \)
have \( \exists d \in D. \; \text{mbal}(\text{disk} s d p) \leq \text{mbal}(\text{dblock} s p) \)
by(\( \text{auto simp add: MajoritySet-def HInv4a-def } H\text{Inv4a2-def Phase1or2Write-def} \))
with True \( i2 \)
show \( \text{bal}(\text{dblock} s q) < \text{mbal}(\text{dblock} s p) \)
by auto
qed
with \( \text{hasread} \) \( \text{dblock'} \) True \( i1 \) \( i2 \) \( i3 \) \( \text{act} \)
show \( \exists \text{thesis} \)
by(\( \text{auto simp add: Phase1or2Write-def} \))
next
case False
with \( \text{act} \) \( i2 \) \( i3 \)
show \( \exists \text{thesis} \)
by(\( \text{auto simp add: Phase1or2Write-def hasRead-def} \))
qed
with \( i1 \)
show \( \exists \text{thesis} \)
by(\( \text{auto simp add: MajoritySet-def} \))
qed

96
lemma HPhase1or2Write-HInv5-q:
assumes act: HPhase1or2Write s s' p d
and inv: HInv5-inner s q
and inv4a: HInv4a s p
and pnq: p ≠ q
shows HInv5-inner s' q
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase s' q = 2
and i2: ∀ D ∈ MajoritySet. ∀ d ∈ D. bal (dblock s' q) < mbal (disk s' d qa)
→ hasRead s' q d qa
from phase' act have phase: phase s q = 2
by (auto simp add: Phase1or2Write-def)
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
proof (rule HPhase1or2Write-HInv5-1 [OF act, of q])
from HPhase1or2Write-HInv5-q2 [OF act pnq inv4a] inv i2 phase
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

theorem HPhase1or2Write-HInv5:
[ HPhase1or2Write s s' p d; HInv5-inner s q;
∈ MajoritySet. ∀ d ∈ D. bal (dblock s' q) < mbal (disk s' d qa)
→ hasRead s' q d qa
from phase' act have phase: phase s q = 2
by (auto simp add: Phase1or2Write-def)
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
proof (rule HPhase1or2Write-HInv5-1 [OF act, of q])
from HPhase1or2Write-HInv5-q2 [OF act pnq inv4a] inv i2 phase
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

lemma HPhase1or2ReadThen-HInv5-1:
assumes act: HPhase1or2ReadThen s s' p d r
and inv5-1: maxBalInp s (bal (dblock s q)) (inp (dblock s q))
shows maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
using assms and HPhase1or2ReadThen-blocksOf [OF act]
by (auto simp add: Phrase1or2ReadThen-def maxBalInp-def allBlocks-def)

lemma HPhase1or2ReadThen-HInv5-p2:
assumes act: HPhase1or2ReadThen s s' p d r
and inv4c: HInv4c s p
and inv2c: Inv2c-inner s p
and phase: phase s p = 2
and inv5-2: ∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal (dblock s p) < mbal (disk s d q)
∧ ¬ hasRead s p d q)
shows ∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal (dblock s' p) < mbal (disk s' d q)
∧ ¬ hasRead s' p d q)
proof
from inv5-2
obtain D q
where i1: IsMajority D
and i2: ∀ d ∈ D. bal (dblock s p) < mbal (disk s d q)
and i3: ∀ d ∈ D. ¬ hasRead s p d q
by (auto simp add: MajoritySet-def)
from inv2c phase
have bal(dblock s p) = mbal(dblock s p)
  by (auto simp add: Inv2c-inner-def)
moreover
from act have mbal (disk s d r) < mbal (dblock s p)
  by (auto simp add: Phase1or2ReadThen-def)
moreover
from i2 have d ∈ D ⟹ bal(dblock s p) < mbal (disk s d q)
  by auto
ultimately have pnr: d ∈ D ⟹ q ≠ r by auto
have pqr: d ∈ D ⟹ q ≠ r by auto
proof
  from inv4c phase obtain D1 where
    r1: IsMajority D1 ∧ (∀ d ∈ D1. mbal(disk s d p) = bal(dblock s p))
  by (auto simp add: HInv4c-def MajoritySet-def)
  with i1 majorities-intersect
  have D ∩ D1 ≠ {} by auto
  then obtain dd where dd ∈ D ∩ D1
  by auto
  with i1 i2 r1
  have bal(dblock s p) < mbal(disk s dd q) ∧ mbal(disk s dd p) = bal(dblock s p)
  by auto
  thus ?thesis by auto
qed
from pnr act
have hasRead': ∀ d ∈ D. hasRead s p d q = hasRead s p d q
  by (auto simp add: Phase1or2ReadThen-def hasRead-def)
from act png
  — dblock and disk do not change
have dblock s' = dblock s
  and ∀ d. disk s' = disk s
  by (auto simp add: Phase1or2ReadThen-def)
with i2 hasRead' i3
have ∀ d ∈ D. bal (dblock s' p) < mbal (disk s' d q) ∧ ¬ hasRead s' p d q
  by auto
with i1
show ?thesis
  by (auto simp add: MajoritySet-def)
qed

lemma HPhase1or2ReadThen-HInv5-p:
  assumes act: HPhase1or2ReadThen s s' p d r
  and inv: HInv5-inner s p
  and inv4: HInv4c s p
  and inv2c: Inv2c s
  shows HInv5-inner s' p
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
  assume phase': phase s' p = 2
  and i2: ∀ D ∈ MajoritySet. ∀ q. ∃ d ∈ D. bal (dblock s' p) < mbal (disk s' d q) ⟹ hasRead s' p d q

98
with act have phase: phase s p = 2
  by(auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal (dblock s' p)) (inp (dblock s p))
proof (rule HPhase1or2ReadThen-HInv5-1[OF act, of p])
  from inv2c
  have Inv2c-inner s p by(auto simp add: Inv2c-def)
  from HPhase1or2ReadThen-HInv5-p2[OF act inv4 this phase]
  show maxBalInp s (bal (dblock s p)) (inp (dblock s p))
    by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

lemma HPhase1or2ReadThen-allBlocks:
  assumes act: HPhase1or2ReadThen s s' p d r
  shows allBlocks s' ⊆ allBlocks s
using HPhase1or2ReadThen-blocksOf[OF act]
by(auto simp add: allBlocks-def)

lemma HPhase1or2ReadThen-HInv5-q2:
  assumes act: HPhase1or2ReadThen s s' p d r
  and pnq: p ≠ q
  and inv4a: HInv4a s p
  and inv5-2: ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
    ∧ ¬ hasRead s q d qq)
  shows ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s' q) < mbal(disk s' d qq)
    ∧ ¬ hasRead s' q d qq)
proof —
  from inv5-2
  obtain D qq
    where i1: IsMajority D
    and i2: ∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
    and i3: ∀ d ∈ D. ¬ hasRead s q d qq
    by(auto simp add: MajoritySet-def)
  from act pnq
    — dblock and hasRead do not change
  have dblock': dblock s' = dblock s
    and disk': disk s' = disk s
    and hasread: ∀ d, hasRead s' q d qq = hasRead s q d qq
    by(auto simp add: Phase1or2ReadThen-def hasRead-def)
  with i2 i3
  have ∀ d ∈ D. bal (dblock s' q) < mbal (disk s' d qq) ∧ ¬ hasRead s' q d qq
    by auto
  with i1
  show ?thesis
    by(auto simp add: MajoritySet-def)
qed

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 ≠ q
shows HInv5-inner s' q

proof(auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase s' q = 2
and i2: ∀D ∈ MajoritySet. ∀qa. ∃d ∈ D. bal (dblock s' q) < mbal (disk s' d qa)
››› hasRead s' q d qa
from phase' act have phase: phase s q = 2
  by(auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal (dblock s q)) (inp (dblock s q))
proof(rule HPhase1or2ReadThen-HInv5-1[OF act, of q])
from HPhase1or2ReadThen-HInv5-q2[OF act pnq inv4a] inv i2 phase
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

theorem HPhase1or2ReadThen-HInv5:
[ HPhase1or2Read s s' p d r; HInv5-inner s q; Inv2c s; HInv4c s p; HInv4a s p ] ⇒ HInv5-inner s' q
by(blast dest: HPhase1or2ReadThen-HInv5-p)

theorem HPhase1or2ReadElse-HInv5:
[ HPhase1or2ReadElse s s' p d r; HInv5-inner s q ] =⇒ HInv5-inner s' q
using HStartBallot-HInv5
by(auto simp add: Phase1or2ReadElse-def)

lemma HEndPhase2-HInv5-p:
HEndPhase2 s s' p ⇒ HInv5-inner s' q
by(auto simp add: EndPhase2-def HInv5-inner-def)

lemma HEndPhase2-allBlocks:
assumes act: HEndPhase2 s s' p
shows allBlocks s' ⊆ allBlocks s
using HEndPhase2-blocksOf[OF act]
by(auto simp add: allBlocks-def)

lemma HEndPhase2-HInv5-q1:
assumes act: HEndPhase2 s s' p
and pnq: p ≠ q
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(auto simp add: maxBalInp-def)
fix bk
assume bk: bk ∈ allBlocks s'
and bal: bal (dblock s' q) ≤ bal bk

100
lemma HEndPhase2-HInv5-q2:
assumes act: HEndPhase2 s s' p
and pnq: 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 –
from act pnq
have disk: disk s' = disk s 
and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
and dblock: dblock s' q = dblock s q
by(auto simp add: EndPhase2-def InitializePhase-def) with inv5-2
show ?thesis
by(auto simp add: hasRead-def)
qed

lemma HEndPhase2-HInv5-q:
assumes act: HEndPhase2 s s' p
and inv: HInv5-inner s q
and pnq: p\neq q
shows HInv5-inner s' q
using assms and HEndPhase2-HInv5-q1[of act pnq] HEndPhase2-HInv5-q2[of act pnq]
by(auto simp add: HInv5-inner-def HInv5-inner-R-def EndPhase2-def HEndPhase2-def)

theorem HEndPhase2-HInv5:
[ HEndPhase2 s s' p; HInv5-inner s q ] \implies HInv5-inner s' q
by(blast dest: HEndPhase2-HInv5-q HEndPhase2-HInv5-p)

lemma HEndPhase1-HInv5-p:
assumes act: HEndPhase1 s s' p
and inv4: HInv4 s
and inv2a: Inv2a s
and inv2a': Inv2a' s'
and inv2c: Inv2c s
and asm4: \neg maxBalImp s' (bal(dblock s' p)) (inp(dblock s' p))
shows (\exists D\in MajoritySet. \exists qq. (\forall d\in D. bal(dblock s' p) < mbal(disk s' d q) \\
\land \neg hasRead s' p d qq))
proof –

101
have $\exists bk \in \text{allBlocks } s. \text{bal(dblock } s \prime \text{ p)} \leq \text{bal } bk \land bk \neq \text{dblock } s \prime \text{ p}$

proof
from asm4
obtain bk
  where p31: $bk \in \text{allBlocks } s' \land \text{bal(dblock } s' \text{ p) } \leq \text{bal } bk \land bk \neq \text{dblock } s' \text{ p}$
  by (auto simp add: maxBalImp-def)
then obtain q where p32: $bk \in \text{blocksOf } s' \text{ q}$
  by (auto simp add: allBlocks-def)
from act
have dblock: $p \neq q \implies \text{dblock } s' \text{ q } = \text{dblock } s \text{ q}$
  by (auto simp add: EndPhase1-def)
have $bk \in \text{blocksOf } s \text{ q}$
proof (cases $p = q$)
case True
  with p32 p31 HEndPhase1-blocksOf[of act]
  show ?thesis
  by auto
next
case False
  from dblock[of False] subsetD[of HEndPhase1-blocksOf[of act, of q] p32]
  show ?thesis
  by (auto simp add: blocksOf-def)
qed
with p31
show ?thesis
by (auto simp add: allBlocks-def)
qed
then obtain bk where p22: $bk \in \text{allBlocks } s \land \text{bal(dblock } s \prime \text{ p)} \leq \text{bal } bk \land bk \neq \text{dblock } s' \text{ p}$
by auto
have $\exists q \in \text{UNIV } \setminus \{p\}. \text{ bk } \in \text{blocksOf } s \text{ q}$
proof
from p22
obtain q where bk: $bk \in \text{blocksOf } s \text{ q}$
  by (auto simp add: allBlocks-def)
from act p22
have mbal(dblock s p) $\leq \text{bal } bk$
  by (auto simp add: EndPhase1-def)
moreover
from act
have phase s p = 1
  by (auto simp add: EndPhase1-def)
moreover
from inv4
have Hinv4b s p by (auto simp add: Hinv4-def)
ultimately
have $p \neq q$
  using bk
  by (auto simp add: Hinv4-def Hinv4b-def)
with bk
show \( ? \text{thesis} \)
by auto
qed

then obtain \( q \) where \( p23: q \in \text{UNIV} - \{ p \} \land bk \in \text{blocksOf } s q \)
by auto
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \, \text{bal}(\text{dblock } s' \, p) \leq \text{mbal}(\text{disk } s \, d \, q) \)
proof -
  from \( p23 \) inv4
  have \( \forall d: \exists D \in \text{MajoritySet}. \forall d \in D. \, \text{bal} \leq \text{mbal} \)
  by (auto simp add: HInv4-def HInv4d-def)
from \( \forall d \, p22 \)
show \( ? \text{thesis} \)
by force
qed

then obtain \( D \) where \( D_{maj}: D \in \text{MajoritySet} \) and \( p24: (\forall d \in D. \, \text{bal}(\text{dblock } s' \, p) \leq \text{mbal}(\text{disk } s \, d \, q) \)
by auto
have \( p25: (\forall d \in D. \, \text{bal}(\text{dblock } s' \, p) < \text{mbal}(\text{disk } s \, d \, q) \)
proof -
  from \( \text{inv2c} \)
  have \( \text{Inv2c-inner } s \, p \)
    by (auto simp add: Inv2c-def)
    with \( \text{act} \)
    have \( \text{bal-pos} \): \( 0 < \text{bal}(\text{dblock } s' \, p) \)
      by (auto simp add: Inv2c-inner-def EndPhase1-def)
    with \( \text{inv2a} \)
    have \( \text{bal}(\text{dblock } s' \, p) \in \text{Ballot } p \cup \{ 0 \} \)
      by (auto simp add: Inv2a-def Inv2a-inner-def

Inv2a-innermost-def blocksOf-def)
    with \( \text{bal-pos} \) have \( \text{bal-in-p} \): \( \text{bal}(\text{dblock } s' \, p) \in \text{Ballot } p \)
by auto
from \( \text{inv2a} \) have \( \text{Inv2a-inner } s \, q \) by (auto simp add: Inv2a-def)
hence \( \forall d \in D. \, \text{mbal}(\text{disk } s \, d \, q) \in \text{Ballot } q \cup \{ 0 \} \)
  by (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with \( p24 \) bal-pos
have \( \forall d \in D. \, \text{mbal}(\text{disk } s \, d \, q) \in \text{Ballot } q \)
by force
with \( \text{Ballot-disj } p23 \) bal-in-p
have \( \forall d \in D. \, \text{mbal}(\text{disk } s \, d \, q) \neq \text{bal}(\text{dblock } s' \, p) \)
by force
with \( p23 \) \( p24 \)
show \( ? \text{thesis} \)
by force
qed
with \( p23 \) \( \text{act} \)
have \( \forall d \in D. \, \text{bal}(\text{dblock } s' \, p) < \text{mbal}(\text{disk } s' \, d \, q) \land \neg \text{hasRead } s' \, p \, d \, q \)
  by (auto simp add: EndPhase1-def InitializePhase-def hasRead-def)
with \( D_{maj} \)
lemma union-inclusion:
\[ A \subseteq A'; B \subseteq B' \] \implies A \cup B \subseteq A' \cup B' 
by blast

lemma HEndPhase1-blocksOf-q:
assumes act: HEndPhase1 s s' p
and png: p\neq q
shows blocksOf s' q \subseteq blocksOf s q

proof –
from act png
have dblock: \{dblock s' q\} \subseteq \{dblock s q\}
and disk: disk s' = disk s
and blk: blocksRead s' q = blocksRead s q
by(auto simp add: EndPhase1-def InitializePhase-def)
from disk
have disk': \{disk s' d q \mid d . d \in UNIV\} \subseteq \{disk s d q \mid d . d \in UNIV\} (is ?D' \subseteq ?D)
by auto
from png act
have (UN qq d. rdBy s' q qq d) \subseteq (UN qq d. rdBy s q qq d)
by(auto simp add: EndPhase1-def InitializePhase-def rdBy-def split: if-split_asm, blast)
hence \{block br \mid br. br \in (UN qq d. rdBy s' q qq d)\} \subseteq \{block br \mid br. br \in (UN qq d. rdBy s q qq d)\} (is ?R' \subseteq ?R)
by auto blast
from union-inclusion[OF dblock union-inclusion[OF disk this]]
show \?thesis
by(auto simp add: blocksOf-def)
qed

lemma HEndPhase1-allBlocks:
assumes act: HEndPhase1 s s' p
shows allBlocks s' \subseteq allBlocks s \cup \{dblock s' p\}

proof(auto simp del: HEndPhase1-def simp add: allBlocks-def
dest: HEndPhase1-blocksOf-q[OF act])
fix x pa
assume x-pa: x \in blocksOf s' pa and
x-nblks: \forall xa. x \notin blocksOf s xa
show \(x=dblock s' p\)

proof(cases p=pa)
case True
from x-nblks
have x \notin blocksOf s p
by auto
with True subsetD[OF HEndPhase1-blocksOf[OF act] x-pa]

104
show ?thesis
  by auto
next
  case False
  from x-nblks subsetD[OF HEndPhase1-blocksOf-q[OF False] x-pa]
  show ?thesis
  by auto
qed

lemma HEndPhase1-HInv5-q:
  assumes act: HEndPhase1 s s' p
  and inv: HInv5 s
  and inv1: Inv1 s
  and inv2a: Inv2a s'
  and inv2a-q: Inv2a q
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  and inv3: HInv3 s
  and phase': phase s' q = 2
  and pmq: p ≠ q
  and asm4: ¬maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
  shows (∃ D∈MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s' q) < mbal(disk s d qq)
      ∧ ¬hasRead s' q d qq))
proof (cases maxBalInp s (bal(dblock s q)) (inp(dblock s q)))
  case True
  have p21: bal(dblock s q) < bal(dblock s' p) ∧ inp(dblock s q) ≠ inp(dblock s' p)
  proof (cases maxBalInp s (bal(dblock s q)) (inp(dblock s q))]
    have p32: bal(dblock s q) ≤ bal(dblock s' p)
      ∧ inp(dblock s q) ≠ inp(dblock s' p)
      by (auto simp add: maxBalInp-def)
    from inv2a
have bal(dblock s' p) ∈ Ballot p ∪ {θ}
  by(auto simp add: Inv2a-def Inv2a-inner-def
       Inv2a-innermost-def blocksOf-def)

moreover
from Ballot-disj Ballot-nzero pnq
have Ballot q ∩ (Ballot p ∪ {θ}) = {}
  by auto
ultimately
have bal(dblock s' p) ≠ bal(dblock s q)
  using bal-dblk-q
  by auto
with p32
show ?thesis
  by auto
qed

have ∃D∈MajoritySet.∀d∈D. bal(dblock s q) < mbal(disk s d p) ∧ hasRead s p d q
proof
  from act
  have ∃D∈MajoritySet.∀d∈D. d∈disksWritten s p ∧ (∀q∈UNIV−{p}. hasRead s p d q)
    by(auto simp add: EndPhase1-def MajoritySet-def)
  then obtain D
    where act1: ∀d∈D. d∈disksWritten s p ∧ (∀q∈UNIV−{p}. hasRead s p d q)
      and Dmaj: D∈MajoritySet
    by auto
  from inv2b
  have ∀d. Inv2b-inner s p d
    by(auto simp add: Inv2b-def)
  with act1 pnq phase-p bal
  have ∀d∈D. bal(dblock s' p) = mbal(disk s d p) ∧ hasRead s p d q
    by(auto simp add: Inv2b-def Inv2b-inner-def)
  with p21 Dmaj
  have ∀d∈D. bal(dblock s q) < mbal(disk s d p) ∧ hasRead s p d q
    by auto
  with Dmaj
  show ?thesis
    by auto
qed

then obtain D
  where p22: D∈MajoritySet ∧ (∀d∈D. bal(dblock s q) < mbal(disk s d p) ∧ hasRead s p d q)
    by auto
  have p23: ∀d∈D.(∃block=dblock q s, proc=q) /∈ blocksRead s p d
    proof
      have dblock s q ∈ allBlocksRead s p → inp(dblock s' p) = inp(dblock s q)
      proof
        assume dblock-q: dblock s q ∈ allBlocksRead s p
        from inv2a-q

  106
have \((\text{bal}(\text{dblock } s \ q) = 0) = (\text{inp}(\text{dblock } s \ q) = \text{NotAnInput})\)
by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def Inv2a-innermost-def)
with \text{bal-dblk-q Ballot-nzero dblock-q InputsOrNi}
have \text{dblock-q-nib}: \text{dblock } s \ q \in \text{nonInitBlks } s \ p
by (auto simp add: nonInitBlks-def blocksSeen-def)
with \text{act}
have \text{dblock-max}: \text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{maxBlk } s \ p)
by (auto simp add: EndPhase1-def)
from \text{maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]}
have \text{max-in-nib}: \text{maxBlk } s \ p \in \text{nonInitBlks } s \ p ..
hence \text{nonInitBlks } s \ p \subseteq \text{allBlocks } s
by (auto simp add: allBlocks-def nonInitBlks-def blocksSeen-def blocksOf-def rdBy-def allBlocksRead-def allRdBlks-def)
with \text{True subsetD[OF this max-in-nib]}
have \text{bal (dblock } s \ q) \leq \text{bal (maxBlk } s \ p) \longrightarrow \text{inp (maxBlk } s \ p) = \text{inp (dblock } s \ q)
by (auto simp add: maxBalInp-def)
with \text{maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]}
dblock-q-nib dblock-max
show \text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{dblock } s \ q)
by auto
qed
with \text{p21}
have \text{dblock } s \ q \notin \text{block } \text{allRdBlks } s \ p
by (auto simp add: allBlocksRead-def)
hence \forall d. \text{dblock } s \ q \notin \text{block } \text{blocksRead } s \ p \ d
by (auto simp add: allRdBlks-def)
thus \text{?thesis}
by force
qed
have \text{p24}: \forall d \in D. \neg (\exists b r \in \text{blocksRead } s \ q \ d. \text{bal}(\text{dblock } s \ q) \leq \text{mbal (block } b r))
proof
from \text{inv2c phase}
have \forall d. \forall b r \in \text{blocksRead } s \ q \ d. \text{mbal (block } b r) < \text{mbal (dblock } s \ q)
and \text{bal}(\text{dblock } s \ q) = \text{mbal (dblock } s \ q)
by (auto simp add: Inv2c-def Inv2c-inner-def)
thus \text{?thesis}
by force
qed
have \text{p25}: \forall d \in D. \neg \text{hasRead } s \ q \ d \ p
proof auto
fix d
assume \text{d-in-D}: d \in D
and \text{hasRead-qdp}: \text{hasRead } s \ q \ d \ p
have \text{p31}: (\text{block=} \text{dblock } s \ p, \text{proc=} p) \in \text{blocksRead } s \ q \ d
proof
from \text{d-in-D \text{p22}
have hasRead-pdq; hasRead s p d q by auto
with hasRead-qdp phase phase-p inv3
have HInv3-R s q p d
  by(auto simp add: HInv3-def HInv3-inner-def HInv3-L-def)
with p23 d-in-D
show ?thesis
  by(auto simp add: HInv3-R-def)
qed
from p21 act
have p32: bal(dblock s q) < mbal(dblock s p)
  by(auto simp add: EndPhase1-def)
with p31 d-in-D hasRead-qdp p24
show False
  by(force)
qed
with p22
show ?thesis
  by auto
next
  case False
  with inv phase
  show ?thesis
    by(auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def)
qed
then obtain D qq
  where D∈MajoritySet ∧ (∀d∈D. bal(dblock s q) < mbal(disk s d qq)
                          ∧ ¬hasRead s q d qq)
    by auto
moreover
  from act pnq
  have ∀d. hasRead s' q d qq = hasRead s q d qq
    by(auto simp add: EndPhase1-def InitializePhase-def hasRead-def)
ultimately show ?thesis
  using disk dblock
    by auto
qed

theorem HEndPhase1-HInv5:
assumes act: HEndPhase1 s s' p
and inv: HInv5 s
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2a': Inv2a s'
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and inv4: HInv4 s
shows HInv5-inner s' q
using HEndPhase1-HInv5-p[OF act inv4 inv2a inv2a' inv2c]
\begin{verbatim}
HEndPhase1-HInv5-q[OF act inv inv1 inv2a' inv2a inv2h inv2c inv3, of q]
by(auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def)

lemma HFail-HInv5-p:
HFail s s' p \Rightarrow HInv5-inner s' p
by(auto simp add: Fail-def HInv5-inner-def)

lemma HFail-blocksOf-q:
assumes act: HFail s s' p
and pnq: p\neq q
shows blocksOf s' q \subseteq blocksOf s q
using assms
by(auto simp add: Fail-def InitializePhase-def blocksOf-def rdBy-def)

lemma HFail-allBlocks:
assumes act: HFail s s' p
shows allBlocks s' \subseteq allBlocks s \cup \{dblock s' p\}
proof(auto simp del: HFail-def simp add: allBlocks-def
dest: HFail-blocksOf-q[OF act])
fix x pa
assume x-pa: x \in blocksOf s' pa and
x-nblks: \forall xa. x \notin blocksOf s xa
show x=dblock s' pa
proof(cases p=pa)
  case True
  from x-nblks
  have x \notin blocksOf s p
  by auto
  with True subsetD[OF HFail-blocksOf[OF act] x-pa]
  show \?thesis
  by auto
  next
  case False
  from x-nblks subsetD[OF HFail-blocksOf-q[OF act False] x-pa]
  show \?thesis
  by auto
qed

ded

lemma HFail-HInv5-q1:
assumes act: HFail s s' p
and pnq: p\neq q
and inv2a: Inv2a-inner s' q
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(auto simp add: maxBalInp-def)
fix bk
assume bk: bk \in allBlocks s'
and bal: bal (dblock s' q) \leq bal bk
\end{verbatim}
from act pnq
have dblock': dblock s' q = dblock s q by (auto simp add: Fail-def)
from subsetD[OF HFail-allBlocks[OF act] bk]
show inp bk = inp (dblock s' q)
proof
assume bk: bk ∈ allBlocks s
with inv5-1 dblock' bal
show ?thesis
  by (auto simp add: maxBalInp-def)
next
assume bk: bk ∈ {dblock s' p}
with act have bk-init: bk = InitDB
with bal
  have bal (dblock s' q)=0
    by (auto simp add: InitDB-def)
with inv2a
  have inp (dblock s' q)= NotAnInput
    by (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with bk-init
  show ?thesis
    by (auto simp add: InitDB-def)
qed
qed

lemma HFail-HInv5-q2:
  assumes act: HFail s s' p
  and pnq: p ≠ q
  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)
  shows \exists D∈MajoritySet. \exists qq. (\forall d ∈ D. bal(dblock s' q) < mbal(disk s' d qq)
                      ∧ ¬hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
    and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
    and dblock: dblock s' q = dblock s q
    by (auto simp add: Fail-def InitializePhase-def)
  with inv5-2
  show ?thesis
    by (auto simp add: hasRead-def)
qed

lemma HFail-HInv5-q:
  assumes act: HFail s s' p
  and inv: HInv5-inner s q
  and pnq: p ≠ q
  and inv2a: Inv2a s'
shows $HInv5$-inner $s' q$
proof (auto simp add: $HInv5$-inner-def $HInv5$-inner-R-def)
  assume phase': phase $s' q = 2$
  and nR2: $\forall D \in$ MajoritySet.
    $\forall qa. \exists d \in D. \bal (\dblock s' q) < \mbal (\disk s' d qa) \rightarrow$
    hasRead $s' q d qa$ (is $?P s'$)
  from $HFail$-$HInv5$-q2[OF act pnq]
  have $\neg (?P s) \implies \neg (?P s')$
    by auto
  with nR2
  have $P$: $?P s$ by blast
  from inv HFail-$HInv5$-q1
  show $maxBalInp s (\bal (\dblock s' q)) (\inp (\dblock s' q))$
    by (auto simp add: $HInv5$-inner-def $HInv5$-inner-R-def)
qed

theorem $HFail$-$HInv5$:
  $[ [ HFail s s' p; HInv5$-inner $s q; Inv2a s' ] ] \implies HInv5$-inner $s' q$
by (blast dest: $HFail$-$HInv5$-q $HFail$-$HInv5$-p)

lemma $HPhase0Read$-$HInv5$-p:
  $HPhase0Read s s' p d \implies HInv5$-inner $s' p$
by (auto simp add: $Phase0Read$-def $HInv5$-inner-def)

lemma $HPhase0Read$-allBlocks:
  assumes act: $HPhase0Read$ s $s' p d$
  shows allBlocks $s' \subseteq$ allBlocks s
  using $HPhase0Read$-blocksOf[OF act]
by (auto simp add: allBlocks-def)

lemma $HPhase0Read$-$HInv5$-1:
  assumes act: $HPhase0Read$ s $s' p d$
  and inv5-1: $maxBalInp s (\bal (\dblock s q)) (\inp (\dblock s q))$
  shows $maxBalInp s' (\bal (\dblock s' q)) (\inp (\dblock s' q))$
  using assms and $HPhase0Read$-blocksOf[OF act]
by (auto simp add: $Phase0Read$-def $maxBalInp$-def allBlocks-def)

lemma $HPhase0Read$-$HInv5$-q2:
  assumes act: $HPhase0Read$ s $s' p d$
  and pnq: $p \neq q$
  and inv5-2: $\exists D \in$ MajoritySet. $\exists qq. (\forall d \in D. \bal (\dblock s q) < \mbal (\disk s d qq)$
    $\wedge \neg$ hasRead $s q d qq)$

111
shows $\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \quad \text{bal} (\text{dblock } s' q) < \text{mbal} (\text{disk } s' d q) \wedge \neg \text{hasRead } s' d q q)$

proof –
from act pnq
have disk s' = disk s
  and blocksRead: $\forall d. \text{blocksRead } s' q d = \text{blocksRead } s q d$
  and dblock: $\text{dblock } s' q = \text{dblock } s q$
  by(auto simp add: Phase0Read-def InitializePhase-def)
with inv5-2
show ?thesis
  by(auto simp add: hasRead-def)
qed

lemma HPhase0Read-HInv5-q:
  assumes act: HPhase0Read s s' p d
  and inv: HInv5-inner s q
  and pnq: p\neq q
  shows HInv5-inner s' q
proof(auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase s' q = 2
and i2: $\forall D \in \text{MajoritySet}. \forall q a. \exists d \in D. \text{bal} (\text{dblock } s' q) < \text{mbal} (\text{disk } s' d q a)$
  → hasRead s' q d qa
from phase' act have phase: phase s q = 2
  by(auto simp add: Phase0Read-def)
show maxBalInp s' (bal (dblock s' q)) (∫ (dblock s' q))
proof(rule HPhase0Read-HInv5-1[OF act, of q])
  from HPhase0Read-HInv5-q2[OF act pnq] inv i2 phase
    show maxBalInp s (bal (dblock s q)) (∫ (dblock s q))
      by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

theorem HPhase0Read-HInv5:
  $\exists \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \quad \text{bal} (\text{dblock } s' q) < \text{mbal} (\text{disk } s' d q) \wedge \neg \text{hasRead } s' d q q)$
by(blast dest: HPhase0Read-HInv5-q HPhase0Read-HInv5-p)

lemma HEndPhase0-HInv5-p:
  HEndPhase0 s s' p =⇒ HInv5-inner s' q
by(auto simp add: EndPhase0-def HInv5-inner-def)

lemma HEndPhase0-blocksOf-q:
  assumes act: HEndPhase0 s s' p
  and pnq: p\neq q
  shows blocksOf s' q ⊆ blocksOf s q
proof –
from act pnq
have dblock: $\{ \text{dblock } s' q \} \subseteq \{ \text{dblock } s q \}$
  and disk: disk s' = disk s
and \( \text{blks} : \text{blocksRead} s' q = \text{blocksRead} s q \)

by (auto simp add: EndPhase0-def InitializePhase-def)
from disk
have \( \text{disk}' : \{ \text{disk} s' d q \mid d \in \text{UNIV} \} \subseteq \{ \text{disk} s d q \mid d \in \text{UNIV} \} \) \((\text{is } ?D' \subseteq ?D)\)
  by auto
from pnq act
have \( \{ \text{UN qq d. rdBy s q qq d} \} \subseteq \{ \text{UN qq d. rdBy s' q qq d} \} \)
  by (auto simp add: EndPhase0-def InitializePhase-def rdBy-def split: if-split_asm, blast)

hence \( \{ \text{block br} \mid br. br \in \{ \text{UN qq d. rdBy s' q qq d} \} \} \subseteq \{ \text{block br} \mid br. br \in \{ \text{UN qq d. rdBy s q qq d} \} \} \)
  by auto blast
from union-inclusion[of dblock union-inclusion[of disk' this]]
show \( \text{thesis} \)
  by (auto simp add: blocksOf-def)
qed

lemma HEndPhase0-allBlocks:
  assumes act: HEndPhase0 s s' p
  shows allBlocks s' \( \subseteq \) allBlocks s \( \cup \) \{ dblock s' p \}
proof (auto simp del: HEndPhase0-def simp add: allBlocks-def dest: HEndPhase0-blocksOf-q[OF act])
  fix x pa
  assume x-pa: \( x \in \text{blocksOf} s' \) pa and
    x-nblks: \( \forall xa. x \notin \text{blocksOf} s xa \)
  show \( x = \text{dblock} s' p \)
  proof (cases p = pa)
    case True
    from x-nblks
    have \( x \notin \text{blocksOf} s p \)
      by auto
    with True subsetD[of HEndPhase0-blocksOf[q[OF act]] x-pa]
    show \( \text{thesis} \)
      by auto
  next
    case False
    from x-nblks subsetD[of HEndPhase0-blocksOf-q[OF act False] x-pa]
    show \( \text{thesis} \)
      by auto
  qed
qed

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

proof(auto simp add: maxBalInp-def)

fix \( bk \)

assume \( bk: bk \in \text{allBlocks} \ s' \)

and \( \text{bal}: \text{bal} \ (\text{dblock} \ s' \ q) \leq \text{bal} \ bk \)

from act pnq
have \( \text{dblock}': \text{dblock} \ s' \ q = \text{dblock} \ s \ q \) by(auto simp add: EndPhase0-def)

from subsetD[OF HEndPhase0-allBlocks[OF act] bk]
show \( \text{inp} \ bk = \text{inp} \ (\text{dblock} \ s' \ q) \)

proof

assume \( bk: bk \in \text{allBlocks} \ s \)

with inv5-1 dblock
show \(?thesis\)

by(auto simp add: maxBalInp-def)

next

assume \( bk: bk \in \{ \text{dblock} \ s' \ p \} \)

with HEndPhase0-some[OF act inv1]
have \( \exists ba \in \text{allBlocksRead} \ s \ p. \ \text{bal} \ ba = \text{bal} \ (\text{dblock} \ s' \ p) \land \text{inp} \ ba = \text{inp} \ (\text{dblock} \ s' \ p) \)

by(auto simp add: EndPhase0-def)

then obtain \( ba \)

where ba-blksread: \( ba \in \text{allBlocksRead} \ s \ p \)

and ba-balinp: \( \text{bal} \ ba = \text{bal} \ (\text{dblock} \ s' \ p) \land \text{inp} \ ba = \text{inp} \ (\text{dblock} \ s' \ p) \)

by auto

have \( \text{allBlocksRead} \ s \ p \subseteq \text{allBlocks} \ s \)

by(auto simp add: allBlocksRead-def allRdBlks-def allBlocks-def blocksOf-def rdBy-def)

from subsetD[OF this ba-blksread] ba-balinp bal bk dblock' inv5-1
show \(?thesis\)

by(auto simp add: maxBalInp-def)

qed

qed

lemma HEndPhase0-Hinv5-q2:

assumes act: \( \text{HEndPhase0} \ s \ s' \ p \)

and png: \( p \neq q \)

and inv5-2: \( \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(\text{dblock} \ s \ q) < \text{mbal}(\text{disk} \ s \ d \ qq) \)

\quad \land \ \neg \text{hasRead} \ s \ q \ d \ qq \)

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

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

proof –

from act png
have \( \text{disk}: \text{disk} \ s' = \text{disk} \ s \)

and blocksRead: \( \forall d. \ \text{blocksRead} \ s' \ q \ d = \text{blocksRead} \ s \ q \ d \)

and dblock: \( \text{dblock} \ s' \ q = \text{dblock} \ s \ q \)

by(auto simp add: EndPhase0-def InitializePhase-def)

with inv5-2
show \(?thesis\)
by (auto simp add: hasRead-def)
qed

lemma HEndPhase0-HInv5-q:
assumes act: HEndPhase0 s s' p
and inv: HInv5-inner s q
and inv1: Inv1 s
and pq: p≠q
shows HInv5-inner s' q
using assms and
  HEndPhase0-HInv5-q1[OF act inv1]
  HEndPhase0-HInv5-q2[OF act pq]
by (auto simp add: HInv5-inner-def HInv5-inner-R-def EndPhase0-def)

theorem HEndPhase0-HInv5:
  [ HEndPhase0 s s' p; HInv5-inner s q; Inv1 s ] ⇒ HInv5-inner s' q
by (blast dest: HEndPhase0-HInv5-p)

HInv1 ∧ HInv2 ∧ HInv3 ∧ HInv4 ∧ HInv5 is an invariant of HNext.

lemma I2c:
assumes nxt: HNext s s'
and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s
shows HInv5 s'
using assms
by (auto simp add: HInv5-def HNext-def Next-def, auto simp add: HStartBallot-HInv5,
  auto intro: HPhase0Read-HInv5,
  auto simp add: HInv4-def intro: HPhase1or2Write-HInv5,
  auto simp add: Phase1or2Read-def
  intro: HPhase1or2ReadThen-HInv5
  HPhase1or2ReadElse-HInv5,
  auto simp add: EndPhase1or2-def HInv1-def HInv4-def HInv5-def
  intro: HEndPhase1-HInv5
  HEndPhase2-HInv5,
  auto intro: HFail-HInv5,
  auto intro: HEndPhase0-HInv5 simp add: HInv1-def)
end

theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

C.6 Lemma I2f

To prove the final conjunct we will use the predicate valueChosen(v). This predicate is true if v is the only possible value that can be chosen as output. It also asserts that, for every disk d in D, if q has already read disksdp, then it has read a block with bal field at least b.
definition valueChosen :: state ⇒ InputsOrNi ⇒ bool
where
valueChosen s v =
(∃ b ∈ (UN p. Ballot p).
maxBalInp s b v
∧ (∃ p. ∃ D∈ MajoritySet.(∀ d∈D. b ≤ bal(disk s d p)
∧ (∀ q.( phase s q = 1
∧ b ≤ mbal(dblock s q)
∧ hasRead s q d p
)) → (∃ br∈blocksRead s q d. b ≤ bal(block br))))

lemma HEndPhase1-valueChosen-inp:
assumes act: HEndPhase1 s s' q
and inv2a: Inv2a s s'
and asm1: b ∈ (UN p. Ballot p)
and bk-blocksOf: bk∈blocksOf s r
and bk bk-blocksOf: bk∈blocksSeen s q
and b-bal: b ≤ bal bk
and asm3: maxBalInp s b v
and inv1: Inv1 s
shows inp(dblock s' q) = v
proof −
from bk-blocksOf inv2a
have inv2a-bk: Inv2a-innermost s r bk
  by(auto simp add: Inv2a-def Inv2a-inner-def)
from Ballot-nzero asm1
have 0 < b  by auto
with b-bal
have 0 < bal bk  by auto
with inv2a-bk
have inp bk ≠ NotAnInput
  by(auto simp add: Inv2a-innermost-def)
with bk InputsOrNi
have bk-noninit: bk ∈ nonInitBlks s q
  by(auto simp add: nonInitBlks-def blocksSeen-def
  allBlocksRead-def allRdBlks-def)
with maxBlk-in-nonInitBlks[OF this inv1] b-bal
have maxBlk-b: b ≤ bal (maxBlk s q)
  by auto
from maxBlk-in-nonInitBlks[OF bk-noninit inv1]
have ∃ p d. maxBlk s q ∈ blocksSeen s p
  by(auto simp add: nonInitBlks-def blocksSeen-def)
  hence ∃ p. maxBlk s q ∈ blocksOf s p
    by(auto simp add: blocksOf-def blocksSeen-def
    allBlocksRead-def allRdBlks-def rdBy-def, force)
with maxBlk-b asm3
have inp(maxBlk s q) = v
  by(auto simp add: maxBalInp-def allBlocks-def)
with bk-noninit act
show ?thesis
  by(auto simp add: EndPhase1-def)
qed

lemma HEndPhase1-maxBalInp:
  assumes act: HEndPhase1 s s' q
    and asm1: b ∈ (UN p. Ballot p)
    and asm2: D ∈ MajoritySet
    and asm3: maxBalInp s b v
    and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
      ∧ (∀ q.( phase s q = 1
                ∧ b ≤ mbal(dblock s q)
                ∧ hasRead s q d p
                  ) −→ (∃ br ∈ blocksRead s q d. b ≤ bal(block br)))
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2b: Inv2b s
shows maxBalInp s' b v
proof(cases b ≤ mbal(dblock s q))
case True
  show ?thesis
  proof(cases p ≠ q)
    assume pnq: p ≠ q
    have ∃ d ∈ D. hasRead s q d p
      proof −
        from act
        have IsMajority({ d. d ∈ disksWritten s q ∧ (∀ r ∈ UNIV − {q}. hasRead s q d r)}) (is IsMajority(?M))
          by(auto simp add: EndPhase1-def)
        with majorities-intersect asm2
        have D ∩ ?M ≠ {}
          by(auto simp add: MajoritySet-def)
        hence ∃ d ∈ D. (∀ r ∈ UNIV − {q}. hasRead s q d r)
          by auto
        with pnq
        show ?thesis
          by auto
      qed
    then obtain d where p41: d ∈ D ∧ hasRead s q d p by auto
    with asm4 asm3 act True
    have p42: ∃ br ∈ blocksRead s q d. b ≤ bal(block br)
      by(auto simp add: EndPhase1-def)
    from True act
    have thesis-L: b ≤ bal(dblock s' q)
      by(auto simp add: EndPhase1-def)
    from p42
    have inp(dblock s' q) = v
  qed
proof auto
  fix br
  assume br: br ∈ blocksRead s q d
    and b-bal: b ≤ bal (block br)
  hence br-rdBy: br ∈ (UN q d. rdBy s (proc br) q d)
    by(auto simp add: rdBy-def)
  hence br-blksof: block br ∈ blocksOf s (proc br)
    by(auto simp add: blocksOf-def)
  from br have br-bseen: block br ∈ blocksSeen s q
    by(auto simp add: blocksSeen-def allBlocksRead-def allRdBlks-def)
  from HEndPhase1-valueChosen-inp[OF act inv2a asm1 br-blksof br-bseen b-bal
asm3 inv1]
    show ?thesis .
qed
next
  case False
  from asm4
  have p41: ∀d∈D. b ≤ bal(disk s d p)
    by auto
  have p42: ∃d∈D. disk s d p = dblock s p
  proof –
    from act
    have IsMajority {d. d∈disksWritten s q ∧ (∀p∈UNIV−{q}. hasRead s q d p)}
      (is IsMajority ?S)
      by(auto simp add: EndPhase1-def)
    with majorities-intersect asm2
    have D ∩ ?S ≠ {}
      by(auto simp add: MajoritySet-def)
    hence ∃d∈D. d∈disksWritten s q
      by auto
    with inv2b False
    show ?thesis
      by(auto simp add: Inv2b-def Inv2b-inner-def)
qed
  have inp(dblock s' q) = v
  proof –
    from p42 p41 False
    have b-bal: b ≤ bal(dblock s q) by auto
    have db-blksof: (dblock s q) ∈ blocksOf s q
      by(auto simp add: blocksOf-def)
    have db-bseen: (dblock s q) ∈ blocksSeen s q
      by(auto simp add: blocksSeen-def)
    from HEndPhase1-valueChosen-inp[OF act inv2a asm1 db-blksof db-bseen
b-bal asm3 inv1]
    show ?thesis .
qed
with asm3 HEndPhase1-allBlocks[OF act]
show ?thesis
  by (auto simp add: maxBalInp-def)
qed
next
case False
have dblock s' q ∈ allBlocks s'
  by (auto simp add: allBlocks-def blocksOf-def)
show ?thesis
  proof (auto simp add: maxBalInp-def)
      fix bk
      assume bk: bk ∈ allBlocks s
      and b-bal: b ≤ bal bk
      from subsetD[OF HEndPhase1-allBlocks[OF act] bk]
      show inp bk = v
          proof
              assume bk: bk ∈ allBlocks s
              with asm3 b-bal
              show ?thesis
              by (auto simp add: maxBalInp-def)
          next
              assume bk: bk ∈ {dblock s' q}
              from act False
              have ¬ b ≤ bal (dblock s' q)
              by (auto simp add: EndPhase1-def)
              with bk b-bal
              show ?thesis
              by (auto)
          qed
          qed
lemma HEndPhase1-valueChosen2:
  assumes act: HEndPhase1 s s' q
  and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
      ∧ (∀ q. (phase s q = 1
        ∧ b ≤ mbal(dblock s q)
        ∧ hasRead s q d p
      ) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br))) (is ?P s)
  shows ?P s'
proof (auto)
  fix d
  assume d: d ∈ D
  with act asm4
  show b ≤ bal (disk s' d p)
      by (auto simp add: EndPhase1-def)
  fix d q
  assume d: d ∈ D
  and phase': phase s' q = Suc 0
\[ \text{and dblk-mbal: } b \leq \text{mbal (dblock s' q)} \]

\textbf{with act}

\textbf{have} \( p31: \text{phase s q} = 1 \)
\textbf{and} \( p32: \text{dblock s' q} = \text{dblock s q} \)
\textbf{by} (auto simp add: EndPhase1-def split: if-split-asm)

\textbf{with dblk-mbal}
\textbf{have} \( b \leq \text{mbal(dblock s q)} \) \textbf{by} auto

\textbf{moreover}
\textbf{assume} \( \text{hasRead: hasRead s' q d p} \)

\textbf{with act}
\textbf{have} \( \text{hasRead s q d p} \)
\textbf{by} (auto simp add: EndPhase1-def InitializePhase-def hasRead-def split: if-split-asm)

\textbf{ultimately}
\textbf{have} \( \exists \text{br} \in \text{blocksRead s q d}. \ b \leq \text{bal(block br)} \)

\textbf{using} \( p31 \) \( \text{asm4 d} \)
\textbf{by} blast

\textbf{with act hasRead}
\textbf{show} \( \exists \text{br} \in \text{blocksRead s' q d}. \ b \leq \text{bal(block br)} \)
\textbf{by} (auto simp add: EndPhase1-def InitializePhase-def hasRead-def)

\textbf{qed}

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

\textbf{proof} –
\textbf{from} \( \text{vc} \)
\textbf{obtain} \( b \ p \ D \) \textbf{where}
\( \text{asm1: } b \in (\text{UN p. Ballot p}) \)
\( \text{and} \text{asm2: } D \in \text{MajoritySet} \)
\( \text{and asm3: } \text{maxBalInp s b v} \)
\( \text{and asm4: } \forall d \in D. \ b \leq \text{bal(disk s d p)} \)
\( \wedge (\forall q. (\text{phase s q} = 1) \wedge b \leq \text{mbal(dblock s q)} \wedge \text{hasRead s q d p}) \rightarrow (\exists \text{br} \in \text{blocksRead s q d}. \ b \leq \text{bal(block br)})) \)
\textbf{by} (auto simp add: valueChosen-def)

\textbf{from} \( \text{HEndPhase1-maxBalInp[OF act asm1 asm2 asm3 asm4 inv1 inv2a inv2b]} \)
\textbf{have} \( \text{maxBalInp s' b v} \).

\textbf{with} \( \text{HEndPhase1-valueChosen2[OF act asm4]} \) \( \text{asm1 asm2} \)
\textbf{show} \( \text{thesis} \)
\textbf{by} (auto simp add: valueChosen-def)

\textbf{qed}
lemma HStartBallot-maxBalInp:
assumes act: HStartBallot s s' q
and asm3: maxBalInp s b v
shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
fix bk
assume bk: bk ∈ allBlocks s'
and b-bal: b ≤ bal bk
from subsetD[OF HStartBallot-allBlocks[OF act] bk]
show inp bk = v
proof
assume bk: bk∈allBlocks s
with asm3 b-bal
show ?thesis
  by (auto simp add: maxBalInp-def)
next
assume bk: bk∈{dblock s' q}
from asm3
have b≤ bal(dblock s q) ⟹ inp(dblock s q) = v
  by (auto simp add: maxBalInp-def allBlocks-def blocksOf-def)
with act bk b-bal
show ?thesis
  by (auto simp add: StartBallot-def)
qed
qed

lemma HStartBallot-valueChosen2:
assumes act: HStartBallot s s' q
and asm4: ∀ d∈ D. b ≤ bal(disk s d p)
∧ (∀ q. (phase s q = 1
∧ b ≤ mbal(dblock s q)
∧ hasRead s q d p
) ⟹ (∃ br∈blocksRead s q d. b ≤ bal(block br))) (is ?P s)
shows ?P s'
proof (auto)
fix d
assume d: d∈ D
with act asm4
show b ≤ bal (disk s' d p)
  by (auto simp add: StartBallot-def)
fix d q
assume d: d∈ D
and phase': phase s' q = Suc 0
and dblk-mbal: b ≤ mbal (dblock s' q)
and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
and p32: dblock s' q = dblock s q
  by (auto simp add: StartBallot-def InitializePhase-def)
hasRead-def split : if-split-asm)

with dblk-mbal
have b≤mbal(dblock s q) by auto
moreover
from act hasRead
have hasRead s q d p
  by(auto simp add: StartBallot-def InitializePhase-def
       hasRead-def split: if-split-asm)
ultimately
have∃ br∈blocksRead s q d. b≤ bal(block br)
  using p$\exists$I asm4 d
  by blast
with act hasRead
show∃ br∈blocksRead s′ q d. b≤ bal(block br)
  by(auto simp add: StartBallot-def InitializePhase-def
       hasRead-def)

qed

theorem HStartBallot-valueChosen:
assumes act: HStartBallot s s′ q
and vc: valueChosen s v
and v-input: v∈ Inputs
shows valueChosen s′ v
proof –
  from vc
  obtain b p D where
    asm1: b∈(UN p. Ballot p)
    and asm2: D∈MajoritySet
    and asm3: maxBalInp s b v
    and asm4: ∀ d∈D. b≤ bal(disk s d p)
     ∧ (∀ q.( phase s q = 1
       ∧ b≤mbal(dblock s q)
       ∧ hasRead s q d p
     ) −→ (∃ br∈blocksRead s q d. b≤ bal(block br))))
    by(auto simp add: valueChosen-def)
  from HStartBallot-maxBalInp[OF act asm3]
  have maxBalInp s′ b v.
  with HStartBallot-valueChosen2[OF act asm4] asm1 asm2
  show ?thesis
    by(auto simp add: valueChosen-def)

qed

lemma HPhase1or2Write-maxBalInp:
assumes act: HPhase1or2Write s s′ q d
    and asm3: maxBalInp s b v
shows maxBalInp s′ b v
proof(auto simp add: maxBalInp-def)
fix bk
assume bk: bk∈allBlocks s′
and $b$-bal: $b \leq \text{bal}\ bk$

from subsetD[OF HPhase1or2Write-allBlocks[OF act] bk] asm3 $b$-bal

show $\text{inp}\ bk = v$

by(auto simp add: maxBalInp-def)

qed

lemma HPhase1or2Write-valueChosen2:

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

and asm2: $D \in \text{MajoritySet}$

and asm4: \( \forall d \in D. \ b \leq \text{bal}\ (\text{disk}\ s\ d\ p) \)
\& (\forall q. (\text{phase}\ s\ q = 1 \\
\& b \leq \text{mbal}\ (\text{dblock}\ s\ q) \\
\& \text{hasRead}\ s\ q\ d\ p) \\
\rightarrow (\exists br \in \text{blocksRead}\ s\ q\ d. \ b \leq \text{bal}(\text{block}\ br)))) \) (is ?P s)

and inv4: HInv4a $s\ pp$

shows ?P $s'$

proof(auto)

fix $d1$

assume $d:\ d1 \in D$

show $b \leq \text{bal}\ (\text{disk}\ s'\ d1\ p)$

proof(cases $d1 = d \land pp = p$)


case True

with inv4 act

have HInv4a2 $s\ p$

by(auto simp add: Phase1or2Write-def HInv4a-def)

with asm2 majorities-intersect

have $\exists dd \in D. \ \text{bal}(\text{disk}\ s\ dd\ p) \leq \text{bal}(\text{dblock}\ s\ p)$

by(auto simp add: HInv4a2-def MajoritySet-def)

then obtain $dd$ where p41: $dd \in D \land \text{bal}(\text{disk}\ s\ dd\ p) \leq \text{bal}(\text{dblock}\ s\ p)$

by auto

from asm4 p41

have $b \leq \text{bal}(\text{disk}\ s\ dd\ p)$

by auto

with p41

have p42: $b \leq \text{bal}(\text{dblock}\ s\ p)$

by auto

from act True

have $\text{dblock}\ s\ p = \text{disk}\ s'\ d\ p$

by(auto simp add: Phase1or2Write-def)

with p42 True

show ?thesis

by auto

next

case False

with act asm4 d

show ?thesis

by(auto simp add: Phase1or2Write-def)

qed

next
fix \( d \ q \)
assume \( d: d \in D \)
\[\text{and phase': phase s' q = Suc 0} \]
\[\text{and dblk-mbal: b \leq mbal (dblock s' q)} \]
\[\text{and hasRead: hasRead s' q d p} \]
from phase' act hasRead
have \( p31: \text{phase s q = 1} \)
\[\text{and p32: dblock s' q = dblock s q} \]
by (auto simp add: Phase1or2Write-def InitializePhase-def
hasRead-def split : if-split-asm)
with dblk-mbal
have \( b \leq \text{mbal (dblock s q)} \) by auto
moreover
from act hasRead
have \( \text{hasRead s q d p} \)
by (auto simp add: Phase1or2Write-def InitializePhase-def
hasRead-def split : if-split-asm)
ultimately
have \( \exists \ br \in \text{blocksRead s q d}. b \leq \text{bal (block br)} \)
using \( p31 \ \text{asm4} \ d \)
by blast
with act hasRead
show \( \exists \ br \in \text{blocksRead s' q d}. b \leq \text{bal (block br)} \)
by (auto simp add: Phase1or2Write-def InitializePhase-def
hasRead-def)

qed

theorem HPhase1or2Write-valueChosen:
assumes \( \text{act: HPhase1or2Write s s' q d} \)
and \( \text{vc: valueChosen s v} \)
and \( \text{v-input: v \in Inputs} \)
and \( \text{inv4: HInv4a s q} \)
shows \( \text{valueChosen s' v} \)
proof –
from vc
obtain \( b p D \) where
\( \text{asm1: b \in (UN p. \text{Ballot p})} \)
and \( \text{asm2: D \in MajoritySet} \)
and \( \text{asm3: maxBalInp s b v} \)
and \( \text{asm4: \( \forall \ d \in D. \ b \leq \text{bal (disk s d p)} \)} \)
\( \land (\forall q.(\text{phase s q = 1}) \land \ b \leq \text{mbal (dblock s q)} \land \text{hasRead s q d p}) \)
\( \longrightarrow (\exists \ br \in \text{blocksRead s q d}. b \leq \text{bal (block br)}) \)
by (auto simp add: valueChosen-def)
from HPhase1or2Write-maxBalInp[OF act asm3]
have maxBalInp s' b v .
with HPhase1or2Write-valueChosen2[OF act asm2 asm4 inv4] asm1 asm2
show \(?thesis \)
by (auto simp add: valueChosen-def)

qed

lemma HPhase1or2ReadThen-maxBalInp:
assumes act: HPhase1or2ReadThen s s' q d p
and asm3: maxBalInp s b v
shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
fix bk
assume bk: bk ∈ allBlocks s'
and b-bal: b ≤ bal bk
from subsetD[OF HPhase1or2ReadThen-allBlocks[OF act] bk] asm3 b-bal
show inp bk = v
  by (auto simp add: maxBalInp-def)
qed

lemma HPhase1or2ReadThen-valueChosen2:
assumes act: HPhase1or2ReadThen s s' q d pp
and asm4: ∀ d ∈ D. b ≤ bal (disk s d p)
∧ (∀ q. (phase s q = 1
  ∧ b ≤ mbal (dblock s q)
  ∧ hasRead s q d p
)) −→ (∃ br ∈ blocksRead s q d. b ≤ bal (block br))) (is ?P s)
shows ?P s'
proof (auto)
fix dd
assume d: dd ∈ D
with act asm4
show b ≤ bal (disk s' dd p)
  by (auto simp add: Phase1or2ReadThen-def)
fix dd qq
assume d: dd ∈ D
and phase': phase s' qq = Suc 0
and dblk-mbal: b ≤ mbal (dblock s' qq)
and hasRead: hasRead s' qq dd p
show (∃ br ∈ blocksRead s' qq dd. b ≤ bal (block br))
proof (cases d = dd ∧ qq = q ∧ pp = p)
case True
from d asm4
have b ≤ bal (disk s dd p)
  by auto
with act True
show ?thesis
  by (auto simp add: Phase1or2ReadThen-def)
next
case False
with phase' act
have p31: phase s qq = 1
and \( p32: \text{dblock } s' \text{ qq } = \text{dblock } s \text{ qq} \)

by (auto simp add: Phase1or2ReadThen-def)

with \( \text{dblkt-mbal} \)

have \( b \leq \text{mbal} (\text{dblock } s \text{ qq}) \) by auto

moreover

from act hasRead False

have hasRead \( s \text{ qq dd p} \)

by (auto simp add: Phase1or2ReadThen-def hasRead-def split: if-split-asm)

ultimately

have \( \exists br \in \text{blocksRead } s \text{ qq dd}. \ b \leq \text{bal} (\text{block } br) \)

using \( p31 \text{ asm4 d} \)

by blast

with act hasRead

show \( \exists br \in \text{blocksRead } s' \text{ qq dd}. \ b \leq \text{bal} (\text{block } br) \)

by (auto simp add: Phase1or2ReadThen-def hasRead-def)

qed

qed

theorem HPhase1or2ReadThen-valueChosen:

assumes act: \( \text{HPhase1or2ReadThen } s \text{ s'} q \text{ d p} \)

and \( \text{vc} : \text{valueChosen } s \text{ v} \)

and \( \text{v-input: } v \in \text{Inputs} \)

shows \( \text{valueChosen } s' \text{ v} \)

proof –

from \( \text{vc} \)

obtain \( b \text{ p D where} \)

\( \text{asm1: } b \in (\text{UN } p. \text{ Ballot } p) \)

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

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

\( \text{and } \text{asm4: } \forall d \in D. \ b \leq \text{bal} (\text{disk } s \text{ d p}) \)

\( \land (\forall q. ( \text{phase } s \text{ q } = 1 \)

\( \land b \leq \text{mbal} (\text{dblock } s \text{ q}) \)

\( \land \text{hasRead } s \text{ q d p} \)

\( ) \rightarrow (\exists br \in \text{blocksRead } s \text{ q d}. \ b \leq \text{bal} (\text{block } br)) ) \)

by (auto simp add: valueChosen-def)

from HPhase1or2ReadThen-maxBalInp[OF act asm3]

have maxBalInp \( s' \text{ b v} . \)

with HPhase1or2ReadThen-valueChosen2[OF act asm4] \( \text{asm1 asm2} \)

show \( \text{thesis} \)

by (auto simp add: valueChosen-def)

qed

theorem HPhase1or2ReadElse-valueChosen:

\[ \text{[ HPhase1or2ReadElse } s \text{ s'} p \text{ d r; valueChosen } s \text{ v}; v \in \text{Inputs} \] \]

\( \rightarrow \text{valueChosen } s' \text{ v} \)

using HStartBallot-valueChosen

by (auto simp add: Phase1or2ReadElse-def)
lemma HEndPhase2-maxBalInp:
assumes act: HEndPhase2 s s' q
and asm3: maxBalInp s b v
shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
fix bk
assume bk: bk ∈ allBlocks s'
and b-bal: b ≤ bal bk
from subsetD[OF HEndPhase2-allBlocks[OF act] bk] asm3 b-bal
show inp bk = v
by (auto simp add: maxBalInp-def)
qed

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)
shows ?P s'
proof (auto)
fix d
assume d: d ∈ D
with act asm4
show b ≤ bal (disk s' d p)
by (auto simp add: EndPhase2-def)
fix d q
assume d: d ∈ D
and phase': phase s' q = Suc 0
and dblk-mbal: b ≤ mbal (dblock s' q)
and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
and p32: dblock s' q = dblock s q
by (auto simp add: EndPhase2-def InitializePhase-def
    hasRead-def split : if-split-thm)
with dblk-mbal
have b ≤ mbal (dblock s q) by auto
moreover
from act hasRead
have hasRead s q d p
by (auto simp add: EndPhase2-def InitializePhase-def
    hasRead-def split : if-split-thm)
ultimately
have ∃ br ∈ blocksRead s q d. b ≤ bal (block br)
using p31 asm4 d
by blast
with act hasRead

127
∃br∈blocksRead s’ q d. \( b \leq \text{bal}(\text{block } br) \)

by(auto simp add: EndPhase2-def InitializePhase-def hasRead-def)

qed

theorem HEndPhase2-valueChosen:
assumes act: HEndPhase2 s s’ q
and vc: valueChosen s v
and v-input: v ∈ Inputs
shows valueChosen s’ v
proof
  from vc
  obtain b p D where
    asm1: b ∈ (UN p. Ballot p)
    and asm2: D ∈ MajoritySet
    and asm3: maxBalInp s b v
    and asm4: \( \forall d ∈ D. \quad b ≤ \text{bal}(\text{disk } s d p) \)
      \( \land \forall q. \quad (\text{phase } s q = 1 \land b ≤ \text{mbal}(\text{dblock } s q) \land \text{hasRead } s q d p) \)
    \( \longrightarrow (\exists br∈\text{blocksRead } s q d. \quad b ≤ \text{bal}(\text{block } br))) \)
  by(auto simp add: valueChosen-def)
  from HEndPhase2-maxBalInp[OF act asm3]
  have maxBalInp s’ b v .
  with HEndPhase2-valueChosen2[OF act asm4] asm1 asm2
  show ?thesis
  by(auto simp add: valueChosen-def)
qed

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(auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk ∈ allBlocks s’
  and b-bal: b ≤ bal bk
  from subsetD[OF HFail-allBlocks[OF act] bk]
  show inp bk = v
proof
  assume bk: bk ∈ allBlocks s
  with asm3 b-bal
  show ?thesis
  by(auto simp add: maxBalInp-def)
next
  assume bk: bk ∈ {dblock s’ q}
  with act
  have bal bk = 0

128
by (auto simp add: Fail-def InitDB-def)
moreover
from Ballot-nzero asm1
have 0 < b
  by auto
ultimately
show ?thesis
  using b-bal
  by auto
qed

lemma HFail-valueChosen2:
  assumes act: HFail s s' q
  and asm4: \( \forall d \in D. \ b \leq \text{bal}(\text{disk} s d p) \land (\forall q. (\text{phase} s q = 1 \land b \leq \text{mbal}(\text{dblock} s q) \land \text{hasRead} s q d p) \rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br))) \) (is ?P s)
  shows ?P s'
proof (auto)
  fix d
  assume d: d \in D
  with act asm4
  show b \leq \text{bal}(\text{disk} s d p)
    by (auto simp add: Fail-def)
  fix d q
  assume d: d \in D
    and phase': phase s' q = Suc 0
    and dblk-mbal: b \leq \text{mbal}(\text{dblock} s' q)
    and hasRead: hasRead s' q d p
  from phase' act hasRead
  have p31: phase s q = 1
    and p32: dblock s' q = dblock s q
  by (auto simp add: Fail-def InitializePhase-def hasRead-def split : if-split-asm)
  with dblk-mbal
  have b \leq \text{mbal}(\text{dblock} s q) by auto
moreover
from act hasRead
have hasRead s q d p
  by (auto simp add: Fail-def InitializePhase-def hasRead-def split : if-split-asm)
ultimately
have \( \exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br) \)
  using p31 asm4 d
  by blast
with act hasRead
show \( \exists br \in \text{blocksRead} s' q d. b \leq \text{bal}(\text{block} br) \)
by (auto simp add: Fail-def InitializePhase-def hasRead-def)

qed

theorem HFail-valueChosen:
  assumes act: HFail s s' q
  and vc: valueChosen s v
  and v-input: v ∈ Inputs
  shows valueChosen s' v
proof -
  from vc
  obtain b p D where
    asm1: b ∈ (UN p. Ballot p)
  and asm2: D ∈ MajoritySet
  and asm3: maxBalInp s b v
  and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
    ∧ (∀ q. (phase s q = 1
      ∧ b ≤ mbal(dblock s q)
      ∧ hasRead s q d p)
    ) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br)))
  by (auto simp add: valueChosen-def)
  from HFail-maxBalInp[OF act asm1 asm3]
  have maxBalInp s' b v.
  with HFail-valueChosen2[OF act asm4] asm1 asm2
  show ?thesis
  by (auto simp add: valueChosen-def)
qed

lemma HPhase0Read-maxBalInp:
  assumes act: HPhase0Read s s' q d
  and asm3: maxBalInp s b v
  shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk ∈ allBlocks s'
  and b-bal: b ≤ bal bk
  from subsetD[OF HPhase0Read-allBlocks[OF act] bk] asm3 b-bal
  show inp bk = v
  by (auto simp add: maxBalInp-def)
qed

lemma HPhase0Read-valueChosen2:
  assumes act: HPhase0Read s s' qq dd
  and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
    ∧ (∀ q. (phase s q = 1
      ∧ b ≤ mbal(dblock s q)
      ∧ hasRead s q d p)
    ) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br))) (is ?P s)
  shows ?P s'
proof (auto)
fix \textit{d}
assume \textit{d} : \textit{d} \in D
with \textit{act} \textit{asm4}
show \textit{b} \leq \textit{bal} (disk \textit{s}' \textit{d} \textit{p})
by (auto simp add: \textit{Phase0Read-def})

next
fix \textit{d} \textit{q}
assume \textit{d} : \textit{d} \in D
and \textit{phase}' : \textit{phase} \textit{s}' \textit{q} = \textit{Suc 0}
and \textit{dblk-mbal} : \textit{b} \leq \textit{mbal} (\textit{dblock} \textit{s}' \textit{q})
and \textit{hasRead} : \textit{hasRead} \textit{s}' \textit{q} \textit{d} \textit{p}
from \textit{phase}' \textit{act}
have \textit{qqnq} : \textit{qq} \neq \textit{q}
by (auto simp add: \textit{Phase0Read-def})
show \exists \textit{br} \in \textit{blocksRead} \textit{s}' \textit{q} \textit{d} . \textit{b} \leq \textit{bal} (\textit{block} \textit{br})
proof 
from \textit{phase}' \textit{act} \textit{hasRead}
have \textit{p31} : \textit{phase} \textit{s} \textit{q} = 1
and \textit{p32} : \textit{dblock} \textit{s}' \textit{q} = \textit{dblock} \textit{s} \textit{q}
by (auto simp add: \textit{Phase0Read-def} \textit{hasRead-def})
with \textit{dblk-mbal}
have \textit{b} \leq \textit{mbal}(\textit{dblock} \textit{s} \textit{q}) by auto
moreover
from \textit{act} \textit{hasRead} \textit{qqnq}
have \textit{hasRead} \textit{s} \textit{q} \textit{d} \textit{p}
by (auto simp add: \textit{Phase0Read-def} \textit{hasRead-def}
split: if-split-asm)
ultimately
have \exists \textit{br} \in \textit{blocksRead} \textit{s} \textit{q} \textit{d} . \textit{b} \leq \textit{bal}(\textit{block} \textit{br})
using \textit{p31} \textit{asm4} \textit{d}
by blast
with \textit{act} \textit{hasRead}
show \exists \textit{br} \in \textit{blocksRead} \textit{s}' \textit{q} \textit{d} . \textit{b} \leq \textit{bal}(\textit{block} \textit{br})
by (auto simp add: \textit{Phase0Read-def} \textit{InitializePhase-def}
\textit{hasRead-def})
qed

theorem \textit{HPhase0Read-valueChosen}:
assumes \textit{act} : \textit{HPhase0Read} \textit{s} \textit{s}' \textit{q} \textit{d}
and \textit{vc} : \textit{valueChosen} \textit{s} \textit{v}
and \textit{v-input} : \textit{v} \in \textit{Inputs}
shows \textit{valueChosen} \textit{s}' \textit{v}
proof 
from \textit{vc}
obtain \textit{b} \textit{p} \textit{D} \textit{where}
\textit{asm1} : \textit{b} \in (\textit{UN} \textit{p}. \textit{Ballot} \textit{p})
and \textit{asm2} : \textit{D} \in \textit{MajoritySet}
and \textit{asm3} : \textit{maxBalnp} \textit{s} \textit{b} \textit{v}
and \( \text{asm}4 \): \( \forall d \in D. \ b \leq \text{bal}(\text{disk } s d p) \)
\( \land (\forall q. (\text{phase } s q = 1 \land b \leq \text{mbal}(\text{dblock } s q) \land \text{hasRead } s q d p) \rightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal}(\text{block } br))) \)
by (auto simp add: valueChosen-def)
from HPhase0Read-maxBalInp[OF act asm3]
have maxBalInp \( s' \) \( b \) \( v \).
with HPhase0Read-valueChosen2[OF act asm4] \( \text{asm1} \) \( \text{asm2} \)
show ?thesis
by (auto simp add: valueChosen-def)
qed

lemma HEndPhase0-maxBalInp:
assumes act: HEndPhase0 \( s \) \( s' \) \( q \)
and \( \text{asm3} \): maxBalInp \( s \) \( b \) \( v \)
and inv1: Inv1 \( s \)
shows maxBalInp \( s' \) \( b \) \( v \)
proof (auto simp add: maxBalInp-def)
fix \( bk \)
assume \( bk \): \( bk \in \text{allBlocks } s' \)
and \( b-\text{bal} \): \( b \leq \text{bal } bk \)
from subsetD[OF HEndPhase0-allBlocks[OF act] \( bk \)]
show \( \text{inp } bk = v \)
proof
assume \( bk \): \( bk \in \text{allBlocks } s \)
with \( \text{asm3} \) \( b-\text{bal} \)
show ?thesis
by (auto simp add: maxBalInp-def)
next
assume \( bk \): \( bk \in \{\text{dblock } s' q\} \)
with HEndPhase0-some[OF act inv1] act
have \( \exists ba \in \text{allBlocksRead } s q. \text{bal } ba = \text{bal } (\text{dblock } s' q) \land \text{inp } ba = \text{inp } (\text{dblock } s' q) \)
by (auto simp add: EndPhase0-def)
then obtain \( ba \)
where \( ba-\text{blksread} \): \( ba \in \text{allBlocksRead } s q \)
and \( ba-\text{balinp} \): \( \text{bal } ba = \text{bal } (\text{dblock } s' q) \land \text{inp } ba = \text{inp } (\text{dblock } s' q) \)
bysubstitution
have allBlocksRead \( s q \subseteq \text{allBlocks } s \)
by (auto simp add: allBlocksRead-def allRdBlks-def
allBlocks-def blocksOf-def rdBy-def)
from subsetD[OF this ba-\text{blksread} ba-\text{balinp} \( bk \) b-bal \( \text{asm3} \)]
show ?thesis
by (auto simp add: maxBalInp-def)
qed
qed
lemma HEndPhase0-valueChosen2:
assumes act: HEndPhase0 s s' q
and asm4: \( \forall d \in D. \ b \leq \text{bal}(\text{disk} \ s \ d \ p) \)
\( \land (\forall q. (\ \text{phase} \ s \ q = 1 \)
\land b \leq \text{mbal}(\text{dblock} \ s \ q) \)
\land \text{hasRead} \ s \ q \ d \ p \)
\( ) \rightarrow (\exists br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ br)) \) (is ?P s)
shows ?P s'
proof (auto)
fix d
assume d: d \in D
with act asm4
show b \leq \text{bal}(\text{disk} \ s' \ d \ p)
by (auto simp add: EndPhase0-def)
fix q
assume d: d \in D
and phase': phase s' q = Suc 0
and dblk-mbal: b \leq \text{mbal}(\text{dblock} \ s' \ q)
and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
and p32: dblock s' q = dblock s q
by (auto simp add: EndPhase0-def InitializePhase-def
hasRead-def split : if-split-asn)
with dblk-mbal
have b \leq \text{mbal}(\text{dblock} \ s \ q) \ by \ auto
moreover
from act hasRead
have hasRead s q d p
by (auto simp add: EndPhase0-def InitializePhase-def
hasRead-def split : if-split-asn)
ultimately
have \( \exists br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ br) \)
using p31 asm4 d
by blast
with act hasRead
show \( \exists br \in \text{blocksRead} \ s' q \ d. \ b \leq \text{bal}(\text{block} \ br) \)
by (auto simp add: EndPhase0-def InitializePhase-def
hasRead-def)
qed

theorem HEndPhase0-valueChosen:
assumes act: HEndPhase0 s s' q
and vc: valueChosen s v
and v-input: v \in \text{Inputs}
and inv1: Inv1 s
shows valueChosen s' v
proof –
from vc
obtain \( b \) \( p \) \( D \) where
- \( asm1: b \in (\text{UN} p. \text{Ballot} p) \)
- \( asm2: D \subseteq \text{MajoritySet} \)
- \( asm3: \maxBalInp s b v \)
- \( asm4: \forall d \in D. \; b \leq \text{bal}(\text{disk} s d p) \)
  \( \land (\forall q. (\text{phase} s q = 1 \land b \leq \text{mbal}(\text{dblock} s q) \land \text{hasRead} s q d p) \rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br))) \)

by (auto simp add: valueChosen-def)

from \( \text{HEndPhase0-maxBalInp}[OG \text{act} \text{asm3} \text{inv1}] \)

have \( \maxBalInp s' b v \).

with \( \text{HEndPhase0-valueChosen2}[OG \text{act} \text{asm4}] \) \( \text{asm1} \) \( \text{asm2} \)

show \( ?\text{thesis} \)

by (auto simp add: valueChosen-def)

qed

end

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

definition \( HInv6 :: \text{state} \Rightarrow \text{bool} \)

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

theorem \( HInit-HInv6: \; HInit s \Rightarrow HInv6 s \)

by (auto simp add: HInit-def Init-def InitDB-def HInv6-def)

lemma \( HEndPhase2-Inv6-1: \)

assumes act: \( \text{HEndPhase2} s s' p \)

and inv: \( HInv6 s \)

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

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

and inv3: \( HInv3 s \)

and inv5: \( HInv5-inner s p \)

and chosen': \text{chosen} s' \neq \text{NotAnInput}

shows \( \text{valueChosen} s' (\text{chosen} s') \)

proof (cases chosen \( s = \text{NotAnInput} \))

from \( \text{inv5 act} \)

have \( \text{inv5R}: \; HInv5-inner-R s p \)

and \( \text{phase}: \; \text{phase} s p = 2 \)

and \( \text{ep2-maj}: \; \text{IsMajority} \{d. \; d \in \text{disksWritten} s p \)
\(\forall q \in \text{UNIV} - \{p\}, \text{hasRead s p d q}\)

by (auto simp add: EndPhase2-def HInv5-inner-def)

case True

have p32: \(\text{maxBalInp s (bal(dblock s p)) (inp(dblock s p))}\)

proof

- have \(\neg (\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal (dblock s p)} < \text{mbal (disk s d q)} \land \neg \text{hasRead s p d q}))\)

proof auto

fix D q

assume Dmaj: \(D \in \text{MajoritySet}\)

from ep2-maj Dmaj majorities-intersect

have \(\exists d \in D. d \in \text{disksWritten s p}\)

\(\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q})\)

by (auto simp add: MajoritySet-def, blast)

then obtain d

where dinD: \(d \in D\)

and ddisk: \(d \in \text{disksWritten s p}\)

and dhasR: \(\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q}\)

by auto

from inv2b

have Inv2b-inner s p d

by (auto simp add: Inv2b-def)

with ddisk

have \(\text{disk s d p} = \text{dblock s p}\)

by (auto simp add: Inv2b-inner-def)

with inv2c phase

have \(\text{bal (dblock s p)} = \text{mbal (disk s d p)}\)

by (auto simp add: Inv2c-def Inv2c-inner-def)

with dhasR dinD

show \(\exists d \in D. \text{bal (dblock s p)} < \text{mbal (disk s d q)} \longrightarrow \text{hasRead s p d q}\)

by auto

qed

with inv5R

show \(?\text{thesis}\)

by (auto simp add: HInv5-inner-R-def)

qed

have p33: \(\text{maxBalInp s' (bal(dblock s' p)) (chosen s')}\)

proof

from act

have \(\text{outpt'}: \text{outpt s'} = (\text{outpt s}) (p:= \text{inp (dblock s p)})\)

by (auto simp add: EndPhase2-def)

have \(\text{outpt' q : } \forall q. p \neq q \longrightarrow \text{outpt s'} q = \text{NotAnInput}\)

proof auto

fix q

assume pnq: \(p \neq q\)

from outpt' pnq

have \(\text{outpt s'} q = \text{outpt s q}\)

by (auto simp add: EndPhase2-def)

with True inv2c
show \( \text{outpt } s' \ q = \text{NotAnInput} \)
  by (auto simp add: Inv2c-def Inv2c-inner-def)
qed

from True act chosen'
have chosen s' = inp (dblock s p)
proof (auto simp add: HNextPart-def split: if-split_asm)
  fix pa
  assume outpt'-pa: outpt s' pa \neq \text{NotAnInput}
  from outpt'-q
  have someeq2: \( \bigwedge_{p} \text{outpt } s' \ pa \neq \text{NotAnInput} \implies pa=p \)
    by auto
  with outpt'-pa
  have outpt s' p \neq \text{NotAnInput}
    by auto
  from some-equality[of \( \lambda p. \text{outpt } s' \ p \neq \text{NotAnInput} \), OF this someeq2]
  have (SOME p. outpt s' p \neq \text{NotAnInput}) = p
    with outpt'
  show outpt s' (SOME p. outpt s' p \neq \text{NotAnInput}) = inp (dblock s p)
    by auto
qed

moreover
from act
have bal(dblock s' p) = bal(dblock s p)
  by (auto simp add: EndPhase2-def)
ultimately
have maxBalInp s (bal(dblock s' p)) (chosen s')
  using p32
    by auto
  with HEndPhase2-allBlocks[of act]
  show \?thesis
    by (auto simp add: maxBalInp-def)
qed

from ep2-maj inv2b majorities-intersect
have \( \exists D \in \text{MajoritySet}. \ (\forall d \in D. \ \text{disk } s \ d \ p = \text{dblock } s \ p \)
  \wedge (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s \ p \ d \ q) \)
  by (auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def)
then obtain D
  where Dmaj: D \in MajoritySet
  and p34: \( \forall d \in D. \ \text{disk } s \ d \ p = \text{dblock } s \ p \)
  \wedge (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s \ p \ d \ q) \)
    by auto
have p35: \( \forall q. \ \forall d \in D. \ (\text{phase } s \ q = 1 \wedge \text{bal(dblock } s \ p) \leq \text{mbal(dblock } s \ q) \wedge \text{hasRead } s \ q \ d \ p) \)
  \implies (\{\text{block=}\text{dblock } s \ p, \text{proc=}p\} \in \text{blocksRead } s \ q \ d)
proof auto
  fix q d
  assume dD: d \in D and phase-q: phase s q = Suc 0
  and bal-mbal: bal(dblock s p) \leq \text{mbal(dblock } s \ q) and hasRead: hasRead s q d p
  from phase inv2c
have \( \text{bal}(\text{dblock } s \ p) = \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)

moreover
from inv2c
have \( \forall \text{br} \in \text{blocksRead } s \ p \ d \ . \ \text{mbal}(\text{block } \text{br}) < \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)

ultimately
have \( p41: \{ \text{block}=\text{dblock } s \ q, \ \text{proc}=p \} \notin \text{blocksRead } s \ p \ d \)
using bal-mbal
by auto

from phase phase-q
have \( p \neq q \) by auto
with \( p34 \ dD \)
have \( \text{hasRead } s \ p \ d q \)
by auto
with \( \text{phase } \text{phase-q} \ \text{hasRead } \text{inv3 } p41 \)
show \( \big( \exists \text{br} \in \text{blocksRead } s \ q \ d \ . \ \text{bal}(\text{block } \text{br}) = \text{bal}(\text{dblock } s \ p) \big) \)
proof (auto)
fix \( q \ d \)
assume \( \text{dD}: \ d \in D \ and \ \text{phase}-q: \ \text{phase } s' q = 0 \)
and \( \text{bal}: \ \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \)
and \( \text{hasRead}: \ \text{hasRead } s' q d p \)
from phase-q act
have \( \text{phase } s' q = \text{phase } s q \land \ \	ext{dblock } s' q = \text{dblock } s q \land \ \text{hasRead } s' q d p = \text{hasRead } s q d p \)
by (auto simp add: EndPhase2-def hasRead-def InitializePhase-def)
with \( p35 \ \text{phase-q } \text{bal} \ \text{hasRead } \text{dD} \)
have \( \{ \text{block}=\text{dblock } s \ p, \ \text{proc}=p \} \in \text{blocksRead } s' q d \)
by auto
thus \( \exists \text{br} \in \text{blocksRead } s' q d \ . \ \text{bal}(\text{block } \text{br}) = \text{bal}(\text{dblock } s \ p) \)
by force
qed

hence \( p36-2: \ \forall \ q, \ \forall \ d \in D. \ \text{phase } s' q = 1 \land \ \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \land \ \	ext{hasRead } s' q d p \)
by force
from act
have \( \text{bal-dblock}: \ \text{bal}(\text{dblock } s' p) = \text{bal}(\text{dblock } s \ p) \)
and \( \text{disk}: \ \text{disk } s' = \text{disk } s \)
by (auto simp add: EndPhase2-def)
from \( \text{bal-dblock } p33 \)
have \( \text{maxBalInp } s' \ (\text{bal}(\text{dblock } s \ p)) = \text{chosen } s' \)
by auto
moreover
from disk p34
have \( \forall d \in D. \) bal(dblock s p) \( \leq \) bal(disk s' d p) 
  by auto
ultimately
have maxBalInp s' (bal(dblock s p)) (chosen s') \( \land \)
  \((\exists D \in MajoritySet.
    \forall d \in D. \) bal(dblock s p) \( \leq \) bal(disk s' d p) \( \land \)
    (\forall q. \) phase s' q = Suc 0 \( \land \)
    bal(dblock s p) \( \leq \) mbal(dblock s' q) \( \land \) hasRead s' q d p \( \rightarrow \)
    (\exists br \in blocksRead s' q d. bal(dblock s p) \( \leq \) bal(block br))))
  using p36-2 Dmaj
  by auto
moreover
from phase inv2c
have bal(dblock s p) \( \in \) Ballot p
  by(auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
show \( ? \)thesis
  by(auto simp add: valueChosen-def)
next
case False
with act
have p31: chosen s' = chosen s
  by(auto simp add: HNextPart-def)
from False inv
have valueChosen s (chosen s)
  by(auto simp add: HInv6-def)
from HEndPhase2-valueChosen[of act this] p31 False InputsOrNi
show \( ? \)thesis
  by auto
qed

lemma valueChosen-equal-case:
  assumes max-v: maxBalInp s b v
  and Dmaj: D \( \in \) MajoritySet
  and asm-v: \( \forall d \in D. \) b \( \leq \) bal(disk s d p)
  and max-w: maxBalInp s ba w
  and Damaj: Da \( \in \) MajoritySet
  and asm-w: \( \forall d \in Da. \) ba \( \leq \) bal(disk s d pa)
  and b-ba: b \( \leq \) ba
  shows v=w
proof –
  have \( \forall d. \) disk s d pa \( \in \) allBlocks s
    by(auto simp add: allBlocks-def blocksOf-def)
  with majorities-intersect Dmaj Damaj
  have \( \exists d \in D \setminus Da. \) disk s d pa \( \in \) allBlocks s
    by(auto simp add: MajoritySet-def, blast)
  then obtain d
where \( \text{dinmaj: } d \in D \cap Da \) and \( \text{dab: disk s d pa } \in \text{allBlocks s} \) by auto

with \( \text{asm-w} \)

have \( \text{ba: } ba \leq \text{bal (disk s d pa)} \) by auto

with \( \text{b-ba} \)

have \( b \leq \text{bal (disk s d pa)} \) by auto

with \( \text{max-v dab} \)

have \( \text{v-value: } \text{inp (disk s d pa)} = v \) by (auto simp add: maxBalInp-def)

from \( \text{ba max-w dab} \)

have \( \text{w-value: } \text{inp (disk s d pa)} = w \) by (auto simp add: maxBalInp-def)

with \( \text{v-value} \)

show \( \text{?thesis by auto} \) qed

lemma \( \text{valueChosen-equal:} \)

assumes \( \text{v: valueChosen s v} \)

and \( \text{w: valueChosen s w} \)

shows \( v = w \) using assms

proof (auto simp add: valueChosen-def)

fix \( a \ b \ aa \ ba \ p \ D \ pa \ Da \)

assume \( \text{max-v: maxBalInp s b v} \)

and \( \text{Dmaj: } D \in \text{MajoritySet} \)

and \( \text{asm-v: } \forall d \in D. \ b \leq \text{bal (disk s d p)} \land \\
(\forall q. \ \text{phase s q = Suc 0 } \land \\
\ b \leq \text{mbal (dblock s q)} \land \text{hasRead s q d p } \rightarrow \\
(\exists br \in \text{blocksRead s q d}. \ b \leq \text{bal (block br)}))) \)

and \( \text{max-w: maxBalInp s ba w} \)

and \( \text{Damaj: Da } \in \text{MajoritySet} \)

and \( \text{asm-w: } \forall d \in Da. \ ba \leq \text{bal (disk s d pa)} \land \\
(\forall q. \ \text{phase s q = Suc 0 } \land \\
\ ba \leq \text{mbal (dblock s q)} \land \text{hasRead s q d pa } \rightarrow \\
(\exists br \in \text{blocksRead s q d}. \ ba \leq \text{bal (block br)}))) \)

from \( \text{asm-v} \)

have \( \text{asm-v: } \forall d \in D. \ b \leq \text{bal (disk s d p)} \) by auto

from \( \text{asm-w} \)

have \( \text{asm-w: } \forall d \in Da. \ ba \leq \text{bal (disk s d pa)} \) by auto

show \( v = w \)

proof (cases \( b \leq ba \))

case \( \text{True} \)

from \( \text{valueChosen-equal-case[OF max-v Dmaj asm-v max-w Damaj asm-w True]} \)

show \( \text{?thesis} \).

next

case \( \text{False} \)

from \( \text{valueChosen-equal-case[OF max-w Damaj asm-w max-v Dmaj asm-v]} \)

show \( \text{?thesis} \)

139
lemma HEndPhase2-Inv6-2:
  assumes act: HEndPhase2 s s' p
  and inv: HInv6 s
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  and inv3: HInv3 s
  and inv5: HInv5-inner s p
  and asm: outpt s' r ≠ NotAnInput
  shows outpt s' r = chosen s'
proof(cases chosen s = NotAnInput)
  case True
  with inv2c
  have ∀ q, outpt s q = NotAnInput
    by(auto simp add: Inv2c-def Inv2c-inner-def)
  with True act asm
  show ?thesis
    by(auto simp add: EndPhase2-def HNextPart-def
         split: if-split-asm)
next
  case False
  with inv
  have p31: valueChosen s (chosen s)
    by(auto simp add: HInv6-def)
  with False act
  have chosen s' ≠ NotAnInput
    by(auto simp add: HNextPart-def)
  from HEndPhase2-Inv6-1[OF act inv inv2b inv2c inv3 inv5 this]
  have p32: valueChosen s' (chosen s') .
  from False InputsOrNi
  have chosen s ∈ Inputs by auto
  from valueChosen-equal[OF HEndPhase2-valueChosen[OF act p31 this] p32]
  have p33: chosen s = chosen s' .
  from act
  have maj: IsMajority {d . d ∈ disksWritten s p
    ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)} (is IsMajority ?D)
    and phase: phase s p = 2
    by(auto simp add: EndPhase2-def)
  show ?thesis
proof(cases outpt s r = NotAnInput)
  case True
  with asm act
  have p41: r = p
    by(auto simp add: EndPhase2-def split: if-split-asm)
  from maj
  have p42: ∃ D ∈ MajoritySet. ∀ d ∈ D. ∀ q ∈ UNIV − {p}. hasRead s p d q
  by(auto simp add: Majority-def)
by(auto simp add: MajoritySet-def)

have p43: 
  \neg(\exists D \in MajoritySet. \exists q. (\forall d \in D. \text{bal}(\text{dblock s} p) < \text{mbal}(\text{disk s} d q) \\
  \land \neg \text{hasRead s} p d q))
  
  proof auto
  fix D q
  assume Dmaj: D \in MajoritySet
  show \exists d \in D. \text{bal}(\text{dblock s} p) < \text{mbal}(\text{disk s} d q) \longrightarrow \text{hasRead s} p d q
    proof (cases p=q)
      assume pq: p=q
      thus ?thesis
      proof auto
        from maj majorities-intersect Dmaj
        have ?D \cap D \neq \{}
          by(auto simp add: MajoritySet-def)
        hence \exists d \in ?D \cap D. d \in \text{disksWritten s} p \ by auto
        then obtain d where d: d \in \text{disksWritten s} p and d \in ?D \cap D
          by auto
        hence dD: d \in D by auto
        from d inv2b
        have \text{disk s} d p = \text{dblock s} p
          by(auto simp add: Inv2b-def Inv2b-inner-def)
        with inv2c phase
        have \text{bal}(\text{dblock s} p) = \text{mbal}(\text{disk s} d p)
          by(auto simp add: Inv2c-def Inv2c-inner-def)
        with dD pq
        show \exists d \in D. \text{bal}(\text{dblock s} q) < \text{mbal}(\text{disk s} d q) \longrightarrow \text{hasRead s} q d q
          by auto
        qed
    qed
    next
    case False
    with p42
    have \exists D \in MajoritySet. \forall d \in D. \text{hasRead s} p d q
      by auto
    with majorities-intersect Dmaj
    show ?thesis
      by(auto simp add: MajoritySet-def, blast)
    qed
  qed
  with inv5 act
  have p44: \text{maxBalInp s} (\text{bal}(\text{dblock s} p)) (\text{inp}(\text{dblock s} p))
    by(auto simp add: EndPhase2-def HInv5-inner-def HInv5-inner-R-def)
  have \exists bk \in \text{allBlocks s}. \exists b \in (\text{UN} p. \text{Ballot p}). (\text{maxBalInp s} b (\text{chosen s})) \land b \leq \text{bal} bk
    
    proof
      have \text{disk-allblk}: \forall d p. \text{disk s} d p \in \text{allBlocks s}
        by(auto simp add: allBlocks-def blocksOf-def)
    from p31
have \( \exists b \in \text{(UN p. Ballot p)}. \maxBalInp s b \ (\text{chosen s}) \land \) 
(\( \exists p. \exists D \in \text{MajoritySet}. (\forall d \in D. \ b \leq \text{bal(disk s d p)}) \)) 
by(auto simp add: valueChosen-def, force) 
with majority-nonempty obtain b p D d 
where IsMajority D \land b \in \text{(UN p. Ballot p)} \land 
\maxBalInp s b \ (\text{chosen s}) \land d \in D \land b \leq \text{bal(disk s d p)} 
by(auto simp add: MajoritySet-def, blast) 
with disk-allblks 
show \(?thesis\) 
by(auto) 
qed 

then obtain bk b 
where p45-bk: bk \in \text{allBlocks s} \land b \leq \text{bal bk} 
and p45-b: b \in \text{(UN p. Ballot p)} \land (\maxBalInp s b \ (\text{chosen s})) 
by auto 
have p46: \text{inp(dblock s p)} = \text{chosen s} 
proof(cases b \leq \text{bal(dblock s p)}) 
  case True 
  have dblock s p \in \text{allBlocks s} 
  by(auto simp add: allBlocks-def blocksOf-def) 
  with p45-b True 
  show \(?thesis\) 
  by(auto simp add: maxBalInp-def) 
next 
  case False 
  from p44 p45-bk False 
  have \text{inp bk} = \text{inp(dblock s p)} 
  by(auto simp add: maxBalInp-def) 
  with p45-b p45-bk 
  show \(?thesis\) 
  by(auto simp add: maxBalInp-def) 
qed 

with p41 p33 act 
show \(?thesis\) 
by(auto simp add: EndPhase2-def) 
next 
  case False 
  from inv2c 
  have inv2c-inner s r 
  by(auto simp add: inv2c-def) 
  with False asm inv2c act 
  have \text{outpt s' r} = \text{outpt s r} 
  by(auto simp add: inv2c-inner-def EndPhase2-def split: if-split-asm) 
  with inv p33 False 
  show \(?thesis\) 
  by(auto simp add: HInv6-def) 
qed 

qed 

142
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(auto simp add: HInv6-def)
assume chosen s' ≠ NotAnInput
from HEndPhase2-Inv6-1[OF act inv inv2b inv2c inv3 inv5 this]
show valueChosen s' (chosen s') .
next
fix p
assume outpt s' p≠ NotAnInput
from HEndPhase2-Inv6-2[OF act inv inv2b inv2c inv3 inv5 this]
show outpt s' p = chosen s' .
qed

lemma outpt-chosen:
  assumes outpt: outpt s = outpt s'
  and inv2c: Inv2c s
  and nextp: HNextPart s s'
  shows chosen s' = chosen s
proof −
from inv2c
have chosen s = NotAnInput → (∀ p. outpt s p = NotAnInput)
  by(auto simp add: Inv2c-inner-def Inv2c-def)
with outpt nextp
show ?thesis
  by(auto simp add: HNextPart-def)
qed

lemma outpt-Inv6:
[ outpt s = outpt s'; ∀ p. outpt s p ∈ {chosen s, NotAnInput};
  Inv2c s; HNextPart s s' ] → ∀ p. outpt s' p ∈ {chosen s', NotAnInput}
using outpt-chosen
by auto

theorem HStartBallot-Inv6:
  assumes act: HStartBallot s s' p
  and inv: HInv6 s
  and inv2c: Inv2c s
  shows HInv6 s'
proof −
from outpt-chosen act inv2c inv
have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
  by(auto simp add: StartBallot-def HInv6-def)
from HStartBallot-valueChosen[OF act] this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
  by auto
from act have outpt: outpt s = outpt s'
  by(auto simp add: StartBallot-def)
from outpt-Inv6[OF outpt] act inv2c inv
have ∀ p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
  by(auto simp add: HInv6-def)
with t1 show ?thesis
  by(simp add: HInv6-def)
qed

theorem HPhase1or2Write-Inv6:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv6 s
  and inv4: HInv4a s p
  and inv2c: Inv2c s
  shows HInv6 s'
proof −
  from outpt-chosen act inv2c inv
  have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
    by(auto simp add: Phase1or2Write-def HInv6-def)
  from HPhase1or2Write-valueChosen[OF act] this InputsOrNi
  have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
    by auto
  from act have outpt: outpt s = outpt s'
    by(auto simp add: Phase1or2Write-def)
  from outpt-Inv6[OF outpt] act inv2c inv
  have ∀ p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
    by(auto simp add: HInv6-def)
  with t1 show ?thesis
    by(simp add: HInv6-def)
qed

theorem HPhase1or2ReadThen-Inv6:
  assumes act: HPhase1or2ReadThen s s' p d q
  and inv: HInv6 s
  and inv2c: Inv2c s
  shows HInv6 s'
proof −
  from outpt-chosen act inv2c inv
  have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
    by(auto simp add: Phase1or2ReadThen-def HInv6-def)
  from HPhase1or2ReadThen-valueChosen[OF act] this InputsOrNi
  have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
by auto
from act
have outpt: outpt s = outpt s'
  by (auto simp add: Phase1or2ReadThen-def)
from outpt-Inv6[of outpt] act inv2c inv
have \( \forall p. \text{outpt } s' p = \text{chosen } s' \lor \text{outpt } s' p = \text{NotAnInput} \)
  by (auto simp add: HInv6-def)
with t1
show ?thesis
  by (simp add: HInv6-def)
qed

theorem HPhase1or2ReadElse-Inv6:
  assumes act: HPhase1or2ReadElse s s' p d q
  and inv: HInv6 s
  and inv2c: Inv2c s
  shows HInv6 s'
  using assms and HStartBallot-Inv6
  by (auto simp add: Phase1or2ReadElse-def)

theorem HEndPhase1-Inv6:
  assumes act: HEndPhase1 s s' p
  and inv: HInv6 s
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  shows HInv6 s'
proof
  from outpt-chosen act inv2c inv
  have chosen s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s'(\text{chosen } s')
    by (auto simp add: EndPhase1-def HInv6-def)
  from HEndPhase1-valueChosen[of act] inv1 inv2a inv2b this InputsOrNi
  have t1: chosen s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s'(\text{chosen } s')
    by auto
  from act
  have outpt: outpt s = outpt s'
    by (auto simp add: EndPhase1-def)
  from outpt-Inv6[of outpt] act inv2c inv
  have \( \forall p. \text{outpt } s' p = \text{chosen } s' \lor \text{outpt } s' p = \text{NotAnInput} \)
    by (auto simp add: HInv6-def)
  with t1
  show ?thesis
    by (simp add: HInv6-def)
qed

lemma outpt-chosen-2:
  assumes outpt: outpt s' = (outpt s) (p:= \text{NotAnInput})
  and inv2c: Inv2c s
and nextp: HNextPart s s' 
shows chosen s = chosen s' 
proof – 
from inv2c 
have chosen s = NotAnInput \rightarrow (\forall p. outpt s p = NotAnInput) 
by(auto simp add: Inv2c-inner-def Inv2c-def) 
with outpt nextp 
show ?thesis 
by(auto simp add: HNextPart-def) 
qed 

lemma outpt-HInv6-2: 
assumes outpt: outpt s' = (outpt s) (p:= NotAnInput) 
and inv: \forall p. outpt s p \in \{chosen s, NotAnInput\} 
and inv2c: Inv2c s 
and nextp: HNextPart s s' 
shows \forall p. outpt s' p \in \{chosen s', NotAnInput\} 
proof – 
from outpt-chosen-2[OF outpt inv2c nextp] 
have chosen s = chosen s' . 
with inv outpt 
show ?thesis 
by auto 
qed 

theorem HFail-Inv6: 
assumes act: HFail s s' p 
and inv: HInv6 s 
and inv2c: Inv2c s 
shows HInv6 s' 
proof – 
from outpt-chosen-2 act inv2c inv 
have chosen s' \neq NotAnInput \rightarrow valueChosen s (chosen s') 
by(auto simp add: Fail-def HInv6-def) 
from HFail-valueChosen[OF act] this InputsOrNi 
have t1: chosen s' \neq NotAnInput \rightarrow valueChosen s' (chosen s') 
by auto 
from act 
have outpt: outpt s' = (outpt s) (p:=NotAnInput) 
by(auto simp add: Fail-def) 
from outpt-HInv6-2[OF outpt] act inv2c inv 
have \forall p. outpt s' p = chosen s' \vee outpt s' p = NotAnInput 
by(auto simp add: HInv6-def) 
with t1 
show ?thesis 
by(simp add: HInv6-def) 
qed 

theorem HPhase0Read-Inv6: 

assumes act: HPhase0Read s s' p d
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'

proof –
from outpt-chosen act inv2c inv
have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
  by(auto simp add: Phase0Read-def HInv6-def)
from HPhase0Read-valueChosen[OF act] this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
  by auto
from act
have outpt: outpt s = outpt s'
  by(auto simp add: Phase0Read-def)
from outpt-Inv6[OF outpt] act inv2c inv
have ∀ p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
  by(auto simp add: HInv6-def)
with t1
show ?thesis
  by(simp add: HInv6-def)
qed

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

proof –
from outpt-chosen act inv2c inv
have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
  by(auto simp add: EndPhase0-def HInv6-def)
from HEndPhase0-valueChosen[OF act] inv1 this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
  by auto
from act
have outpt: outpt s = outpt s'
  by(auto simp add: EndPhase0-def)
from outpt-Inv6[OF outpt] act inv2c inv
have ∀ p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
  by(auto simp add: HInv6-def)
with t1
show ?thesis
  by(simp add: HInv6-def)
qed

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

lemma I2f:
assumes \( \text{nxt} : H\text{Next} \ s \ s' \)
and \( \text{inv} : H\text{Inv} \ s \land H\text{Inv} \ s' \land H\text{Inv} \ s' \land H\text{inv} \ s \land H\text{inv} \ s' \land H\text{inv} \ s \land H\text{inv} \ s' \land H\text{inv} \ s' \land H\text{inv} \ s \land H\text{Inv} \ s' \land H\text{Inv} \ s' \land H\text{Inv} \ s' \)
shows \( H\text{Inv} \ s' \) using asms
by (auto simp add: HNext-def Next-def,
    auto simp add: HInv2-def intro: HStartBallot-Inv6,
    auto intro: HPhase0Read-Inv6,
    auto simp add: HInv4-def intro: HPhase1or2Write-Inv6,
    auto simp add: Phase1or2Read-def
      intro: HPhase1or2ReadThen-Inv6
      HPhase1or2ReadElse-Inv6,
    auto simp add: EndPhase1or2-def HInv1-def HInv5-def
      intro: HEndPhase1-Inv6
      HEndPhase2-Inv6,
    auto intro: HFail-Inv6,
    auto intro: HEndPhase0-Inv6)

end

theory DiskPaxos-Invariant imports DiskPaxos-Inv6 begin

C.8 The Complete Invariant

definition \( H\text{inv} :: \text{state} \Rightarrow \text{bool} \)
where
\[
H\text{inv} \ s = (H\text{Inv} \ s \\
\land H\text{Inv} \ s') \\
\land H\text{Inv} \ s' \\
\land H\text{Inv} \ s' \\
\land H\text{Inv} \ s' \\
\land H\text{Inv} \ s')
\]

theorem I1:
\( H\text{Init} \ s \implies H\text{inv} \ s \)
using HInit-HInv1 HInit-HInv2 HInit-HInv3 HInit-HInv4 HInit-HInv5 HInit-HInv6
by (auto simp add: HInv-def)

theorem I2:
assumes \( \text{inv} : H\text{inv} \ s \)
and \( \text{nxt} : H\text{Next} \ s \ s' \)
shows \( H\text{inv} \ s' \)
using \( \text{inv} \ I2a[\text{OF} \ \text{nxt}] \ I2b[\text{OF} \ \text{nxt}] \ I2c[\text{OF} \ \text{nxt}] \ I2d[\text{OF} \ \text{nxt}] \ I2e[\text{OF} \ \text{nxt}] \)
\( I2f[\text{OF} \ \text{nxt}] \)
by (simp add: HInv-def)

end
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 (∃ip ∈ iallInput s. ichosen s' = ichosen s)
      ∧ iallInput s' = iallInput s ∪ {ip}))
  ∧ iinput s' = iinput s ∧ iallInput s' = iallInput s)

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

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

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

theorem R1:
  [ IInit s; is = s2is s ] ⇒ IInit is
by(auto simp add: HInit-def IInit-def s2is-def Init-def)

theorem R2b:
  assumes inv: HInv s
  and inv': HInv s'
  and nxt: HNext s s'
  and srel: is=s2is s ∧ is'=s2is s'
  shows (∃ p. IFail is is' p ∨ IChoose is is' p) ∨ is = is'
proof(auto)
  assume chg-vars: is≠is'
  with srel
  have s-change: inpt s ≠ inpt s' ∨ outpt s ≠ outpt s'
    ∨ chosen s ≠ chosen s' ∨ allInput s ≠ allInput s'
    by(auto simp add: s2is-def)
  from inv
  have inv2c5: ∀ p. inpt s p ∈ allInput s
    ∧ (chosen s = NotAnInput → outpt s p = NotAnInput)
    by(auto simp add: HInv-def HInv2-def Inv2c-def Inv2c-inner-def)
  from nxt s-change inv2c5
  have inpt s' ≠ inpt s ∨ outpt s' ≠ outpt s
    by(auto simp add: HNext-def Next-def HNextPart-def)
  with nxt
  have ∃ p. Fail s s' p ∨ EndPhase2 s s' p
    by(auto simp add: HNext-def Next-def
      StartBallot-def Phase0Read-def Phase1or2Write-def
      Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def
      EndPhase1or2-def EndPhase1-def EndPhase0-def)
  then obtain p where fail-or-endphase2: Fail s s' p ∨ EndPhase2 s s' p
    by(auto)
  from inv
  have inv2c: Inv2c-inner s p
    by(auto simp add: HInv-def HInv2-def Inv2c-def)
  from fail-or-endphase2 have IFail is is' p ∨ IChoose is is' p
  proof
    assume fail: Fail s s' p
    hence phase': phase s' p = 0
      and outpt: outpt s' = (outpt s) (p:= NotAnInput)
      by(auto simp add: Fail-def)
    have IFail is is' p
    proof
      from fail srel
      have ioutput is' = (ioutput is) (p:= NotAnInput)
        by(auto simp add: Fail-def s2is-def)
      moreover
      from nxt
      have all-nxt: allInput s' = allInput s ∪ (range (inpt s'))
        by(auto simp add: HNext-def HNextPart-def)
      from fail srel
      have ∃ ip ∈ Inputs. iinput is' = (iinput is)(p:= ip)
by (auto simp add: Fail-def s2is-def)
then obtain \( ip \) where \( ip\text{-Input}: ip \in \text{Inputs} \) and \( ii\text{input is}' = (ii\text{input is})(p:=ip) \)

by auto

with inv2c5 srel all-nxt
have \( ii\text{input is}' = (ii\text{input is})(p:=ip) \)
\( \land \ all\text{Input is}' = all\text{Input is} \cup \{ip\} \)
by (auto simp add: s2is-def)

moreover
from outpt srel nxt inv2c
have \( i\text{chosen is}' = i\text{chosen is} \)
by (auto simp add: HNext-def HNextPart-def s2is-def Inv2c-inner-def)

ultimately
show ?thesis
using \( ip\text{-Input} \)
by (auto simp add: IFail-def)

qed
thus ?thesis
by auto

next
assume endphase2; \( \text{EndPhase2} s \ s' \ p \)
from endphase2
have \( \text{phase} s \ p = 2 \)
by (auto simp add: EndPhase2-def)

with inv2c \( \text{Ballot-}n\text{zero} \)
have \( \text{bal-}d\text{blk-}n\text{zero}: \text{bal}(d\text{block} s \ p) \neq 0 \)
by (auto simp add: Inv2c-inner-def)

moreover
from inv
have inv2a-dblock: \( \text{Inv2a-innermost} s \ p \ (d\text{block} s \ p) \)
by (auto simp add: HInv-def HInv2-def Inv2a-def Inv2a-inner-def blocksOf-def)

ultimately
have p22: \( \text{inp} (d\text{block} s \ p) \in \text{allInput} s \)
by (auto simp add: Inv2a-innermost-def)

from inv
have allInput s \( \subseteq \) Inputs
by (auto simp add: HInv-def HInv1-def)

with p22 \( \text{NotAnInput} \) endphase2
have outpt-nni: outpt s' \( \neq \) NotAnInput
by (auto simp add: EndPhase2-def)

show ?thesis

proof (cases chosen \( s = \) NotAnInput)

  case True
  with inv2c5
  have p31: \( \forall q. \text{outpt} s q = \text{NotAnInput} \)
  by auto

  with endphase2
  have p32: \( \forall q \in \text{UNIV} - \{p\}. \text{outpt} s' q = \text{NotAnInput} \)
  by (auto simp add: EndPhase2-def)
**hence** some-eq: (\(\forall x. \text{outpt } s' x \neq \text{NotAnInput} \implies x = p\))
by auto

from p32 True nxt some-equality[of \(\lambda p. \text{outpt } s' p \neq \text{NotAnInput}, \text{OF } \text{outpt-nni}\)]

have p33: chosen \(s' = \text{outpt } s' p\)
by(auto simp add: HNext-def HNextPart-def)

with endphase2

have chosen \(s' = \text{inp}(\text{dblock } s p) \land \text{outpt } s' = (\text{outpt } s)(p:=\text{inp}(\text{dblock } s p))\)
by(auto simp add: EndPhase2-def)

with True p22

have if (chosen \(s = \text{NotAnInput}\))
then (\(\exists ip \in \text{allInput } s. \text{chosen } s' = ip\)
\land \text{outpt } s' = (\text{outpt } s)(p := ip))
else (\(\text{outpt } s' = (\text{outpt } s)(p := \text{chosen } s)\)
\land \text{chosen } s' = \text{chosen } s)

by auto

moreover
from endphase2 inv2c5 nxt

have inpt \(s' = \text{inpt } s \land \text{allInput } s' = \text{allInput } s\)
by(auto simp add: EndPhase2-def HNext-def HNextPart-def)

ultimately

show ?thesis
using srel p31
by(auto simp add: IChoose-def s2is-def)

next

case False
with nxt

have p31: chosen \(s' = \text{chosen } s\)
by(auto simp add: HNext-def HNextPart-def)

from inv'

have inv6: HInv6 \(s'\)
by(auto simp add: HInv-def)

have p32: outpt \(s' p = \text{chosen } s\)

proof --
from endphase2

have outpt \(s' p = \text{inp}(\text{dblock } s p)\)
by(auto simp add: EndPhase2-def)

moreover
from inv6 p31

have outpt \(s' p \in \{\text{chosen } s, \text{NotAnInput}\}\)
by(auto simp add: HInv6-def)

ultimately

show ?thesis
using outpt-nni
by auto

qed
from srel False

have IChoose is is' \(p\)

proof(clarsimp simp add: IChoose-def s2is-def)
from endphase2 inv2c
have outpt s p = NotAnInput
  by (auto simp add: EndPhase2-def Inv2c-inner-def)
moreover
from endphase2 p31 p32 False
have outpt s' = (outpt s) (p := chosen s) ∧ chosen s' = chosen s
  by (auto simp add: EndPhase2-def)
moreover
from endphase2 nxt inv2c5
have inpt s' = inpt s ∧ allInput s' = allInput s
  by (auto simp add: EndPhase2-def HNext-def HNextPart-def)
ultimately
  show outpt s p = NotAnInput
    ∧ outpt s' = (outpt s) (p := chosen s) ∧ chosen s' = chosen s
    ∧ inpt s' = inpt s ∧ allInput s' = allInput s
  by auto
qed
thus ?thesis
  by auto
qed
thus ∃p. IFail is is' p ∨ IChoose is is' p
  by auto
qed
end