Abstract

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

Contents

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

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

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

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

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

2 The Disk Paxos Algorithm

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

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

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

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

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

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

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

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

In each block, processors maintain three values:

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

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

2.2 Disk Paxos and its TLA+ Specification

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

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

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

$HDiskSynodSpec \triangleq HInit \land \Box\lbrack HNext \rbrack_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle}$

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

Analogously, we have the specification of the Inner module:

$ISpec \triangleq IInit \land \Box\lbrack INext \rbrack_{\langle \text{input}, \text{output}, \text{chosen}, \text{allInput} \rangle}$

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

THEOREM $R1$ $HInit \Rightarrow IInit$

THEOREM $R2$ $HInit \land \Box\lbrack HNext \rbrack_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \Box\lbrack INext \rbrack_{\text{ivars}}$

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

THEOREM $R2a$ $HInit \land \Box\lbrack HNext \rbrack_{\langle \text{vars}, \text{chosen}, \text{allInput} \rangle} \Rightarrow \Box HInv$

THEOREM $R2b$ $HInv \land HInv' \land HNext \Rightarrow INext \lor \lbrack \text{UNCHANGED ivars} \rbrack$

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

Isabelle/HOL

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

\( \text{choose } x. P\ x \)

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

\( \varepsilon x. P\ x \)

phase' = [phase EXCEPT ![p] = 1]

phase' = (phase s)(p := 1)

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

UN \ p. blocksOf \ s\ p

UNCHANGED \ v

\( v\ s' = v\ s \)

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

**THEOREM I1**  \( H\text{Init} \Rightarrow H\text{Inv} \)

**THEOREM I2**  \( H\text{Inv} \land H\text{Next} \Rightarrow H\text{Inv}' \)

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

Finding a predicate \( H\text{Inv} \) that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present \( H\text{Inv} \) as a conjunction of 6 predicates \( H\text{Inv_1}, \ldots, H\text{Inv_6} \), where \( H\text{Inv_1} \) is a simple “type invariant” and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of \( H\text{Inv_j} \) by the algorithm’s next-state relation relies on all \( H\text{Inv_j} \) (for \( j \leq i \)) being true in the state before the transition. In our proofs we are going to use exactly the same tactic.

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

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

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

#### 3.1 Typed vs. Untyped

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

---

\(^1\)There is no point in using the existing TLA encoding in Isabelle. Since the encoding is also based on HOL and we only prove safety, we would have gained nothing.
This process is not automatic and requires some thought to find the right abstractions. Furthermore, simple types may not be expressive enough to represent exactly the set in the specification. In some cases, these sets could be modelled by algebraic datatypes, but this would make the specification more complex and less directly related to the original one. In this work, we have chosen to stick to simple types and add additional axioms to account for their lack of expressiveness.

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

This example shows the kind of difficulties that can arise when trans-
TLA+:
\[
\text{Phase1or2Write}(p, d) \triangleq \\
\quad \land \ phase[p] \in \{1, 2\} \\
\quad \land \ disk'[disk \setminus ![d][p] = \text{dblock}[p]] \\
\quad \land \ disksWritten'[disksWritten \setminus ![p] = @ \cup \{d\}] \\
\quad \land \ \text{UNCHANGED}(\text{input}, \text{output}, \text{phase}, \text{dblock}, \text{blocksRead})
\]

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

Figure 3: Translation of an action

3.2 Primed Variables

In TLA+, to denote the value of a variable in the state resulting from an action, there exists the concept of primed variables. In Isabelle/HOL, there is no built-in notion of state, so we define the state as a record of the variables involved. Instead of defining a “priming” operator, we just define actions as predicates that take any two states, intended to be the previous state and the next state. In this way, \( P s s' \) will be true iff executing an action \( P \) in the \( s \) state could result in the \( s' \) state. In figure 3 we can see how the action \textit{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 \textit{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 \textit{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, 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 Phase1or2ReadThen the condition of the if-then-else is present as a state formula (i.e. it is an enabling condition) while in Phase1or2ReadElse we add the negation of this condition.

Another example is 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 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 Inv2a, but only for some process q and block (dblock s q).

4 Structure of the Correctness Proof

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

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

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv_4$ and $HInv_5$ hold in the previous state to prove lemma $I2f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate $I$ was an invariant of $Next$, we preferred proving the invariance of $I$ for each action, rather than a big theorem proving the invariance of $I$ for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $HInv_3$ for the $EndPhase0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

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

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

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

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

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

References


A TLA\(^+\) correctness specification

---

**MODULE Synod**

EXTENDS Naturals

CONSTANT N, Inputs

ASSUME \((N \in \text{Nat}) \land (N > 0)\)

\(Proc \triangleq 1..N\)

\(NotAnInput \triangleq \text{CHOOSE } c : c \notin Inputs\)

VARIABLES inputs, output

---

**MODULE Inner**

VARIABLES allInput, chosen

---

\(IInit \triangleq \land \text{input} \in [Proc \rightarrow Inputs]\)

\(\land \text{output} = [p \in Proc \mapsto NotAnInput]\)

\(\land \text{chosen} = NotAnInput\)

\(\land \text{allInput} = \text{input}[p] : p \in Proc\)

\(IChoose(p) \triangleq\)

\(\land \text{output}[p] = NotAnInput\)

\(\land \text{IF } chosen = NotAnInput\)

\(\text{THEN } \text{ip} \in \text{allInput} : \land \text{chosen'} = \text{ip}\)

\(\land \text{output'} = [\text{output except } ![p] = \text{ip}]\)

\(\text{ELSE } \land \text{output'} = [\text{output except } ![p] = \text{chosen}]\)

\(\land \text{UNCHANGED chosen}\)

\(\land \text{UNCHANGED } \langle \text{input, allInput} \rangle\)

\(IFail(p) \triangleq\)

\(\land \text{output'} = [\text{output except } ![p] = NotAnInput]\)

\(\land \exists \text{ip} \in Inputs : \land \text{input'} = [\text{input except } ![p] = \text{ip}]\)

\(\land \text{allInput'} = \text{allInput} \cup \{\text{ip}\}\)

\(INext \triangleq \exists p \in Proc : IChoose(p) \lor IFail(p)\)

\(ISpec \triangleq IInit \land \Box [INext](input, output, chosen, allInput)\)

\(IS(\text{chosen, allInput}) \triangleq \text{INSTANCE Inner}\)

\(SynodSpec \triangleq \exists \text{chosen, allInput} : IS(\text{chosen, allInput})!ISpec\)
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 contr)
  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 =
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 =
\((\text{disksWritten } s') = (\text{disksWritten } s)(p := \{\})\) & \((\text{blocksRead } s') = (\text{blocksRead } s)(p := (\lambda d. \{\}))\)

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

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

definition \text{Phase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} 
where
\text{Phase1or2ReadThen} s s' p d q = 
(\text{phase } s \in \{1, 2\}) & \text{StartBallot } s s' p 
\& \text{blocksSeen } s' = \text{blocksSeen } s\)

definition \text{Phase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} 
where
\text{Phase1or2ReadElse} s s' p d q = 
(\text{phase } s \in \{1, 2\}) & \text{StartBallot } s s' p 
\& \text{blocksSeen } s' = \text{blocksSeen } s\)

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

definition blocksSeen :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock set}
where \( \text{blocksSeen } s \ p = \text{allBlocksRead } s \ p \cup \{ \text{dblock} \ s \ p \} \)

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

definition maxBlk :: state \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock}
where 
maxBlk \ s \ p = 
(SOME \ b. \ b \in \text{nonInitBlks } s \ p \land (\forall c \in \text{nonInitBlks } s \ p. \ \text{bal } c \leq \text{bal } b))

definition EndPhase1 :: state \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
where
EndPhase1 \ s \ s' \ p = 
(IsMajority \{ d \ . \ d \in \text{disksWritten } s \ p \\
\land (\forall q \in \text{UNIV} \setminus \{ 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) \\
\land \text{inp } := \\
(if \ \text{nonInitBlks } s \ p = \{} \\
\text{then inp } s \ p \\
\text{else inp } (\text{maxBlk } s \ p)) \\
\land \text{outpt } s' = \text{outpt } s \\
\land \text{phase } s' = (\text{phase } s) \ (p := \text{phase } s \ p + 1) \\
\land \text{InitializePhase } s \ s' \ p \\
\land \text{inp } s' = \text{inp } s \land \text{disk } s' = \text{disk } s)

definition EndPhase2 :: state \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
where
EndPhase2 \ s \ s' \ p = 
(IsMajority \{ d \ . \ d \in \text{disksWritten } s \ p \\
\land (\forall q \in \text{UNIV} \setminus \{ 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{inp } s' = \text{inp } s \land \text{disk } s' = \text{disk } s)

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

definition Fail :: state \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
where
Fail \ s \ s' \ p = 
(\exists \text{ip } \in \text{Inputs}. \ \text{inp } s' = (\text{inp } s) \ (p := \text{ip}) \\
\land \text{phase } s' = (\text{phase } s) \ (p := 0) \\
\land \text{dblock } s' = (\text{dblock } s) \ (p := \text{InitDB})
\[ \land \text{outpt } s' = (\text{outpt } s) (p := \text{NotAnInput}) \]
\[ \land \text{InitializePhase } s \ s' \ p \]
\[ \land \text{disk } s' = \text{disk } s \]

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

**where**

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

**definition** EndPhase0 :: state ⇒ state ⇒ Proc ⇒ bool

**where**

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

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

**where**

\[ \text{Next } s \ s' = (\exists p. \]
\[ \text{StartBallot } s \ s' \ p \]
\[ \lor (\exists d. \ \text{Phase0Read } s \ s' \ p \ d) \]
\[ \lor \text{Phase1or2Write } s \ s' \ p \ d \]
\[ \lor (\exists q. \ q \neq p \land \text{Phase1or2Read } s \ s' \ p \ d \ q) \]
\[ \lor \text{EndPhase1or2 } s \ s' \ p \]
\[ \lor \text{Fail } s \ s' \ p \]
\[ \lor \text{EndPhase0 } s \ s' \ p \]

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**

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

**Definition**

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

where

\[
\text{HNextPart} \ s \ s' =
\]

\[
(\text{chosen} \ s \neq \text{NotAnInput} \lor (\forall \ p. \ \text{outpt} \ s' \ p = \text{NotAnInput}))
\]

\[
\text{then chosen} \ s \quad \text{else } \text{outpt} \ s'
\]

\[
(\text{SOME} \ p. \ \text{outpt} \ s' \ p \neq \text{NotAnInput})
\]

\[
\land \text{allInput} \ s' = \text{allInput} \ s \cup (\text{range} \ (\text{inpt} \ s'))
\]

**Definition**

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

where

\[
\text{HNext} \ s \ s' =
\]

\[
(\text{Next} \ s \ s' \land \text{HNextPart} \ s \ s')
\]

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

**Definition**

\[
\text{HPhase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

where

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

**Definition**

\[
\text{HEndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

where

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

**Definition**

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

where

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

**Definition**

\[
\text{HPhase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

where

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

**Definition**

\[
\text{HPhase1or2ReadElse} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

where

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

**Definition**

\[
\text{HEndPhase2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}
\]

where

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

**Definition**

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

where

\[
\text{HFail} \ s \ s' = (\text{Fail} \ s \ s' \ p \land \text{HNextPart} \ s \ s')
\]
definition
HPhase0Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool where
HPhase0Read s s' p d = (Phase0Read s s' p d ∧ HNextPart s s')

definition
HEndPhase0 :: state ⇒ state ⇒ Proc ⇒ bool where
HEndPhase0 s s' p = (EndPhase0 s s' p ∧ HNextPart s s')

Since these definitions are the conjunction of two other definitions declaring them as simplification rules should be harmless.
declare HPhase1or2ReadThen-def [simp]
declare HPhase1or2ReadElse-def [simp]
declare HEndPhase1-def [simp]
declare HStartBallot-def [simp]
declare HPhase1or2Write-def [simp]
declare HEndPhase2-def [simp]
declare HFail-def [simp]
declare HPhase0Read-def [simp]
declare HEndPhase0-def [simp]
end

C Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1 Invariant 1

This is just a type Invariant.
definition Inv1 :: state ⇒ bool where
Inv1 s = (∀p. inpt s p ∈ Inputs ∧ phase s p ≤ 3 ∧ finite (allRdBlks s p))
definition HInv1 :: state ⇒ bool where
HInv1 s = (Inv1 s ∧ allInput s ⊆ Inputs)
declare HInv1-def [simp]

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

**lemma HNextPart-Inv1:** \[ HInv1 s; HNextPart s s’; Inv1 s’ \] \[\implies HInv1 s’ \]
by(auto simp add: HNextPart-def Inv1-def)

**theorem HInit-HInv1:** HInit s \[\implies HInv1 s \]
by(auto simp add: HInit-def Inv1-def Init-def allRdBlks-def)

**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
fix pp
from inv
have \(\forall p. \text{finite } (\text{allRdBlks } s p)\)
  by(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)
qued

**theorem HPhase1or2ReadThen-HInv1:**
assumes inv1: HInv1 s
and act: HPhase1or2ReadThen s s’ p d q
shows HInv1 s’
proof
— 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)
qued

**theorem HEndPhase1-HInv1:**
assumes inv1: HInv1 s
and act: HEndPhase1 s s’ p
shows HInv1 s’
proof
from inv1 act
have \textit{Inv1} s',
by (auto simp add: \textit{Inv1-def} \textit{EndPhase1-def} \textit{InitializePhase-def} \textit{allRdBlks-def})
with \textit{inv1} act
show ?thesis
by (auto simp del: \textit{HInv1-def} dest: \textit{HNextPart-Inv1})

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

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

qed
definition \textit{HPhase1or2ReadElse-HInv1}
assumes \textit{act}: \textit{HPhase1or2ReadElse} s s' p d q
and \textit{inv1}: \textit{HInv1} s
shows \textit{HInv1} s'
using \textit{HStartBallot-HInv1}\{OF \textit{inv1}\} act
by (auto simp add: \textit{Phase1or2ReadElse-def})
definition \textit{HEndPhase2-HInv1}
assumes \textit{inv1}: \textit{HInv1} s
and \textit{act}: \textit{HEndPhase2} s s' p
shows \textit{HInv1} s'
proof
from \textit{inv1} act
have \textit{Inv1} s'
by (auto simp add: \textit{Inv1-def} \textit{EndPhase2-def} \textit{InitializePhase-def} \textit{allRdBlks-def})
with \textit{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 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)
qed

\textbf{declare} \emph{HInv1-def \ [simp del]} \\[HInv1\] is an invariant of \emph{HNext}

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

\textbf{end}

\textbf{theory DiskPaxos-Inv2 imports DiskPaxos-Inv1 begin}

\textbf{C.2 Invariant 2}

The second invariant is split into three main conjuncts called \emph{Inv2a}, \emph{Inv2b}, and \emph{Inv2c}. The main difficulty is in proving the preservation of the first conjunct.

\textbf{definition} \emph{rdBy :: state} ⇒ \emph{Proc} ⇒ \emph{Proc} ⇒ \emph{Disk} ⇒ \emph{BlockProc set}
\textbf{where}
\begin{itemize}
\item \emph{rdBy} \ s \ p \ q \ d = \\
\{ br . \ br \in \ blocksRead \ s \ q \ d \wedge \ proc \ br = p \} \\
\end{itemize}

\textbf{definition} \emph{blocksOf :: state} ⇒ \emph{Proc} ⇒ \emph{DiskBlock set}
\textbf{where}
\begin{itemize}
\item \emph{blocksOf} \ s \ p = \\
\{ \text{dblock} \ s \ p \} \\
\cup \{ \text{disk} \ s \ d \ p \ | \ d . \ d \in \ UNIV \} \\
\cup \{ \text{block} \ br \ | \ br . \ br \in (\text{UN} \ q \ d . \ \text{rdBy} \ s \ p \ q \ d) \}
\end{itemize}

\textbf{definition} \emph{allBlocks :: state} ⇒ \emph{DiskBlock set}

\textbf{end}
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 ∥ inp t p ∈ allInput s
  ∧ (chosen s = NotAnInput → outpt s p = NotAnInput)})

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

definition Inv2c-inner :: state ⇒ Proc ⇒ bool
where
Inv2c-inner s p =
  ((phase s p = 0 →
    (dblock s p = InitDB
∧ disksWritten s p = {}
∧ (∀ d, ∀ br ∈ blocksRead s p d.
    proc br = p ∧ block br = disk s d 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 ∥ inp t p ∈ allInput s
  ∧ (chosen s = NotAnInput → outpt s p = NotAnInput)})

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

**definition** \( H\text{Inv2} :: \text{state} \Rightarrow \text{bool} \)

where \( H\text{Inv2} \ s = (\text{Inv2a} \ s \land \text{Inv2b} \ s \land \text{Inv2c} \ s) \)

### C.2.1 Proofs of Invariant 2 a

**theorem** \( H\text{Init-Inv2a}: H\text{Init} \ s \longrightarrow \text{Inv2a} \ s \)

by (auto simp add: \( H\text{Init-def} \) \( \text{Init-def} \) \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \) \( \text{rdBy-def} \) \( \text{blocksOf-def} \) \( \text{InitDB-def} \))

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-\( \text{Inv2a-dblock} \).

**lemma** \( H\text{Phase1or2ReadThen-blocksOf}: \)

\[ (\text{HPhase1or2ReadThen} \ s \ s' p d q) \Rightarrow \text{blocksOf} \ s' r \subseteq \text{blocksOf} \ s r \]

by (auto simp add: \( \text{Phase1or2ReadThen-def} \) \( \text{blocksOf-def} \) \( \text{rdBy-def} \) \( \text{blocksOf-def} \) \( \text{rdBy-def} \))

**theorem** \( H\text{Phase1or2ReadThen-Inv2a}: \)

assumes \( \text{inv} \): \( \text{Inv2a} \ s \) and \( \text{act} \): \( H\text{Phase1or2ReadThen} \ s \ s' p d q \)

shows \( \text{Inv2a} \ s' \)

proof (clarsimp simp add: \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \))

fix \( pp \ bk \)

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

with \( \text{inv} \) \( H\text{Phase1or2ReadThen-blocksOf}[OF \text{act}] \)

have \( \text{Inv2a-innermost} \ s pp bk \)

by (auto simp add: \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \))

with \( \text{act} \)

show \( \text{Inv2a-innermost} \ s' pp bk \)

by (auto simp add: \( \text{Inv2a-innermost-def} \) \( \text{HNextPart-def} \))

qed

**lemma** \( \text{InitializePhase-rdB} \):

\( \text{InitializePhase} \ s s' p \Rightarrow \text{rdBy} \ s s' pp qq dd \subseteq \text{rdBy} \ s pp qq dd \)

by (auto simp add: \( \text{InitializePhase-def} \) \( \text{rdBy-def} \))

**lemma** \( H\text{StartBallot-blocksOf}: \)

\( H\text{StartBallot} \ s s' p \Rightarrow \text{blocksOf} \ s' q \subseteq \text{blocksOf} \ s q \cup \{ \text{dblock} \ s' q \} \)

by (auto simp add: \( \text{StartBallot-def} \) \( \text{blocksOf-def} \) \( \text{dest: subsetD}[OF \text{InitializePhase-rdB}] \))

**lemma** \( H\text{StartBallot-Inv2a-dblock}: \)

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

and \( \text{inv2a} \): \( \text{Inv2a-innermost} \ s p \ (\text{dblock} \ s p) \)

25
shows Inv2a-innermost s' p (dblock s' p)

proof –
from act 
have mbal': mbal (dblock s' p) ∈ Ballot p 
  by(auto simp add: StartBallot-def)
from act 
have bal': bal (dblock s' p) = bal (dblock s p) 
  by(auto simp add: StartBallot-def)
with act 
have inp': inp (dblock s' p) = inp (dblock s p) 
  by(auto simp add: StartBallot-def)
from act 
have mbal (dblock s p) ≤ mbal (dblock s' p) 
  by(auto simp add: StartBallot-def)
with bal' inv2a 
have bal-mbal: bal (dblock s' p) ≤ mbal (dblock s' p) 
  by(auto simp add: Inv2a-innermost-def)
from act 
have allInput s ⊆ allInput s' 
  by(auto simp add: HNextPart-def InitializePhase-def Inv2a-innermost-def)
with mbal' bal' inp' bal-mbal act inv2a 
show ?thesis 
by(auto simp add: Inv2a-innermost-def)
qed

lemma HStartBallot-Inv2a-dblock-q:
  assumes act: HStartBallot s s' p 
  and inv2a: Inv2a-innermost s q (dblock s q) 
  shows Inv2a-innermost s' q (dblock s' q)
proof(cases p=q)
  case True 
  with act inv2a 
  show ?thesis 
    by(blast dest: HStartBallot-Inv2a-dblock)
next
  case False 
  with act inv2a 
  show ?thesis 
    by(clarsimp simp add: StartBallot-def HNextPart-def InitializePhase-def Inv2a-innermost-def)
qed

theorem HStartBallot-Inv2a:
  assumes inv: Inv2a s 
  and act: HStartBallot s s' p 
  shows Inv2a s' 
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk 
  assume bk: bk ∈ blocksOf s' q

26
with inv
have oldBlks: bk ∈ blocksOf s q → Inv2a-innermost s q bk
  by (auto simp add: Inv2a-def Inv2a-inner-def)
from bk HStartBallot-blocksOf[OF act]
have bk ∈ {dblock s' q} ∪ blocksOf s q
  by blast
thus Inv2a-innermost s' q bk
proof
  assume bk-dblock: bk ∈ {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 inv bk-dblock
  show ?thesis
    by (blast dest: HStartBallot-Inv2a-dblock-q)
next
  assume bk-in-blocks: bk ∈ blocksOf s q
  with oldBlks
  have Inv2a-innermost s q bk ..
  with act
  show ?thesis
    by (auto simp add: StartBallot-def HNextPart-def
                      InitializePhase-def Inv2a-innermost-def)
qed

lemma HPhase1or2Write-blocksOf:
  \[ HPhase1or2Write s s' p d \] ⇒ blocksOf s' r ⊆ blocksOf s r
by (auto simp add: Phase1or2Write-def blocksOf-def rdBy-def)

theorem HPhase1or2Write-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase1or2Write s s' p d
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ blocksOf s' q
  from inv bk HPhase1or2Write-blocksOf[OF act]
  have inp-q-bk: Inv2a-innermost s q bk
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
  with act
  show Inv2a-innermost s' q bk
    by (auto simp add: Inv2a-innermost-def HNextPart-def)
qed

theorem HPhase1or2ReadElse-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase1or2ReadElse s s' p d q
  shows 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:
  \[ HEndPhase2 s s' p \implies \text{blocksOf } q \subseteq \text{blocksOf } s q \]
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 \text{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 \text{blocksOf } q \subseteq \text{blocksOf } s q \cup \{\text{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 thesis by (auto simp add: Fail-def HNextPart-def InitializePhase-def Inv2a-innermost-def)

qed

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

qed

lemma HPhase0Read-blocksOf:
  HPhase0Read s s' p d ⇒ blocksOf s' q ⊆ blocksOf s q
by (auto simp add: Phase0Read-def InitializePhase-def blocksOf-def rdBy-def)

theorem HPhase0Read-Inv2a:
  assumes inv: Inv2a s
  and  act: HPhase0Read s s' p d
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ 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 ⟹ blocksOf s′ q ⊆ blocksOf s q ∪ \{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 ∃ d. blocksRead s p d ≠ {}
proof –
  from act
  have IsMajority({d. hasRead s p d p}) by (simp add: EndPhase0-def)
  hence {d. hasRead s p d p} ≠ {} 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 ⟹ ?P (Eps ?P).

lemma HEndPhase0-some:
  assumes act: HEndPhase0 s s′ p
  and inv1: Inv1 s
  shows (∃ b. b ∈ allBlocksRead s p
              ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal b)
              ) ∈ allBlocksRead s p
        ∧ (∀ t ∈ allBlocksRead s p.
           bal t ≤ bal (SOME b. b ∈ allBlocksRead s p
                                ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal b)))
proof –
  from inv1 have finite (bal ′ allBlocksRead s p) (is finite ?S)
    by (simp add: Inv1-def allBlocksRead-def)
  moreover
  from HEndPhase0-blocksRead[OF act]
  have ?S ≠ {}
    by (auto simp add: allBlocksRead-def allRdBlks-def)
  ultimately
  have Max ?S ∈ ?S and ∀ t ∈ ?S. t ≤ Max ?S by auto
  hence ∃ r ∈ ?S. ∀ t ∈ ?S. t ≤ r ..
  then obtain mblk
  where mblk ∈ allBlocksRead s p
        ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk) (is ?P mblk)
by auto
thus \(?thesis\)
by (rule someI)
qed

lemma HEndPhase0-dblock-allBlocksRead:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
shows dblock s' p \(\in (\lambda x. (mbal:= mbal(dblock s' p)))) \cdot \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 inp (dblock s' p) \(\in \text{allInput s' \cup \{NotAnInput\}}\)
proof -
from act
have allInput s' = allInput s \cup (range (inpt s'))
  by(simp add: HNextPart-def)
moreover
from inv2a
have inp (dblock s' p) \(\in \text{allInput s' \cup \{NotAnInput\}}\)
  by(simp add: Inv2a-innermost-def)
ultimately show \(?thesis\)
  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 \(\forall t \in (\lambda x. (mbal:= mbal (dblock s' p)))) \cdot \text{allBlocksRead s p. Inv2a-innermost s p t}\)
proof -
from act
have mbal': mbal (dblock s' p) \(\in \text{Ballot p}\)
  by(auto simp add: EndPhase0-def)
from inv2c act
have allproc-p: \(\forall d, \forall \text{br} \in \text{blocksRead s p d, proc br = p}\)
  by(simp add: Inv2c-inner-def EndPhase0-def)
with inv2a
have allBlocks-inv2a: \(\forall t \in \text{allBlocksRead s p. Inv2a-innermost s p t}\)
proof(auto simp add: Inv2a-inner-def allBlocksRead-def
  allRdBkls-def blocksOf-def rdBy-def)
fix d bk
assume bk-in-blocksRead: bk \(\in \text{blocksRead s p d}\)
and inv2a-bk: \(\forall x \in \{u. \exists d. u = disk s d p\}\)
  \(\cup \{\text{block \ br | \br. (\exists q d. \br \in \text{blocksRead s q d})}\}\)
∧ proc br = p}. Inv2a-innermost s p x

  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: ∀ bk ∈ allBlocksRead s p. mbal bk ≤ 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 ∈ allBlocksRead s p
with allBlocks-inv2a
have_bal t ≤ mbal t by (auto simp add: Inv2a-innermost-def)
moreover
from t-blocksRead mbal'gt
have mbal t ≤ mbal (dblock s' p) by blast
ultimately show bal t ≤ 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: ∀ t ∈ (λx. x (λmbal.:= mbal (dblock s' p))) \ allBlocksRead s p.
  Inv2a-innermost s p t
  by (blast dest: HEndPhase0-Inv2a-allBlocksRead)
from act inv1
have dblock s' p ∈ (λx. x (λmbal.:= mbal(dblock s' p))) \ 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) ∈ allInput s' ∪ {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: Inv1 s}

\text{and}

\text{inv2a: Inv2a-inner s q}

\text{and}

\text{inv2c: Inv2c-inner s p}

shows \text{Inv2a-innermost s' q (dblock s' q)}

proof(cases p=q)

case True

with act inv2a inv2c inv1

show \text{thesis}

by(blast dest: HEndPhase0-Inv2a-dblock)

next

case False

from inv2a

have inv-q-dblock: \text{Inv2a-innermost s q (dblock s q)}

by(auto simp add: Inv2a-inner-def blocksOf-def)

with False act

show \text{thesis}

by(clarsimp simp add: EndPhase0-def HNextPart-def

InitializePhase-def Inv2a-innermost-def)

qed

theorem HEndPhase0-Inv2a:

assumes

\text{inv: Inv2a s}

\text{and}

\text{act: HEndPhase0 s s' p}

\text{and}

\text{inv1: Inv1 s}

\text{and}

\text{inv2c: Inv2c-inner s p}

shows \text{Inv2a s'}

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

fix q bk

assume bk: \text{bk \in blocksOf s' q}

with HEndPhase0-blocksOf[OF act]

have dblock-blocks: bk \in \{dblock s' q\} \cup 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: Inv2a-def)

assume bk \in \{dblock s' q\}

with act inv1 inv2c inv-q

show \text{thesis}

by(blast dest:HEndPhase0-Inv2a-dblock-q)

next

assume bk-in-blocks: bk \in blocksOf s q

with inv

have \text{Inv2a-innermost s q bk}

by(auto simp add: Inv2a-def Inv2a-inner-def)

with act show \text{thesis}

by(auto simp add: EndPhase0-def HNextPart-def

InitializePhase-def Inv2a-innermost-def)
qed

\textbf{lemma} \texttt{HEndPhase1-blocksOf}:
\texttt{HEndPhase1} \ s \ s' \ p \ \implies \ \texttt{blocksOf} \ s' \ q \ \subseteq \ \texttt{blocksOf} \ s \ q \ \cup \ \{ \texttt{dblock} \ s' \ q \}
\textbf{by} (\textit{auto simp add: EndPhase1-def blocksOf-def dest: subsetI[OF InitializePhase-rdBy])}

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

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

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

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

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 inv2
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 inv2: 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 inv
  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 inv: Inv2a \( s \)
and inv2c: Inv2c-inner \( s \ p \)
shows Inv2a \( s' \)

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

proof
  assume bk \in \{dblock s' q\}
  with act inv1 inv2c inv
  show \(?thesis\)
  by (blast dest: HEndPhase1-Inv2a-dblock-q)
next
  assume bk-in-blocks: \( bk \in \) blocksOf \( s \ q \)
  with inv
  have Inv2a-innermost \( s \ q \ bk \)
    by (auto simp add: Inv2a-def Inv2a-inner-def)
  with act show \(?thesis\)
    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 HInit-Inv2b: HInit \( s \longrightarrow \) Inv2b \( s \)
by (auto simp add: HInit-def Init-def Inv2b-def
  Inv2b-inner-def InitDB-def)

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

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

theorem HPhase1or2Write-Inv2b:
  [ \( \text{Inv2b} \ s; \text{HPhase1or2Write} \ s \ s' \ p \ d \] \implies \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 s' p ∈ 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 blocksOof-def)
  with act inv outpt'
  have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
    by (auto dest: HNextPart-Inv2c-chosen)
  from act inv
have \( \forall p \cdot \text{inpt} s' p \in \text{allInput} s' \)
\( \land (\text{chosen} s' = \text{NotAnInput} \rightarrow \text{outpt} s' p = \text{NotAnInput}) \)
by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show \( ?\text{thesis} \)
by (auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)
qed

theorem HStartBallot-Inv2c:
assumes inv: Inv2c s
and act: HStartBallot s s' p
and inv2a: Inv2a s
shows 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': mbal(dblock s' p) \in \text{Ballot} p
  by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
from inv phase
have bal(dblock s p) \in \text{Ballot} p \cup \{0\}
  by (auto simp add: Inv2c-def Inv2c-inner-def)
with act
have bal': bal(dblock s' p) \in \text{Ballot} p \cup \{0\}
  by (auto simp add: StartBallot-def)
from act inv phase phase'
have blks': (\( \forall d. \forall br \in \text{blocksRead} s' p d. 
  mbal(block br) < mbal(dblock s' p) \))
  by (auto simp add: StartBallot-def InitializePhase-def 
  Inv2c-def Inv2c-inner-def)
from inv2a act
have inv2a': Inv2a s'
  by (blast dest: HStartBallot-Inv2a)
from act inv
have outpt': \( \forall p. \text{outpt} s' p = (\text{if phase} s' p = 3 
  \text{then inp(dblock s' p) 
  else NotAnInput}) \)
  by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: \( \forall p. \text{inp (dblock s' p) \in allInput s' \cup \{NotAnInput\} \)
  by (auto simp add: Inv2a-def Inv2a-inner-def 
  Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen s' \in allInput s' \cup \{NotAnInput\}
  by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \( \forall p. \text{inpt} s' p \in \text{allInput} s' \)
    \( \land (\text{chosen} s' = \text{NotAnInput} \implies \text{outpt} s' p = \text{NotAnInput}) \)
    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 Inv2c s'
proof –
from inv2a act
have inv2a': Inv2a s'
    by(blast dest: HPhase1or2Write-Inv2a)
from act inv
have outpt': \( \forall p. \text{outpt} s' p = \text{(if phase} s' p = 3 \text{then inp(dblock s'} p) \text{else NotAnInput}) \)
    by(auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: \( \forall p. \text{inp (dblock s'} p) \in \text{allInput} s' \cup \{\text{NotAnInput}\} \)
    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: \( \forall p. \text{inpt} s' p \in \text{allInput} s' \land (\text{chosen} s' = \text{NotAnInput} \implies \text{outpt} s' p = \text{NotAnInput}) \)
    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:
[ Inv2c s; HPhase1or2ReadElse s s' p d q; Inv2a s ] \implies 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' p ∪ {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' p ∪ {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
show Inv2c s'
proof – 
  from inv2a act

42
have \textit{inv2a': Inv2a s'}
by (blast dest: \textit{HEndPhase2-Inv2a})

from \textit{act inv}
have \textit{outpt': \forall p. outpt s' p = (if phase s' p = 3 then inp (dblock s' p)
else NotAnInput)}
by (auto simp add: \textit{EndPhase2-def Inv2c-def Inv2c-inner-def})

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

with \textit{act inv outpt'}
have \textit{chosen': chosen s' \in allInput s' \cup \{NotAnInput\}}
by (auto dest: \textit{HNextPart-Inv2c-chosen})

from \textit{act inv}
have \textit{allinp: \forall p. inpt s' p \in allInput s'
\& (chosen s' = NotAnInput
\rightarrow outpt s' p = NotAnInput)}
by (auto dest: \textit{HNextPart-chosen HNextPart-allInput})

with \textit{outpt' chosen' act inv}
show \textit{?thesis}
by (auto simp add: \textit{EndPhase2-def InitializePhase-def Inv2c-def Inv2c-inner-def})

qed

\textbf{theorem} \textit{HFail-Inv2c:}
\textbf{assumes} inv: \textit{Inv2c s}
\textbf{and} act: \textit{HFail s s' p}
\textbf{and} inv2a: \textit{Inv2a s}
\textbf{shows} \textit{Inv2c s'}

\textbf{proof} –
from \textit{inv2a act}
have \textit{inv2a': Inv2a s'}
by (blast dest: \textit{HFail-Inv2a})

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

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

with \textit{act inv outpt'}
have \textit{chosen': chosen s' \in allInput s' \cup \{NotAnInput\}}
by (auto dest: \textit{HNextPart-Inv2c-chosen})

from \textit{act inv}
have \textit{allinp: \forall p. inpt s' p \in allInput s' \& (chosen s' = NotAnInput
\rightarrow outpt s' p = NotAnInput)}
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. outpt s' p = (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 \{NotAnInput\}
    by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen s' \in allInput s' \cup \{NotAnInput\}
    by (auto dest: HNextPart-Inv2c-chosen)
  from act inv
  have allinp: \forall p. inpt s' p \in allInput s'
    \land (chosen s' = NotAnInput
    \land outpt s' p = 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 \textit{inv2a'}: Inv2a s'
  by (blast dest: HEndPhase0-Inv2a)

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

\textit{from act inv}

have \textit{mbal'}: mbal(dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase0-def Inv2c-def Inv2c-inner-def)

\textit{from act inv}

have \textit{blks'}: (∀d. ∀br ∈ blocksRead s' p d.
    mbal(block br) < mbal(dblock s' p))
  by (auto simp add: EndPhase0-def InitializePhase-def
       Inv2c-def Inv2c-inner-def)

\textit{from act inv}

have \textit{outpt'}: ∀p. outpt s' p = (if phase s' p = 3
    then inp(dblock s' p)
    else NotAnInput)
  by (auto simp add: EndPhase0-def Inv2c-def Inv2c-inner-def)

\textit{from \textit{inv2a'}}

have \textit{dblk}: ∀p. inp(dblock s' p) ∈ allInput s' ∪ \{NotAnInput\}
  by (auto simp add: Inv2a-def Inv2a-inner-def
       Inv2a-innermost-def blocksOf-def)

\textit{with \textit{act inv outpt'}}

have \textit{chosen'}: chosen s' ∈ allInput s' ∪ \{NotAnInput\}
  by (auto dest: HNextPart-Inv2c-chosen)

\textit{from act inv}

have \textit{allinp}: ∀p. inpt s' p ∈ allInput s' ∧ (chosen s' = NotAnInput
    → outpt s' p = NotAnInput)
  by (auto dest: HNextPart-chosen HNextPart-allInput)

with \textit{mbal'} \textit{bal'} \textit{blks'} \textit{outpt'} \textit{chosen'} \textit{act inv}

show ?thesis
  by (auto simp add: EndPhase0-def InitializePhase-def
       Inv2c-def Inv2c-inner-def)

qed

\textbf{theorem} \textit{HInit-HInv2}:
  HInit s \implies HInv2 s
\textbf{using} HInit-Inv2a HInit-Inv2b HInit-Inv2c
\textbf{by} (auto simp add: HInv2-def)

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

\textbf{lemma} \textit{I2b}:
  \textbf{assumes} \textit{nxt}: HNext s s'
  \textbf{and} \textit{inv}: HInv1 s \land HInv2 s
  \textbf{shows} HInv2 s'
\textbf{proof} (auto simp add: HInv2-def)

\textbf{show} \textit{Inv2a} s' \textbf{using} \textit{assms}
  \textbf{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: EndPhase1or2-2-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: EndPhase1or2-2-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: EndPhase1or2-2-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
\[
H_{\text{inv3-L}} s p q d = (\text{phase } s p \in \{1,2\} \\
\land \text{phase } s q \in \{1,2\} \\
\land \text{hasRead } s p d q \\
\land \text{hasRead } s q d p)
\]

definition \(H_{\text{inv3-R}}\) :: \(\text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}\)
where
\(H_{\text{inv3-R}} s p q d = (\text{\{block = dblock } s q, \text{proc} = q\} \in \text{blocksRead } s p d \\
\lor \text{\{block = dblock } s p, \text{proc} = p\} \in \text{blocksRead } s q d)\)

definition \(H_{\text{inv3-inner}}\) :: \(\text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool}\)
where \(H_{\text{inv3-inner}} s p q d = (H_{\text{inv3-L}} s p q d \rightarrow H_{\text{inv3-R}} s p q d)\)

definition \(H_{\text{inv3}}\) :: \(\text{state} \Rightarrow \text{bool}\)
where \(H_{\text{inv3}} s = (\forall p q d. \text{\(H_{\text{inv3-inner}} s p q d\)})\)

### C.3.1 Proofs of Invariant 3

**theorem** \(H_{\text{init}} \Rightarrow H_{\text{inv3}}\): \(H_{\text{init}} s \Rightarrow H_{\text{inv3}} s\)
by (simp add: \(H_{\text{init-def}} \Rightarrow H_{\text{inv3-def}}\))

**lemma** \(\text{InitPhase}\)-\(H_{\text{inv3-p}}\):
\([\text{InitializePhase } s s' p; H_{\text{inv3-L}} s' p q d] \Rightarrow H_{\text{inv3-R}} s' p q d\)
by (auto simp add: \(\text{InitializePhase-def} \Rightarrow H_{\text{inv3-inner-def}}\))

**lemma** \(\text{InitPhase}\)-\(H_{\text{inv3-q}}\):
\([\text{InitializePhase } s s' q; H_{\text{inv3-L}} s' p q d] \Rightarrow H_{\text{inv3-R}} s' p q d\)
by (auto simp add: \(\text{InitializePhase-def} \Rightarrow H_{\text{inv3-inner-def}}\))

**lemma** \(H_{\text{inv3-L-sym}}\): \(H_{\text{inv3-L}} s p q d \Rightarrow H_{\text{inv3-L}} s q p d\)
by (auto simp add: \(H_{\text{inv3-L-def}}\))

**lemma** \(H_{\text{inv3-R-sym}}\): \(H_{\text{inv3-R}} s p q d \Rightarrow H_{\text{inv3-R}} s q p d\)
by (auto simp add: \(H_{\text{inv3-R-def}}\))

**lemma** \(\text{Phase1or2ReadThen}\)-\(H_{\text{inv3-pq}}\):
assumes \(\text{act} : \text{Phase1or2ReadThen } s s' p d q\)
and \(\text{inv-L'} : H_{\text{inv3-L}} s' p q d\)
and \(\text{pq} : p \neq q\)
and \(\text{inv2b} : \text{Inv2b } s\)
shows \(H_{\text{inv3-R}} s' p q d\)
proof –
from \(\text{inv-L'}\) \(\text{act} pq\)
have \(\text{phase } s q \in \{1,2\} \land \text{hasRead } s q d p\)
by (auto simp add: \(\text{Phase1or2ReadThen-def} \Rightarrow H_{\text{inv3-L-def}}\))
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\ne p \ve qq\ne q \ve dd\ne d ] ]
  \implies ~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\ne 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: Phase1or2ReadThen-def)
next
  case False
  assume nh3l: ~ 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 ~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 ]
  ⇒ 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 ]
  ⇒ HInv3-R s' p q d
by(auto simp add: StartBallot-def dest: InitPhase-HInv3-q)

lemma StartBallot-HInv3-nL:
  [ StartBallot s s' t; ¬HInv3-L s p q d; t≠p; t≠ q ]
  ⇒ ¬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≠p; t≠ q ]
  ⇒ 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≠p; t≠ q ]
  ⇒ 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 ∨ 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:
  [ HStartBallot s s' p; HInv3 s ] \Rightarrow HInv3 s'
by (auto simp add: Hinv3-def dest: StartBallot-HInv3)

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

theorem HPhase1or2Write-HInv3:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv3 s
  shows HInv3 s'
proof (auto simp add: HInv3-def)
  fix pp qq dd
  show HInv3-inner s' pp qq dd
proof (cases HInv3-L 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
                      Phase1or2Write-def)
next
  case False
  with act
  have \neg 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
qed

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

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

lemma EndPhase1-HInv3-nL:
lemma EndPhase1-HInv3-R:
[ EndPhase1 s s' t; HInv3-R s p q d ]
⇒ 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 ]
⇒ 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 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: EndPhase1-HInv3-p EndPhase1-HInv3-q)
next
  case False
  with inv act
  show ?thesis
    by(auto simp add: HInv3-inner-def dest: EndPhase1-HInv3-t)
qed

theorem HEndPhase1-HInv3:
[ HEndPhase1 s s' p; HInv3 s ] ⇒ 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 ] ⇒ 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 ] ⇒ HInv3-R s' p q d
by(auto simp add: EndPhase2-def dest: InitPhase-HInv3-q)

lemma EndPhase2-HInv3-nL:
[ EndPhase2 s s' t; ¬HInv3-L s p q d; t≠p; t≠ q ]
⇒ ¬HInv3-L s' p q d
by (auto simp add: EndPhase2-def InitializePhase-def 
HInv3-L-def hasRead-def)

\textbf{lemma} \ EndPhase2-HInv3-R: 
\[ \text{EndPhase2} s s' t ; \ HInv3-R s p q d \quad \text{t} \neq p ; \ t \neq q \]  
\[ \Longrightarrow \ HInv3-R s' p q d \]  
by (auto simp add: EndPhase2-def InitializePhase-def 
HInv3-R-def hasRead-def)

\textbf{lemma} \ EndPhase2-HInv3-t: 
\[ \text{EndPhase2} s s' t ; \ HInv3-inner s p q d \quad \text{t} \neq p ; \ t \neq q \]  
\[ \Longrightarrow \ HInv3-inner s' p q d \]  
by (auto simp add: HInv3-inner-def 
dest: EndPhase2-HInv3-nL EndPhase2-HInv3-R)

\textbf{lemma} \ EndPhase2-HInv3: 
\textbf{assumes} \ act: \text{EndPhase2} s s' t 
\textbf{and} \ inv: \ HInv3-inner s p q d 
\textbf{shows} \ HInv3-inner s' p q d 
proof \ (cases t = p \lor t = q) 
\begin{itemize} 
    \item \textbf{case} \ True 
    \item \textbf{with} \ act \ inv 
    \item \textbf{show} \ ?thesis 
        by (auto simp add: HInv3-inner-def 
            dest: EndPhase2-HInv3-p EndPhase2-HInv3-q) 
    \end{itemize} 
\textbf{next} 
\begin{itemize} 
    \item \textbf{case} \ False 
    \item \textbf{with} \ inv \ act 
    \item \textbf{show} \ ?thesis 
        by (auto simp add: HInv3-inner-def dest: EndPhase2-HInv3-t) 
\end{itemize} 
qed

\textbf{theorem} \ HEndPhase2-HInv3: 
\[ \text{HEndPhase2} s s' p ; \ HInv3 s \]  
\[ \Longrightarrow \ HInv3 s' \]  
by (auto simp add: HInv3-def dest: EndPhase2-HInv3)

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

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

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

52
lemma Fail-HInv3-R:
\[
\begin{array}{l}
\text{Fail s s'; HInv3-R s p q d; t} \neq p; t \neq q \implies HInv3-R s' p q d \\
\text{by (auto simp add: Fail-def InitializePhase-def HInv3-R-def hasRead-def)}
\end{array}
\]

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

lemma Fail-HInv3:
\[
\begin{array}{l}
\text{assumes act: Fail s s' t and inv: HInv3-inner s p q d shows HInv3-inner s' p q d}
\end{array}
\]
\[
\begin{array}{l}
\text{proof (cases t=p \lor t=q)} \\
\text{case True with inv act show ?thesis by (auto simp add: HInv3-inner-def HInv3-R-def Phase0Read-def)}
\end{array}
\]

next
\[
\begin{array}{l}
\text{case False with inv act show ?thesis by (auto simp add: HInv3-inner-def dest: Fail-HInv3-t)}
\end{array}
\]

qed

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

theorem HPhase0Read-HInv3:
\[
\begin{array}{l}
\text{assumes act: HPhase0Read s s' p d and inv: HInv3 s shows HInv3 s'}
\end{array}
\]
\[
\begin{array}{l}
\text{proof (auto simp add: HInv3-def)} \\
\text{fix pp qq dd} \\
\text{show HInv3-inner s' pp qq dd proof (cases HInv3-L s pp qq dd)} \\
\text{case True with inv have HInv3-R s pp qq dd by (simp add: HInv3-def HInv3-inner-def)} \\
\text{with act show ?thesis by (auto simp add: HInv3-inner-def HInv3-R-def Phase0Read-def)}
\end{array}
\]
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

lemma 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)

lemma 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)

lemma 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)

lemma 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)

lemma 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)

lemma 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)
next
\textbf{case} False \\
\textbf{with} inv act \\
\textbf{show} \texttt{?thesis} \\
\hspace{2em} \textbf{by} (auto simp add: HInv3-inner-def dest: EndPhase0-HInv3-t) \\
\textbf{qed}

\textbf{theorem} HEndPhase0-HInv3: \\
\[ [ \texttt{HEndPhase0\ s\ s'}\ p; \texttt{HInv3\ s} ] \implies \texttt{HInv3\ s'} \] \\
\textbf{by} (auto simp add: HInv3-def dest: EndPhase0-HInv3)

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

\textbf{lemma} \texttt{I2c}:
\begin{itemize}
\item \textbf{assumes} \(\texttt{next: HNext\ s\ s'}\)
\item \textbf{and} \(\texttt{inv: HInv1\ s} \land \texttt{HInv2\ s} \land \texttt{HInv3\ s}\)
\item \textbf{shows} \(\texttt{HInv3\ s'}\) \textbf{using} \(\texttt{assms}\)
\end{itemize}
\textbf{by} (auto simp add: HNext-def Next-def, \\
\hspace{2em} auto intro: HStartBallot-HInv3, \\
\hspace{2em} auto intro: HPhase0Read-HInv3, \\
\hspace{2em} auto intro: HPhase1or2Write-HInv3, \\
\hspace{2em} auto simp add: Phase1or2Read-def HInv2-def \\
\hspace{4em} intro: HPhase1or2ReadThen-HInv3 \\
\hspace{4em} HPhase1or2ReadElse-HInv3, \\
\hspace{2em} auto simp add: EndPhase1or2-def \\
\hspace{4em} intro: HEndPhase1-HInv3 \\
\hspace{4em} HEndPhase2-HInv3, \\
\hspace{2em} auto intro: HFail-HInv3, \\
\hspace{2em} auto intro: HEndPhase0-HInv3)

end

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

\textbf{C.4} \textbf{Invariant 4}

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

\textbf{definition} \textit{MajoritySet} :: Disk set set \\
\hspace{2em} \textbf{where} \textit{MajoritySet} = \{ D. \textit{IsMajority}(D) \}

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

definition $HInv4a2 :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

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

\land \ \text{bal} (\text{disk} \ s \ d \ p) \leq \text{bal} (\text{dblock} \ s \ p))$

definition $HInv4a :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where $HInv4a \ s \ p = (\text{phase} \ s \ p \neq 0 \rightarrow HInv4a1 \ s \ p \ \land \ HInv4a2 \ s \ p)$

definition $HInv4b :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where $HInv4b \ s \ p = (\text{phase} \ s \ p = 1 \rightarrow (\forall bk \in \text{blocksOf} \ s \ p. \ \text{bal} \ bk < \text{mbal} (\text{dblock} \ s \ p)))$

definition $HInv4c :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

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

definition $HInv4d :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

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

definition $HInv4 :: \text{state} \Rightarrow \text{bool}$

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 $HInit-HInv4$: $HInit \ s \Rightarrow HInv4 \ s$

using $\text{Disk-isMajority}$

by (auto simp add: $HInit$-def $Init$-def $HInv4$-def $HInv4a1$-def
     $HInv4a2$-def $HInv4b$-def $HInv4c$-def $HInv4d$-def
     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 $act \ p'$ and conjunct $x \in a, b, c, d$ of $HInv4xs'p$, we prove two lemmas. The first lemma $act-HInv4x-p'$ proves the case of $p = q$, while lemma $act-HInv4x-q$ proves the other case.

C.4.1 Proofs of Invariant 4a

lemma $HStartBallot-HInv4a1$:

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

shows $HInv4a1 \ s' \ p$

proof (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 ?thesis
by(auto simp add: \text{Inv2a-innermost-def Inv2a-inner-def blocksOf-def})
next
assume \( bk \in \text{blocksOf } s p \)
with \( \text{inv act} \)
show ?thesis
by(auto simp add: \text{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: \text{HInv4a2-def})
fix \( D \)
assume \( \text{Dmaj: } D \in \text{MajoritySet} \)
from \( \text{inv Dmaj} \)
have \( \exists d \in D. \ mbal (\text{disk } s d p) \leq \text{mbal } (\text{dblock } s p) \)
\( \land \text{bal } (\text{disk } s d p) \leq \text{bal } (\text{dblock } s p) \)
by(auto simp add: \text{HInv4a2-def})
then obtain \( d \)
where \( d \in D \)
\( \land mbal (\text{disk } s d p) \leq \text{mbal } (\text{dblock } s p) \)
\( \land \text{bal } (\text{disk } s d p) \leq \text{bal } (\text{dblock } s p) \)
by auto
with \( \text{act} \)
have \( d \in D \)
\( \land mbal (\text{disk } s' d p) \leq \text{mbal } (\text{dblock } s' p) \)
\( \land \text{bal } (\text{disk } s' d p) \leq \text{bal } (\text{dblock } s' p) \)
by(auto simp add: \text{StartBallot-def})
with \( \text{Dmaj} \)
show \( \exists d \in D. \ mbal (\text{disk } s' d p) \leq \text{mbal } (\text{dblock } s' p) \)
\( \land \text{bal } (\text{disk } s' d p) \leq \text{bal } (\text{dblock } s' p) \)
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 $H_{Inv4a}$ $s'$ $p$

using act inv inv2a

proof

- from act
  have phase: $0 < \text{phase } s \ p$
    by(auto simp add: StartBallot-def)

- from 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 act: $H_{StartBallot} \ s \ s' \ p$
and inv: $H_{Inv4a} \ s \ q$
and pnq: $p \neq q$

shows $H_{Inv4a} \ s' \ q$

proof

- from act pnq
  have blocksOf $s' \ q \subseteq \text{blocksOf } s \ q$
    by(auto simp add: StartBallot-def InitializePhase-def
          blocksOf-def rdBy-def)

  with act inv pnq
  show ?thesis
    by(auto simp add: StartBallot-def HInv4a-def
          HInv4a1-def HInv4a2-def)

qed

theorem HStartBallot-HInv4a:

assumes act: $H_{StartBallot} \ s \ s' \ p$
and inv: $H_{Inv4a} \ s \ q$
and inv2a: $Inv2a \ s'\$0

shows $H_{Inv4a} \ s' \ q$

proof(cases $p = q$)

- case True
  from inv2a
  have $Inv2a$-$inner \ s' \ p$
    by(auto simp add: Inv2a-def)

  with act inv True
  show ?thesis
    by(blast dest: HStartBallot-HInv4a-p)

next

- case False
  with act inv
  show ?thesis
    by(blast dest: HStartBallot-HInv4a-q)

qed

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

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

theorem HPhase1or2Write-HInv4a:
assumes act: HPhase1or2Write s s' p d
and inv: HInv4a s q
shows HInv4a s' q
proof
from act
have phase': phase s = phase s'
  by (simp add: Phase1or2Write-def)
show \?thesis
proof (cases phase s q = 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

lemma HPhase1or2ReadThen-HInv4a1-p:
assumes act: HPhase1or2ReadThen s s' p d q
and inv: HInv4a1 s p
shows HInv4a1 s' p
proof (auto simp: HInv4a1-def)
fix bk
assume bk: bk \in blocksOf s' p
with HPhase1or2ReadThen-blocksOf[OF act]
have bk \in 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:
[ HPhase1or2ReadThen s s' p d r; HInv4a2 s q ] \implies HInv4a2 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
and
shows \( HInv4a1 \) \( s' \) \( p \)
proof(auto simp add: HInv4a1-def)

fix \( bk \)

assume \( bk: bk \in \blocksOf s' p \)

from \( bk \) \( HEndPhase1 \)-blocksOf[OF act]
have \( bk \in \{\dblock s' p\} \cup \blocksOf s p \)
  by blast

with \( act \) \( inv \)
show \( \bal bk \leq \mbal (\dblock s' p) \)
  by(auto simp add: HInv4a-def HInv4a1-def EndPhase1-def)
qed

lemma \( HEndPhase1-HInv4a2 \):
assumes \( act: HEndPhase1 s s' p \)
and \( inv: HInv42 s p \)
and \( inv2a: Inv2a s \)
shows \( HInv4a2 s' p \)
proof(auto simp add: HInv4a2-def)

fix \( D \)

assume \( Dmaj: D \in \MajoritySet \)

from \( inv \) \( Dmaj \)
have \( \exists d \in D. \ \mbal (\disk s d p) \leq \mbal (\dblock s p) \)
  \( \land \ \bal (\disk s d p) \leq \bal (\dblock s p) \)
  by(auto simp add: HInv4a2-def)

then obtain \( d \)
  where \( d-cond: d \in D \)
  \( \land \ \mbal (\disk s d p) \leq \mbal (\dblock s p) \)
  \( \land \ \bal (\disk s d p) \leq \bal (\dblock s p) \)
  by(auto simp add: blocksOf-def)

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 \( d \in D \)
  \( \land \ \mbal (\disk s' d p) \leq \mbal (\dblock s' p) \)
  \( \land \ \bal (\disk s' d p) \leq \bal (\dblock s' p) \)
  by(auto simp add: EndPhase1-def)

with \( Dmaj \)
show \( \exists d \in D. \ \mbal (\disk s' d p) \leq \mbal (\dblock s' p) \)
  \( \land \ \bal (\disk s' d p) \leq \bal (\dblock s' p) \)
  by(auto)
qed

lemma \( HEndPhase1-HInv4a-p \):
assumes \( act: HEndPhase1 s s' p \)
and \( inv: HInv4a s p \)
and \( inv2a: Inv2a s \)

61
shows $HInv4a_s' p$

proof –

from act
have phase: $0 < \text{phase } s p$
  by (auto simp add: EndPhase1-def)
with act inv inv2a
show ?thesis
  by (auto simp del: HEndPhase1-def simp add: HInv4a-def
       elim: HEndPhase1-HInv4a1 HEndPhase1-HInv4a2)
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 $s' q = \text{dblock } s q \land \text{disk } s' = \text{disk } s$
  by (auto simp add: EndPhase1-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: EndPhase1-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 blocksOf $s' q \subseteq \text{blocksOf } s q$
  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** \( \text{HEndPhase2-HInv4a} \):

\[
\left[ \text{HEndPhase2 } s \ s' \ p; \ \text{HInv4a } s \ q \right] \implies \text{HInv4a } s' \ q
\]

by (auto simp add: EndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)

**lemma** \( \text{allSet} \):

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

shows \( Q \ rb \)

proof –

from \( aPQ \) have \( \forall r \in P \ d. \ Q r \) by auto

with \( rb \)

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

qed

**lemma** \( \text{EndPhase0-44} \):

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

and \( \text{bk}: \ bk \in \text{blocksOf } s \ p \)

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 \( \text{bk inv4d} \) 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 \( \forall d. \ \forall rb \in \text{blocksRead } s \ p \ d. \ \text{block } rb = \text{disk } s \ d \ p \) — 5.1

by (simp add: Inv2c-inner-def)

hence \( \forall d. \ \text{hasRead } s \ p \ d \)

\[ \rightarrow (\exists rb \in \text{blocksRead } s \ p \ d. \ \text{block } rb = \text{disk } s \ d \ p) \] — 5.2

(is \( \forall d. \ ?H d \rightarrow ?P d \)

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 \( \text{?thesis} \)

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 \( \text{?thesis} \)

by force

qed
lemma \( H\text{EndPhase0-}H\text{Inv4a1-p} \):
assumes act: \( H\text{EndPhase0 s s'} \)
and inv2a': \( \text{Inv2a s'} \)
and inv2c: \( \text{Inv2c-inner s p} \)
and inv4d: \( H\text{Inv4d s p} \)
shows \( H\text{Inv4a1 s'} \)

\begin{proof}
  (auto simp add: \( H\text{Inv4a1-def} \))
  fix \( bk \)
  assume \( bk \in \text{blocksOf s p} \)
  with \( H\text{EndPhase0-blocksOf[OF act]} \)
  have \( bk \in \{ \text{dblock s'} \} \cup \text{blocksOf s p} \) by auto
  thus \( \text{bal bk} \leq \text{mbal (dblock s'} \) by auto
  proof
    assume \( bk: bk \in \{ \text{dblock s'} \} \)
    with inv2a'
    have \( \text{Inv2a-innermost s'} \) by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
    with \( bk \) show \( \text{?thesis} \)
    by (auto simp add: Inv2a-innermost-def)
  next
    assume \( bk: bk \in \text{blocksOf s p} \)
    from act
    have \( \forall r \in \text{allBlocksRead s p}. \text{mbal r < mbal (dblock s'} \) by (auto simp add: 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: EndPhase0-44)
    with \( f1 \) show \( \text{?thesis} \)
    by (auto simp add: EndPhase0-def allBlocksRead-def allRdBlks-def dest: allSet)
  qed
\end{proof}

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

\begin{proof}
  fix \( d \)
  assume \( d: d \in \{ \text{d. hasRead s p d} \} \) (is \( d \in \{ \text{d} \})
  hence \( \text{br-ne: blocksRead s p d} \neq \{ \})
  by (auto simp add: hasRead-def)
  from inv2c phase
  have \( \forall br \in \text{blocksRead s p d}. \text{block br = disk s d p} \)
  by (auto simp add: Inv2c-inner-def)
  with \( \text{br-ne} \)
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 ∃ D∈MajoritySet. ∀ 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: ∃ D∈MajoritySet. ∀ d∈D. P d
  shows ∀ D∈MajoritySet. ∃ 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: ∀ x∈D1. P x
  from D1 D2 majorities-intersect
  have ∃ d∈D1. d∈D2 by auto
  with Px
  show ∃ x∈D2. P x
    by auto
qed

lemma HEndPhase0-HInv4a2-p:
  assumes act: HEndPhase0 s s’ p
  and inv1: Inv1 s
and \( inv2c: \text{Inv2c-inner s p} \)
shows \( H\text{Inv4a2} s' p \)
proof\((\text{simp add: HInv4a2-def})\)
from act
have disk': disk s' = disk s
  by\((\text{simp add: EndPhase0-def})\)
from act inv1 inv2c
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \ mbal(disk s d p) \leq mbal(dblock s' d p) \)
  \& \( bal(disk s d p) \leq bal(dblock s' d p) \)
  by\((\text{blast dest: HEndPhase0-41})\)
from Majority-exQ[\( OF \ this \)]
have \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal(disk s d p) \leq mbal(dblock s' d p) \)
  \& \( bal(disk s d p) \leq bal(dblock s' d p) \)
(is ?P (disk s)) .
from subst[\( OF \ disk', of ?P, OF this \)]
show \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal(disk s' d p) \leq mbal(dblock s' d p) \)
  \& \( bal(disk s' d p) \leq bal(dblock s' d p) \).
qed

lemma HEndPhase0-HInv4a-p:
assumes act: HEndPhase0 s s' p
and \( \text{inv2a}: \text{Inv2a s} \)
and \( \text{inv2}: \text{Inv2c s} \)
and \( \text{inv4d}: \text{HInv4d s p} \)
and \( \text{inv1}: \text{Inv1 s} \)
and \( \text{inv}: \text{HInv4a s p} \)
shows \( \text{HInv4a s' p} \)
proof –
from inv2
have inv2c: Inv2c-inner s p
  by\((\text{auto simp add: Inv2c-def})\)
with inv1 inv2a act
have inv2a': Inv2a s'
  by\((\text{blast dest: HEndPhase0-Inv2a})\)
from act
have phase s' p = 1
  by\((\text{auto simp add: EndPhase0-def})\)
with act inv inv2c inv4d inv2a' inv1
show ?thesis
  by\((\text{auto simp add: HInv4a-def simp del: HEndPhase0-def elim: HEndPhase0-HInv4a1-p HEndPhase0-HInv4a2-p})\)
qed

lemma HEndPhase0-HInv4a-q:
assumes act: HEndPhase0 s s' p
and \( \text{inv}: \text{HInv4a s q} \)
and \( \text{pnq}: p \neq q \)
shows \( \text{HInv4a s' q} \)
proof –
from act pnq
have dblock $s' q = dblock s q \land disk s' = disk s$
  by (auto simp add: EndPhase0-def)
moreover
from act pnq
have \(\forall p d. \ rdBy s' q p d \subseteq \ rdBy s q p d\)
  by (auto simp add: EndPhase0-def InitializePhase-def rdBy-def)
hence \((UN p d. \ rdBy s' q p d) \subseteq (UN p d. \ rdBy s q p d)\)
  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: EndPhase0-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

theorem HEndPhase0-HInv4a:
[ [ HEndPhase0 s s' p; HInv4a s q; HInv4d s p; 
  Inv2a s; Inv1 s; Inv2a s; Inv2c s ] ]
\Rightarrow HInv4a s' q
by (blast dest: HEndPhase0-HInv4a-p HEndPhase0-HInv4a-q)

C.4.2 Proofs of Invariant 4b

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

lemma HEndPhase0-dblock-mbal:
[ [ HEndPhase0 s s' p ] ]
\Rightarrow \forall br \in \text{allBlocksRead } s p. \ \text{mbal } br < \text{mbal(dblock } s' p) 
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 \(\text{bal(dblock } s' p) < \text{mbal(dblock } s' p)\)
proof –
  from act have phase s p = 0 by (auto simp add: EndPhase0-def)
  with inv2c
  have \(\forall d. \forall br \in \text{blocksRead } s p d. \ \text{proc } br = p \land \text{block } br = \text{disk } s d p\)
    by (auto simp add: Inv2c-inner-def)
hence allBlks-in-blocksOf: \(\text{allBlocksRead } s p \subseteq \text{blocksOf } s p\)
  by (auto simp add: allBlocksRead-def allRdBlks-def blocksOf-def)
  from act HEndPhase0-some[OF act inv1]
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 ?thesis
  by auto
qed

lemma HEndPhase0-HInv4b-p-blocksOf:
  assumes act: HEndPhase0 \, s \, s' \, p
  and inv4d: HInv4d \, s \, p
  and inv2c: Inv2c-inner \, s \, p
  and bk: bk \, \in \, \text{blocksOf} \, s \, p
  shows bal \, bk \, < \, \text{mbal} \,(\text{dblock} \, s \, p) 
proof –
  from inv4d majorities-intersect bk
  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)
  have \( \exists \, br \in \text{allBlocksRead} \, s \, p \, . \, \text{bal} \,(\text{dblock} \, s \, p) \leq \text{mbal} \, br \)
    by auto
  with p43
  have \( \exists \, br \in \text{allBlocksRead} \, s \, p \, . \, \text{bal} \,(\text{dblock} \, s \, p) \leq \text{mbal} \, br \)
    by auto
  with maj
  show ?thesis
    by (auto simp add: MajoritySet-def)
qed

show ?thesis
  by auto
qed
lemma \textit{HEndPhase0-HInv4b-p}:
assumes act: \textit{HEndPhase0} \(s\ s'\ p\)
and inv4d: \textit{HInv4d} \(s\ p\)
and inv1: \textit{Inv1} \(s\)
and inv2a: \textit{Inv2a} \(s\)
and inv2c: \textit{Inv2c-inner} \(s\ p\)
shows \textit{HInv4b} \(s'\ p\)
proof(\textit{clarsimp simp add: HInv4b-def})
from act
have phase: \textit{phase} \(s\ p = 0\)
  by(\textit{auto simp add: EndPhase0-def})
fix bk
assume bk: \(bk \in \textit{blocksOf} \ s'\ p\)
with \textit{HEndPhase0-blocksOf[OF act]}
have \((bk \in \textit{dblock} \ s'\ p) \lor (bk \in \textit{blocksOf} \ s\ p)\)
  by blast
thus bal bk < mbal (\textit{dblock} \ s'\ p)
proof
  assume bk: \(bk \in \textit{dblock} \ s'\ p\)
  with act inv1 inv2a inv2c
  show \(\textit{thesis}\)
    by(\textit{auto simp del: HEndPhase0-def dest: HEndPhase0-HInv4b-p-dblock})
next
  assume bk: \(bk \in \textit{blocksOf} \ s\ p\)
  with act inv2c inv4d
  show \(\textit{thesis}\)
    by(\textit{blast dest: HEndPhase0-HInv4b-p-blocksOf})
qed
qed

lemma \textit{HEndPhase0-HInv4b-q}:
assumes act: \textit{HEndPhase0} \(s\ s'\ p\)
and pnq: \(p \neq q\)
and inv: \textit{HInv4b} \(s\ q\)
shows \textit{HInv4b} \(s'\ q\)
proof –
from act pnq
have disk': \textit{disk} \(s' = \textit{disk} \ s\)
  and dblock': \textit{dblock} \(s' \ q = \textit{dblock} \ s \ q\)
  and phase': \textit{phase} \(s' \ q = \textit{phase} \ s \ q\)
  by(\textit{auto simp add: EndPhase0-def})
from act pnq
have blocksRead': \(\forall q. \textit{allRdBlks} \ s' \ q \subseteq \textit{allRdBlks} \ s \ q\)
  by(\textit{auto simp add: EndPhase0-def InitializePhase-def allRdBlks-def})
with disk' dblock'
have blocksOf' \(s' \ q \subseteq \textit{blocksOf} \ s \ q\)
  by(\textit{auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast})
with inv phase' dblock'

69
\text{show} \ ?\text{thesis} \\
\quad \text{by} (\text{auto simp add: HInv4b-def}) \\
\text{qed}

\text{theorem} \ H\text{EndPhase0-HInv4b}:
\begin{align*}
\text{assumes} & \quad \text{act}: H\text{EndPhase0} \ s \ s' \ p \\
\text{and} & \quad \text{inv}: H\text{Inv4b} \ s \ q \\
\text{and} & \quad \text{inv4d}: H\text{Inv4d} \ s \ p \\
\text{and} & \quad \text{inv1}: \text{Inv1} \ s \\
\text{and} & \quad \text{inv2a}: \text{Inv2a} \ s \\
\text{and} & \quad \text{inv2c}: \text{Inv2c-inner} \ s \ p \\
\text{shows} & \quad H\text{Inv4b} \ s' \ q
\end{align*}
\text{proof} (cases \ p=q) \\
\text{case} \ True \\
\text{with} \ H\text{EndPhase0-HInv4b-p}[OF \ \text{act} \ \text{inv4d} \ \text{inv1} \ \text{inv2a} \ \text{inv2c}] \\
\text{show} \ ?\text{thesis} \ \text{by simp}
\text{next} \\
\text{case} \ False \\
\text{from} \ H\text{EndPhase0-HInv4b-q}[OF \ \text{act} \ \text{False} \ \text{inv}] \\
\text{show} \ ?\text{thesis} . \\
\text{qed}

\text{lemma} \ H\text{StartBallot-HInv4b-p}:
\begin{align*}
\text{assumes} & \quad \text{act}: H\text{StartBallot} \ s \ s' \ p \\
\text{and} & \quad \text{inv2a}: \text{Inv2a-innermost} \ s \ p \ (\text{dblock} \ s \ p) \\
\text{and} & \quad \text{inv4b}: H\text{Inv4b} \ s \ p \\
\text{and} & \quad \text{inv4a}: H\text{Inv4a} \ s \ p \\
\text{shows} & \quad H\text{Inv4b} \ s' \ p
\end{align*}
\text{proof} (clarsimp simp add: HInv4b-def) \\
\text{fix} \ bk \\
\text{assume} \ bk: bk \in \text{blocksOf} \ s' \ p \\
\text{from} \ \text{act} \\
\text{have} \ \text{phase'}: \text{phase} \ s' \ p = 1 \\
\quad \text{and} \ \text{phase}: \text{phase} \ s \ p \in \{1,2\} \\
\quad \text{by} (\text{auto simp add: StartBallot-def}) \\
\text{from} \ \text{act} \\
\text{have} \ p42: \ m\text{bal} \ (\text{dblock} \ s \ p) < m\text{bal} \ (\text{dblock} \ s' \ p) \\
\quad \wedge \ \text{bal} (\text{dblock} \ s \ p) = \text{bal} (\text{dblock} \ s' \ p) \\
\quad \text{by} (\text{auto simp add: StartBallot-def}) \\
\text{from} \ H\text{StartBallot-blocksOf}[OF \ \text{act}] \ bk \\
\text{have} \ bk \in \{\text{dblock} \ s' \ p\} \cup \text{blocksOf} \ s \ p \\
\quad \text{by blast} \\
\text{thus} \ \text{bal} \ bk < m\text{bal} \ (\text{dblock} \ s' \ p) \\
\text{proof} \\
\text{assume} \ bk: bk \in \{\text{dblock} \ s' \ p\} \\
\text{from} \ \text{inv2a} \\
\text{have} \ \text{bal} (\text{dblock} \ s \ p) \leq m\text{bal} \ (\text{dblock} \ s \ p) \\
\quad \text{by} (\text{auto simp add: Inv2a-innermost-def}) \\
\text{with} \ p42 \ bk
show \( ? \)thesis by auto

next

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

from phase inv4a

have p41: \( H\text{Inv4a1 } s \ p \)

by (auto simp add: HInv4a-def)

with p42 bk

show \( ? \)thesis

by (auto simp add: HInv4a1-def)

qed

lemma \( H\text{StartBallot-HInv4b-q} \):

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

and pnq: \( p \neq q \)

and inv: \( H\text{Inv4b } s \ q \)

shows \( H\text{Inv4b } s \ s' \ q \)

proof

from act pnq

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

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

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

by (auto simp add: \text{StartBallot-def})

from act pnq

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

by (auto simp add: \text{StartBallot-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)

with inv phase' dblock'

show \( ? \)thesis

by (auto simp add: HInv4b-def)

qed

theorem \( H\text{StartBallot-HInv4b} \):

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

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

and inv4b: \( H\text{Inv4b } s \ q \)

and inv4a: \( H\text{Inv4a } s \ p \)

shows \( H\text{Inv4b } s' q \)

using act inv2a inv4b inv4a

proof (cases \( p = q \))

case True

from inv2a

have inv2a-innermost s p (dblock s p)

by (auto simp add: inv2a-def inv2a-inner-def blocksOf-def)

with act True inv4b inv4a

show \( ? \)thesis

by (blast dest: HStartBallot-HInv4b-p)
next
  case False
  with act inv4b
  show ?thesis
    by (blast dest: HStartBallot-HInv4b-q)
qed

theorem HPhase1or2Write-HInv4b:
  [ HPhase1or2Write s s' p d; HInv4b s q ] \implies HInv4b s' q
by (auto simp add: Phase1or2Write-def HInv4b-def
    blockOf-def rdBy-def)

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

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

theorem HPhase1or2ReadThen-HInv4b:
  [ HPhase1or2ReadThen s s' p d q; HInv4b s r ] \implies HInv4b s' r
by (blast dest: HPhase1or2ReadThen-HInv4b-p
    HPhase1or2ReadThen-HInv4b-q)

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

lemma HEndPhase1-HInv4b-p:
  HEndPhase1 s s' p \implies HInv4b s' p
by (auto simp add: EndPhase1-def HInv4b-def)

lemma HEndPhase1-HInv4b-q:
  assumes act: HEndPhase1 s s' p
  and pnq: p \neq 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: EndPhase1-def)
from act pnq
have blocksRead′: ∀ q. allRdBlks s′ q ⊆ allRdBlks s q
by (auto simp add: EndPhase1-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′
show ?thesis
by (auto simp add: HInv4b-def)
qed

theorem HEndPhase1-HInv4b:
assumes act: HEndPhase1 s s′ p
and inv: HInv4b s q
shows HInv4b s′ q
proof (cases p = q)
case True
with HEndPhase1-HInv4b-p [OF act]
show ?thesis by simp
next
case False
from HEndPhase1-HInv4b-q [OF act False inv]
show ?thesis .
qed

lemma HEndPhase2-HInv4b-p:
HEndPhase2 s s′ p → HInv4b s′ p
by (auto simp add: EndPhase2-def HInv4b-def)

lemma HEndPhase2-HInv4b-q:
assumes act: HEndPhase2 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: EndPhase2-def)
from act pnq

73
have \( \text{blocksRead}' : \forall q. \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
by (auto simp add: EndPhase2-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 \(?\text{thesis}\)
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 \(?\text{thesis}\) by simp

next
  case False
  from \( \text{HEndPhase2-HInv4b-q[OF act False inv]} \)
  show \(?\text{thesis}\).

qed

lemma \( \text{HFail-HInv4b-p} \):
\( \text{HFail} s s' p \rightarrow \text{HInv4b} s s' p \)
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
  from \( \text{act \ pnq} \)
  have \( \text{disk'}: \text{disk} s'=\text{disk} s \)
  and \( \text{dblock'}: \text{dblock} s' q=\text{dblock} s q \)
  and \( \text{phase'}: \text{phase} s' q=\text{phase} s q \)
  by (auto simp add: Fail-def)

  from \( \text{act \ pnq} \)
  have \( \text{blocksRead'}: \forall q. \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
  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 \(?\text{thesis}\)
  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 pq: p ≠ q
  and inv: HInv4b s q
  shows HInv4b s' q
proof (cases)
  from act pq
  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 \texttt{HStartBallot-HInv4c-p}:
\[\begin{array}{c}
\text{HStartBallot } s \ s' \ p ; \ HInv4c \ s \ p \\
\Rightarrow \ HInv4c \ s' \ p
\end{array}\]
by (auto simp add: StartBallot-def HInv4c-def)

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

theorem \texttt{HStartBallot-HInv4c}:
\[\begin{array}{c}
\text{HStartBallot } s \ s' \ p ; \ HInv4c \ s \ q \\
\Rightarrow \ HInv4c \ s' \ q
\end{array}\]
by (blast dest: \texttt{HStartBallot-HInv4c-p} \texttt{HStartBallot-HInv4c-q})

lemma \texttt{HPhase1or2Write-HInv4c-p}:
assumes act: \texttt{HPhase1or2Write } s \ s' \ p \ d
and inv: \texttt{HInv4c } s \ p
and inv2c: \texttt{Inv2c } s
shows \texttt{HInv4c } s' \ p
proof (cases phase s' \ p = 2)
assume phase': \texttt{phase' } s' \ p = 2
show ?thesis
proof (auto simp add: HInv4c-def phase' MajoritySet-def)
from act phase'
have bal: \texttt{bal } (\texttt{dblock } s' \ p) = \texttt{bal } (\texttt{dblock } s \ p)
and phase: \texttt{phase } s \ p = 2
by (auto simp add: Phase1or2Write-def)
from phase' inv2c act
have mbal (\texttt{disk } s' \ d \ p) = \texttt{mbal } (\texttt{dblock } s \ p)
by (auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
with bal
have \texttt{bal } (\texttt{dblock } s' \ p) = \texttt{mbal } (\texttt{disk } s' \ d \ p)
by auto
with inv phase act
show \ \exists D. \ \texttt{IsMajority } D \wedge (\forall d \in D. \ \texttt{mbal } (\texttt{disk } s' \ d \ p) = \texttt{bal } (\texttt{dblock } s' \ p))
by (auto simp add: HInv4c-def Phase1or2Write-def MajoritySet-def)
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 \neq 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: \forall 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:
  \[ [ HPhase1or2Write s s' p d; HInv4c s q; Inv2c s ] \implies HInv4c s' q \]
  by (blast dest: HPhase1or2Write-HInv4c-p
       HPhase1or2Write-HInv4c-q)

lemma HPhase1or2ReadThen-HInv4c-p:
  \[ [ HPhase1or2ReadThen s s' p d q; HInv4c s p ] \implies HInv4c s' p \]
  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 \neq 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:
\[ \text{HPhase1or2ReadThen } s \ s' \ p \ d \ r ; \text{ HInv4c } s \ q \]
\[ \implies \text{ HInv4c } s' \ q \]
by (blast dest: HPhase1or2ReadThen-HInv4c-p HPhase1or2ReadThen-HInv4c-q)

theorem HPhase1or2ReadElse-HInv4c:
\[ \text{HPhase1or2ReadElse } s \ s' \ p \ d \ r ; \text{ HInv4c } s \ q \]
\[ \implies \text{ HInv4c } s' \ q \]
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: Is Majority \( \{ d. \ d \in \text{disksWritten } s \ p \ \wedge (\forall q \in (\text{UNIV} - \{ p \}). \text{ hasRead } s \ p \ d \ q) \} \)
is Is Majority \( \exists M \)
by (simp add: EndPhase1-def)
from inv2b
have \( \forall d \in \exists M. \text{ disk } s \ d \ p = \text{ dblock } s \ p \)
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 \( \neq \) 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:
\[ \text{HEndPhase1 } s \ s' \ p ; \text{ HInv4c } s \ q ; \text{ Inv2b } s \]
\[ \implies \text{ HInv4c } s' \ q \]
by (blast dest: HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q)
lemma \(\text{HEndPhase2-HInv4c-p:} \)
\[
\begin{align*}
& [ \text{HEndPhase2 } s \; s' \; p; \; \text{HInv4c } s \; p ] \implies \text{HInv4c } s' \; p
\end{align*}
\]
by(auto simp add: EndPhase2-def HInv4c-def)

lemma \(\text{HEndPhase2-HInv4c-q:} \)
assumes act: \(\text{HEndPhase2 } s \; s' \; p\)
and inv: \(\text{HInv4c } s \; q\)
and \(\text{pnq: } p \neq q\)
shows \(\text{HInv4c } s' \; q\)
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: EndPhase2-def)
with inv
show \(?\text{thesis}\)
by(auto simp add: HInv4c-def)
qed

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

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

lemma \(\text{HFail-HInv4c-q:} \)
assumes act: \(\text{HFail } s \; s' \; p\)
and inv: \(\text{HInv4c } s \; q\)
and \(\text{pnq: } p \neq q\)
shows \(\text{HInv4c } s' \; q\)
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: Fail-def)
with inv
show \(?\text{thesis}\)
by(auto simp add: HInv4c-def)
qed

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

lemma \(\text{HPhase0Read-HInv4c-p:} \)
lemma \( HPhase0Read-HInv4c-q \):

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

proof -

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: Phase0Read-def)
with inv
show \(?\)thesis
by(auto simp add: HInv4c-def)

qed

theorem \( HPhase0Read-HInv4c \):

\[ [ \text{HPhase0Read } s s' p d; \text{HInv4c } s q ] \implies \text{HInv4c } s' q \]
by(blast dest: HPhase0Read-HInv4c-p HPhase0Read-HInv4c-q)

lemma \( HEndPhase0-HInv4c-p \):

\[ [ \text{HEndPhase0 } s s' p; \text{HInv4c } s p ] \implies \text{HInv4c } s' p \]
by(auto simp add: EndPhase0-def)

lemma \( HEndPhase0-HInv4c-q \):

assumes act: \( HEndPhase0 s s' p \)
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: EndPhase0-def)
with inv
show \(?\)thesis
by(auto simp add: HInv4c-def)

qed

theorem \( HEndPhase0-HInv4c \):

\[ [ \text{HEndPhase0 } s s' p; \text{HInv4c } s q ] \implies \text{HInv4c } s' q \]
by(blast dest: HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q)
C.4.4 Proofs of Invariant 4d

**lemma** \(H\text{StartBallot-HInv4d-}\text{p}\):
**assumes** \(\text{act}: H\text{StartBallot } s s' p\)
**and inv**: \(H\text{inv4d } s p\)
**shows** \(H\text{inv4d } s' p\)
**proof**(clarsimp simp add: \(H\text{inv4d-def}\))
fix \(bk\)
assume \(bk\): \(bk \in \text{blocksOf } s' p\)
from \(\text{act}\)
have \(\text{bal'': } \text{bal (dblock } s' p) = \text{bal (dblock } s p)\)
  by(auto simp add: \(\text{StartBallot-def}\))
from \(\text{subsetD[OF } H\text{StartBallot-blocksOf[OF } \text{act } \text{bk}]\)
have \(\exists D \in \text{MajoritySet. } \forall d \in D. \text{ bal } bk \leq \text{mbal (disk } s d p)\)
**proof**
assume \(bk\): \(bk \in \text{blocksOf } s p\)
with \(\text{inv}\)
show \(\text{thesis}\)
  by(auto simp add: \(H\text{inv4d-def}\))
next
assume \(bk\): \(bk \in \{\text{dblock } s' p\}\)
with \(\text{bal'}\ \text{inv}\)
show \(\text{thesis}\)
  by(auto simp add: \(H\text{inv4d-def blocksOf-def}\))
qed
with \(\text{act}\)
show \(\exists D \in \text{MajoritySet. } \forall d \in D. \text{ bal } bk \leq \text{mbal (disk } s' d p)\)
  by(auto simp add: \(\text{StartBallot-def}\))
qed

**lemma** \(H\text{StartBallot-HInv4d-q}\):
**assumes** \(\text{act}: H\text{StartBallot } s s' p\)
**and inv**: \(H\text{inv4d } s q\)
**and pnq**: \(p \neq q\)
**shows** \(H\text{inv4d } s' q\)
**proof**
from \(\text{act pnq}\)
have \(\text{disk': disk } s' = \text{disk } s\)
  and \(\text{dblock': dblock } s' q = \text{dblock } s q\)
  by(auto simp add: \(\text{StartBallot-def}\))
from \(\text{act pnq}\)
have \(\text{blocksRead': } \forall q. \text{ allRdBlks } s' q \subseteq \text{allRdBlks } s q\)
  by(auto simp add: \(\text{StartBallot-def InitializePhase-def allRdBlks-def}\))
with \(\text{disk' dblock'}\)
have \(\text{blocksOf } s' q \subseteq \text{blocksOf } s q\)
  by(auto simp add: \(\text{allRdBlks-def blocksOf-def rdBy-def, blast}\))
from \(\text{subsetD[OF } \text{this } \text{inv}\)
have \(\forall bk \in \text{blocksOf } s' q\)
  \(\exists D \in \text{MajoritySet. } \forall d \in D. \text{ bal } bk \leq \text{mbal (disk } s d q)\)
  by(auto simp add: \(H\text{inv4d-def}\))

81
with \( \text{disk}' \)
show \(?\text{thesis}\)
by (auto simp add: \text{HInv4d-def})
qed

**Theorem** HStartBallot-HInv4d:
\[
[HStartBallot \ s \ s' \ p; \ HInv4d \ s \ q] \implies HInv4d \ s' \ q
\]
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: \( bk \in \text{blocksOf} \ s \ p \)
from act have ddisk: \( \forall \ d. \ \text{disk} \ s' \ d \ p = \) (if \( d = d \)
then \( \text{dblock} \ s \ p \)
else \( \text{disk} \ s \ d \ p \))
and phase: \( \text{phase} \ s \ p \neq 0 \)
by (auto simp add: HPhase1or2Write-def)
from inv subsetD[OF HPhase1or2Write-blocksOf[OF act] bk]
have asm3: \( \exists D \in \text{MajoritySet} \ . \ \forall dd \in D. \ \text{bal} \ bk \leq \text{mbal} \) (disk \( s \ d \ d \ p \))
by (auto simp add: HInv4d-def)
from phase inv4a subsetD[OF HPhase1or2Write-blocksOf[OF act] bk] ddisk
have p41: \( \text{bal} \ bk \leq \text{mbal} \) (disk \( s' \ d \ p \))
by (auto simp add: HInv4a-def HInv4a1-def)
with ddisk asm3
show \( \exists D \in \text{MajoritySet} . \ \forall dd \in D. \ \text{bal} \ bk \leq \text{mbal} \) (disk \( s' \ d \ d \ p \))
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: \( p \neq q \)
shows HInv4d \( s' \ q \)
proof –
from act pnq
have disk': \( \forall d. \ \text{disk} \ s' \ d \ q = \text{disk} \ s \ d \ q \)
by (auto simp add: HPhase1or2Write-def)
from act pnq
have blocksRead': \( \forall q. \ \text{allRdBlks} \ s' \ q \subseteq \text{allRdBlks} \ s \ q \)
by (auto simp add: HPhase1or2Write-def
InitializePhase-def allRdBlks-def)
with act pnq
have blocksOf \( s' \ q \subseteq \text{blocksOf} \ s \ q \)

82
by (auto simp add: Phase1or2Write-def allRdBlks-def blocksOf-def rdBy-def)
from subsetD[OF this] inv
have \( \forall bk \in \text{blocksOf } s' q. \exists D \in \text{MajoritySet}. \forall d \in D. \text{bal } bk \leq \text{mbal} (\text{disk } s \ d \ q) \)
  by (auto simp add: HInv4d-def)
with disk'
show ?thesis
by (auto simp add: HInv4d-def)
qed

theorem HPhase1or2Write-HInv4d:
[ HPhase1or2Write s s' p d; HInv4d s q; HInv4a s p ] \implies 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': bal (\text{dblock } s' p) = 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': disk s' = disk s
  by (auto simp add: Phase1or2ReadThen-def)
from act pnq
have blocksOf s' q \subseteq 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 ] ⇒ HInv4d s' q
by (blast dest: HPhase1or2ReadThen-HInv4d-p
HPhase1or2ReadThen-HInv4d-q)

theorem HPhase1or2ReadElse-HInv4d:
[ HPhase1or2ReadElse s s' p d r; HInv4d s q ] ⇒ 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 ∈ blocksOf s' p
from HEndPhase1-HInv4c[OF act inv4c inv2b]
have HInv4c s' p ,
with act
have p31: "∃ D ∈ MajoritySet. ∀ d ∈ 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 "∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s' d p)"
proof
assume bk: bk ∈ blocksOf s p
with inv disk'
show ?thesis
  by (auto simp add: HInv4d-def)
next
assume bk: bk ∈ {dblock s' p}
with p31
show ?thesis
  by force
qed
qed

lemma HEndPhase1-HInv4d-q:
assumes act: HEndPhase1 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: EndPhase1-def)
from act pnq
have blocksRead': \( \forall q \cdot \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
  by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
with disk'
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 \( \forall \text{bk} \in \text{blocksOf} s' q \). \exists D \in \text{MajoritySet}. \forall d \in D. \text{bal} \text{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:
[ HEndPhase1 s s' p; HInv4d s q; Inv2b s; HInv4c s p ]
\implies HInv4d s' q
  by (blast dest: HEndPhase1-HInv4d-p HEndPhase1-HInv4d-q)

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

lemma HEndPhase2-HInv4d-q:
assumes act: HEndPhase2 s s' p
and inv: HInv4d s q
and png: p ≠ q
shows HInv4d s' q

proof –
from act png
have disk': disk s' = disk s
  by(auto simp add: EndPhase2-def)
from act png
have blocksOf s' q ⊆ blocksOf s q
  by(auto simp add: EndPhase2-def InitializePhase-def
      allRdBlks-def blocksOf-def rdBy-def)
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 HEndPhase2-HInv4d:
[ HEndPhase2 s s' p; HInv4d s q ] ⇒ HInv4d s' q
by(blast dest: HEndPhase2-HInv4d-p HEndPhase2-HInv4d-q)

lemma HFail-HInv4d-p:
assumes act: HFail s s' p
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 disk': disk s' = disk s
  by(auto simp add: Fail-def)
from subsetD[OF HFail-blocksOf[OF act] bk]
show ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal(disk s' d p)
proof
assume bk: bk ∈ {dblock s' p}
with inv disk'
show ?thesis
  by(auto simp add: HInv4d-def)
next
assume bk: bk ∈ {dblock s' p}
with act
have bal bk = 0
  by(auto simp add: Fail-def InitDB-def)
with Disk-isMajority
show ?thesis
  by(auto simp add: MajoritySet-def)
lemma \textit{HFail-HInv4d-q}:
assumes \textit{act}: \textit{HFail} s s' p
and \textit{inv}: \textit{HInv4d} s q
and \textit{pq} \not= q
shows \textit{HInv4d} s' q
proof –
from \textit{act} \textit{pq}
have \textit{disk'}: \textit{disk} s' = disk s
and \textit{dblock'}: \textit{dblock} s' q = dblock s q
by (auto simp add: \textit{Fail-def})
from \textit{act} \textit{pq}
have \textit{blocksRead'}: \forall q. \textit{allRdBlks} s' q \subseteq \textit{allRdBlks} s q
by (auto simp add: \textit{Fail-def} \textit{InitializePhase-def} \textit{allRdBlks-def})
with \textit{disk'} \textit{dblock'}
have \textit{blocksOf} s' q \subseteq \textit{blocksOf} s q
by (auto simp add: \textit{Phase0Read-def} \textit{blocksOf-def} \textit{rdBy-def}, blast)
from \textit{subsetD}[OF this] \textit{inv}
have \forall bk \in \textit{blocksOf} s' q.
\exists D \in \textit{MajoritySet}. \forall d \in D. \textit{bal} bk \leq \textit{mbal} (\textit{disk} s d q)
by (auto simp add: \textit{HInv4d-def})
with \textit{disk'}
show \textit{?thesis}
by (auto simp add: \textit{HInv4d-def})
qed

theorem \textit{HFail-HInv4d}:
\[
\begin{array}{c}
\textit{HFail} s s' p; \textit{HInv4d} s q \\
\end{array} \implies \textit{HInv4d} s' q
\]
by (blast dest: \textit{HFail-HInv4d-p} \textit{HFail-HInv4d-q})

lemma \textit{HPhase0Read-HInv4d-p}:
assumes \textit{act}: \textit{HPhase0Read} s s' p d
and \textit{inv}: \textit{HInv4d} s p
shows \textit{HInv4d} s' p
proof (clarsimp simp add: \textit{HInv4d-def})
fix \textit{bk}
assume \textit{bk}: \textit{bk} \in \textit{blocksOf} s' p
from \textit{act}
have \textit{bal'}: \textit{bal} (\textit{dblock} s' p) = \textit{bal} (\textit{dblock} s p)
by (auto simp add: \textit{Phase0Read-def})
from \textit{subsetD}[OF \textit{HPhase0Read-blocksOf}[OF \textit{act}] \textit{bk}] \textit{inv}
have \exists D \in \textit{MajoritySet}. \forall d \in D. \textit{bal} \textit{bk} \leq \textit{mbal} (\textit{disk} s d p)
by (auto simp add: \textit{HInv4d-def})
with \textit{act}
show \exists D \in \textit{MajoritySet}. \forall d \in D. \textit{bal} \textit{bk} \leq \textit{mbal} (\textit{disk} s' d p)
by (auto simp add: \textit{Phase0Read-def})
qed
lemma HPhase0Read-HInv4d-q:
  assumes act: HPhase0Read s s′ p d
  and inv: HInv4d s q
  and pnq: p\neq q
  shows HInv4d s′ q
proof –
  from act pnq
  have disk′: disk s′=disk s
    by (auto simp add: Phase0Read-def)
  from act pnq
  have blocksOf s′ q \subseteq blocksOf s q
    by (auto simp add: Phase0Read-def allRdBlks-def
                  blocksOf-def rdBy-def)
  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 HPhase0Read-HInv4d:
  [ HPhase0Read s s′ p d; HInv4d s q ] \implies HInv4d s′ q
by (blast dest: HPhase0Read-HInv4d-p HPhase0Read-HInv4d-q)

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

lemma HEndPhase0-HInv4d-p:
  assumes act: HEndPhase0 s s′ p
  and inv: HInv4d s p
  and inv2c: Inv2c s
  and inv1: Inv1 s
  shows HInv4d s′ p
proof(clarsimp simp add: HInv4d-def)
  fix bk

88
assume \( bk: bk \in \text{blocksOf } s' p \)

from \( \text{subsetD}[OF \ HEndPhase0\text{-blocksOf}[OF \ text{act}] \ bk] \)

have \( \exists D\in \text{MajoritySet}. \forall d\in D. \ bal \ bk \leq \text{mbal}(disk \ s \ d \ p) \)

proof

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

with \( \text{inv} \)

show \( ?\text{thesis} \)

by\( (\text{auto simp add: } H\text{inv4d-def}) \)

next

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

from \( \text{inv2c} \)

have \( \text{inv2c-inner}: \text{Inv2c-inner } s \ p \)

by\( (\text{auto simp add: Inv2c-def}) \)

from \( bk \ H\text{EndPhase0-some}[OF \ text{act inv1}] \)

\( H\text{EndPhase0-blocksOf}[OF \ text{act inv2c-inner}] \text{act} \)

have \( \text{bal } bk \in \text{bal }'(\text{blocksOf } s \ p) \)

by\( (\text{auto simp add: EndPhase0-def}) \)

with \( \text{inv} \)

show \( ?\text{thesis} \)

by\( (\text{auto simp add: H\text{inv4d-def}}) \)

qed

with \( \text{act} \)

show \( \exists D\in \text{MajoritySet}. \forall d\in D. \ bal \ bk \leq \text{mbal}(disk \ s' \ d \ p) \)

by\( (\text{auto simp add: EndPhase0-def}) \)

qed

lemma \( H\text{EndPhase0-H\text{inv4d-q}}: \)

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

and \( \text{inv}: H\text{inv4d } s \ q \)

and \( \text{pnq}: p \neq q \)

shows \( H\text{inv4d } s' \ q \)

proof

from \( \text{act pnq} \)

have \( \text{dblock } s' \ q = \text{dblock } s \ q \land \text{disk } s' = \text{disk } s \)

by\( (\text{auto simp add: EndPhase0-def}) \)

moreover

from \( \text{act pnq} \)

have \( \forall p \ d. \ rdBy \ s' \ q \ p \ d \subseteq rdBy \ s \ q \ p \ d \)

by\( (\text{auto simp add: EndPhase0-def } \text{InitializePhase-def } \text{rdBy-def}) \)

hence \( (UN \ p \ d. \ rdBy \ s' \ q \ p \ d) \subseteq (UN \ p \ d. \ rdBy \ s \ q \ p \ d) \)

by\( (\text{auto, blast}) \)

ultimately

have \( \text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q \)

by\( (\text{auto simp add: blocksOf-def, blast}) \)

from \( \text{subsetD}[OF \ this] \ text{inv} \)

have \( \forall bk\in\text{blocksOf } s' \ q. \)

\( \exists D\in \text{MajoritySet}. \forall d\in D. \ bal \ bk \leq \text{mbal}(disk \ s \ d \ q) \)

by\( (\text{auto simp add: H\text{inv4d-def}}) \)

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

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

definition HInv5-inner-R :: state ⇒ Proc ⇒ bool 
  where 
  HInv5-inner-R $s$ $p$ = 
    ($\forall$ $bk$$\in$$allBlocks$ $s$, $b$ $\leq$ $bal$ $bk$ $\rightarrow$ $inp$ $bk$ $=$ $v$) 
    $\forall$ ($\exists$ $D$$\in$MajoritySet. $\exists$ $q$. ($\forall$ $d$$\in$$D$. $bal$($dblock$ $s$ $p$) $<$ $mbal$($disk$ $s$ $d$ $q$) 
        $\land$$\neg$hasRead $s$ $p$ $d$ $q$))

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

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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

theorem \( HInit-HInv5 \): \( HInit \ s \Rightarrow HInv5 \ s \)
    using Disk-isMajority
    by (auto simp add: 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 \( \text{action-}HInv5\text{-p} \) and \( \text{action-}HInv5\text{-q} \) for each action, for the cases \( p = q \) and \( p \neq q \) respectively.

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

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

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

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

92
show \(?\)thesis
by auto
next
case False
from \(x\text{-}nblks\) subsetD[OF \(H\text{StartBallot}\)-blocksOf-q[OF act False] \(x\text{-}pa\)]
show \(?\)thesis
by auto
qed
qed

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

lemma \(H\text{StartBallot}\)-HInv5-q2:
assumes \(\text{act}: \ H\text{StartBallot} s s' p\)
and \(\text{pnq}: p \neq q\)
and \(\text{inv5-2}: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d qq)\)
\(\wedge \neg \text{hasRead} s q d qq)\)
shows \(\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s' d qq)\)
\(\wedge \neg \text{hasRead} s' q d qq)\)
proof
from act pnq
have \(\text{disk}: \text{disk} s' = \text{disk} s\)

93
and blocksRead: \( \forall d. \) blocksRead \( s' q d = \text{blocksRead} \ s q d \)
and dblock: \( d\text{block} s' q = \text{dblock} s q \)
by (auto simp add: StartBallot-def InitializePhase-def)

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

lemma HStartBallot-HInv5-q:
assumes act: HStartBallot \( s s' p \)
and inv: HInv5-inner \( s q \)
and pnq: \( p \neq q \)
shows 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 HStartBallot-HInv5:
\( [\ ] \ HStartBallot \ s s' p; HInv5-inner \ s q \] \( \Rightarrow \) HInv5-inner \( s' q \)
by (blast dest: HStartBallot-HInv5-q HStartBallot-HInv5-p)

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

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

proof
from inv5-2
obtain \( D \ q \)
where i1: IsMajority \( D \)
and i2: \( \forall d \in D. \ \text{bal}(\text{dblock} \ s p) < \text{mbal}(\text{disk} \ s d q) \)
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 \( D1 \) \( \land \) \( \forall d \in D1. \ \text{mbal}(\text{disk} \ s d p) = \text{bal} (\text{dblock} \ s p) \)

94
by (auto simp add: HInv4c Def MajoritySet-def)
with i1 majorities-intersect
have $D \cap D^1 \neq \{\} \text{ by auto}
then obtain $dd$ where $dd \in D \cap D^1$
  by auto
with i1 i2 r1
have $bal(dblock s p) < mbal(disk s dd q) \land mbal(disk s dd p) = bal(dblock s p)$
  by auto
thus $?thesis$ by auto
qed

from act pnq
— dblock and hasRead do not change
have $dblock s' = dblock s$
  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 $\forall d \in D. \text{bal} (dblock s' p) < \text{mbal} (disk s' d q) \land \neg \text{hasRead s'} p d q$
  by auto
with i1
show $?thesis$
  by (auto simp add: MajoritySet-def)
qed

lemma HPhase1or2Write-HInv5-p:
assumes act: $HPhase1or2Write s s' p d$
and inv: $HInv5-inner s p$
and inv4: $HInv4c s p$
shows $HInv5-inner s' p$
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': $phase s' p = 2$
and i2: $\forall D \in \text{MajoritySet}. \forall q. \exists d \in D. \text{bal} (dblock s' p) < \text{mbal} (disk s' d q)$
  $\rightarrow \text{hasRead s'} p d q$
with act have phase: $phase s p = 2$
  by (auto simp add: Phase1or2Write-def)
show maxBalImp 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 maxBalImp 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: $HPhase1or2Write s s' p d$
shows allBlocks s' $\subseteq$ allBlocks s
using HPhase1or2Write-blocksOf [OF act]
by (auto simp add: allBlocks-def)
lemma \texttt{HPhase1or2Write-HInv5-q2:}
assumes \texttt{act: HPhase1or2Write s s' p d}
and \texttt{pnq: p \neq q}
and \texttt{inv4a: HInv4a s p}
and \texttt{inv5-2: \exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \tb{bal(dblock s q) < mbal(disk s d qq) \wedge \neg \text{hasRead s q d qq})}}
shows \texttt{\exists D \in \text{MajoritySet}. \exists q q. (\forall d \in D. \tb{bal(dblock s' q) < mbal(disk s' d qq) \wedge \neg \text{hasRead s' q d qq})}}

proof –
from \texttt{inv5-2}
obtain D q q
where \texttt{i1: IsMajority D}
and \texttt{i2: \forall d \in D. \tb{bal(dblock s q) < mbal(disk s d qq) \wedge \neg \text{hasRead s q d qq}}}
and \texttt{i3: \forall d \in D. \tb{\neg \text{hasRead s q d qq}}}
by (auto simp add: MajoritySet-def)
from act pnq
— \texttt{dblock and hasRead do not change}
have \texttt{dblock': dblock s' = dblock s}
and \texttt{hasread: \forall d. \text{hasRead s' q d qq = hasRead s q d qq}}
by (auto simp add: Phase1or2Write-def hasRead-def)
have \texttt{\forall d \in D. \tb{bal(dblock s' q) < mbal(disk s' d qq) \wedge \neg \text{hasRead s' q d qq}}}
proof (cases q q=p)
case True
have \texttt{bal(dblock s q) < mbal(dblock s p)}
proof –
from \texttt{inv4a act i1}
have \texttt{\exists d \in D. \tb{mbal(disk s d p) \leq mbal(dblock s p)}}
by (auto simp add: MajoritySet-def HInv4a-def HInv4a2-def Phase1or2Write-def)
with True i2
show \texttt{bal(dblock s q) < mbal(dblock s p)}
by auto
qed
with \texttt{hasread dblock' True i1 i2 i3 act}
show \texttt{?thesis}
by (auto simp add: Phase1or2Write-def)
next
case False
with \texttt{act i2 i3}
show \texttt{?thesis}
by (auto simp add: Phase1or2Write-def hasRead-def)
qed
with \texttt{i1}
show \texttt{?thesis}
by (auto simp add: MajoritySet-def)
qed
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. ∀q. ∃d∈D. bal (dblock s' q) < mbal (disk s' d q)
→ hasRead s' q d q a
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;
HInv5c s p; HInv4a s p ] → HInv5-inner s' q
by (blast dest: HPhase1or2Write-HInv5-p)

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: Phase1or2ReadThen-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 \( \text{bal}(\text{dblock} \ s \ p) = \text{mbal}(\text{dblock} \ s \ p) \)
by (auto simp add: \text{Inv2c-inner-def})

moreover

from \( \text{act} \) have \( \text{mbal}(\text{disk} \ s \ d \ r) < \text{mbal}(\text{dblock} \ s \ p) \)
by (auto simp add: \text{Phase1or2ReadThen-def})

moreover

from \( i2 \) have \( d \in D \rightarrow \text{bal}(\text{dblock} \ s \ p) < \text{mbal}(\text{disk} \ s \ d \ q) \) by auto

ultimately have \( \text{pnr}: \ d \in D \rightarrow q \neq r \) by auto
have \( \text{pnq}: p \neq q \)

proof –
from \( \text{inv4c phase} \)

obtain \( D1 \) where \( \text{r1: IsMajority} \ D1 \land (\forall d \in D1. \text{mbal}(\text{disk} \ s \ d \ p) = \text{bal}(\text{dblock} \ s \ p)) \)
by (auto simp add: \text{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 \( ?\text{thesis} \) by auto

qed

from \( \text{pnr act} \)
have \( \text{hasRead}': \forall d \in D. \text{hasRead}' \ p \ d \ q = \text{hasRead} \ s \ p \ d \ q \)
by (auto simp add: \text{Phase1or2ReadThen-def hasRead-def})

from \( \text{act pnq} \)
— \( \text{dblock and disk do not change} \)
have \( \text{dblock} \ s' = \text{dblock} \ s \)
and \( \forall d. \text{disk} \ s' = \text{disk} \ s \)
by (auto simp add: \text{Phase1or2ReadThen-def})

with \( i2 \) \( \text{hasRead}' \ i3 \)
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 \( ?\text{thesis} \)
by (auto simp add: \text{MajoritySet-def})

qed

lemma \( \text{HPhase1or2ReadThen-HInv5-p} \):\nassumes \( \text{act}: \text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r \)
and \( \text{inv}: \text{HInv5-inner} \ s \ p \)
and \( \text{inv4}: \text{HInv4c} \ s \ p \)
and \( \text{inv2c}: \text{Inv2c} \ s \)
shows \( \text{HInv5-inner} \ s' \ p \)

proof (auto simp add: \text{HInv5-inner-def \ HInv5-inner-R-def})
assume \( \text{phase}'': \text{phase} \ s' \ p = 2 \)
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: Phase1or2ReadThen-def)
show maxBallInp s' (bal (dblock s' p)) (inp (dblock s' p))
proof (rule HPhase1or2ReadThen-HInv5-1[OF act, of p])
  from inv2e
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  from HPhase1or2ReadThen-HInv5-p2[OF act inv4 this phase] inv i2 phase
  show maxBallInp 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: \( \forall D \in \text{MajoritySet}. \forall qa. \exists d\in D. \text{bal (dblock s' q)} < m\text{bal (disk s' d qa)} \)
\( \rightarrow \) 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:
[ [ HPhase1or2ReadThen s s' p d r; HInv5-inner s q; HInv4c s p; HInv4a s p ] ] \( \Rightarrow \) HInv5-inner s' q
by(blast dest: HPhase1or2ReadThen-HInv5-1 HPhase1or2ReadThen-HInv5-p)

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

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

lemma HEndPhase2-allBlocks:
assumes act: HEndPhase2 s s' p
shows allBlocks s' \( \subseteq \) 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 \( \in \) allBlocks s'
and bal: bal (dblock s' q) \( \leq \) bal bk

qed
from \text{act} \ \text{pnq}

\text{have } \text{dblock'} s' q = \text{dblock} s q \ \text{by}(\text{auto simp add: EndPhase2-def})

\text{from } \text{subsetD}[OF \text{HEndPhase2-allBlocks}[OF \text{act}] \text{ bk}] \text{ inv5-1 \text{dblock'}} \text{ bal}

\text{show } \text{inp} \text{ bk} = \text{inp} (\text{dblock} s' q)

\quad \text{by}(\text{auto simp add: maxBalInp-def})

\text{qed}

\text{lemma } \text{HEndPhase2-HInv5-q2}: \ \text{assumes} \ \text{act}: \text{HEndPhase2} s s' p

\quad \text{and } \text{pnq}: p \neq q

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

\quad \quad \wedge \neg \text{hasRead} s d qq)

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

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

\text{proof --}

\text{from } \text{act} \ \text{pnq}

\text{have } \text{disk}: \text{disk} s' = \text{disk} s

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

\quad \text{and } \text{dblock}: \text{dblock} s' q = \text{dblock} s q

\quad \text{by}(\text{auto simp add: EndPhase2-def InitializePhase-def})

\text{with } \text{inv5-2}

\text{show } \text{thesis}

\quad \text{by}(\text{auto simp add: hasRead-def})

\text{qed}

\text{lemma } \text{HEndPhase2-HInv5-q}: \ \text{assumes} \ \text{act}: \text{HEndPhase2} s s' p

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

\quad \text{and } \text{pnq}: p \neq q

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

\text{using } \text{assms} \ \text{and } \text{HEndPhase2-HInv5-q1}[OF \text{act} \text{ pnq}] \text{ HEndPhase2-HInv5-q2}[OF \text{act} \text{ pnq}]

\quad \text{by}(\text{auto simp add: HInv5-inner-def HInv5-inner-R-def EndPhase2-def})

\text{theorem } \text{HEndPhase2-HInv5}:

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

\quad \text{by}(\text{blast dest: HEndPhase2-HInv5-q HEndPhase2-HInv5-q})

\text{lemma } \text{HEndPhase1-HInv5-p}: \ \text{assumes} \ \text{act}: \text{HEndPhase1} s s' p

\quad \text{and } \text{inv4}: \text{HInv4} s

\quad \text{and } \text{inv2a}: \text{Inv2a} s

\quad \text{and } \text{inv2b}: \text{Inv2a s'}

\quad \text{and } \text{inv2c}: \text{Inv2c s}

\quad \text{and } \text{asm4}: \neg \text{maxBalInp} s' (\text{bal}(\text{dblock} s' p)) (\text{inp}(\text{dblock} s' p))

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

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

\text{proof --}

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

proof –
from asm4
obtain bk
  where p31: \( bk \in \text{allBlocks } s' \land \text{bal}(\text{dblock } s' \ p) \leq \text{bal } bk \land bk \neq \text{dblock } s' \ p \)
  by(auto simp add: maxBalImp-def)
then obtain q where p32: \( bk \in \text{blocksOf } s' \ q \)
  by(auto simp add: allBlocks-def)
from act
have dblock: \( p \neq q =\Rightarrow \text{dblock } s' \ q = \text{dblock } s \ q \)
  by(auto simp add: EndPhase1-def)
have bk \( \in \text{blocksOf } s \ q \)
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: \( bk \in \text{allBlocks } s \land \text{bal}(\text{dblock } s \ p) \leq \text{bal } bk \land bk \neq \text{dblock } s \ p \)
by(auto)
have \( \exists q \in \text{UNIV} \setminus \{p\}. \ bk \in \text{blocksOf } s \ q \)
proof –
from p22
obtain q where \( bk : bk \in \text{blocksOf } s \ q \)
  by(auto simp add: allBlocks-def)
from act p22
have mbal(dblock s p) \( \leq \) 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 \( bk \)

102
show \text{thesis} \\
by auto
\text{qed}

then obtain \( q \) where \( p_{23}: q \in \text{UNIV} - \{p\} \land bk \in \text{blocksOf} \ s \ q \)
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 \( p_{23} \ \text{inv4} \)
  have \( \text{inv4d}: \exists D \in \text{MajoritySet} \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ q) \)
    by (auto simp add: HInv4-def HInv4d-def)
  from \( \text{inv4d} \ p_{22} \)
  show \text{thesis} \by force
\text{qed}

then obtain \( D \) where \( D_{maj}: D \in \text{MajoritySet} \) and \( p_{24}: (\forall d \in D. \ \text{bal} (\text{dblock} \ s' \ p) \leq \text{mbal} (\text{disk} \ s \ d \ q) \)
by auto
have \( p_{25}: (\forall d \in D. \ \text{bal} (\text{dblock} \ s' \ p) < \text{mbal} (\text{disk} \ s \ d \ q) \)
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 \( \text{bal} (\text{dblock} \ s' \ p) \in \text{Ballot} \ p \cup \{b\} \)
    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 \( \forall d \in D. \ \text{mbal} (\text{disk} \ s \ d \ q) \in \text{Ballot} \ q \cup \{b\} \)
    by (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with \( p_{24} \) \( \text{bal-pos} \)
  have \( \forall d \in D. \ \text{mbal} (\text{disk} \ s \ d \ q) \in \text{Ballot} \ q \)
    by force
  with \( \text{Ballot-disj} \ p_{23} \) \( \text{bal-in-p} \)
  have \( \forall d \in D. \ \text{mbal} (\text{disk} \ s \ d \ q) \neq \text{bal} (\text{dblock} \ s' \ p) \)
    by force
  with \( p_{23} \ p_{24} \)
  show \text{thesis} \by force
\text{qed

with \( \text{p23} \) \( \text{act} \)
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 \( \text{Dmaj} \)
show ?thesis
  by blast
qed

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\#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 blks: \{blocksRead s' q = blocksRead s q\}
  by(auto simp add: EndPhase1-def InitializePhase-def)
  from disk
  have disk': \{disk s' d q | d . d\in UNIV\} \subseteq \{disk s d q | 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 | br . br \in (\UN qq d . rdBy s' q qq d)\} \subseteq \{block br | 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]
show \(
\text{thesis}
\)
by auto
next
  case False
  from \(x\text{-nblks} \text{ subsetD}[OF \text{ HEndPhase1-blocsOf-q}[OF \text{ act False}] \text{ x-pa}]
\)
show \(
\text{thesis}
\)
by auto
qed

lemma \(\text{HEndPhase1-HInv5-q}:
\)
assumes \(\text{act: HEndPhase1 s s' p}
\)
and \(\text{inv: HInv5 s}
\)
and \(\text{inv1: Inv1 s}
\)
and \(\text{inv2a: Inv2a s'}
\)
and \(\text{inv2a-q: Inv2a s}
\)
and \(\text{inv2b: Inv2b s}
\)
and \(\text{inv2c: Inv2c s}
\)
and \(\text{inv3: HInv3 s}
\)
and \(\text{phase': phase s' q = 2}
\)
and \(\text{pnq: p \neq q}
\)
and \(\text{asm4\': \sim \text{maxBalInp s'} (bal(dblock s' q)) (inp(dblock s' q))}
\)
shows \((\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{ bal(dblock s' q) < mbal(disk s' d qq)\)}
\)
\(\wedge \sim \text{hasRead s' q d qq}))
\)
proof –
  from \(\text{act pnq}
\)
  have \(\text{phase s' q = phase s q}
\)
  and \(\text{phase-p: phase s p = 1}
\)
  and \(\text{disk: disk s' = disk s}
\)
  and \(\text{dblock: dblock s' q = dblock s q}
\)
  and \(\text{bal: bal(dblock s' p) = mbal(dblock s p)}
\)
  by(auto simp add: \text{EndPhase1-def InitializePhase-def})
  with \(\text{phase'}
\)
  have \(\text{phase: phase s q = 2 by auto}
\)
  from \(\text{phase inv2c}
\)
  have \(\text{bal-dblk-q: bal(dblock s q) \in \text{Ballot q}}
\)
  by(auto simp add: \text{Inv2c-def Inv2c-inner-def})
  have \(\exists D \in \text{MajoritySet. } \exists qq. (\forall d \in D. \text{ bal(dblock s q) < mbal(disk s d qq)\)}
\)
\(\wedge \sim \text{hasRead s q d qq}))
  proof(cases \text{maxBalInp s (bal(dblock s q)) (inp(dblock s q))})
  case True
  have \(p21: \text{bal(dblock s q) < bal(dblock s' p) \wedge inp(dblock s q) \neq inp(dblock s' p)}
\)
  proof –
    from \(\text{True asm4 dblock HEndPhase1-allBlocks[OF act]}
\)
    have \(p32: \text{ bal(dblock s q) \leq bal(dblock s' p)}
\)
    \(\wedge \text{ inp(dblock s q) \neq inp(dblock s' p)}
\)
    by(auto simp add: \text{maxBalInp-def})
    from \(\text{inv2a}
\)
\)
have \( \text{bal} (\text{dblock} \ s' \ p) \in \text{Ballot} \ p \cup \{\emptyset\} \)
by (auto simp add: Inv2a-def Inv2a-inner-def
Inv2a-innermost-def blocksOf-def)

moreover from Ballot-disj Ballot-nzero pnq
have Ballot q \cap (\text{Ballot} \ p \cup \{\emptyset\}) = \{\}
by auto
ultimately have \( \text{bal} (\text{dblock} \ s' \ p) \neq \text{bal} (\text{dblock} \ s \ q) \)
using bal-dblk-q
by auto
with p32
show ?thesis
by auto
qed

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

then obtain \( D \)
where p22: \( D \in \text{MajoritySet} \wedge (\forall \ d \in \ D. \ \text{bal}(\text{dblock} \ s \ q) < \text{mbal}(\text{disk} \ s \ d \ p) \wedge \text{hasRead} \ s \ p \ d \ q) \)
by auto
have p23: \( \forall \ d \in \ D. (\text{block} = \text{dblock} \ s \ q, \ \text{proc} = q) \notin \text{blocksRead} \ s \ p \ d \)
proof –
have \( \text{dblock} \ s \ q \in \text{allBlocksRead} \ s \ p \rightarrow \text{inp}(\text{dblock} \ s' \ p) = \text{inp}(\text{dblock} \ s \ q) \)
proof auto
assume dblock-q: \( \text{dblock} \ s \ q \in \text{allBlocksRead} \ s \ p \)
from inv2a-q
have \((\text{bal}(\text{dblock } s \ q) = 0) \Rightarrow (\text{inp}(\text{dblock } s \ q) = \text{NotAnInput})\)

by \((\text{auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def Inv2a-innermost-def})\)

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 \((\text{auto simp add: nonInitBlks-def blocksSeen-def})\)

with \(\text{act}\)

have \(\text{dblock-max}: \text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{maxBlk } s \ p)\)

by \((\text{auto simp add: EndPhase1-def})\)

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 \((\text{auto simp add: allBlocks-def nonInitBlks-def blocksSeen-def blocksOf-def rdBy-def allBlocksRead-def allRdBlks-def})\)

with \(\text{True subsetD[OF this max-in-nib]}\)

have \(\text{max-in-nib}\)

by \(\text{auto}\)

qed

have \(\text{p21}: \forall d \in D. \neg(\exists br \in \text{blocksRead } s \ q \ d. \text{bal}(\text{dblock } s \ q) \leq \text{mbal}(\text{block } br))\)

proof

−

from \(\text{inv2c phase}\)

have \(\forall d. \forall br \in \text{blocksRead } s \ q \ d. \text{mbal}(\text{block } br) \bowtie \text{mbal}(\text{dblock } s \ q)\)

and \(\text{bal}(\text{dblock } s \ q) = \text{mbal}(\text{dblock } s \ q)\)

by \((\text{auto simp add: Inv2c-def Inv2c-inner-def})\)

thus \(?\text{thesis}\)

by \(\text{force}\)

qed

have \(\text{p24}: \forall d \in D. \neg(\exists \text{hasRead } s \ q \ d \ p)\)

proof

auto

fix \(d\)

assume \(\text{d-in-D}: d \in D\)

and \(\text{hasRead-qdp}: \text{hasRead } s \ q \ d \ p\)

have \(\text{p31}: (\exists \text{block } s \ p, \text{proc } p) \in \text{blocksRead } s \ q \ d\)

proof

−

from \(\text{d-in-D p22}\)

107
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]
lemma HFail-HInv5-p:
  HFail s s' p \implies HInv5-inner s' p
by(auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-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 \not\in blocksOf s xa
show x=dblock s' p
proof(cases p=pa)
  case True
    from x-nblks
    have x \not\in blocksOf s p
    by auto
  with True subsetD[OF HFail-blocksOf-q[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

lemma HFail-HInv5-q1:
  assumes act: HFail s s' p
  and pnq: p \neq q
  and inv2a: Inv2a-inner s' q
  and inv5-1: maxBalImp s (bal(dblock s q)) (inp(dblock s q))
  shows maxBalImp s' (bal(dblock s' q)) (inp(dblock s' q))
proof(auto simp add: maxBalImp-def)
fix bk
assume bk: bk \in allBlocks s'
  and bal: bal (dblock s' q) \leq bal bk

qed
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

lemma HFail-HInv5-q2:
  assumes act: HFail 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 blocksRead: ∀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-inner 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: ∀D∈MajoritySet.
    ∀qa. ∃d∈D. bal (dblock s' q) < mbal (disk s' d qa) → hasRead s' q d qa (is ?P s')
from HFail-HInv5-q2[OF act pnq]
have ¬ (?P s) ===> ¬ (?P s')
  by auto
with nR2
have P: ?P s
  by blast
from inv2a
have inv2a': Inv2a-inner s' q by (auto simp add: Inv2a-def)
from act pnq phase'
have phase s q = 2
  by (auto simp add: Fail-def split: if-split-asm)
with inv HFail-HInv5-q1[OF act pnq inv2a'] P
show maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q)) (is ?P s')
  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' ] ==> HInv5-inner s' q
by (blast dest: HFail-HInv5-q HFail-HInv5-p)

lemma HPhase0Read-HInv5-p:
HPhase0Read s s' p d ==> 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' ⊆ 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-HInv5-2:
assumes act: HPhase0Read s s' p d
and pnq: p≠q
and inv5-2: ∃D∈MajoritySet. ∃qq. (∀d∈D. bal(dblock s q) < mbal(disk s d qq))
    ∧ ~hasRead s q d qq)
shows $\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ (\text{bal}(\text{dblock} s' q) < \text{mbal} (\text{disk} s' d q)) \\
\land \neg \text{hasRead} s' q d qq)$

proof –
from act pnq
have disk: 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: Phase0Read-def InitializePhase-def)
with inv5-2
show $\exists$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: $\forall D \in \text{MajoritySet}. \forall qa. \exists d \in D. \ (\text{bal} (\text{dblock} s' q) < \text{mbal} (\text{disk} s' d qa)) \\
\rightarrow \text{hasRead} s' q d qa$
from phase' act have phase: phase $s q = 2$
  by(auto simp add: Phase0Read-def)
show maxBalInp $s' (\text{bal} (\text{dblock} s' q)) (\text{inp} (\text{dblock} s' q))$
proof(rule HPhase0Read-HInv5-1[OF act, of q])
from HPhase0Read-HInv5-q2[OF act pnq] inv i2 phase
show maxBalInp $s (\text{bal} (\text{dblock} s q)) (\text{inp} (\text{dblock} s q))$
  by(auto simp add: Hinv5-inner-def Hinv5-inner-R-def, blast)
qed
qed

theorem HPhase0Read-HInv5:
  ![HPhase0Read $s s' p d; \text{HInv5-inner} s q \Rightarrow \text{HInv5-inner} s' q]
by(blast dest: HPhase0Read-HInv5-q HPhase0Read-HInv5-p)

lemma HEndPhase0-HInv5-p:
HEndPhase0 $s s' p \Rightarrow \text{HInv5-inner} s' p$
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 $s' q \subseteq \text{blocksOf} s q$
proof –
from act pnq
have dblock: $\{\text{dblock} s' q\} \subseteq \{\text{dblock} s q\}$
  and disk: disk $s' = \text{disk} s$
and \( \text{blks: blocksRead } s' q = \text{blocksRead } s q \)

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

from disk

have \( \text{disk'}: \{ \text{disk } s' d q | d \in \text{UNIV} \} \subseteq \{ \text{disk } s d q | d \in \text{UNIV} \} \) (is \( ?D' \subseteq ?D \))
  by auto

from pnq act

have \( \{ \text{block } br | \text{block } br \in (\text{UN } qq d. \text{rdBy } s' q qq d) \} \subseteq \{ \text{block } br | \text{block } br \in (\text{UN } qq d. \text{rdBy } s q qq d) \} \)
  (is \( ?R' \subseteq ?R \))
  by auto blast

from union-inclusion[OF dblock union-inclusion[OF disk' this]]

show \( \text{thesis} \)
  by (auto simp add: blocksOf-def)

qed

lemma \( \text{HEndPhase0-allBlocks} \):

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

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

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[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 \( \text{HEndPhase0-HInv5-q1} \):

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

and \( \text{pnq: } p \neq q \)

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

and \( \text{inv5-1: maxBalInp } s (\text{bal}(\text{dblock } s q)) (\text{inp}(\text{dblock } s q)) \)
shows $\maxBalInp s' (\text{bal}(\text{dblock } s' q)) (\text{inp}(\text{dblock } s' q))$

proof (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 $\text{thesis}$
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 allBlocksRead $s p \subseteq \text{allBlocks } s$
by (auto simp add: allBlks-def allRdBlks-def)

from subsetD[OF this ba-blksread ba-balinp]
show $\text{thesis}$
by (auto simp add: maxBalInp-def)

qed

qed

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

proof
from act pnq
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 $\text{thesis}$
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 pnq: p≠q
  shows HInv5-inner s' q
  using assms and
  HEndPhase0-HInv5-q1 [OF act pnq inv1]
  HEndPhase0-HInv5-q2 [OF act pnq]
  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-q 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: HInv2-def intro: 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(ν). This predicate is true if ν 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). 
  maxBallInp s b v 
  ∧ (∃ p. ∃ D∈MajoritySet.(∀ d∈D. b ≤ bal(disk s d p) 
      ∧ (∀ q.( phase s q = 1 
        ∧ b ≤ mbal(dblock s q) 
        ∧ hasRead s q d p 
      ) → (∃ br∈blocksRead s q d. b ≤ bal(block br)))))
)

lemma HEndPhase1-valueChosen-inp: 
assumes act: HEndPhase1 s s' q 
and inv2a: Inv2a s 
and asm1: b ∈ (UN p. Ballot p) 
and bk-blocksOf: bk∈blocksOf s r 
and bk: bk∈ blocksSeen s q 
and b-bal: b ≤ bal bk 
and asm3: maxBallInp 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 ∃ d. maxBlk s q ∈ blocksSeen s p 
  by(auto simp add: nonInitBlks-def blocksSeen-def) 
  hence ∃ d. 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: maxBallInp-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: maxBallnp 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
  next
  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

117
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 ∈ blocksRead s q d
  by(auto simp add: blocksRead-def allRdBlks-def)
from HEndPhase1-valueChosen-mp[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-mp[OF act inv2a asm1 db-blksof db-bseen
b-bal asm3 inv1]
  show ?thesis .
qed
with \textit{asm3} HEndPhase1-allBlocks\{OF \textit{act}\n
show \textit{?thesis}
by(auto simp add: maxBalInp-def)
qed

next

\textbf{case} False

\textbf{have} dblock s' q \in allBlocks s'
by(auto simp add: allBlocks-def blocksOf-def)

show \textit{?thesis}
proof(auto simp add: maxBalInp-def)
fix \textit{bk}
\textbf{assume} bk: \textit{bk} \in allBlocks \textit{s'}
\textbf{and} \textit{b-bal}: b \leq bal \textit{bk}
from \textit{subsetD}(OF HEndPhase1-allBlocks\{OF \textit{act}\} \textit{bk})

show \textit{inp bk} = \textit{v}
proof
\textbf{assume} \textit{bk}: \textit{bk} \in allBlocks \textit{s}
\textbf{with} \textit{asm3} \textit{b-bal}
\textbf{show} \textit{?thesis}
by(auto simp add: maxBalInp-def)

next
\textbf{assume} \textit{bk}: \textit{bk} \in \{dblock \textit{s'} \textit{q}\}
from \textit{act} False
\textbf{have} \neg b \leq bal (dblock \textit{s'} \textit{q})
by(auto simp add: EndPhase1-def)
\textbf{with} \textit{bk} \textit{b-bal}
\textbf{show} \textit{?thesis}
by(auto)
qed
qed

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

\textbf{shows} \textit{?P} \textit{s'}
proof(auto)
fix \textit{d}
\textbf{assume} \textit{d}: \textit{d} \in \textit{D}
\textbf{with} \textit{act} \textit{asm4}
\textbf{show} b \leq bal (\textit{disk} \textit{s'} \textit{d} \textit{p})
by(auto simp add: EndPhase1-def)
fix \textit{d} \textit{q}
\textbf{assume} \textit{d}: \textit{d} \in \textit{D}
\& \textit{phase'}: \textit{phase} \textit{s'} \textit{q} = \textit{Suc 0}

119
and `dblk-mbal`: \( b \leq \text{mbal}(\text{dblock } s' q) \)

with `act`
have `p31`: `phase s q = 1`
  and `p32`: `dblock s' q = dblock s q`
  by (auto simp add: `EndPhase1-def` split: `if-split-asm`)
with `dblk-mbal`
have `b \leq \text{mbal}(\text{dblock } s q)` by auto
moreover
assume `hasRead`: `hasRead s' q d p`
with `act`
have `hasRead s q d p`
  by (auto simp add: `EndPhase1-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: `EndPhase1-def` `InitializePhase-def` `hasRead-def`)
qed

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

**Proof** –
from `vc`

obtain `b p D` where
  `asm1`: `b \in (\text{UN } p. \text{Ballot } p)`
  and `asm2`: `D \in \text{MajoritySet}`
  and `asm3`: `maxBalInp s b v`
  and `asm4`: `\forall d \in D. b \leq \text{bal}(\text{disk } s d p)`
  \( \wedge (\forall q. (\text{phase } s q = 1 \wedge b \leq \text{mbal}(\text{dblock } s q) \wedge \text{hasRead } s q d p) \longrightarrow (\exists br \in \text{blocksRead } s q d. b \leq \text{bal}(\text{block } br)))) \)
  by (auto simp add: `valueChosen-def`)
from `HEndPhase1-maxBalInp[\text{OF } act \text{ asm1} \text{ asm2} \text{ asm3} \text{ asm4} \text{ inv1} \text{ inv2a} \text{ inv2b}]`
have `maxBalInp s' b v`.
with `HEndPhase1-valueChosen2[\text{OF } act \text{ asm4}]` `asm1` `asm2`
show `?thesis`
  by (auto simp add: `valueChosen-def`)
qed
lemma \( H_{\text{StartBallot-maxBalInp}} \):
assumes \( \text{act} : H_{\text{StartBallot}} s s' q \)
and \( \text{asm3} : \text{maxBalInp} s b v \)
shows \( \text{maxBalInp} s' b v \)
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 HStartBallot-allBlocks[OF act] bk]
show \( \text{inp} bk = v \)
proof
assume \( bk : bk \in \text{allBlocks} s \)
with \( \text{asm3} b-\text{bal} \)
show \( \text{thesis} \)
by (auto simp add: maxBalInp-def)
next
assume \( bk : bk \in \{ \text{dblock} s' q \} \)
from \( \text{asm3} \)
have \( b \leq \text{bal} (\text{dblock} s q) \Rightarrow \text{inp}(\text{dblock} s q) = v \)
by (auto simp add: maxBalInp-def allBlocks-def blocksOf-def)
with \( \text{act} bk b-\text{bal} \)
show \( \text{thesis} \)
by (auto simp add: StartBallot-def)
qed
qed

lemma \( H_{\text{StartBallot-valueChosen2}} \):
assumes \( \text{act} : H_{\text{StartBallot}} s s' q \)
and \( \text{asm4} : \forall d \in D. \ b \leq \text{bal}(\text{disk} s d p) \)
\( \land (\forall q . (\text{phase} s q = 1 \land b \leq m\text{bal}(\text{dblock} s q) \land \text{hasRead} s q d p) \Rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br))) \) (is \( ?P s \))
shows \( ?P s' \)
proof (auto)
fix \( d \)
assume \( d : d \in D \)
with \( \text{asm4} \)
show \( b \leq \text{bal} (\text{disk} s' d p) \)
by (auto simp add: StartBallot-def)
fix \( d q \)
assume \( d : d \in D \)
and \( \text{phase'} : \text{phase} s' q = \text{Suc} 0 \)
and \( \text{dblk-mbal} : b \leq m\text{bal} (\text{dblock} s' q) \)
and \( \text{hasRead} : \text{hasRead} s' q d p \)
from \( \text{phase'} \text{ act hasRead} \)
have \( p31 : \text{phase} s q = 1 \)
and \( p32 : \text{dblock} s' q = \text{dblock} s q \)
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§1 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: \( b \leq bal bk \)

from subsetD[\( OF \) HPhase1or2Write-allBlocks[\( OF \) act] bk] asm3 b-bal
show \( inp bk = v \)
  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: \( \forall d \in D. \quad b \leq bal(disk s d p) \)
    \( \land (\forall q. ( \quad \text{phase } s q = 1 \)
        \land b \leq mbal(dblock s q)
        \land \text{hasRead } s q d p ) \)
    \( \rightarrow (\exists br \in \text{blocksRead } s q d. \quad b \leq bal(block br))) \) (is \( ?P \) s)
  and inv4: HInv4a s pp
shows \( ?P s' \)
proof(auto)
fix d1
assume d: \( d1 \in D \)
show \( b \leq bal(disk s' d1 p) \)
proof(cases d1 = d \( \land pp = p \))
  case True
    with inv4 act
    have HInv4a2 s pp
      by(auto simp add: Phase1or2Write-def HInv4a-def)
    with asm2 majority-intersect
    have \( \exists d2 \in D. \quad bal(disk s d2 p) \leq bal(dblock s p) \)
      by(auto simp add: HInv4a2-def MajoritySet-def)
    then obtain d2 where p41: \( d2 \in D \land bal(disk s d2 p) \leq bal(dblock s p) \)
      by auto
    from asm4 p41
    have b \( \leq bal(disk s dd p) \)
      by auto
    with p41
    have p42: \( b \leq bal(dblock s p) \)
      by auto
    from act True
    have dblock s p = 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∈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: Phase1or2Write-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: Phase1or2Write-def InitializePhase-def
      hasRead-def split : if-split-asm)
ultimately
have ∃ br∈blocksRead s q d. b≤ bal(block br)
  using p31 asm4 d
by blast
with act hasRead
show ∃ br∈blocksRead s' q d. b≤ bal(block br)
  by (auto simp add: Phase1or2Write-def InitializePhase-def
       hasRead-def)
qed

theorem HPhase1or2Write-valueChosen:
assumes act: HPhase1or2Write s s' q d
and vc: valueChosen s v
and v-input: v ∈ Inputs
and inv4: HInv4a s q
shows valueChosen s' v
proof -
from vc
obtain b p D where
  asm1: b ∈ (UN p. Ballot p)
and asm2: D∈MajoritySet
and asm3: maxBallInp 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 HPhase1or2Write-maxBallInp[OF act asm3]
have maxBallInp 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 

\[ \text{p32: } \text{dblock } s' \text{ } qq = \text{dblock } s \text{ } qq \]

by (auto simp add: Phase1or2ReadThen-def)

with 

\[ \text{dblk-mbal} \]

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

moreover

from act hasRead False

have hasRead \[ s \text{ } qq \text{ } dd \text{ } p \]

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

ultimately

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

using \[ p31 \text{ } asm4 \text{ } d \]

by blast

with act hasRead

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

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

qed

qed

theorem HPhase1or2ReadThen-valueChosen:

assumes act: HPhase1or2ReadThen \[ s \text{ } s' \text{ } q \text{ } d \text{ } p \]

and vc: valueChosen \[ s \text{ } v \]

and v-input: \[ v \in \text{Inputs} \]

shows valueChosen \[ s' \text{ } v \]

proof –

from vc

obtain \[ b \text{ } p \text{ } D \text{ where} \]

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

and \[ \text{asm2: } D \in \text{MajoritySet} \]

and \[ \text{asm3: } \text{maxBalInp } s \text{ } b \text{ } v \]

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

\[ \wedge (\forall \text{ } q. ( \text{phase } s \text{ } q = 1 \]

\[ \wedge \ b \leq \text{mbal(dblock } s \text{ } q) \]

\[ \wedge \text{hasRead } s \text{ } q \text{ } d \text{ } p \]

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

by (auto simp add: valueChosen-def)

from HPhase1or2ReadThen-maxBalInp[OF act asm3]

have maxBalInp \[ s' \text{ } b \text{ } v \text{ } . \]

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

show \text{thesis}

by (auto simp add: valueChosen-def)

qed

theorem HPhase1or2ReadElse-valueChosen:

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

\[ \Rightarrow \text{valueChosen } s' \text{ } v \]

using HStartBallot-valueChosen

by (auto simp add: Phase1or2ReadElse-def)
lemma HE ndPhase2-maxBalInp:
assumes act: HE ndPhase2 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 HE ndPhase2-allBlocks[OF act] bk] asm3 b-bal
show inp bk = v
by(auto simp add: maxBalInp-def)
qed

lemma HE ndPhase2-valueChosen2:
assumes act: HE ndPhase2 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-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: EndPhase2-def InitializePhase-def
hasRead-def split: if-split-asm)
ultimately
have ∃ br ∈ blocksRead s q d. b ≤ bal(block br)
using p31 asm4 d
by blast
with act hasRead

127
show $\exists br \in \text{blocksRead} s' q d. \ b \leq \text{bal(block br)}$
by (auto simp add: EndPhase2-def InitializePhase-def hasRead-def) 
qed 

definition HEndPhase2-valueChosen:
  assumes act: HEndPhase2 s s' q
  and vc: valueChosen s v
  and v-input: $v \in \text{Inputs}$
  shows valueChosen s' v
proof 
  from vc
  obtain $b p D$ where
    asm1: $b \in (\text{UN p. Ballot p})$
    and asm2: $D \in \text{MajoritySet}$
    and asm3: maxBalInp s b v
    and 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 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 

definition HFail-maxBalInp:
  assumes act: HFail s s' q
  and asm1: $b \in (\text{UN p. Ballot p})$
  and asm3: maxBalInp s b v
  shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk $\in \text{allBlocks} s'$
    and b-bal: $b \leq \text{bal bk}$
  from subsetD[OF HFail-allBlocks[OF act] bk]
  show inp bk $=$ v
  proof 
    assume bk: bk $\in \text{allBlocks} s$
      with asm3 b-bal
    show ?thesis
    by (auto simp add: maxBalInp-def) 
  next
    assume bk: bk$\in \{\text{dblock} s' q\}$
      with act
    have bal bk $=$ 0
  qed
by(auto simp add: Fail-def InitDB-def)
moreover
from Ballot-nzero asm1
have $0 < b$
  by auto
ultimately
show $\exists!thesis$
  using $b$-bal
  by auto
qed
qed

lemma $H$Fail-valueChosen2:
assumes act: $H$Fail $s\ q$
  and asm4: $\forall d \in D.\ b \leq bal(disk\ s\ d\ p)\ 
\land(\forall q.(\ phase\ s\ q = 1\ 
\land b \leq mbal(dblock\ s\ q)\ 
\land hasRead\ s\ q\ d\ p)\ ) \rightarrow (\exists br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$ (is $\exists! P\ s$
shows $\exists! P\ s$
proof(auto)
  fix $d$
  assume $d: d \in D$
  with act asm4
  show $b \leq bal(disk\ s'\ d\ p)$
    by(auto simp add: Fail-def)
  fix $d\ q$
  assume $d: d \in D$
    and phase': $phase\ s'\ q = Suc\ 0$
    and dblk-mbal: $b \leq mbal(dblock\ s'\ q)$
    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: Fail-def InitializePhase-def 
    hasRead-def split : if-split-asm)
  with dblk-mbal
  have $b \leq mbal(dblock\ s\ q)$ 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 $\exists br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)$
  using $p31$ asm4 $d$
  by blast
with act hasRead
show $\exists br \in blocksRead\ s'\ q\ d.\ b \leq bal(block\ br)$
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' q q d d
  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: Phase0Read-def)

next
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
have qqnq: qq ≠ q
  by (auto simp add: Phase0Read-def)
show 3 br∈blocksRead s' q d. b ≤ bal (block br)
proof –
  from phase' act hasRead
  have p31: phase s q = 1
    and p32: dblock s' q = dblock s q
    by (auto simp add: Phase0Read-def hasRead-def)
  with dblk-mbal
  have b ≤ mbal (dblock s q) by auto
moreover
from act hasRead qqnq
have hasRead s q d p
  by (auto simp add: Phase0Read-def hasRead-def
    split: if-split-asm)
ultimately
have 3 br∈blocksRead s q d. b ≤ bal (block br)
  using p31 asm4 d
  by blast
with act hasRead
show 3 br∈blocksRead s' q d. b ≤ bal (block br)
  by (auto simp add: Phase0Read-def InitializePhase-def
    hasRead-def)

qed
qed

theorem HPhase0Read-valueChosen:
  assumes act: HPhase0Read s s' q d
  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 \( \text{asm4: } \forall d \in D. \ b \leq \text{bal}(\text{disk s d p}) \wedge \forall q. (\text{phase s q = 1} \wedge b \leq \text{mbal}(\text{dblock s q}) \wedge \text{hasRead s q d p}) \rightarrow (\exists br \in \text{blocksRead s q d}. \ b \leq \text{bal}(\text{block br})) \)

by (auto simp add: valueChosen-def)

from \( \text{HPhase0Read-maxBalInp[OF act asm3]} \)
have \( \text{maxBalInp s' b v} \).

with \( \text{HPhase0Read-valueChosen2[OF act asm4]} \) \( \text{asm1 asm2} \)

show ?thesis
by (auto simp add: valueChosen-def)

qed

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

proof (auto simp add: maxBalInp-def)

fix \( \text{bk} \)
assume \( \text{bk: } bk \in \text{allBlocks s'} \)
and \( \text{b-bal: } b \leq \text{bal bk} \)
from \( \text{subsetD[OF HEndPhase0-allBlocks[OF act] bk]} \)

show \( \text{inp bk = v} \)

proof
assume \( \text{bk: } bk \in \text{allBlocks s} \)
with \( \text{asm3 b-bal} \)

show ?thesis
by (auto simp add: maxBalInp-def)

next
assume \( \text{bk: } bk \in \{\text{dblock s' q}\} \)
with \( \text{HEndPhase0-some[OF act inv1] act} \)

have \( \exists \text{ba } \in \text{allBlocksRead s q}. \ \text{bal ba = bal (dblock s' q)} \wedge \text{inp ba = inp (dblock s' q)} \)

by (auto simp add: EndPhase0-def)

then obtain \( \text{ba} \)
where \( \text{ba-blksread: } \text{ba} \in \text{allBlocksRead s q} \)
and \( \text{ba-balinp: } \text{bal ba = bal (dblock s' q)} \wedge \text{inp ba = inp (dblock s' q)} \)

by auto

have \( \text{allBlocksRead s q } \subseteq \text{allBlocks s} \)
by (auto simp add: allBlocksRead-def allRdBlks-def allBlocks-def blocksOf-def rdBy-def)

from \( \text{subsetD[OF this ba-blksread] ba-balinp bk b-bal asm3} \)

show ?thesis
by (auto simp add: maxBalInp-def)

qed

qed
lemma HEndPhase0-valueChosen2:
assumes act: HEndPhase0 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: EndPhase0-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: EndPhase0-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: EndPhase0-def InitializePhase-def
hasRead-def split: if-split-asm)
ultimately
have ∃br∈blocksRead s q d. b ≤ bal(block br)
using p31 asm4 d
by blast
with act hasRead
show ∃br∈blocksRead s' q d. b ≤ bal(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 ∈ Inputs
and inv1: Inv1 s
shows valueChosen s' v
proof –
from vc

133
obtain \( b p D \) where

\begin{align*}
\text{asm1:} & \quad b \in (\text{UN p. Ballot p}) \\
\text{and asm2:} & \quad D \subseteq \text{MajoritySet} \\
\text{and asm3:} & \quad \text{maxBalInp} s b v \\
\text{and asm4:} & \quad \forall d \in D. \quad b \leq \text{bal(disk s d p)} \\
& \quad \land (\forall q.(\text{phase s q} = 1 \\
& \quad \land b \leq \text{mbal(dblock s q)} \\
& \quad \land \text{hasRead s q d p}) \\
& \quad \longrightarrow (\exists br \in \text{blocksRead s q d}. \ b \leq \text{bal(block br)}))
\end{align*}

by (auto simp add: valueChosen-def) from HEndPhase0-maxBalInp[OF act asm3 inv1]

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

with HEndPhase0-valueChosen2[OF act asm4] asm1 asm2

show \(?thesis\)

by (auto simp add: valueChosen-def)

qed

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

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

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

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

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

theorem \( HInit-HInv6: HInit s \Longrightarrow HInv6 s \)

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

lemma HEndPhase2-Inv6-1:

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

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

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

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

and \( \text{inv3: HInv3 s} \)

and \( \text{inv5: HInv5-inner s p} \)

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

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

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

from inv5 act

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

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

and \( \text{ep2-maj: IsMajority \{d \in disksWritten s p}} \)
∀ q ∈ UNIV − {p}. hasRead s p d q}

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

case True

have p32: maxBalInp s (bal (dblock s p)) (inp (dblock s p))

proof −

  have "¬ (∃ D ∈ MajoritySet. ∃ q. (∀ d ∈ D. bal (dblock s p) < mbal (disk s d q) ∧ ¬ hasRead s p d q))"
    proof
      assume Dmaj: D ∈ MajoritySet
      from ep2-maj Dmaj majorities-intersect
      have "∃ d ∈ D. d ∈ disksWritten s p ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)"
      by (auto simp add: MajoritySet-def, blast)
      then obtain d
      where dinD: d ∈ D
      and ddisk: d ∈ disksWritten s p
      and dhasR: ∀ q ∈ UNIV − {p}. hasRead s p d q
      by auto
      from inv2b
      have "Inv2b-inner s p d"
      by (auto simp add: Inv2b-def)
      with ddisk
      have "disk s d p = dblock s p"
      by (auto simp add: Inv2b-inner-def)
      with inv2c phase
      have "bal (dblock s p) = mbal (disk s d p)"
      by (auto simp add: Inv2c-def Inv2c-inner-def)
      with dhasR dinD
      show "∃ d ∈ D. bal (dblock s p) < mbal (disk s d q) −→ hasRead s p d q"
      by auto

    qed

    with inv5R

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

  qed

have p33: maxBalInp s' (bal (dblock s' p')) (chosen s')

proof −

  from act
  have outpt': outpt s' = (outpt s) (p := inp (dblock s p))
  by (auto simp add: EndPhase2-def)
  have outpt' q: ∀ q. p ≠ q −→ outpt s' q = NotAnInput
  proof auto
  fix q
  assume "p ≠ q"
  from outpt' p
  have outpt' q: outpt s q
  by (auto simp add: EndPhase2-def)
  with True inv2c


show outpt s' q = 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 ≠ NotAnInput
  from outpt'-q
  have someeq2: \(\forall pa. \) outpt s' pa ≠ NotAnInput \implies pa=p
    by auto
  with outpt'-pa
  have outpt s' p ≠ NotAnInput
    by auto
  from some-equality[of \(\lambda p. \) outpt s' p ≠ NotAnInput, OF this someeq2]
  have (SOME p. outpt s' p ≠ NotAnInput) = p .
  with outpt'
  show outpt s' (SOME p. outpt s' p ≠ 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 \(\exists D \in \text{MajoritySet.} \ (\forall d \in D. \ disk s d p = dblock s p\)
  \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s q d)\)
  by (auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def)
then obtain D
  where Dmaj: D \in MajoritySet
  and p34: \(\forall d \in D. \ disk s d p = dblock s p\)
  \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s q d)\)
  by auto
have p35: \(\forall q. \forall d \in D. \ (\text{phase } s q = 1 \land \text{bal} (\text{dblock } s p) \leq \text{mbal} (\text{dblock } s q) \land \text{hasRead } s q d)\)
  \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: \(\text{bal} (\text{dblock } s p) \leq \text{mbal} (\text{dblock } s q)\) and hasRead: \(\text{hasRead } s q d\)
  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 phase
have \( \forall br \in \text{blocksRead } s \ p \ d. \ \text{mbal}(\text{block } br) < \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)

ultimately
have \( p41: \{ \text{block}=\text{dblock } s \ q, \ \text{proc}=q \} \in \text{blocksRead } s \ p \ d \)
using bal-mbal
by auto
from phase phase-q
have \( p \neq q \) by auto
with \( p34 \ dD \)
have hasRead \( s \ p \ d \)
by auto
with phase phase-q hasRead inv3 p41

show \( \{ \text{block}=\text{dblock } s \ p, \ \text{proc}=p \} \in \text{blocksRead } s \ q \ d \)
by (auto simp add: HInv3-def HInv3-inner-def HInv3-L-def HInv3-R-def)

qed

have \( p36: \forall q. \forall d \in D. \ \text{phase } s' q = 1 \land \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \land \text{hasRead } s' q d p \)

\[ \rightarrow (\exists br \in \text{blocksRead } s' q d. \ \text{bal}(\text{block } br) = \text{bal}(\text{dblock } s p)) \]

proof (auto)

fix \( q \ d \)

assume \( dD: d \in D \) and phase-q: phase \( s' q = \text{Suc 0} \)
and bal: bal (dblock \( s \ p)) \leq \text{mbal}(\text{dblock } s' q) \land \text{hasRead}: \text{hasRead } s' q d p \)

from phase-q act
have phase \( s' q = \text{phase } s q \land \text{dblock } s' q = \text{dblock } s q \land \text{hasRead } s' q d p = \text{hasRead } s q d p \)
by (auto simp add: EndPhase2-def hasRead-def InitializePhase-def)

with \( p35 \ phase-q \) bal hasRead dD
have \( \{ \text{block}=\text{dblock } s \ p, \ proc=p \} \in \text{blocksRead } s' q d \)
by auto

thus \( \exists br \in \text{blocksRead } s' q d. \ \text{bal}(\text{block } br) = \text{bal}(\text{dblock } s p) \)
by force

qed

hence \( p36-2: \forall q. \forall d \in D. \ \text{phase } s' q = 1 \land \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \land \text{hasRead } s' q d p \)

\[ \rightarrow (\exists br \in \text{blocksRead } s' q d. \ \text{bal}(\text{block } br) \leq \text{bal}(\text{block } s p)) \]

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

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

by auto

with \text{asm-w}

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

by auto

with b-ba

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

by auto

with \text{max-v dab}

have \text{v-value}: \text{inp} (\text{disk } s \text{ d } pa) = v

by (auto simp add: maxBalInp-def)

from \text{ba max-w dab}

have \text{w-value: inp} (\text{disk } s \text{ d } pa) = w

by (auto simp add: maxBalInp-def)

with \text{v-value}

show \text{thesis} by auto

qed

lemma \text{valueChosen-equal}:

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

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

\text{shows } v = w \text{ using assms}

proof (auto simp add: valueChosen-def)

fix a b aa ba p D pa Da

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

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

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

(\forall q. \text{ phase } s \text{ q } = \text{Suc } 0 \land

\text{b} \leq \text{mbal} (\text{dblock } s \text{ q}) \land \text{hasRead } s \text{ q } d \text{ p } \longrightarrow

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

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

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

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

(\forall q. \text{ phase } s \text{ q } = \text{Suc } 0 \land

\text{ba} \leq \text{mbal} (\text{dblock } s \text{ q}) \land \text{hasRead } s \text{ q } d \text{ pa } \longrightarrow

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

from \text{asm-v}

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

from \text{asm-w}

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

show \text{v } = \text{w}

proof (cases \text{b} \leq \text{ba})

case True

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

show \text{thesis} .

next
case False

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

\text{False}
show ?thesis
  by auto
qed

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: \( \exists D \in \text{MajoritySet}. \forall d \in D. \forall q \in \text{UNIV} - \{p\}. \text{hasRead } s \ p \ d \ q \)
by (auto simp add: MajoritySet-def)

have p43: \( \neg(\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{ bal}(dblock s p d) < \text{mbal}(disk s d q) \wedge \neg \text{hasRead } s \ p \ d \ q)) \)
proof auto
fix D q
assume Dmaj: \( D \in \text{MajoritySet} \)
show \( \exists d \in D. \text{ bal}(dblock s p d) < \text{mbal}(disk s d q) \longrightarrow \text{hasRead } s \ p \ d \ q \)
proof (cases \( p=q \))
assume pq: \( p=q \)
thus ?thesis
proof auto
from maj majors-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 disk s d p = dblock s p
by (auto simp add: Inv2b-def Inv2b-inner-def)
with inv2c phase
have bal(dblock s p) = mbal(disk s d p)
by (auto simp add: Inv2c-def Inv2c-inner-def)
with dD pq
show \( \exists d \in D. \text{ bal}(dblock s q) < \text{mbal}(disk s d q) \longrightarrow \text{hasRead } s \ q \ d \ q \)
by auto
qed
next
case False
with p42
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \text{hasRead } s \ p \ d \ q \)
by auto
with majors-intersect Dmaj
show ?thesis
by (auto simp add: MajoritySet-def, blast)
qed

have p44: \( \text{maxBalInp } s \ (\text{bal}(dblock s p)) (\text{inp}(dblock s p)) \)
by (auto simp add: EndPhase2-def Hinv5-inner-def Hinv5-inner-R-def)

have \( \exists b k \in \text{allBlocks } s. \exists b \in (\text{UN } p. \text{ Ballot } p). (\text{maxBalInp } s \ b \ (\text{chosen } s)) \wedge b \leq \text{bal } bk \)
proof –
have disk-allblks: \( \forall d \ p. \ \text{disk } s \ d \ p \in \text{allBlocks } s \)
by (auto simp add: allBlocks-def blocksOf-def)
from p31
have \( \exists b \in (\bigcup_p \text{Ballot } p), \text{maxBalInp } s \ b \ (\text{chosen } s) \ \land \\
(\exists p, \exists D \in \text{MajoritySet}(\forall d \in D. \ b \leq \text{bal}(\text{disk } s \ d \ p))) \\
\) by(auto simp add: valueChosen-def, force)
with majority-nonempty obtain b p D d
  where IsMajority D \(\land\) b \(\in\) (\(\bigcup_p \text{Ballot } p\) \(\land\) maxBalInp s b (chosen s) \(\land\) d \(\in\) D \(\land\) b \(\leq\) 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 \(\in\) allBlocks s \(\land\) b \(\leq\) bal bk
  and p45-b: b \(\in\) (\(\bigcup_p \text{Ballot } p\) \(\land\) (maxBalInp s b (chosen s))
  by(auto)
have p46: inp(dblock s p) = chosen s
proof(cases b \(\leq\) bal(dblock s p))
case True
have dblock s p \(\in\) 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 inp bk = 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 outpt s' r = 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

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' \neq \text{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 \neq \text{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 = \text{NotAnInput} \implies (\forall p. \text{outpt s p} = \text{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'; \forall p. outpt s p \in \{\text{chosen s}, \text{NotAnInput}\};
  \quad \text{Inv2c s; HNextPart s s' } ] \implies \forall p. \text{outpt s' p} \in \{\text{chosen s'}, \text{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' \neq \text{NotAnInput} \implies valueChosen s (chosen s')
by (auto simp add: StartBallot-def HInv6-def)
from HStartBallot-valueChosen[OF act] this InputsOrNi
have \( t1 \colon \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
  by auto
from act
have outpt: outpt \( s = \text{outpt } s' \)
  by (auto simp add: StartBallot-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 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 \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s (\text{chosen } s') \)
    by (auto simp add: Phase1or2Write-def HInv6-def)
  from HPhase1or2Write-valueChosen[OF act] inv4 this InputsOrNi
  have \( t1 \colon \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
    by auto
  from act
  have outpt: outpt \( s = \text{outpt } s' \)
    by (auto simp add: Phase1or2Write-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 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 \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s (\text{chosen } s') \)
    by (auto simp add: Phase1or2ReadThen-def HInv6-def)
  from HPhase1or2ReadThen-valueChosen[OF act] this InputsOrNi
  have \( t1 \colon \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
    by auto
  from act
  have outpt: outpt \( s = \text{outpt } s' \)
    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
have \( t1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \) (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
  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' \) (chosen \( s' \))
    by(auto simp add: EndPhase1-def HInv6-def)
  from HEndPhase1-valueChosen[of act] inv1 inv2a inv2b this InputsOrNi
  have \( t1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \) (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 \( \text{inv2c: Inv2c } s \)
and \( \text{nextp: HNextPart } s s' \)
shows \( \text{chosen } s = \text{chosen } s' \)

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

lemma \( \text{outpt-HInv6-2} \):
assumes \( \text{outpt: outpt } s' = (\text{outpt } s) (p := \text{NotAnInput}) \)
and \( \text{inv: } \forall p. \text{outpt } s p \in \{\text{chosen } s, \text{NotAnInput}\} \)
and \( \text{inv2c: Inv2c } s \)
and \( \text{nextp: HNextPart } s s' \)
shows \( \forall p. \text{outpt } s' p \in \{\text{chosen } s', \text{NotAnInput}\} \)

proof –
from \( \text{outpt-chosen-2}[OF outpt inv2c nextp] \)
have \( \text{chosen } s = \text{chosen } s' \).
with \( \text{inv outpt} \)
show ?thesis
by auto
qed

theorem \( \text{HFail-Inv6} \):
assumes \( \text{act: HFail } s s' p \)
and \( \text{inv: HInv6 } s \)
and \( \text{inv2c: Inv2c } s \)
shows \( \text{HInv6 } s' \)

proof –
from \( \text{outpt-chosen-2 act inv2c inv} \)
have \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s (\text{chosen } s') \)
by (auto simp add: Fail-def HInv6-def)
from \( \text{HFail-valueChosen}[OF act] \) this InputsOrNi
have \( \text{t1: chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
by auto
from \( \text{act} \)
have \( \text{outpt: outpt } s' = (\text{outpt } s) (p := \text{NotAnInput}) \)
by (auto simp add: Fail-def)
from \( \text{outpt-HInv6-2}[OF outpt] \) 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 \( \text{t1} \)
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 \( \forall p.\) outpt s' p = chosen s' \( \lor \) 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 nxt: HNext s s'
  and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s ∧ HInv6 s
  shows HInv6 s' using assms
  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 HInv :: state ⇒ bool
where
  HInv s = (HInv1 s 
    ∧ HInv2 s 
    ∧ HInv3 s 
    ∧ HInv4 s 
    ∧ HInv5 s 
    ∧ HInv6 s)

theorem I1:
  HInit s ⇒ HInv s 
  using HInit-HInv1 HInit-HInv2 HInit-HInv3 
  HInit-HInv4 HInit-HInv5 HInit-HInv6 
  by (auto simp add: HInv-def)

theorem I2:
  assumes inv: HInv s 
  and nxt: HNext s s' 
  shows HInv s' 
  by (simp add: HInv-def)
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

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

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

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

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

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

theorem R1:
[ \text{HInit } s; \text{is } = s_{2is} s ] \implies \text{HInit is }

\text{by}\ (\text{auto simp add: HInit-def HInit-def s2is-def Init-def})

\text{theorem } R2b:
\text{ assumes } \text{inv}: \text{HInv } s
\text{ and } \text{inv}' : \text{HInv } s'
\text{ and } \text{nxt: HNext } s s'
\text{ and } \text{srel: is } = s_{2is} s \land is' = s_{2is} s'
\text{ shows } (\exists p. \text{IFail is is }' p \lor \text{IChoose is is }' p) \lor \text{is } = \text{is}'

\text{proof}\ (\text{auto})
\text{ assume } \text{chg-vars: is } \neq \text{is}'
\text{ with } \text{srel}
\text{ have } \text{s-change: } \text{inpt } s \neq \text{inpt } s' \lor \text{outpt } s \neq \text{outpt } s'
\text{ by}\ (\text{auto simp add: s2is-def})
\text{ from } \text{inv}
\text{ have } \text{inv2c5: } \forall p. \text{inpt } s p \in \text{allInput } s
\text{ by}\ (\text{auto simp add: HInv-def HInv2-def Inv2c-def Inv2c-inner-def})
\text{ from } \text{nxt s-change inv2c5}
\text{ have } \text{inpt } s' \neq \text{inpt } s \lor \text{outpt } s' \neq \text{outpt } s
\text{ by}\ (\text{auto simp add: HNext-def Next-def HNextPart-def})
\text{ with } \text{nxt}
\text{ have } \exists p. \text{Fail } s s' p \lor \text{EndPhase2 } s s' p
\text{ by}\ (\text{auto simp add: HNext-def Next-def})
\text{ StartBallot-def Phase0Read-def Phase1or2Write-def}
\text{ Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def}
\text{ EndPhase1or2-def EndPhase1-def EndPhase0-def})
\text{ then obtain } p \text{ where fail-or-endphase2: } \text{Fail } s s' p \lor \text{EndPhase2 } s s' p
\text{ by}\ \text{auto}
\text{ from } \text{inv}
\text{ have } \text{inv2c: Inv2c-inner s p}
\text{ by}\ (\text{auto simp add: HInv-def HInv2-def Inv2c-def})
\text{ from } \text{fail-or-endphase2 have } \text{IFail is is }' p \lor \text{IChoose is is }' p
\text{ proof}
\text{ assume } \text{fail: } \text{Fail } s s' p
\text{ hence } \text{phase': } \text{phase } s' p = 0
\text{ and } \text{outpt: } \text{outpt } s' = (\text{outpt } s) (p:= \text{NotAnInput})
\text{ by}\ (\text{auto simp add: Fail-def})
\text{ have } \text{IFail is is }' p
\text{ proof}
\text{ from } \text{fail srel}
\text{ have } \text{ioutput is }' = (\text{ioutput is}) (p:= \text{NotAnInput})
\text{ by}\ (\text{auto simp add: Fail-def s2is-def})
\text{ moreover}
\text{ from } \text{nxt}
\text{ have } \text{all-nxt: allInput } s' = \text{allInput } s \cup (\text{range } (\text{inpt } s'))
\text{ by}\ (\text{auto simp add: HNext-def HNextPart-def})
\text{ from } \text{fail srel}
have \( \exists \, ip \in \text{Inputs}. \) \( \text{iinput is}' = (\text{iinput is})(p:= ip) \)
by (auto simp add: Fail-def s2is-def)
then obtain ip where ip-Input: ip \( \in \) Inputs and \( \text{iinput is}' = (\text{iinput is})(p:= ip) \)
by auto

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

moreover
from outpt srel nxt inv2c
have ichosen is' = ichosen is
by (auto simp add: HNext-def HNextPart-def s2is-def Inv2c-inner-def)

ultimately
show ?thesis
using ip-Input
by (auto simp add: IFail-def)
qed

thus ?thesis
by auto

next
assume endphase2: EndPhase2 \( s \ s' \ p \)
from endphase2
have phase \( s \ p = 2 \)
by (auto simp add: EndPhase2-def)
with inv2c: Ballot-nzero
have bal-dblk-nzero: bal(dblock \( s \ p \)) \( \neq 0 \)
by (auto simp add: Inv2c-inner-def)

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

ultimately
have p22: \( \text{inp} (\text{dblock} \( s \ p \)) \in \text{allInput} \ s \)
by (auto simp add: Inv2a-innermost-def)
from inv
have allInput \( s \subseteq \text{Inputs} \)
by (auto simp add: HInv-def HInv1-def)
with p22 NotAnInput endphase2
have outpt-nni: outpt \( s' \ p \neq \text{NotAnInput} \)
by (auto simp add: EndPhase2-def)
show ?thesis
proof (cases chosen \( s = \text{NotAnInput} \))
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: (\x. outpt s' x \neq NotAnInput \implies x = p)
by auto

from p32 True nxt some-equality[of \p. outpt s' p \neq NotAnInput, OF outpt-nni
some-eq]

have p33: chosen s' = outpt s' p
  by (auto simp add: HNext-def HNextPart-def)
with endphase2
have chosen s' = inp(dblock s p) \land outpt s' = (outpt s)(p:=inp(dblock s p))
  by (auto simp add: EndPhase2-def)
with True p22
have if (chosen s = NotAnInput)
  then (\exists ip \in allInput s. chosen s' = ip
  \land outpt s' = (outpt s)(p := ip))
  else (outpt s' = (outpt s)(p := chosen s)
  \land chosen s' = chosen s)

  by auto
moreover
from endphase2 inv2c5 nxt
have inpt s' = inpt s \land allInput s' = 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' = 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 = chosen s
proof
  from endphase2
  have outpt s' p = inp(dblock s p)
    by (auto simp add: EndPhase2-def)
moreover
  from inv6 p31
  have outpt s' p \in \{chosen s, 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
qed
thus ∃p. IFail is is' p ∨ IChoose is is' p
  by auto
qed
end