Proving the Correctness of Disk Paxos in
Isabelle/HOL

Mauro Jaskelioff    Stephan Merz

October 14, 2005

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 \(HHn^1\) and \(HHn^3\)) 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: allInput and chosen. Our Synod module will be obtained by existentially quantifying these variables of the Inner module.

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

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

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

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

Analogously, we have the specification of the Inner module:

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

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

\begin{align*}
\text{THEOREM R1} & \quad HInit \Rightarrow IInit \\
\text{THEOREM R2} & \quad HInit \land \square [HNext]_{\langle \text{vars, chosen, allInput} \rangle} \Rightarrow \square [INext]_{ivars}
\end{align*}

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:

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

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

<table>
<thead>
<tr>
<th>TLA^*</th>
<th>Isabelle/HOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{choose} , x.! P , x</td>
<td>\in x. , P , x</td>
</tr>
<tr>
<td>\text{phase}' = \text{phase} \setminus {p\mid</td>
<td>p</td>
</tr>
<tr>
<td>\text{UNION} {\text{blocksOf}(p) : p \in \text{Proc}}</td>
<td>\text{UN} , p. , \text{blocksOf s p}</td>
</tr>
<tr>
<td>\text{UNCHANGED} , v</td>
<td>v' = v , s</td>
</tr>
</tbody>
</table>

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

\text{THEOREM I1} \quad HInit \Rightarrow HInv
\text{THEOREM I2} \quad HInv \land HNext \Rightarrow HInv'

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

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

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

3 Translating from TLA^+ to Isabelle/HOL

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

3.1 Typed vs. Untyped

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

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

CONSTANT Inputs
NotAnInput ≜ CHOOSE c : c ∉ Inputs
DiskBlock ≜ [mbal : (UNION Ballot(p) : p ∈ Proc) ∪ {0},
bal : (UNION Ballot(p) : p ∈ Proc) ∪ {0},
inp : Inputs ∪ {NotAnInput}]

Isabelle/HOL:

typedec InputsOrNi

c consts
Inputs :: InputsOrNi set
NotAnInput :: InputsOrNi

axioms
NotAnInput: NotAnInput ∉ Inputs
InputsOrNi: (UNIV :: InputsOrNi set) = Inputs ∪ {NotAnInput}

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

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

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

For example, see figure 2. The type 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⁺:

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

Isabelle/HOL:

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

Figure 3: Translation of an action

3.2 Primed Variables

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

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

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

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

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

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

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

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

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

we write:

\[
\begin{align*}
\text{Inv2a-innermost} & \colon \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \Rightarrow \text{bool} \\
\text{Inv2a-innermost } s \ p \ bk & \equiv \ldots
\end{align*}
\]

\[
\begin{align*}
\text{Inv2a-inner} & \colon \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{Inv2a-inner } s \ p & \equiv \forall bk \in \text{blocksOf } s \ p. \text{Inv2a-innermost } s \ p \ bk
\end{align*}
\]

\[
\begin{align*}
\text{Inv2a} & \colon \text{state} \Rightarrow \text{bool} \\
\text{Inv2a } s & \equiv \forall p. \text{Inv2a-inner } s \ p
\end{align*}
\]

Now we can express that we want to obtain the fact

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

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

4 Structure of the Correctness Proof

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

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

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase_1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv_4$ and $HInv_5$ hold in the previous state to prove lemma $I_2 f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate $I$ was an invariant of $Next$, we preferred proving the invariance of $I$ for each action, rather than a big theorem proving the invariance of $I$ for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $HInv_3$ for the $EndPhase_0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

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

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

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

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

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

References


A TLA+ correctness specification

---

**MODULE Synod**

EXTENDS Naturals

CONSTANT $N$, $\text{Inputs}$

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

$\text{Proc} \triangleq 1..N$

$\text{NotAnInput} \triangleq \text{choose } c : c \notin \text{Inputs}$

VARIABLES $\text{inputs}, \text{output}$

---

**MODULE Inner**

VARIABLES $\text{allInput}, \text{chosen}$

---

$IInit \triangleq \land input \in [\text{Proc} \to \text{Inputs}]$

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

$\land \text{chosen} = \text{NotAnInput}$

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

$IChoose(p) \triangleq$

$\land output[p] = \text{NotAnInput}$

$\land \text{if chosen = NotAnInput}$

$\text{then } ip \in \text{allInput} : \land \text{chosen'} = ip$

$\land output' = [output \text{ except } ![p] = ip]$;

$\text{else } \land output' = [output \text{ except } ![p] = \text{chosen}]$

$\land \text{UNCHANGED chosen}$

$\land \text{UNCHANGED } \langle \text{input, allInput} \rangle$

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

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

$\land allInput' = allInput \cup \{ip\}$

$INext \triangleq \exists p \in \text{Proc} : IChoose(p) \lor IFail(p)$

$I\text{Spec} \triangleq IInit \land \Box [INext] (input, output, chosen, allInput)$

---

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

$\text{SynodSpec} \triangleq \exists \text{chosen, allInput} : IS(\text{chosen, allInput})!I\text{Spec}$
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

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

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

lemma ballots-not-zero [simp]:
  b ∈ Ballot p −→ 0 < b
proof (rule ccontr)
  assume b: b ∈ Ballot p
  and contr: ¬ (0 < b)
  from Ballot-nzero
  have 0 ∉ Ballot p ..
  with b contr
  show False
    by auto
qed

lemma majority-nonempty [simp]: IsMajority(S) −→ S ≠ {}
proof (auto)
  from majorities-intersect
  have IsMajority({}) ∧ IsMajority({}) −→ {} ∩ {} ≠ {}
    by auto
  thus IsMajority {} −→ False
    by auto
qed

constdefs
  AllBallots :: nat set
  AllBallots ≡ UN p. Ballot p
record
  DiskBlock =
  mbal :: nat
  bal :: nat
  inp :: InputsOrNi

constdefs
  InitDB :: DiskBlock
  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

constdefs
  hasRead :: state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
  hasRead s p d q ≡ ∃ br ∈ blocksRead s p d. proc br = q

  allRdBlks :: state ⇒ Proc ⇒ BlockProc set
  allRdBlks s p ≡ UN d. blocksRead s p d

  allBlocksRead :: state ⇒ Proc ⇒ DiskBlock set
  allBlocksRead s p ≡ block ' ( allRdBlks s p )

constdefs
  Init :: state ⇒ bool
  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. { } )
constdefs

InitializePhase :: state ⇒ state ⇒ Proc ⇒ bool
InitializePhase s s′ p ≡
  disksWritten s′ = (disksWritten s)(p := {})
& blocksRead s′ = (blocksRead s)(p := (λ d. {}))

constdefs

StartBallot :: state ⇒ state ⇒ Proc ⇒ bool
StartBallot s s′ p ≡
  phase s p ∈ {1, 2}
& phase s′ = (phase s)(p := 1)
& (∃ b ∈ Ballot p.
  mbal (dblock s p) < b
& dblock s′ = (dblock s)(p := (dblock s p)(mbal := b))
& InitializePhase s s′ p
& inpt s′ = inpt s & outpt s′ = outpt s & disk s′ = disk s

constdefs

Phase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool
Phase1or2Write s s′ p d ≡
  phase s p ∈ {1, 2}
& disk s′ = (disk s)(d := (disk s d)(p := dblock s p))
& disksWritten s′ = (disksWritten s)(p := (disksWritten s p) ∪ {d})
& inpt s′ = inpt s & outpt s′ = outpt s
& phase s′ = phase s & dblock s′ = dblock s
& blocksRead s′ = blocksRead s

constdefs

Phase1or2ReadThen :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
Phase1or2ReadThen s s′ p d q ≡
  d ∈ disksWritten s p
& mbal(disk s d q) < mbal(dblock s p)
& blocksRead s′ = (blocksRead s)(p := (blocksRead s p)(d :=
  (blocksRead s p d) ∪ {(λ block = disk s d q,
  proc = q)}))
& inpt s′ = inpt s & outpt s′ = outpt s
& disk s′ = disk s & phase s′ = phase s
& dblock s′ = dblock s & disksWritten s′ = disksWritten s

constdefs

Phase1or2ReadElse :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
Phase1or2ReadElse s s′ p d q ≡
  d ∈ disksWritten s p
& StartBallot s s′ p

constdefs

Phase1or2Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
Phase1or2Read s s′ p d q ≡
\texttt{Phase1or2ReadThen} \; s \; s' \; p \; d \; q \\
\lor \; \texttt{Phase1or2ReadElse} \; s \; s' \; p \; d \; q

\texttt{constdefs}

\texttt{blocksSeen} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock set} \\
\texttt{blocksSeen} \; s \; p \equiv \texttt{allBlocksRead} \; s \; p \cup \{ \texttt{dblock} \; s \; p \}

\texttt{nonInitBlks} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock set} \\
\texttt{nonInitBlks} \; s \; p \equiv \{ \texttt{bs} . \; \texttt{bs} \in \texttt{blocksSeen} \; s \; p \land \texttt{inp} \; \texttt{bs} \in \texttt{Inputs} \}

\texttt{maxBlk} :: \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{DiskBlock} \\
\texttt{maxBlk} \; s \; p \equiv \texttt{SOME} \; \texttt{b} . \; \texttt{b} \in \texttt{nonInitBlks} \; s \; p \land (\forall \; \texttt{c} \in \texttt{nonInitBlks} \; s \; p . \; \texttt{bal} \; \texttt{c} \leq \texttt{bal} \; \texttt{b})

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

\texttt{EndPhase2} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\texttt{EndPhase2} \; s \; s' \; p \equiv \\
\texttt{IsMajority} \; \{ \texttt{d} . \; \texttt{d} \in \texttt{disksWritten} \; s \; p \land (\forall \; \texttt{q} \in \texttt{UNIV} - \{ p \} . \texttt{hasRead} \; s \; p \; d \; q) \} \\
\land \; \texttt{phase} \; s \; p = 2 \\
\land \; \texttt{outpt} \; s' = (\texttt{outpt} \; s) \; (p := \texttt{inp} \; (\texttt{dblock} \; s \; p)) \\
\land \; \texttt{dblock} \; s' = \texttt{dblock} \; s \\
\land \; \texttt{phase} \; s' = (\texttt{phase} \; s) \; (p := \texttt{phase} \; s \; p + 1) \\
\land \; \texttt{InitializePhase} \; s \; s' \; p \\
\land \; \texttt{inpt} \; s' = \texttt{inpt} \; s \land \texttt{disk} \; s' = \texttt{disk} \; s

\texttt{EndPhase1or2} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\texttt{EndPhase1or2} \; s \; s' \; p \equiv \texttt{EndPhase1} \; s \; s' \; p \lor \texttt{EndPhase2} \; s \; s' \; p

\texttt{constdefs}

\texttt{Fail} :: \texttt{state} \Rightarrow \texttt{state} \Rightarrow \texttt{Proc} \Rightarrow \texttt{bool} \\
\texttt{Fail} \; s \; s' \; p \equiv \\
\exists \; \texttt{ip} \in \texttt{Inputs}. \; \texttt{inpt} \; s' = (\texttt{inpt} \; s) \; (p := \texttt{ip})
∧ \text{phase} \ s' = (\text{phase} \ s) (p := 0) \\
∧ \text{dblock} \ s' = (\text{dblock} \ s) (p := \text{InitDB}) \\
∧ \text{outpt} \ s' = (\text{outpt} \ s) (p := \text{NotAnInput}) \\
∧ \text{InitializePhase} \ s \ s' \ p \\
∧ \text{disk} \ s' = \text{disk} \ s

\textbf{constdefs}

\text{Phase0Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \\
\text{Phase0Read} \ s \ s' \ p \ d \equiv \\
\text{phase} \ s \ p = 0 \\
∧ \text{blocksRead} \ s' = (\text{blocksRead} \ s) (p := \text{blocksRead} \ s \ p \ d) \\
∪ \{(\text{block} = \text{disk} \ s \ d \ p, \text{proc} = p \ [])\} \\
∧ \text{inpt} \ s' = \text{inpt} \ s \ & \text{outpt} \ s' = \text{outpt} \ s \\
∧ \text{disk} \ s' = \text{disk} \ s \ & \text{phase} \ s' = \text{phase} \ s \\
∧ \text{dblock} \ s' = \text{dblock} \ s \ & \text{disksWritten} \ s' = \text{disksWritten} \ s

\textbf{constdefs}

\text{EndPhase0} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{EndPhase0} \ s \ s' \ p \equiv \\
\text{phase} \ s \ p = 0 \\
∧ \text{IsMajority} (\{d. \ \text{hasRead} \ s \ p \ d \ p\}) \\
∧ (\exists b \in \text{Ballot} \ p. \\
\ (\forall r \in \text{allBlocksRead} \ s \ p. \ m\text{bal} \ r < b) \\
∧ \text{dblock} \ s' = (\text{dblock} \ s) (p := \\
\ (\text{SOME} \ r. \ r \in \text{allBlocksRead} \ s \ p \\
∧ (\forall s \in \text{allBlocksRead} \ s \ p. \ \text{bal} \ s \leq \text{bal} \ r)) (\text{mbal} := b \ [])) \\
∧ \text{InitializePhase} \ s \ s' \ p \\
∧ \text{phase} \ s' = (\text{phase} \ s) (p := 1) \\
∧ \text{inpt} \ s' = \text{inpt} \ s \ & \text{outpt} \ s' = \text{outpt} \ s \ & \text{disk} \ s' = \text{disk} \ s

\textbf{constdefs}

\text{Next} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{bool} \\
\text{Next} \ s \ s' \equiv \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.

\textbf{constdefs}

\text{HInit} :: \text{state} \Rightarrow \text{bool} \\
\text{HInit} \ s \equiv \\
\text{Init} \ s \\
& \text{chosen} \ s = \text{NotAnInput}
\& \text{all} \text{input} \ s = \text{range} \ (\text{inpt} \ s)

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

\textbf{constdefs}
\begin{align*}
\text{HNNextPart} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{bool} \\
\text{HNNextPart} \ s \ s' \equiv \\
\begin{array}{l}
\text{chosen} \ s' = \\
\quad (\text{if chosen} \ s \neq \text{NotAnInput} \lor (\forall \ p. \ \text{outpt} \ s' \ p = \text{NotAnInput} )) \\
\quad \text{then chosen} \ s \\
\quad \text{else outpt} \ s' \ (\text{SOME} \ p. \ \text{outpt} \ s' \ p \neq \text{NotAnInput})) \\
\wedge \text{all} \text{input} \ s' = \text{all} \text{input} \ s \cup (\text{range} \ (\text{inpt} \ s'))
\end{array}
\end{align*}

\textbf{constdefs}
\begin{align*}
\text{HNext} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{bool} \\
\text{HNext} \ s \ s' \equiv \\
\begin{array}{c}
\text{Next} \ s \ s' \\
\wedge \text{HNNextPart} \ s \ s'
\end{array}
\end{align*}

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

\textbf{constdefs}
\begin{align*}
\text{HPhase1or2ReadThen} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q \equiv \text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \wedge \text{HNNextPart} \ s \ s'
\end{align*}

\textbf{constdefs}
\begin{align*}
\text{HEndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{HEndPhase1} \ s \ s' \ p \equiv \text{EndPhase1} \ s \ s' \ p \wedge \text{HNNextPart} \ s \ s'
\end{align*}

\textbf{constdefs}
\begin{align*}
\text{HPhase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{HPhase1or2Write} \ s \ s' \ p \ d \equiv \text{Phase1or2Write} \ s \ s' \ p \ d \wedge \text{HNNextPart} \ s \ s'
\end{align*}

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

\textbf{declare} \text{HPhase1or2ReadThen-def} \ [\text{simp}]
\textbf{declare} \text{HPhase1or2ReadElse-def} \ [\text{simp}]
\textbf{declare} \text{HEndPhase1-def} \ [\text{simp}]

18
C Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1 Invariant 1

This is just a type Invariant.

constdefs
  Inv1 :: state ⇒ bool
  Inv1 s ≡ ∀ p.
    inpt s p ∈ Inputs
    ∧ phase s p ≤ 3
    ∧ finite (allRdBlks s p)

constdefs
  HInv1 :: state ⇒ bool
  HInv1 s ≡ Inv1 s
  ∧ allInput s ⊆ Inputs

declare HInv1-def [simp]

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

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

lemma HNextPart-Inv1: [ HInv1 s; HNextPart s s'; Inv1 s' ] ⇒ HInv1 s'
by(auto simp add: HNextPart-def Inv1-def)

theorem HInit-HInv1: HInit s → HInv1 s
by(auto simp add: HInit-def Inv1-def Init-def allRdBlks-def)

lemma allRdBlks-finite:
  assumes inv: HInv1 s
  and asm: ∀ p. allRdBlks s' p ⊆ insert bk (allRdBlks s p)
  shows ∀ p. finite (allRdBlks s' p)
proof
fix \( pp \)
from inv
have \( \forall p. \text{finite} \ (allRdBlks \ s \ p) \)
  by (simp add: Inv1-def)
hence \( \text{finite} \ (allRdBlks \ s \ pp) \)
  by blast
with asm
show \( \text{finite} \ (allRdBlks \ s' \ pp) \)
  by (auto intro: finite-subset)
qed

theorem HPhase1or2ReadThen-HInv1:
assumes inv1: HInv1 \( s \)
and act: HPhase1or2ReadThen \( s \ s' \ p \ d \ q \)
sows 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: split-if-asm)
with inv1
have \( \forall p. \text{finite} \ (\text{allRdBlks} \ s' \ p) \)
  by (blast dest: allRdBlks-finite)
  — the others conjuncts are trivial
with inv1 act
show ?thesis
  by (auto simp add: Inv1-def Phase1or2ReadThen-def HNextPart-def)
qed

theorem HEndPhase1-HInv1:
assumes inv1: HInv1 \( s \)
and act: HEndPhase1 \( s \ s' \ p \)
sows HInv1 \( s' \)
proof –
  from inv1 act
  have HInv1 \( s' \)
    by (auto simp add: Inv1-def EndPhase1-def InitializePhase-def allRdBlks-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def dest: HNextPart-Inv1)
qed

theorem HStartBallot-HInv1:
assumes inv1: HInv1 \( s \)
and act: HStartBallot \( s \ s' \ p \)
sows HInv1 \( s' \)
proof –
  from inv1 act
  have HInv1 \( s' \)
    by (auto simp add: Inv1-def EndPhase1-def InitializePhase-def allRdBlks-def)
  with inv1 act
  show ?thesis
    by (auto simp del: HInv1-def dest: HNextPart-Inv1)
have Inv1 s'
  by (auto simp add: Inv1-def StartBallot-def InitializePhase-def allRdBlks-def)
with inv1 act
show ?thesis
  by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

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

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

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

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

qed

theorem HPhase0Read-HInv1:
  assumes inv1: HInv1 s
  and act: HPhase0Read s s' p d
  shows HInv1 s'

proof
  — we focus on the last conjunct of Inv1
  from act
  have \( \forall pp. \text{allRdBlks } s' pp \subseteq \text{allRdBlks } s pp \cup \{ (\text{block = disk } s d p, \text{proc = p}) \} \)
    by (auto simp add: Phase0Read-def allRdBlks-def
      split: split-if-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
  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 HEndPhase0-def allRdBlks-def InitializePhase-def)
  with inv1 act
  show \(?thesis \)
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)

qed

declare HInv1-def [simp del]

HInv1 is an invariant of HNext

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

end

theory DiskPaxos-Inv2 imports DiskPaxos-Inv1 begin

C.2 Invariant 2

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

constdefs
  rdBy :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ BlockProc set
  rdBy s p q d ≡ 
  \{ br . br ∈ blocksRead s q d ∧ proc br = p\}

blocksOf :: state ⇒ Proc ⇒ DiskBlock set
  blocksOf s p ≡ 
  \{ dblock s p \} 
  ∪ \{ disk s d p | d . d ∈ UNIV \} 
  ∪ \{ block br | br . br ∈ (UN q d. rdBy s p q d) \}

constdefs
  allBlocks :: state ⇒ DiskBlock set
  allBlocks s ≡ UN p. blocksOf s p

constdefs
  Inv2a-innermost :: state ⇒ Proc ⇒ DiskBlock ⇒ bool
  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\}

  Inv2a-inner :: state ⇒ Proc ⇒ bool
  Inv2a-inner s p ≡ ∀ bk ∈ blocksOf s p. Inv2a-innermost s p bk

  Inv2a :: state ⇒ bool
Inv2a $s \equiv \forall p. \ Inv2a-inner s p$

constdefs
Inv2b-inner :: state $\Rightarrow$ Proc $\Rightarrow$ Disk $\Rightarrow$ bool
Inv2b-inner $s \ p \ d \equiv$
\[
(d \in \text{disksWritten} \ s \ p \quad \wedge \quad (\text{phase} \ s \ p \in \{1,2\} \wedge \text{disk} \ s \ d \ p = \text{dblock} \ s \ p))
\wedge (\text{blocksRead} \ s \ p \ d \neq \{} \quad \wedge \quad \neg \text{hasRead} \ s \ p \ d \ p))
\]
Inv2b $::$ state $\Rightarrow$ bool
Inv2b $s \equiv \forall p \ d. \ Inv2b-inner \ s \ p \ d$

constdefs
Inv2c-inner :: state $\Rightarrow$ Proc $\Rightarrow$ bool
Inv2c-inner $s \ p \equiv$
\[
(\text{phase} \ s \ p = 0 \quad \wedge \quad \text{dblock} \ s \ p = \text{InitDB}
\wedge \quad \text{blocksWritten} \ s \ p = \{\}
\wedge (\forall d. \forall br \in \text{blocksRead} \ s \ p \ d.
\quad \text{proc} \ br = p \wedge \text{block} \ br = \text{disk} \ s \ d \ p))
\wedge (\text{phase} \ s \ p \neq 0 \quad \wedge \quad \text{mbal} \ (\text{dblock} \ s \ p) \in \text{Ballot} \ p
\wedge \quad \text{bal} \ (\text{dblock} \ s \ p) \in \text{Ballot} \ p \cup \{0\}
\wedge (\forall d. \forall br \in \text{blocksRead} \ s \ p \ d.
\quad \text{mbal} \ (\text{block} \ br) \leq \text{mbal} \ (\text{dblock} \ s \ p)))
\wedge (\text{phase} \ s \ p \in \{2,3\} \quad \wedge \quad \text{bal} \ (\text{dblock} \ s \ p) = \text{mbal} \ (\text{dblock} \ s \ p)
\wedge \quad \text{outpt} \ s \ p = (\text{if} \ \text{phase} \ s \ p = 3 \ \text{then} \ \text{inp} \ (\text{dblock} \ s \ p) \ \text{else} \ \text{NotAnInput})
\wedge \quad \text{chosen} \ s \in \text{allInput} \ s \cup \{\text{NotAnInput}\}
\wedge (\forall p. \inpt \ s \ p \in \text{allInput} \ s
\quad \wedge (\text{chosen} \ s = \text{NotAnInput} \quad \wedge \quad \text{outpt} \ s \ p = \text{NotAnInput}))
\]
Inv2c $::$ state $\Rightarrow$ bool
Inv2c $s \equiv \forall p. \ Inv2c-inner \ s \ p$

constdefs
HInv2 $::$ state $\Rightarrow$ bool
HInv2 $s \equiv \text{Inv2a} \ s \wedge \text{Inv2b} \ s \wedge \text{Inv2c} \ s$

C.2.1 Proofs of Invariant 2 a

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

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

**lemma** \( HPhase1or2ReadThen-blocksOf \):

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

**theorem** \( HPhase1or2ReadThen-Inv2a \):

assumes \( inv: Inv2a \ s \)
and \( act: HPhase1or2ReadThen \ s \ s' p d q \)
shows \( Inv2a \ s' \)

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

fix \( pp \ bk \)
assume \( bk: bk \in \text{blocksOf } s' pp \)
with \( inv HPhase1or2ReadThen-blocksOf[OF act] \)
have \( Inv2a-innermost \ s \ pp bk \)
  by(auto simp add: \( Inv2a-def \ Inv2a-inner-def \))
with \( act \)
show \( Inv2a-innermost \ s' pp bk \)
  by(auto simp add: \( Inv2a-innermost-def \ HNextPart-def \))
qed

**lemma** \( InitializePhase-rdBy \):

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

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

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

**lemma** \( HStartBallot-Inv2a-db\):

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

**proof**

from \( act \)
have \( mbal': \text{mbal} (dblock s' p) \in \text{Ballot } p \)
  by(auto simp add: StartBallot-def)
from \( act \)
have \( bal': \text{bal} (dblock s' p) = \text{bal} (dblock s \ p) \)
  by(auto simp add: StartBallot-def)
with \( act \)
have \( inp': \text{inp} (dblock s' p) = \text{inp} (dblock s \ p) \)
  by(auto simp add: StartBallot-def)
from \( act \)
have \( mbal (dblock s \ p) \leq \text{mbal} (dblock s' \ p) \)
  by(auto simp add: StartBallot-def)
with \( bal' \ inv2a \)
have \( \text{bal-mbal} : \text{bal}(\text{dblock } s' p) \leq \text{mbal}(\text{dblock } s' p) \)
by (auto simp add: Inv2a-innermost-def)

from act
have allInput s \(\subseteq\) allInput s'
  by (auto simp add: HNextPart-def)
with \( \text{mbal'} \text{bal'} \text{inp'} \text{bal-mbal} \text{act} \text{inv2a} \)
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 \( q = p \))
case True
with act inv2a
show ?thesis
  by (blast dest: HStartBallot-Inv2a-dblock)
next
case False
hence \( q \neq p \) by clarsimp
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: \( bk \in \text{blocksOf } s' q \)
with inv
have oldBlks: \( bk \in \text{blocksOf } s q \implies \text{Inv2a-innermost } s q bk \)
  by (auto simp add: Inv2a-def Inv2a-inner-def)
from bk HStartBallot-blocksOf[OF act]
have bk \( \in \{ \text{dblock } s' q \} \cup \text{blocksOf } s q \)
  by blast
thus Inv2a-innermost s' q bk
proof
  assume bk-dblock: \( bk \in \{ \text{dblock } s' q \} \)
  from inv
  have inv-q-dblock: Inv2a-innermost s q (dblock s q)
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act inv bk-dblock
  show ?thesis
by (blast dest: HStartBallot-Inv2a-dblock-q)

next
assume bk-in-blocks: bk ∈ blocksOf s q
with oddBlks
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)
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 ] ⇒ blocksOf s' q ⊆ blocksOf s q
by (auto simp add: EndPhase2-def blocksOf-def
    dest: subsetD[OF InitializePhase-rdBy])

27
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 ∈ 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 ⟹ blocksOf s' q ⊆ blocksOf s q ∪ {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 \text{Inv2a-innermost } s' q bk

proof
  assume \text{bk-dblock}: bk \in \{ \text{dblock } s' q \}
  from inv
  have \text{inv-q-dblock}: \text{Inv2a-innermost } s q (\text{dblock } s q)
  by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act bk-dblock
  show ?thesis
    by (blast dest: HFail-Inv2a-dblock-q)

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

qed

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

theorem HPhase0Read-Inv2a:
  assumes inv: Inv2a s
  and act: \text{HPhase0Read } s s' p d
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume \text{bk}: bk \in \text{blocksOf } s' q
  from inv bk \text{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:
  \text{HEndPhase0 } s s' p \Rightarrow \text{blocksOf } s' q \subseteq \text{blocksOf } s q \cup \{ \text{dblock } s' q \}
by (auto simp add: EndPhase0-def blocksOf-def dest: subsetD[OF InitializePhase-ndBy])

lemma HEndPhase0-blocksRead:
assumes act: HEndPhase0 s s' p
shows \( \exists d. \text{blocksRead } s \; p \; d \neq {} \)
proof –
  from act
  have IsMajority\({d \cdot \text{hasRead } s \; p \; d}\) by (simp add: EndPhase0-def)
  hence \(d \cdot \text{hasRead } s \; p \; d\) \(\neq {}\) by (rule majority-nonempty)
  thus \(?\text{thesis}\)
  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: \(?P \; ?x \; \Rightarrow \; ?P\) (SOME \(x\). \(?P\) \(x\)).

lemma HEndPhase0-some:  
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
shows (SOME \(b\). \(b \in \text{ allBlocksRead } s \; p\) 
\(\land \; (\forall \; t \in \text{ allBlocksRead } s \; p. \; \text{ bal } t \leq \text{ bal } b)\)) 
\(\in \; \text{ allBlocksRead } s \; p\)
\(\land \; (\forall \; t \in \text{ allBlocksRead } s \; p. \; \text{ bal } t \leq \text{ bal } (\text{SOME } b. \; b \in \text{ allBlocksRead } s \; p\) 
\(\land \; (\forall \; t \in \text{ allBlocksRead } s \; p. \; \text{ bal } t \leq \text{ bal } b)))\)
proof –
  from inv1 have finite \((\text{bal } \cdot \; \text{allBlocksRead } s \; p)\) (is finite \(?S\) )
  by (simp add: inv1-def allBlocksRead-def)
moreover
  from HEndPhase0-blocksRead[OF act]
  have \(?S \neq {}\) 
  by (auto simp add: allBlocksRead-def allRdBlks-def)
ultimately
  have Max \(?S \in \?S \; \land \; \forall \; t \in \?S. \; t \leq \text{ Max } \?S\) by auto
  hence \(\exists r \in \?S. \; \forall \; t \in \?S. \; t \leq r \) 
  then obtain mblk
  where mblk \(\in \text{ allBlocksRead } s \; p\)
  \(\land \; (\forall \; t \in \text{ allBlocksRead } s \; p. \; \text{ bal } t \leq \text{ bal } \text{ mblk})\) (is \(?P\) \text{ mblk})
  by auto
  thus \(?\text{thesis}\)
  by (rule someI)
qed 

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

lemma HNextPart-allInput:  
assumes act: HNextPart s s'

30
and inv2a: Inv2a-innermost s p (dblock s' p)
shows inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
proof –
  from act
  have allInput s' = allInput s ∪ (range (inpt s'))
    by (simp add: HNxtPart-def)
  moreover
  from inv2a
  have inp (dblock s' p) ∈ allInput s ∪ {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 ∀ t ∈ (λx. x (| mbal := mbal (dblock s' p)]) t allBlocksRead s p. Inv2a-innermost s p t
proof –
  from act
  have mbal': mbal (dblock s' p) ∈ Ballot p
    by (auto simp add: EndPhase0-def)
  from inv2c act
  have allproc-p: ∀ d. ∀ br ∈ blocksRead s p d. proc br = p
    by (simp add: Inv2c-inner-def EndPhase0-def)
  with inv2a
  have allBlocks-inv2a: ∀ t ∈ allBlocksRead s p. Inv2a-innermost s p t
    proof (auto simp add: Inv2a-innermost-def allBlocksRead-def)
      fix d bk
      assume bk-in-blocksRead: bk ∈ blocksRead s p d
      and inv2a-bk: ∀ x ∈ {u. ∃ d. u = disk s d p}
        ∪ {block br | br. (∃ q d. br ∈ 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
      show ?thesis
    qed
qed

31
have \( b \leq mb \) by (auto simp add: Inv2a-innermost-def)
moreover
from t-blocksRead \( mb \)-gt
have \( mb \leq mb \) (dblock s p) by blast
ultimately show \( b \leq mb \) (dblock s p)
  by auto
qed

lemma HEndPhase0-Inv2a-dblock:
  assumes act: HEndPhase0 s s' p
  and inv1: Inv1 s
  and inv2a: Inv2a-inner s p
  and inv2c: Inv2c-inner s p
  shows Inv2a-innermost s' p (dblock s p)
proof
  from act inv2a inv2c
  have \( t1 \): \( \forall t \in (\lambda x. x (| mb := mb (dblock s p)) ) \cdot \) allBlocksRead s p.
    Inv2a-innermost s p t
    by(blast dest: HEndPhase0-Inv2a-allBlocksRead)
  from act inv1
  have \( db \in (\lambda x. x (| mb := mb (dblock s p)) ) \cdot \) allBlocksRead s p
    by(simp, blast dest: HEndPhase0-dblock-allBlocksRead)
  with \( t1 \)
  have inv2-dblock: Inv2a-innermost s p (dblock s' p) by auto
  with act
  have \( inp \cdot (dblock s' p) \in allInput s' \cup \{ NotAnInput \} \)
    by(auto dest: HNextPart-allInput)
  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 inv1: Inv1 s
  and inv2a: Inv2a-inner s q
  and inv2c: Inv2c-inner s p
  shows Inv2a-innermost s' q (dblock s q)
proof(cases q=p)
case True
  with act inv2a inv2c inv1
  show \( \)thesis
    by(blast dest: HEndPhase0-Inv2a-dblock)
next
case False
  from inv2a
  have inv-q-dblock: Inv2a-innermost s q (dblock s q)
    by(auto simp add: Inv2a-inner-def blocksOf-def)
with False act
show ?thesis
  by (clarsimp simp add: EndPhase0-def HNextPart-def InitializePhase-def Inv2a-innermost-def)
qed

theorem HEndPhase0-Inv2a:
  assumes inv: Inv2a s
  and act: HEndPhase0 s s' p
  and inv1: Inv1 s
  and inv2c: Inv2c-inner s p
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ blocksOf s'
  with HEndPhase0-blocksOf[OF act]
  have dblock-blocks: bk ∈ {dblock s' q} ∪ blocksOf s q
    by blast
  thus Inv2a-innermost s' q bk
proof
  from inv
  have inv-q: Inv2a-inner s q
    by (auto simp add: Inv2a-def)
  assume bk ∈ {dblock s' q}
  with act inv1 inv2c inv-q
  show ?thesis
    by (blast dest: HEndPhase0-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: EndPhase0-def HNextPart-def InitializePhase-def Inv2a-innermost-def)
qed

lemma HEndPhase1-blocksOf:
  HEndPhase1 s s' p =⇒ blocksOf s' q ⊆ blocksOf s q ∪ {dblock s' q}
by (auto simp add: EndPhase1-def blocksOf-def dest: subsetD[OF InitializePhase-rdBy])

lemma maxBlk-in-nonInitBlks:
  assumes b: b ∈ nonInitBlks s p
  and inv1: Inv1 s
  shows maxBlk s p ∈ nonInitBlks s p ∧ (∀ c ∈ nonInitBlks s p. bal c ≤ bal (maxBlk s p))
proof –
have nibals-finite: finite (bal ' (nonInitBlks s p)) (is finite ?S)

proof (rule finite-imageI)
  from inv1
  have finite (allRdBlks s p)
    by (auto simp add: Inv1-def)
  hence finite (allBlocksRead s p)
    by (auto simp add: allBlocksRead-def)
  hence finite (blocksSeen s p)
    by (simp add: blocksSeen-def)
  thus finite (nonInitBlks s p)
    by(auto simp add: nonInitBlks-def intro: finite-subset)
qed

from b have bal ' nonInitBlks s p ≠ {}
  by auto
with nibals-finite
have Max ?S ∈ ?S and ∀ bb ∈ ?S. bb ≤ Max ?S by auto
hence ∃ mb ∈ ?S. ∀ bb ∈ ?S. bb ≤ mb ..
then obtain mblk
  where mblk ∈ nonInitBlks s p
   ∧ (∀ c ∈ nonInitBlks s p. bal c ≤ bal mblk)
   (is ?P mblk)
  by auto
hence ?P (SOME b. ?P b)
  by (rule someI)
thus ?thesis
  by (simp add: maxBlk-def)
qed

lemma blocksOf-nonInitBlks:
(∀ p bk. bk ∈ blocksOf s p −→ P bk)
  −→ bk ∈ nonInitBlks s p −→ P bk
by(auto simp add: allRdBlks-def blocksOf-def nonInitBlks-def
blocksSeen-def allBlocksRead-def rdBy-def,
blast)

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

proof –
  from inv
  have blocks: ∀ p bk. bk ∈ blocksOf s p
    −→ inp bk ∈ (allInput s) ∪ {NotAnInput}
  by(auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
  from mblk NotAnInput
  have inp (maxBlk s p) ≠ NotAnInput
    by(auto simp add: nonInitBlks-def)
  with mblk blocksOf-nonInitBlks[OF blocks]
  show ?thesis

34
by auto
qed

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

lemma HEndPhase1-Inv2a-dblock:
  assumes act: HEndPhase1 s s' p
  and inv1: HInv1 s
  and inv2: Inv2a s
  and inv2c: Inv2c-inner s p
  shows Inv2a-innermost s' p (dblock s' p)
proof –
  from inv1 act have inv1': HInv1 s'
    by (blast dest: HEndPhase1-HInv1)
  from inv2
  have inv2a: Inv2a-innermost s p (dblock s p)
    by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
  from act inv2c
  have mbal': mbal (dblock s' p) ∈ Ballot p
    by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
  moreover
  from act
  have bal': bal (dblock s' p) = mbal (dblock s p)
    by (auto simp add: EndPhase1-def)
  moreover
  from act inv1 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 inv: Inv2a s
and inv2c: Inv2c-inner s p
shows Inv2a-innermost s' q (dblock s' q)
proof (cases q = p)
  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 ∈ blocksOf s' q
  with HEndPhase1-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 s' q}
  with act inv1 inv2c inv
  show ?thesis
by (blast dest: HEndPhase1-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: 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 → Inv2b s
by (auto simp add: HInit-def Init-def Inv2b-def
    Inv2b-inner-def InitDB-def)

theorem HPhase1or2ReadThen-Inv2b:
    [ Inv2b s; HPhase1or2ReadThen s s' p d q ]
  ⇒ Inv2b s'
by (auto simp add: Phase1or2ReadThen-def Inv2b-def
    Inv2b-inner-def hasRead-def)

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

theorem HPhase1or2Write-Inv2b:
    [ Inv2b s; HPhase1or2Write s s' p d ]
  ⇒ 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)
**C.2.3 Proofs of Invariant 2 c**

**theorem HInit-Inv2c:**

\[
\begin{align*}
\text{HInit } &\leftarrow \text{Inv2c } \\
&\text{by (auto simp add: HInit-def Init-def Inv2c-def Inv2c-inner-def)}
\end{align*}
\]

**lemma HNextPart-Inv2c-chosen:**

assumes \( hnp: \text{HNextPart } s s' \)
and \( \text{inv2c: Inv2c } s \)
and \( \text{outpt': } \forall p. \text{outpt } s' p = (\text{if phase } s' p = 3 \quad \text{then } \text{inp}(\text{dblock } s' p) \quad \text{else NotAnInput}) \)
and \( \text{inp-dblk: } \forall p. \text{inp } (\text{dblock } s' p) \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
shows \( \text{chosen } s' \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
using \( hnp \text{ outpt' inp-dblk inv2c} \)
proof(auto simp add: HNextPart-def Inv2c-def Inv2c-inner-def split: split-if-asm)
qed

**lemma HNextPart-chosen:**

assumes \( hnp: \text{HNextPart } s s' \)
shows \( \text{chosen } s' = \text{NotAnInput } \quad (\forall p. \text{outpt } s' p = \text{NotAnInput}) \)
using \( hnp \)
proof(auto simp add: HNextPart-def split: split-if-asm)
fix \( p \) \( \text{pa} \)
assume \( o1: \text{outpt } s' p \neq \text{NotAnInput} \)
and \( o2: \text{outpt } s' (\text{SOME } p. \text{outpt } s' p \neq \text{NotAnInput}) = \text{NotAnInput} \)
from \( o1 \)
have \( \exists p. \text{outpt} \ s' \ p \neq \text{NotAnInput} \)
by auto

hence \( \text{outpt} \ s' (\text{SOME} \ p. \text{outpt} \ s' \ p \neq \text{NotAnInput}) \neq \text{NotAnInput} \)
by (rule someI-ex)

with 02
show \( \text{outpt} \ s' \ p \ a = \text{NotAnInput} \)
by simp

de

lemma \( H\text{NextPart-allInput}: \)
\[
[ \ H\text{NextPart} \ s \ s' ; \ H\text{Inv2c} \ s \ ] \implies \forall p. \text{inpt} \ s' \ p \in \text{allInput} \ s'
\]
by (auto simp add: HNextPart-def Inv2c-def Inv2c-inner-def)

theorem \( H\text{Phase1or2ReadThen-Inv2c}: \)
assumes \( \text{inv}: \text{Inv2c} \ s \)
and \( \text{act}: H\text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \)
and \( \text{inv2a}: \text{Inv2a} \ s \)
shows \( \text{Inv2c} \ s' \)

proof

from \( \text{inv2a} \)
have \( \text{inv2a'}: \text{Inv2a} \ s' \)
by (blast dest: HPhase1or2ReadThen-Inv2a)

from \( \text{act} \ \text{inv} \)
have \( \text{outpt'}: \forall p. \text{outpt} \ s' \ p = (\text{if phase} \ s' \ p = 3 \\
\qquad \text{then inp}(\text{dblock} \ s' \ p) \\
\qquad \text{else NotAnInput}) \)
by (auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)

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

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

from \( \text{act} \ \text{inv} \)
have \( \forall p. \text{inpt} \ s' \ p \in \text{allInput} \ s' \\
\quad \wedge (\text{chosen} \ s' = \text{NotAnInput} \implies \text{outpt} \ s' \ p = \text{NotAnInput}) \)
by (auto dest: HNextPart-chosen HNextPart-allInput)

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

de

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

proof


from act
have phase': phase s' p = 1
  by (simp add: StartBallot-def)
from act
have phase: phase s p ∈ {1,2}
  by (simp add: StartBallot-def)
from act inv
have mbal': mbal(dblock s' p) ∈ Ballot p
  by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
from inv phase
have bal'': bal(dblock s p) ∈ Ballot p ∪ {0}
  by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
with act
have bal'': bal(dblock s p) ∈ Ballot p ∪ {0}
  by (auto simp add: StartBallot-def)
from act inv phase phase'
have blks': (∀ d. ∀ br ∈ 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': ∀ p. outpt s' p = (if phase s' p = 3
  then inp(dblock s' p)
  else NotAnInput)
  by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk': ∀ p. inp(dblock s' p) ∈ allInput s' ∪ {NotAnInput}
  by (auto simp add: Inv2a-def Inv2a-inner-def
          Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
  by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: ∀ p. inp s' p ∈ allInput s'
  ∧ (chosen s' = NotAnInput
      → outpt s' p = NotAnInput)
  by (auto dest: HNextPart-chosen HNextPart-allInput)
with 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 \( \text{inv2a: Inv2a } s \)
shows \( \text{Inv2c } s' \)
proof –
from \( \text{inv2a act} \)
have \( \text{inv2a': Inv2a } s' \)
  by\((\text{blast dest: HPhase1or2Write-Inv2a})\)
from \( \text{act \ inv} \)
have \( \text{outpt': } \forall \ p. \ \text{outpt } s' p = (\text{if phase } s' p = 3 \\
\text{then } \text{inp}(\text{dblock } s' p) \\text{else NotAnInput}) \)
by\((\text{auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def})\)
from \( \text{inv2a'} \)
have \( \text{dblkl: } \forall \ p. \ \text{inp}(\text{dblock } s' p) \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
by\((\text{auto simp add: Inv2a-def Inv2a-inner-def} \\
\text{Inv2a-innermost-def blocksOf-def})\)
with \( \text{act \ inv \ outpt'} \)
have \( \text{chosen': chosen } s' \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
by\((\text{auto dest: HNextPart-Inv2c-chosen})\)
from \( \text{act \ inv} \)
have \( \text{allinp: } \forall \ p. \ \text{inpt } s' p \in \text{allInput } s' \wedge (\text{chosen } s' = \text{NotAnInput} \\
\implies \text{outpt } s' p = \text{NotAnInput}) \)
by\((\text{auto dest: HNextPart-chosen HNextPart-allInput})\)
with \( \text{outpt'} \text{ chosen'} \text{ act \ inv} \)
show \?thesis
  by\((\text{auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def})\)
qed

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

theorem \( \text{HEndPhase1-Inv2c:} \)
assumes \( \text{inv: Inv2c } s \)
and \( \text{act: HEndPhase1 } s s' p \)
and \( \text{inv2a: Inv2a } s \)
and \( \text{inv1: HInv1 } s \)
shows \( \text{Inv2c } s' \)
proof –
from \( \text{inv} \)
have \( \text{Inv2c-inner } s p \) by\((\text{auto simp add: Inv2c-def})\)
with \( \text{inv2a act inv1} \)
have \( \text{inv2a': Inv2a } s' \)
  by\((\text{blast dest: HEndPhase1-Inv2a})\)
from \( \text{act \ inv} \)
have \( \text{mbal': mbal(\text{dblock } s' p) \in Ballot } p \)
  by\((\text{auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def})\)
from \( \text{act} \)
have \( \text{bal': bal(\text{dblock } s' p) = mbal(\text{dblock } s' p)} \)
  by\((\text{auto simp add: EndPhase1-def})\)
from act inv
have blk': (∀ d. ∀ br ∈ blocksRead s' p d.
    mbal(block br) < mbal(dblock s' p))
  by(auto simp add: EndPhase1-def InitializePhase-def
      Inv2c-def Inv2c-inner-def)

from act inv
have outpt': ∀ p. outpt s' p = (if phase s' p = 3
    then inp(dblock s' p)
    else NotAnInput)
  by(auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)

from inv2a'
have dblk: ∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
  by(auto simp add: Inv2a-def Inv2a-inner-def
      Inv2a-innermost-def blocksOf-def)

with act inv outpt'
have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
  by(auto dest: HNextPart-Inv2c-chosen)

from act inv
have allinp: ∀ p. inpt s' p ∈ allInput s' ∧ (chosen s' = NotAnInput
  —→ outpt s' p = NotAnInput)
  by(auto dest: HNextPart-chosen HNextPart-allInput)

with mbal bal' blk' outpt' chosen' act inv
show ?thesis
  by(auto simp add: EndPhase1-def InitializePhase-def
      Inv2c-def Inv2c-inner-def)

qed

theorem HEndPhase2-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase2 s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
proof —
from inv2a act
have inv2a': Inv2a s'
  by(blast dest: HEndPhase2-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: EndPhase2-def Inv2c-def Inv2c-inner-def)

from inv2a'
have dblk: ∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
  by(auto simp add: Inv2a-def Inv2a-inner-def
      Inv2a-innermost-def blocksOf-def)

with act inv outpt'
have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
  by(auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \( \forall p. \) inpt \( s' \) \( p \in \text{allInput} \) \( s' \)
\( \land (\text{chosen} \ s' = \text{NotAnInput} \)
\( \rightarrow \text{outpt} \ s' \ p = \text{NotAnInput} ) \)
by(auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
by(auto simp add: EndPhase2-def InitializePhase-def
Inv2c-def Inv2c-inner-def)
qed

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

theorem \( \text{HEndPhase0-Inv2c} \):
assumes inv: \( \text{Inv2c } s \)
and act: \( \text{HEndPhase0 } s s' p \)
and inv2a: \( \text{Inv2a } s \)
and inv1: \( \text{Inv1 } s \)
shows \( \text{Inv2c } s' \)
proof
  from inv
  have \( \text{Inv2c-inner } s p \) by (auto simp add: \( \text{Inv2c-def} \))
  with inv2a act inv1
  have \( \text{inv2a}' : \text{Inv2a } s' \)
    by (blast dest: \( \text{HEndPhase0-Inv2a} \))
  hence bal': \( \text{bal(dblock } s' p) \in \text{Ballot } p \cup \{0\} \)
    by (auto simp add: \( \text{Inv2c-def Inv2c-inner-def} \))
  from act inv
  have mbal': \( \text{mbal(dblock } s' p) \in \text{Ballot } p \)
    by (auto simp add: \( \text{EndPhase0-def Inv2c-def Inv2c-inner-def} \))
  from act inv
  have bkss': \( \forall d. \forall \text{br } \in \text{blocksRead } s' p d. \)
    \( \text{mbal(block } \text{br) < mbal(dblock } s' p) \)
    by (auto simp add: \( \text{EndPhase0-def InitializePhase-def} \))
  from act inv
have \textit{outpt}': \forall p. \textit{outpt} s' p = (\text{if phase} s' p = 3 \\
then \text{inp}(\text{dblock} s' p)  \\
else \text{NotAnInput})  \\
by(auto simp add: \text{EndPhase0-def Inv2c-def Inv2c-inner-def})

from \textit{inv2a'}

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

with \textit{act inv outpt}'

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

from \textit{act inv}

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

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

show \theta \textit{thesis}  \\
by(auto simp add: \text{EndPhase0-def InitializePhase-def} \text{Inv2c-def Inv2c-inner-def})

qed

theorem \textit{HInit-HInv2}:
\textit{HInit} s \Rightarrow \textit{HInv2} s

using \textit{HInit-Inv2a} \textit{HInit-Inv2b} \textit{HInit-Inv2c}

by(auto simp add: \text{HInv2-def})

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

lemma I2b:
assumes \textit{nxt}: \textit{HNext} s s'
and \textit{inv}: \textit{HInv1} s \land \textit{HInv2} s
shows \textit{HInv2} s'

proof(auto simp add: \text{HInv2-def})

show \textit{Inv2a} s'

by (auto simp add: \text{HInv2-def HNext-def Next-def}, 
auto intro: \text{HStartBallot-Inv2a}, 
auto intro: \text{HPhase1or2Write-Inv2a}, 
auto simp add: \text{Phase1or2Read-def} 
\hspace{1em} intro: \text{HPhase1or2ReadThen-Inv2a} 
\hspace{1em} \text{HPhase1or2ReadElse-Inv2a}, 
auto intro: \text{HPhase0Read-Inv2a}, 
auto simp add: \text{EndPhase1or2-def Inv2c-def} 
\hspace{1em} intro: \text{HEndPhase1-Inv2a} 
\hspace{1em} \text{HEndPhase2-Inv2a}, 
auto intro: \text{HFail-Inv2a}, 
auto simp add: \text{HInv1-def} 
\hspace{1em} intro: \text{HEndPhase0-Inv2a})

show \textit{Inv2b} s'

by(auto simp add: \text{HInv2-def HNext-def Next-def}, 
auto intro: \text{HStartBallot-Inv2b},
auto intro: HPhase0Read-Inv2b,
auto intro: HPhase1or2Write-Inv2b,
auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2b
    HPhase1or2ReadElse-Inv2b,
auto simp add: EndPhase1or2-def
    intro: HEndPhase1-Inv2b HEndPhase2-Inv2b,
auto intro: HFail-Inv2b HEndPhase0-Inv2b)

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

qed

end

theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

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

constsdefs
HInv3-L :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ bool
HInv3-L s p q d ≡  
    phase s p ∈ {1,2}  
    ∧  phase s q ∈ {1,2}  
    ∧  hasRead s p d q  
    ∧  hasRead s q d p

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

HInv3-inner :: state ⇒ Proc ⇒ Proc ⇒ Disk ⇒ bool
HInv3-inner s p q d ≡  
    HInv3-L s p q d → HInv3-R s p q d

HInv3 :: state ⇒ bool
\[ H_{inv3} \equiv \forall p q d. \; H_{inv3-\text{inner}} s p q d \]

### C.3.1 Proofs of Invariant 3

**Theorem** \( H_{init} - H_{inv3} : H_{init} s \Rightarrow H_{inv3} s \)

by (simp add: \( H_{init-def} \) Init-def \( H_{inv3-def} \)
\( H_{inv3-\text{inner-def}} \) \( H_{inv3-L-def} \) \( H_{inv3-R-def} \))

**Lemma** \( \text{InitPhase-}H_{inv3-p} \):

\[
\begin{align*}
\text{[ \ InitializePhase } s \; s' \; p ; \; H_{inv3-L} \; s' \; p \; q \; d \ ] \Rightarrow H_{inv3-R} \; s' \; p \; q \; d
\end{align*}
\]

by (auto simp add: \( \text{InitializePhase-def} \) \( H_{inv3-\text{inner-def}} \)
\( \text{hasRead-def} \) \( H_{inv3-L-def} \) \( H_{inv3-R-def} \))

**Lemma** \( \text{InitPhase-}H_{inv3-q} \):

\[
\begin{align*}
\text{[ \ InitializePhase } s \; s' \; q ; \; H_{inv3-L} \; s' \; p \; q \; d \ ] \Rightarrow H_{inv3-R} \; s' \; p \; q \; d
\end{align*}
\]

by (auto simp add: \( \text{InitializePhase-def} \) \( H_{inv3-\text{inner-def}} \)
\( \text{hasRead-def} \) \( H_{inv3-L-def} \) \( H_{inv3-R-def} \))

**Lemma** \( H_{inv3-L\text{-sym}} \):

\[
\begin{align*}
H_{inv3-L} \; s \; p \; q \; d \Rightarrow H_{inv3-L} \; s \; q \; p \; d
\end{align*}
\]

by (auto simp add: \( H_{inv3-L-def} \))

**Lemma** \( H_{inv3-R\text{-sym}} \):

\[
\begin{align*}
H_{inv3-R} \; s \; p \; q \; d \Rightarrow H_{inv3-R} \; s \; q \; p \; d
\end{align*}
\]

by (auto simp add: \( H_{inv3-R-def} \))

**Lemma** \( \text{Phase1or2ReadThen-}H_{inv3-pq} \):

assumes act: \( \text{Phase1or2ReadThen } s \; s' \; p \; d \; q \)
and inv-L' : \( H_{inv3-L} \; s' \; p \; q \; d \)
and pq: \( p \neq q \)
and inv2b: Inv2b s

shows \( H_{inv3-R} \; s' \; p \; q \; d \)

**Proof** –

from inv-L' act pq

have phase \( s \; q \in \{1,2\} \land \text{hasRead} \; s \; q \; d \; p \)

by (auto simp add: \( \text{Phase1or2ReadThen-def} \) \( H_{inv3-L-def} \)
\( \text{hasRead-def} \) split: split_if_asm)

with inv2b

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

by (auto simp add: Inv2b-def Inv2b-inner-def
\( \text{hasRead-def} \))

with act

show ?thesis

by (auto simp add: \( \text{Phase1or2ReadThen-def} \) \( H_{inv3-def} \)
\( H_{inv3-\text{inner-def}} \) \( H_{inv3-R-def} \))

qed

**Lemma** \( \text{Phase1or2ReadThen-}H_{inv3-hasRead} \):

\[
\begin{align*}
\neg \text{hasRead } s \; p \; d \; q;
\end{align*}
\]

47
\[\neg \text{hasRead } s' \text{ pp dd qq} \]

**theorem HPhase1or2ReadThen-HInv3:**

assumes act: \(HPhase1or2ReadThen s s' p d q\)
and inv: \(HInv3 s\)
and pq: \(p \neq q\)
and inv2b: \(Inv2b s\)

shows \(HInv3 s'\)

**proof(clarsimp simp add: HInv3-def HInv3-inner-def)**

fix pp qq dd
assume h3l': \(HInv3-L s pp qq dd\)

show \(HInv3-R s pp qq dd\)
proof (cases \(HInv3-L s pp qq dd\))
  case True
  with inv
  have \(HInv3-R s pp qq dd\)
  by (clarsimp simp add: HInv3-def HInv3-inner-def)
  with act h3l'
  show \(?thesis\)
  by (clarsimp simp add: HInv3-R-def HInv3-L-def Phase1or2ReadThen-def)
next
  case False
  assume nh3l: \(\neg HInv3-L s pp qq dd\)
  show \(HInv3-R s' pp qq dd\)
  proof (cases \((pp = p \land qq = q) \lor (pp = q \land qq = p)\) \land dd = d) 
    case True
    with act pq inv2b h3l' HInv3-L-sym[OF h3l']
    show ?thesis
    by (clarsimp dest: Phase1or2ReadThen-HInv3-pq HInv3-R-sym)
  next
    case False
    from nh3l h3l' act
    have \((\neg \text{hasRead } s pp dd qq) \lor (\neg \text{hasRead } s qq dd pp)\)
    \land \(\text{hasRead } s' pp dd qq \land \text{hasRead } s' qq dd pp\)
    by (clarsimp simp add: HInv3-L-def Phase1or2ReadThen-def)
    with act False
    show ?thesis
    by (clarsimp dest: Phase1or2ReadThen-HInv3-hasRead)
  qed
qed

**lemma StartBallot-HInv3-p:**

\[\text{StartBallot } s s' p; HInv3-L s' p q d \]

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

by(auto simp add: StartBallot-def dest: InitPhase-HInv3-p)
lemma StartBallot-HInv3-q:

\[\begin{align*}
\text{StartBallot } & s \ s' \ q; \ HInv3-L \ s' \ p \ q \ d \\
\implies & \ HInv3-R \ s' \ p \ q \ d \\
\end{align*}\]

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

lemma StartBallot-HInv3-nL:

\[\begin{align*}
\text{StartBallot } & s \ s' \ t; \ \neg HInv3-L \ s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies & \ \neg HInv3-L \ s' \ p \ q \ d \\
\end{align*}\]

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

lemma StartBallot-HInv3-R:

\[\begin{align*}
\text{StartBallot } & s \ s' \ t; \ HInv3-R \ s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies & \ HInv3-R \ s' \ p \ q \ d \\
\end{align*}\]

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

lemma StartBallot-HInv3-t:

\[\begin{align*}
\text{StartBallot } & s \ s' \ t; \ HInv3-inner \ s \ p \ q \ d; \ t \neq p; \ t \neq q \\
\implies & \ HInv3-inner \ s' \ p \ q \ d \\
\end{align*}\]

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

lemma StartBallot-HInv3:

assumes act: \text{StartBallot } s \ s' \ t 
and inv: \text{HInv3-inner } s \ p \ q \ d 
shows \ HInv3-inner \ s' \ p \ q \ d 

proof (cases \( t = p \lor t = q \))
  case True
  with act inv
  show ?thesis
    by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-p StartBallot-HInv3-q)
  next 
  case False
  with inv act
  show ?thesis
    by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-t)
qed

theorem HStartBallot-HInv3:

\[\begin{align*}
\text{HStartBallot } & s \ s' \ p; \ HInv3 \ s \\
\implies & \ HInv3 \ s' \\
\end{align*}\]

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

theorem HPhase1or2ReadElse-HInv3:

\[\begin{align*}
\text{HPhase1or2ReadElse } & s \ s' \ p \ d \ q; \ HInv3 \ s \\
\implies & \ HInv3 \ s' \\
\end{align*}\]

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 ~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 ] \implies 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 ] \implies HInv3-R s' p q d
by (auto simp add: EndPhase1-def dest: InitPhase-HInv3-q)

lemma EndPhase1-HInv3-nL:
  [ EndPhase1 s s' t; ~HInv3-L s p q d; t\neq p; t\neq q ]
  \implies ~HInv3-L s' p q d
by (auto simp add: EndPhase1-def InitializePhase-def HInv3-L-def hasRead-def)

lemma EndPhase1-HInv3-R:
  [ EndPhase1 s s' t; HInv3-R s p q d; t\neq p; t\neq q ]
  \implies HInv3-R s' p q d
by (auto simp add: EndPhase1-def InitializePhase-def HInv3-R-def hasRead-def)

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

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

lemma EndPhase2-HInv3-t:
[ [ EndPhase2 s s' t; HInv3-inner s p q d; t ≠ p; t ≠ q ] ]
  ⇒ HInv3-inner s' p q d
by (auto simp add: HInv3-inner-def dest: EndPhase2-HInv3-nL EndPhase2-HInv3-R)
lemma EndPhase2-HInv3:
  assumes act: EndPhase2 s s' t
  and  inv: HInv3-inner s p q d
  shows  HInv3-inner s' p q d
proof(cases t=p ∨ t=q)
  case True
  with act inv
  show  ?thesis
    by(auto simp add: HInv3-inner-def dest: EndPhase2-HInv3-p EndPhase2-HInv3-q)
next
  case False
  with inv act
  show  ?thesis
    by(auto simp add: HInv3-inner-def dest: EndPhase2-HInv3-t)
qed

theorem HEndPhase2-HInv3:
[ HEndPhase2 s s' p; HInv3 s ] ==> HInv3 s'
by(auto simp add: HInv3-def dest: EndPhase2-HInv3)

lemma Fail-HInv3-p:
[ Fail s s' p; HInv3-L s' p q d ] ==> HInv3-R s' p q d
by(auto simp add: Fail-def dest: InitPhase-HInv3-p)

lemma Fail-HInv3-q:
[ Fail s s' q; HInv3-L s' p q d ] ==> HInv3-R s' p q d
by(auto simp add: Fail-def dest: InitPhase-HInv3-q)

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

lemma Fail-HInv3-R:
[ Fail s s' t; HInv3-R s p q d; t≠p; t≠ q ]
  ==> HInv3-R s' p q d
by(auto simp add: Fail-def InitializePhase-def
           HInv3-R-def hasRead-def)

lemma Fail-HInv3-t:
[ Fail 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: Fail-HInv3-nL Fail-HInv3-R)

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

proof (cases t = p ∨ t = q)
  case True
  with act inv
  show ?thesis
    by (auto simp add: HInv3-inner-def dest: Fail-HInv3-p Fail-HInv3-q)
next
  case False
  with inv act
  show ?thesis
    by (auto simp add: HInv3-inner-def dest: Fail-HInv3-t)
qed

theorem HFail-HInv3:
[ [ HFail s s' p; HInv3 s ] ] ⇒ HInv3 s'
by (auto simp add: HInv3-def dest: Fail-HInv3)

theorem HPhase0Read-HInv3:
assumes act: HPhase0Read s s' p d
and inv: HInv3 s
shows HInv3 s'
proof (auto simp add: HInv3-def)
  fix pp qq dd
  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 HPhase0Read-def)
  next
    case False
    with act
    have ¬ HInv3-L s' pp qq dd
      by (auto simp add: HInv3-L-def hasRead-def HPhase0Read-def)
    thus ?thesis
      by (simp add: HInv3-inner-def)
  qed
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
  case False
  with inv act
  show ?thesis
   by(auto simp add: HInv3-inner-def dest: EndPhase0-HInv3-t)
qed

theorem HEndPhase0-HInv3:
[ HEndPhase0 s s' p; HInv3 s ] ⇒ HInv3 s'
by(auto simp add: HInv3-def dest: EndPhase0-HInv3)

HInv1 ∧ HInv2 ∧ HInv3 is an invariant of HNext.

lemma I2c:
assumes nxt: HNext s s'
and inv: HInv1 s ∧ HInv2 s ∧ HInv3 s
shows \( HInv^3 s' \)

by (auto simp add: HNext-def Next-def,
    auto intro: HStartBallot-HInv3,
    auto intro: HPhase0Read-HInv3,
    auto intro: HPhase1or2Write-HInv3,
    auto simp add: Phase1or2Read-def HInv2-def
    intro: HPhase1or2ReadThen-HInv3
    HPhase1or2ReadElse-HInv3,
    auto simp add: EndPhase1or2-def
    intro: HEndPhase1-HInv3
    HEndPhase2-HInv3,
    auto intro: HFail-HInv3,
    auto intro: HEndPhase0-HInv3)

end

theory DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

C.4 Invariant 4

This invariant expresses relations among \( mbal \) and \( bal \) values of a processor and of its disk blocks. \( HInv^4a \) asserts that, when \( p \) is not recovering from a failure, its \( mbal \) value is at least as large as the \( bal \) field of any of its blocks, and at least as large as the \( mbal \) field of its block on some disk in any majority set. \( HInv^4b \) conjunct asserts that, in phase 1, its \( mbal \) value is actually greater than the \( bal \) field of any of its blocks. \( HInv^4c \) asserts that, in phase 2, its \( bal \) value is the \( mbal \) field of all its blocks on some majority set of disks. \( HInv^4d \) asserts that the \( bal \) field of any of its blocks is at most as large as the \( mbal \) field of all its disk blocks on some majority set of disks.

constdefs
    MajoritySet :: Disk set set
    MajoritySet \( \equiv \{ D. \text{IsMajority}(D) \} \)

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

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

    HInv4a :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool
    HInv4a \( s \) \( p \) \( \equiv \) phase \( s \ p \neq 0 \relimplies HInv4a1 \ s \ p \land HInv4a2 \ s \ p \)

    HInv4b :: state \( \Rightarrow \) Proc \( \Rightarrow \) bool
\( H_{inv4b} \ s \ b \ p \equiv \text{phase} \ s \ b \ p = 1 \implies (\forall bk \in \text{blocksOf} \ s \ b \ p. \ \text{bal} \ bk < \text{mbal} (\text{dblock} \ s \ b \ p)) \)

\( H_{inv4c} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
\( H_{inv4c} \ s \ b \ p \equiv \text{phase} \ s \ b \ p \in \{2, 3\} \implies (\exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} (\text{dblock} \ s \ d \ p) = \text{bal} (\text{dblock} \ s \ p)) \)

\( H_{inv4d} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
\( H_{inv4d} \ s \ b \ p \equiv \forall bk \in \text{blocksOf} \ s \ b \ p. \ \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} \ s \ d \ p) \)

The initial state implies Invariant 4.

**Theorem** \( H_{init-Hinv4} : H_{init} \ s \implies H_{inv4} \ s \)
using \( \text{Disk-isMajority} \)
by \((\text{auto simp add: } H_{init-def} \ H_{inv4-def} \ H_{inv4a-def} \ H_{inv4a1-def} \ H_{inv4a2-def} \ H_{inv4b-def} \ H_{inv4c-def} \ H_{inv4d-def} \ MajoritySet-def \ blocksOf-def \ InitDB-def \ rdBy-def)) \)

To prove that the actions preserve \( H_{inv4} \), we do it for one conjunct at a time.

For each action \( actionss'q \) and conjunct \( x \in a, b, c, d \) of \( H_{inv4x} s'p \), we prove two lemmas. The first lemma \( action-H_{inv4x}-p \) proves the case of \( p = q \), while lemma \( action-H_{inv4x}-q \) proves the other case.

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

**Lemma** \( H_{startBallot-Hinv4a1} : \)
assumes act: \( H_{startBallot} \ s \ s' p \)
and inv: \( H_{inv4a1} \ s \ p \)
and inv2a: \( \text{Inv2a-inner} \ s \ p \)
shows \( H_{inv4a1} \ s' p \)
**Proof** (auto simp add: \( H_{inv4a1-def} \))
fix \( bk \)
assume \( bk \in \text{blocksOf} \ s' p \)
with \( H_{startBallot}-\text{blocksOf} [OF \ \text{act}] \)
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: \( \text{Inv2a-innermost-def} \ \text{Inv2a-inner-def} \ \text{blocksOf-def} \))
next
assume \( bk \in \text{blocksOf} \ s \ p \)
with \( \text{inv act} \)
show thesis
by (auto simp add: StartBallot-def HInv4a1-def)
qed

lemma HStartBallot-HInv4a2:
assumes act: HStartBallot s s' p
and inv: HInv4a2 s p
shows HInv4a2 s' p
proof (auto simp add: HInv4a2-def)
fix D
assume Dmaj: D ∈ MajoritySet
from inv Dmaj
have ∃d ∈ D. mbal (disk s d p) ≤ mbal (dblock s p)
∧ bal (disk s d p) ≤ bal (dblock s p)
by (auto simp add: HInv4a2-def)
then obtain d
where d ∈ D
∧ mbal (disk s d p) ≤ mbal (dblock s p)
∧ bal (disk s d p) ≤ bal (dblock s p)
by auto
with act
have d ∈ D
∧ mbal (disk s' d p) ≤ mbal (dblock s' p)
∧ bal (disk s' d p) ≤ bal (dblock s' p)
by (auto simp add: StartBallot-def)
with Dmaj
show ∃d ∈ D. mbal (disk s' d p) ≤ mbal (dblock s' p)
∧ bal (disk s' d p) ≤ bal (dblock s' p)
by auto
qed

lemma HStartBallot-HInv4a-p:
assumes act: HStartBallot s s' p
and inv: HInv4a s p
and inv2a: Inv2a-inner s' p
shows HInv4a s' p
using act inv inv2a
proof –
from act
have phase: 0 < 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 \( \text{act}: \text{HStartBallot} \ s \ s' \ p \)
and \( \text{inv}: \text{HInv4a} \ s \ q \)
and \( \text{pnq}: \ p \neq q \)
shows \( \text{HInv4a} \ s' \ q \)

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

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

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

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

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

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

proof –

58
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 add: 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 \in \{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)

59
lemma HPhase1or2ReadThen-HInv4a-q:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv4a s q
and pq: p≠q
shows HInv4a s' q
proof –
from act pq
have blocksOf s' q ⊆ blocksOf s q
by(auto simp add: Phase1or2ReadThen-def InitializePhase-def
blocksOf-def rdBy-def)
with act inv pq
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: dest: HStartBallot-HInv4a )
qed

lemma HEndPhase1-HInv4a1:
assumes act: HEndPhase1 s s' p
and inv: HInv4a1 s p
shows HInv4a1 s' p
proof(auto simp add: HInv4a1-def)
fix bk
assume bk: bk ∈ blocksOf s' p
from bk HEndPhase1-blocksOF[OF act]
have bk ∈ {dblock s' p} ∪ blocksOf s p
by blast
with act inv
show bal bk ≤ 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: HInv4a2 s p 
and inv2a: Inv2a s 
shows HInv4a2 s' p 
proof(auto simp add: HInv4a2-def) 
fix D 
assume Dmaj: D ∈ MajoritySet 
from inv Dmaj 
have ∃ d∈D. mbal (disk s d p) ≤ mbal (dblock s p) 
  ∧ bal (disk s d p) ≤ bal (dblock s p) 
  by(auto simp add: HInv4a2-def) 
then obtain d 
  where d-cond: d∈D 
    ∧ mbal (disk s d p) ≤ mbal (dblock s p) 
    ∧ bal (disk s d p) ≤ bal (dblock s p) 
  by auto 
have disk s d p ∈ blocksOf s p 
  by(auto simp add: blocksOf-def) 
with inv2a 
have bal(disk s d p) ≤ mbal (disk s d p) 
  by(auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def) 
with act d-cond 
have d∈D 
  ∧ mbal (disk s' d p) ≤ mbal (dblock s' p) 
  ∧ bal (disk s' d p) ≤ bal (dblock s' p) 
  by(auto simp add: EndPhase1-def) 
with Dmaj 
show ∃ d∈D. mbal (disk s' d p) ≤ mbal (dblock s' p) 
  ∧ bal (disk s' d p) ≤ 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 
shows HInv4a s' p 
proof – 
  from act 
  have phase: 0 < phase s p 
    by(auto simp add: EndPhase1-def) 
  with act inv inv2a 
  show ?thesis 
    by(auto simp del: HEndPhase1-def simp add: HInv4a-def 
        elim: HEndPhase1-HInv4a1 HEndPhase1-HInv4a2) 
qed 

lemma HEndPhase1-HInv4a-q: 
assumes act: HEndPhase1 s s' p
and inv: HInv4a s q
and pnq: p ≠ q
shows HInv4a s' q

proof –
from act pnq
have dblock s' q = dblock s q ∧ disk s' = disk s
  by (auto simp add: EndPhase1-def)
moreover
from act pnq
have ∀ p d. rdBy s' q p d ⊆ rdBy s q p d
  by (auto simp add: EndPhase1-def InitializePhase-def rdBy-def)
hence (UN p d. rdBy s' q p d) ⊆ (UN p d. rdBy s q p d)
  by (auto, blast)
ultimately
have blocksOf s' q ⊆ blocksOf s q
  by (auto simp add: blocksOf-def)
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'; HInv4a s q; Inv2a s ] ⇒ HInv4a s' q
by (blast dest: HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q)

theorem HFail-HInv4a:
[ HFail s s'; HInv4a s q ] ⇒ 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; HInv4a s q ] ⇒ HInv4a s' q
by (auto simp add: Phase0Read-def HInv4a-def HInv4a1-def
   HInv4a2-def InitializePhase-def
   blocksOf-def rdBy-def)

theorem HEndPhase2-HInv4a:
[ HEndPhase2 s s' p; HInv4a s q ] ⇒ HInv4a s' q
by (auto simp add: EndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def
   InitializePhase-def blocksOf-def rdBy-def)

lemma allSet:
  assumes aPQ: ∀ a. ∀ r ∈ P a. Q r and rb: rb ∈ P d
  shows Q rb
proof –
from aPQ have ∀ r ∈ P d. Q r by auto
  with rb
  show ?thesis by auto

62
lemma EndPhase0-44:
  assumes act: EndPhase0 s s' p
  and bk: bk ∈ blocksOf s p
  and inv4d: HInv4d s p
  and inv2c: Inv2c-inner s p
  shows ∃ d. ∃ rb ∈ blocksRead s p d. bal bk ≤ mbal(block rb)
proof –
  from bk inv4d
  have ∃ D1 ∈ MajoritySet. ∀ d ∈ D1. bal bk ≤ mbal(disk s d p) — 4.2
    by (auto simp add: HInv4d-def)
  with majorities-intersect
  have p43: ∀ D∈MajoritySet. ∃ d∈D. bal bk ≤ mbal(disk s d p)
    by (simp add: MajoritySet-def, blast)
  from act
  have phase s p = 0 by (simp add: EndPhase0-def)
  with inv2c
  have ∀ d. ∀ rb ∈ blocksRead s p d. block rb = disk s d p — 5.1
    by (simp add: Inv2c-inner-def)
  hence ∀ d. hasRead s p d p
    — (∃ rb ∈ blocksRead s p d. block rb = disk s d p) — 5.2
    by (auto simp add: hasRead-def)
  with act
  have p53: ∃ D ∈ MajoritySet. ∀ d∈D. ?P d
    by (auto simp add: MajoritySet-def EndPhase0-def)
  show ?thesis
proof –
  from p43 p53
  have ∃ D ∈ MajoritySet. (∃ d∈D. bal bk ≤ mbal(disk s d p))
    ∧ (∀ d∈D. ?P d)
    by auto
  thus ?thesis
    by force
qed

lemma HEndPhase0-HInv4a1-p:
  assumes act: HEndPhase0 s s' p
  and inv2a': Inv2a s'
  and inv2c: Inv2c-inner s p
  and inv4d: HInv4d s p
  shows HInv4a1 s' p
proof(auto simp add: HInv4a1-def)
fix bk
assume bk ∈ blocksOf s' p
with HEndPhase0-blocksOf[OF act]
have bk ∈ {dblock s' p} ∪ blocksOf s p by auto
thus \( \text{bal bk} \leq \text{mbal (dblock s') p} \)

**proof**

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

**with** inv2a'

**have** Inv2a-innermost s' p bk

**by**/auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def

**with** bk **show** ?thesis

**by**/auto simp add: Inv2a-innermost-def

**next**

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

**from** act

**have** \( f1: \forall r \in \text{allBlocksRead s p}. \text{mbal r} < \text{mbal (dblock s' p)} \)

**by**/auto simp add: EndPhase0-def

**with** act \( \text{inv4d inv2c bk} \)

**have** \( \exists d. \exists rb \in \text{blocksRead s p d. bal bk} \leq \text{mbal (block rb)} \)

**by**/auto dest: EndPhase0-44

**with** \( f1 \)

**show** ?thesis

**by**/auto simp add: EndPhase0-def allBlocksRead-def allRdBlks-def dest: allSet

qed

lemma hasRead-allBlks:

**assumes** inv2c: Inv2c-inner s p

and \( \text{phase} = 0 \)

**shows** \( \forall d \in \{d. \text{hasRead s p d p}\}. \text{disk s d p} \in \text{allBlocksRead s p} \)

**proof**

**fix** d

**assume** d: d \( \in \{d. \text{hasRead s p d p}\}\) (is d \( \in ?D \))

**hence** br-ne: blocksRead s p d \( \neq \{\} \)

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

**from** inv2c phase

**have** \( \forall br \in \text{blocksRead s p d}. \text{block br} = \text{disk s d p} \)

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

**with** br-ne

**have** \( \text{disk s d p} \in \text{blocksRead s p d} \)

**by** force

**thus** \( \text{disk s d p} \in \text{allBlocksRead s p} \)

**by** (auto simp add: allBlocksRead-def allRdBlks-def)

qed

lemma HEndPhase0-41:

**assumes** act: HEndPhase0 s s' p

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

and inv2c: Inv2c-inner s p

**shows** \( \exists D \in \text{MajoritySet}. \forall d \in D. \text{mbal(disk s d p) \leq mbal(dblock s' p)} \)

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

64
proof
from act HEndPhase0-some[OF act inv1]
have p51: \( \forall \, \text{br} \in \text{allBlocksRead} \, s \, p \cdot \text{mbal}(\text{br}) < \text{mbal}(\text{dblock}(s' \, p)) \)
\[ \land \, \text{bal}(\text{br}) \leq \text{bal}(\text{dblock}(s' \, p)) \]
and a: IsMajority\({\{d. \text{hasRead} \, s \, p \, d \, p}\}}\)
and phase: phase \, s \, p = 0
by (auto simp add: EndPhase0-def)+
from inv2c phase
have (\( \forall \, d \in \{d. \text{hasRead} \, s \, p \, d \, p\} \cdot \text{disk} \, s \, d \, p \in \text{allBlocksRead} \, s \, p \))
by (auto dest: hasRead-allBlks)
with p51
have (\( \forall \, d \in \{d. \text{hasRead} \, s \, p \, d \, p\} \cdot \text{mbal}(\text{disk} \, s \, d \, p) \leq \text{mbal}(\text{dblock}(s' \, p)) \)
\[ \land \, \text{bal}(\text{disk} \, s \, d \, p) \leq \text{bal}(\text{dblock}(s' \, p)) \])
by force
with a show \( ? \text{thesis} \)
by (auto simp add: MajoritySet-def)
qed

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

lemma HEndPhase0-HInv4a2-p:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows HInv4a2 s' p
proof (simp add: HInv4a2-def)
from act
have disk': disk' = disk s
by (simp add: EndPhase0-def)
from act inv1 inv2c
have \( \exists \, D \in \text{MajoritySet} \cdot \forall \, d \in D. \, \text{mbal}(\text{disk} \, s \, d \, p) \leq \text{mbal}(\text{dblock}(s' \, p)) \)
\[ \land \, \text{bal}(\text{disk} \, s \, d \, p) \leq \text{bal}(\text{dblock}(s' \, p)) \]
by (blast dest: HEndPhase0-4T)
from Majority-exQ[OF this]
have \( \forall \, D \in \text{MajoritySet} \cdot \exists \, d \in D. \, \text{mbal}(\text{disk} \, s \, d \, p) \leq \text{mbal}(\text{dblock}(s' \, p)) \)
\[ \land \, \text{bal}(\text{disk} \, s \, d \, p) \leq \text{bal}(\text{dblock}(s' \, p)) \]
(is \( ?P \) (disk s)) .
from subst[OF disk', of ?P, OF this]
show \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal \ (\text{disk s' } d p) \leq \ mbal \ (\text{dblock s' } p) \)
\( \land \ \text{bal} \ (\text{disk s' } d p) \leq \text{bal} \ (\text{dblock s' } p) \).
qed

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

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

66
\textbf{C.4.2 Proofs of Invariant 4b}

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

\textbf{lemma HEndPhase0-dblock-mbal:}
\[ \begin{align*}
& \text{HEndPhase0} s s' p \\
& \implies \forall br \in \text{allBlocksRead} s p. \text{mbal} br < \text{mbal} (\text{dblock} s' p)
\end{align*} \]
by (auto simp add: EndPhase0-def)

\textbf{lemma HEndPhase0-HInv4b-p-dblock:}
\begin{itemize}
  \item \textbf{assumes act:} HEndPhase0 s s' p
  \item \textbf{and inv1:} Inv1 s
  \item \textbf{and inv2a:} Inv2a s
  \item \textbf{and inv2c:} Inv2c-inner s p
\end{itemize}
\textbf{shows} bal (dblock s' p) < mbal (dblock s' p)
\begin{itemize}
  \item \textbf{proof} –
  \item \textbf{from} act \textbf{have} phase s p = 0 \textbf{by} (auto simp add: EndPhase0-def)
  \item \textbf{with} inv2c \textbf{have} \forall d. \forall br \in \text{blocksRead} s p d. \text{proc} br = p \land \text{block} br = \text{disk} s d p
  \textbf{by} (auto simp add: Inv2c-inner-def)
  \item \textbf{hence} allBlks-in-blocksOf: allBlocksRead s p \subseteq blocksOf s p
  \textbf{by} (auto simp add: allBlocksRead-def allRdBlks-def blocksOf-def)
  \item \textbf{from} act \textbf{HEndPhase0-some[OF act inv1]}
  \item \textbf{have p53:} \exists br \in \text{allBlocksRead} s p. \text{bal} (\text{dblock} s' p) = \text{bal} br
  \textbf{by} (auto simp add: EndPhase0-def)
  \item \textbf{from} inv2a \textbf{have} i2: \forall p. \forall bk \in \text{blocksOf} s p. \text{bal} bk \leq \text{mbal} bk
    \textbf{by} (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
  \item \textbf{with} allBlks-in-blocksOf \textbf{have} \forall bk \in \text{allBlocksRead} s p. \text{bal} bk \leq \text{mbal} bk
    \textbf{by} \text{auto}
  \item \textbf{with} p53 \textbf{have} \exists br \in \text{allBlocksRead} s p. \text{bal} (\text{dblock} s' p) \leq \text{mbal} br
    \textbf{by} \text{force}
  \item \textbf{with} HEndPhase0-dblock-mbal[OF act]
  \textbf{show} \textbf{?thesis}
\end{itemize}
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 ∈ blocksOf s p
  shows bal bk < mbal(dblock s' p)
proof –
  from inv4d majorities-intersect bk
  have p43: ∀ D ∈ MajoritySet.∃ d ∈ D. bal bk ≤ mbal(disk s d p)
    by(auto simp add: HInv4d-def MajoritySet-def Majority-exQ)
  have ∃ br ∈ allBlocksRead s p. bal bk ≤ mbal br
proof –
  from act
  have maj: IsMajority({d. hasRead s p d}) (is IsMajority(?D))
    and phase: phase s p = 0
    by(simp add: EndPhase0-def)+
  have br-ne: ∀ d ∈ ?D. blocksRead s d p = {}
    by(auto simp add: hasRead-def)
  from phase inv2c
  have ∀ d ∈ ?D. ∀ br ∈ blocksRead s d p. block br = disk s d p
    by(auto simp add: Inv2c-inner-def)
  with br-ne
  have ∀ d ∈ ?D. ∃ br ∈ allBlocksRead s d p. br = disk s d p
    by(blast dest: blocksRead-allBlocksRead)
  with p43 maj
  show ?thesis
    by(auto simp add: MajoritySet-def)
qed

with HEndPhase0-dblock-mbal[OF act]

show ?thesis
  by auto

qed

lemma HEndPhase0-HInv4b-p:
  assumes act: HEndPhase0 s s' p
  and inv4d: HInv4d s p
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2c: Inv2c-inner s p
  shows HInv4b s' p
proof(clarsimp simp add: HInv4b-def)
  from act
  have phase: phase s p = 0
    by(auto simp add: EndPhase0-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
with $\text{HEndPhase0}-\text{blocksOf}[\text{OF} \; \text{act}]$

have \( \forall bk \in \{\text{dblock} \; s' \; p \} \lor bk \in \text{blocksOf} \; s \; p \)
by blast
thus \( \forall bk < \text{mbal} \; (\text{dblock} \; s' \; p) \)
proof
assume \( bk : bk \in \{\text{dblock} \; s' \; p \} \)
with \( \text{act} \; \text{inv1} \; \text{inv2a} \; \text{inv2c} \)
show \(?\text{thesis}\)
by (auto simp del: $\text{HEndPhase0-def}$
dest: $\text{HEndPhase0-HInv4b-p-dblock}$)
next
assume \( bk : bk \in \text{blocksOf} \; s \; p \)
with \( \text{act} \; \text{inv2c} \; \text{inv4d} \)
show \(?\text{thesis}\)
by (blast dest: $\text{HEndPhase0-HInv4b-p-blocksOf}$)
qed

lemma $\text{HEndPhase0-HInv4b-q}$:
assumes \( \text{act} : \text{HEndPhase0} \; s \; s' \; p \)
and \( \text{pnq} : p \neq q \)
and \( \text{inv} : \text{HInv4b} \; s \; q \)
shows \( \text{HInv4b} \; s' \; q \)
proof
from \( \text{act} \; \text{pnq} \)
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: $\text{EndPhase0-def}$)

from \( \text{act} \; \text{pnq} \)
have \( \text{blocksRead'} : \forall q. \; \text{allRdBlks} \; s' \; q \subseteq \text{allRdBlks} \; s \; q \)
by (auto simp add: $\text{EndPhase0-def InitializePhase-def allRdBlks-def}$)
with \( \text{disk'} \; \text{dblock'} \)
have \( \text{blocksOf} \; s' \; q \subseteq \text{blocksOf} \; s \; q \)
by (auto simp add: $\text{blocksOf-def} \; \text{blocksOf-def rdBy-def, blast}$)
with \( \text{inv} \; \text{phase'} \; \text{dblock'} \)
show \(?\text{thesis}\)
by (auto simp add: $\text{HInv4b-def}$)

qed

theorem $\text{HEndPhase0-HInv4b}$:
assumes \( \text{act} : \text{HEndPhase0} \; s \; s' \; p \)
and \( \text{inv} : \text{HInv4b} \; s \; q \)
and \( \text{inv4d} : \text{HInv4d} \; s \; p \)
and \( \text{inv1} : \text{Inv1} \; s \)
and \( \text{inv2a} : \text{Inv2a} \; s \)
and \( \text{inv2c} : \text{Inv2c-inner} \; s \; p \)
shows \( \text{HInv4b} \; s' \; q \)
proof (cases $p = q$)
case True
with HEndPhase0-HInv4b-p[OF act inv4d inv1 inv2a inv2c]
show ?thesis by simp
next
case False
from HEndPhase0-HInv4b-q[OF act False inv]
show ?thesis .
qed

lemma HStartBallot-HInv4b-p:
assumes act: HStartBallot s s' p
and inv2a: Inv2a-innermost s p (dblock s p)
and inv4b: HInv4b s p
and inv4a: HInv4a s p
shows HInv4b s' p
proof (clarsimp simp add: HInv4b-def)
fix bk
assume bk: bk ∈ blocksOf s' p
from act
have phase': phase s' p = 1
  and phase: phase s p ∈ {1,2}
  by (auto simp add: StartBallot-def)
from act
have p42: mbal (dblock s p) < mbal (dblock s' p)
  ∧ bal(dblock s p) = bal(dblock s' p)
  by (auto simp add: StartBallot-def)
from HStartBallot-blocksOf[OF act] bk
have bk ∈ {dblock s' p} ∪ blocksOf s p
  by blast
thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk ∈ {dblock s' p}
  from inv2a
  have bal (dblock s p) ≤ mbal (dblock s p)
    by (auto simp add: Inv2a-innermost-def)
  with p42 bk
  show ?thesis by auto
next
  assume bk: bk ∈ blocksOf s p
  from phase inv4a
  have p41: HInv4a1 s p
    by (auto simp add: HInv4a-def)
  with p42 bk
  show ?thesis
    by (auto simp add: HInv4a1-def)
qed

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

proof –
from \( \text{act} \ \text{pq} \)
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\((\text{auto simp add: StartBallot-def})\)
from \( \text{act} \ \text{pq} \)
have \( \text{blocksRead}'': \forall \ q. \text{allRdBlks} \ s' \ q \subseteq \text{allRdBlks} \ s \ q \)
by\((\text{auto simp add: StartBallot-def InitializePhase-def allRdBlks-def})\)
with \( \text{disk}' \ \text{dblock}' \)
have \( \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \)
by\((\text{auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast})\)
with \( \text{inv} \ \text{phase}' \ \text{dblock}' \)
show \( \exists \text{thesis} \)
by\((\text{auto simp add: HInv4b-def})\)
qed

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

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

lemma \( \text{HPhase1or2ReadThen-HInv4b-p} \):
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 HPhase1or2ReadThen-blocksOf[OF act]
by (auto simp add: Phase1or2ReadThen-def HInv4b-def)

theorem HPhase1or2ReadThen-HInv4b:
\[
\begin{array}{c}
\text{HPhase1or2ReadThen } s s' p d q; HInv4b s r \\
\text{\Rightarrow } HInv4b s' r
\end{array}
\]
by (blast dest: HPhase1or2ReadThen-HInv4b-p
\quad HPhase1or2ReadThen-HInv4b-q)

theorem HPhase1or2ReadElse-HInv4b:
\[
\begin{array}{c}
\text{HPhase1or2ReadElse } s s' p d q; HInv4b s r; \\
\text{Inv2a } s; HInv4a s p \}
\text{\Rightarrow } HInv4b s' r
\end{array}
\]
using HStartBallot-HInv4b
by (auto simp add: Phase1or2ReadElse-def)

lemma HEndPhase1-HInv4b-p:
HEndPhase1 s s' p \Rightarrow 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': \(\forall q. \text{allRdBlks } s' q \subseteq \text{allRdBlks } s q\)
  by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
with disk' dblock'
have blocksOf s' q \subseteq blocksOf s q

72
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 act pnq
  have disk': disk s' = disk s
    and dblock': dblock s' q = dblock s q
    and phase': phase s' q = phase s q
    by (auto simp add: EndPhase2-def)
  from act pnq
  have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
    by (auto simp add: EndPhase2-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q \subseteq blocksOf s q
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
  show ?thesis
    by (auto simp add: HInv4b-def)

qed

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

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

lemma HEndPhase2-HInv4b-q:
  assumes act: HEndPhase2 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: EndPhase2-def)
  from act pnq
  have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
    by (auto simp add: EndPhase2-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q \subseteq blocksOf s q
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
  show ?thesis
    by (auto simp add: HInv4b-def)

qed

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

proof (cases $p=q$)
  case True
  with $HEndPhase2-HInv4b\_p[\text{OF act}]$
  show $\text{thesis by simp}$
next
  case False
  from $HEndPhase2-HInv4b\_q[\text{OF act False inv}]$
  show $\text{thesis }$.
qed

lemma $HFail-HInv4b\_p$:
  $HFail \ s \ s' \ p \implies HInv4b \ s' \ p$
  by (auto simp add: Fail-def $HInv4b\_def$)

lemma $HFail-HInv4b\_q$:
  assumes act: $HFail \ s \ s' \ p$
  and $\text{pnq: } p \neq q$
  and inv: $HInv4b \ s \ q$
  shows $HInv4b \ s' \ q$
proof (cases $p=q$)
  from act $\text{pnq}$
  have disk\': disk $s'=\text{disk } s$
    and dblock\': dblock $s' \ q=dblock \ s \ q$
    and phase\': phase $s' \ q=\text{phase } s \ q$
    by (auto simp add: Fail-def)
  from act $\text{pnq}$
  have blocksRead\': $\forall q. \ allRdBlks \ s' \ q \subseteq allRdBlks \ s \ q$
    by (auto simp add: Fail-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)
  with inv phase' 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 $HFail-HInv4b\_p[\text{OF act}]$
  show $\text{thesis by simp}$
next
  case False
  from $HFail-HInv4b\_q[\text{OF act False inv}]$
  show $\text{thesis }$.
qed
qed

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

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

theorem HPhase0Read-HInv4b:
  assumes act: HPhase0Read s s' p d
  and inv: HInv4b s q
  shows HInv4b s' q
proof (cases p = q)
  case True
  with HPhase0Read-HInv4b-p[OF act]
  show ?thesis by simp
next
  case False
  from HPhase0Read-HInv4b-q[OF act False inv]
  show ?thesis.
qed

C.4.3 Proofs of Invariant 4c

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

lemma HStartBallot-HInv4c-q:
  assumes act: HStartBallot s s' p
  and inv: HInv4c s q
  and pnq: p ≠ 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: StartBallot-def)
with inv
show \(\text{thesis}\)
by (auto simp add: HInv4c-def)
qed

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

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

76
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:
  [ HPhase1or2ReadThen s s' p d r; HInv4c s q ]
  \implies HInv4c s' q
  by(blast dest: HPhase1or2ReadThen-HInv4c-p
      HPhase1or2ReadThen-HInv4c-q)

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

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

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

theorem HEndPhase1-HInv4c:
[ HEndPhase1 s s' p; HInv4c s q; Inv2b s ] ⇒ HInv4c s' q
by(blast dest: HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q)

lemma HEndPhase2-HInv4c-p:
[ HEndPhase2 s s' p; HInv4c s p ] ⇒ HInv4c s' p
by(auto simp add: EndPhase2-def HInv4c-def)

lemma HEndPhase2-HInv4c-q:
assumes act: HEndPhase2 s s' p
and inv: HInv4c s q
and pnq: p≠q
shows HInv4c s' q
proof –
from act pnq
have phase: phase s' q = phase s q

and dblock: dblock s q = dblock s' q
and disk: disk s' = disk s
by (auto simp add: EndPhase2-def)
with inv
show ?thesis
by (auto simp add: HInv4c-def)
qed

theorem HEndPhase2-HInv4c:
[ HEndPhase2 s s' p; HInv4c s q ] \implies HInv4c s' q
by (blast dest: HEndPhase2-HInv4c-p HEndPhase2-HInv4c-q)

lemma HFail-HInv4c-p:
[ HFail s s' p; HInv4c s p ] \implies HInv4c s' p
by (auto simp add: Fail-def HInv4c-def)

lemma HFail-HInv4c-q:
assumes act: HFail 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: Fail-def)
  with inv
  show ?thesis
  by (auto simp add: HInv4c-def)
qed

theorem HFail-HInv4c:
[ HFail s s' p; HInv4c s q ] \implies HInv4c s' q
by (blast dest: HFail-HInv4c-p HFail-HInv4c-q)

lemma HPhase0Read-HInv4c-p:
[ HPhase0Read s s' p d; HInv4c s p ] \implies HInv4c s' p
by (auto simp add: Phase0Read-def HInv4c-def)

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: phase s' q = phase s q
  and dblock: dblock s q = dblock s' q
and disk: disk s' = disk s
by (auto simp add: Phase0Read-def)
with inv
show ?thesis
by (auto simp add: HInv4c-def)
qed

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

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

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

theorem HEndPhase0-HInv4c:
[ HEndPhase0 s s' p; HInv4c s q ] =⇒ HInv4c s' q
by (blast dest: HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q)

C.4.4 Proofs of Invariant 4d

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

lemma HStartBallot-HInv4d-q:
  assumes act: HStartBallot s s' p
  and inv: HInv4d s q
  and pnq: p ≠ q
  shows HInv4d s' q
proof
  from act pnq
  have disk': disk s'=disk s
  and dblock': dblock s' q=dblock s q
  by(auto simp add: StartBallot-def)
  from act pnq
  have blocksRead': \(∀ q.\) allRdBlks s' q ⊆ allRdBlks s q
  by(auto simp add: StartBallot-def InitializePhase-def allRdBlks-def)
  with disk'
  have blocksOf s' q ⊆ blocksOf s q
  by(auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  from subsetD[OF this] inv
  have \(∀ bk∈blocksOf s' q.\)
    \(∃ D∈MajoritySet.\) \(∀ d∈D.\) bal bk ≤ mbal(disk s d q)
  by(auto simp add: HInv4d-def)
  with disk'
  show ?thesis
  by(auto simp add: HInv4d-def)
qed

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

lemma HPhase1or2Write-HInv4d-p:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv4d s p
  and inv4a: HInv4a s p
shows $HInv_4d \; s' \; p$

proof (clarsimp simp add: HInv4d-def)
  fix $bk$
  assume $bk$: $bk \in \text{blocksOf} \; s'$
  from act
  have $ddisk$: $\forall \; dd. \; \text{disk} \; s' \; dd \; \Rightarrow \; (\text{if} \; d = \; dd \; \text{then} \; \text{dblock} \; s \; p \; \text{else} \; \text{disk} \; s \; dd \; p)$
    and $phase$: $phase \; s \; p \neq 0$
    by (auto simp add: Phase1or2Write-def)
  from inv subsetD [OF HPhase1or2Write-blocksOf [OF act] $bk$]
  have $asm3$: $\exists \; D \in \text{MajoritySet}. \; \forall \; dd \in D. \; \text{bal} \; bk \leq \text{mbal} \; (\text{disk} \; s \; dd \; p)$
    by (auto simp add: HInv4d-def)
  from phase inv4a subsetD [OF HPhase1or2Write-blocksOf [OF act] $bk$] $ddisk$
  have $p41$: $\text{bal} \; bk \leq \text{mbal} \; (\text{disk} \; s' \; d \; p)$
    by (auto simp add: HInv4a-def HInv4a1-def)
  with $ddisk \; asm3$
  show $\exists \; D \in \text{MajoritySet}. \; \forall \; dd \in D. \; \text{bal} \; bk \leq \text{mbal} \; (\text{disk} \; s' \; dd \; p)$
    by (auto simp add: MajoritySet-def split: split-if-asm)
qed

lemma $HPhase1or2Write-HInv4d-q$:
  assumes $act$: $HPhase1or2Write \; s \; s' \; p \; d$
  and $inv$: $HInv_4d \; s \; q$
  and $pnq$: $p \neq q$
  shows $HInv_4d \; s' \; q$

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

theorem $HPhase1or2Write-HInv4d$:
  $[ \; HPhase1or2Write \; s \; s' \; p \; d \; ; \; HInv_4d \; s \; q \; ; \; HInv_4a \; s \; p \; ] \Rightarrow \; HInv_4d \; s' \; q$

82
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 ∈ blocksOf s' p
from act
have bal': bal (dblock s' p) = bal (dblock s p)
  by (auto simp add: Phase1or2ReadThen-def)
from subsetD[OF HPhase1or2ReadThen-blocksOf[OF act] bk] inv
have ∃D∈MajoritySet. ∀d∈D. bal bk ≤ mbal (disk s d p)
  by (auto simp add: HInv4d-def)
with act
show ∃D∈MajoritySet. ∀d∈D. bal bk ≤ mbal (disk s' d p)
  by (auto simp add: Phase1or2ReadThen-def)
qed

lemma HPhase1or2ReadThen-HInv4d-q:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv4d s q
and pnq: p≠q
shows HInv4d s' q
proof
  from act pnq
  have disk': disk s'=disk s
    by (auto simp add: Phase1or2ReadThen-def)
  from act pnq
  have blocksOf s' q ⊆ blocksOf s q
    by (auto simp add: Phase1or2ReadThen-def allRdBlks-def blocksOf-def rdBy-def)
  from subsetD[OF this] inv
  have ∃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 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 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'}
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 ≠ 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': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
by(auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
with disk' dblock'
have \( \text{blocksOf } s' q \subseteq \text{blocksOf } s q \)
by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
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 HEndPhase1-HInv4d:
\[
[ H\text{EndPhase1 } s s' p; H\text{Inv4d } s q; \text{Inv2b } s; H\text{Inv4c } s p ] \implies H\text{Inv4d } s' q
\]
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 (\text{dblock } s' p) = bal (\text{dblock } s p)
by (auto simp add: EndPhase2-def)
from subsetD[OF HEndPhase2-blocksOf[OF act] bk] inv
have \(\exists D \in \text{MajoritySet}. \forall d \in D. \text{bal } bk \leq \text{mbal} (\text{disk } s d p) \)
by (auto simp add: HInv4d-def)
with act
show \(\exists D \in \text{MajoritySet}. \forall d \in D. \text{bal } bk \leq \text{mbal} (\text{disk } s' d p) \)
by (auto simp add: EndPhase2-def)
qed

lemma HEndPhase2-HInv4d-q:
assumes act: HEndPhase2 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
by (auto simp add: EndPhase2-def)
from act pnq
have \(\text{blocksOf } s' q \subseteq \text{blocksOf } s q \)
by (auto simp add: EndPhase2-def InitializePhase-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 HEndPhase2-HInv4d:
\[ [ \text{HEndPhase2} s \ s' \ p; \ HInv4d s \ q] \implies \text{HInv4d } s' \ q \]

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 \in \text{blocksOf } s \ p

from act

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

by (auto simp add: Fail-def)

from subsetD [OF HFail-blocksOf [OF act] bk]

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

proof

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

with inv disk'

show \?thesis

by (auto simp add: HInv4d-def)

next

assume bk: bk \in \{ \text{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)

qed

qed

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

from act pnq

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

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

by (auto simp add: Fail-def)
from act pnq
have blocksRead': '∀ q. allRdBlks s' q ⊆ allRdBlks s q
  by (auto simp add: Fail-def 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)
from subsetD[OF this] inv
have ∀ bk ∈ blocksOf s' q.
  ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s d q)
  by (auto simp add: HInv4d-def)
with disk'
show ?thesis
by (auto simp add: HInv4d-def)
qed

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

lemma HPhase0Read-HInv4d-p:
  assumes act: HPhase0Read s s' p d
  and inv: HInv4d s p
  shows HInv4d s' p
proof (clarsimp simp add: HInv4d-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
  from act
  have bal': bal (dblock s' p) = bal (dblock s p)
    by (auto simp add: Phase0Read-def)
  from subsetD[OF HPhase0Read-blocksOf[OF act] bk] inv
  have ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s d p)
    by (auto simp add: HInv4d-def)
  with act
  show ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s' d p)
    by (auto simp add: Phase0Read-def)
qed

lemma HPhase0Read-HInv4d-q:
  assumes act: HPhase0Read s s' p d
  and inv: HInv4d s q
  and pnq: p ≠ 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 ⊆ blocksOf s q
    by (auto simp add: Phase0Read-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: \(H\text{inv4d-def}\))
with disk'
show \(?thesis\)
by(auto simp add: \(H\text{inv4d-def}\))

qed

theorem \(H\text{Phase0Read-Hinv4d}:\)
[ \(H\text{Phase0Read} s s' p d; H\text{inv4d} s q\] \(\Rightarrow\) \(H\text{inv4d} s' q\)
by(blast dest: \(H\text{Phase0Read-Hinv4d-p H\text{Phase0Read-Hinv4d-q}\})

lemma \(H\text{EndPhase0-blocksOf2}:\)
assumes act: \(H\text{EndPhase0} s s' p\)
and inv2c: \(\text{Inv2c-inner} s p\)
shows allBlocksRead s p \(\subseteq\) blocksOf s p
proof –
from act 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: EndPhase0-def Inv2c-inner-def)
thus \(?thesis\)
by(auto simp add: allBlocksRead-def allRdBlks-def blocksOf-def)

qed

lemma \(H\text{EndPhase0-Hinv4d-p}:\)
assumes act: \(H\text{EndPhase0} s s' p\)
and inv: \(H\text{inv4d} s p\)
and inv2c: \(\text{Inv2c} s\)
and inv1: \(\text{Inv1} s\)
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 subsetD[OF \(H\text{EndPhase0-blocksOf}\[\text{OF act}\] bk]
have \(\exists D \in \text{MajoritySet}. \forall d \in D. \text{bal} bk \leq \text{mbal}(\text{disk} s d p)\)
proof
  assume bk: \(bk \in \text{blocksOf} s p\)
  with inv
  show \(?thesis\)
  by(auto simp add: \(H\text{inv4d-def}\))
next
assume bk: \(bk \in \{ \text{dblock} s' p\}\)
from inv2c
have inv2c-inner: \(\text{Inv2c-inner} s p\)
  by(auto simp add: Inv2c-def)
from bk HEndPhase0-some[OF act inv1] 
HEndPhase0-blocksOf2[OF act inv2c-inner] act
have bal bk ∈ bal ' (blocksOf s p)
  by(auto simp add: EndPhase0-def)
with inv
show ?thesis
  by(auto simp add: HInv4d-def)
qed

with act
show ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s' d p)
  by(auto simp add: EndPhase0-def)
qed

lemma HEndPhase0-HInv4d-q:
assumes act: HEndPhase0 s s' p
and inv: HInv4d s q
and pnq: p ≠ q
shows HInv4d s' q
proof –
from act pnq
have dblock s' q = dblock s q ∧ disk s' = disk s
  by(auto simp add: EndPhase0-def)
moreover
from act pnq
have ∀ p d. rdBy s' q p d ⊆ rdBy s q p d
  by(auto simp add: EndPhase0-def InitializePhase-def rdBy-def)
  hence (UN p d. rdBy s' q p d) ⊆ (UN p d. rdBy s q p d)
    by(auto, blast)
ultimately
have blocksOf s' q ⊆ blocksOf s q
  by(auto simp add: blocksOf-def, blast)
from subsetD[OF this] inv
have ∀ bk ∈ blocksOf s' q.
  ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s d q)
    by(auto simp add: HInv4d-def)
with 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 ∧ HInv2 ∧ HInv4 is also an invariant of HNext.

lemma I2d:
assumes nxt: HNext s s'

89
and \( \text{inv: } H\text{Inv1 }s \land H\text{Inv2 }s \land H\text{Inv2 }s' \land H\text{Inv4 }s \) shows \( H\text{Inv4emento }s' \) proof(auto! simp add: HInv4-def)
fix \( p \) show \( H\text{Inv4a }s' p \) 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 \( H\text{Inv4b }s' p \) 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 \( H\text{Inv4c }s' p \) 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 \( H\text{Inv4d }s' p \) by(auto! simp add: HInv4-def HNext-def Next-def, auto simp add: HInv2-def intro: HStartBallot-HInv4d,
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 \( bal(dblocksp) \).

\[
\begin{align*}
\text{constdefs} & \quad \text{maxBalInp} :: \text{state} \Rightarrow \text{nat} \Rightarrow \text{InputsOrNi} \Rightarrow \text{bool} \\
& \quad \text{maxBalInp} s b v \equiv \forall b_k \in \text{allBlocks} s. \ b \leq \text{bal} b_k \implies \text{inp} b_k = v \\
\text{constdefs} & \quad \text{HInv5-inner-R} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
& \quad \text{HInv5-inner-R} s p \equiv \text{maxBalInp} s (\text{bal}(dblock s p)) (\text{inp}(dblock s p)) \\
& \quad \lor (\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(dblock s p) < \text{mbal}(\text{disk} s d q) \\
& \quad \land \neg \text{hasRead} s p d q)) \\
\text{HInv5-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
& \quad \text{HInv5-inner} s p \equiv \text{phase} s p = 2 \implies \text{HInv5-inner-R} s p \\
\text{HInv5} :: \text{state} \Rightarrow \text{bool} \\
& \quad \text{HInv5} s \equiv \forall p. \ \text{HInv5-inner} s p
\end{align*}
\]

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.
theorem \(HInit-HInv5\): \(HInit \, s \implies HInv5 \, s\)
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 \(action-HInv5-p\) and \(action-HInv5-q\) for each action, for the cases \(p = q\) and \(p \neq q\) respectively.

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

lemma \(HStartBallot-HInv5-p\):
assumes \(act\): 
\(HStartBallot \, s \, s' \, p\)
and \(inv\): \(HInv5-inner \, s \, p\)
shows \(HInv5-inner \, s' \, p\)
by (auto simp add: \(StartBallot-def\) \(HInv5-inner-def\))

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

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

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

lemma HStartBallot-HInv5-q:
assumes act: HStartBallot s s' p
and inv: HInv5-inner s q
and pnq: p ≠ q
shows HInv5-inner s' q
using HStartBallot-HInv5-q1[OF act pnq] HStartBallot-HInv5-q2[OF act pnq]
bym(auto! simp add: HInv5-inner-def HInv5-inner-R-def StartBallot-def)

theorem HStartBallot-HInv5:
[ HStartBallot s s' p; HInv5-inner s q ] ⇒ HInv5-inner s' q
by(blast dest: HStartBallot-HInv5-q1 HStartBallot-HInv5-q2)

lemma HPhase1or2Write-HInv5-1:
assumes act: HPhase1or2Write s s' p d
and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
shows maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
using HPhase1or2Write-blocksOf[OF act]
bym(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: ∃ 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
bym(auto simp add: MajoritySet-def)
have pnq: p ≠ q
proof
from inv4c phase
obtain D1 where r1: IsMajority D1 ∧ (∀ d ∈ D1. mbal(disk s d p) = bal(dblock s p))
bym(auto simp add: HInv4c-def MajoritySet-def)
with i1 majorities-intersect
have D ∩ D1 ≠ {} by auto
then obtain dd where dd ∈ D ∩ D1
bym(auto
with i1 i2 r1
have bal(dblock s p) < mbal(disk s dd q) ∧ mbal(disk s dd p) = bal(dblock s p)
bym(auto
thus ?thesis bym(auto
qed
from act pnq
— dblock and hasRead do not change
have dblock s' = dblock s
and ∀ d. hasRead s' p d q = hasRead s p d q
— In all disks q blocks don’t change
and ∀ d. disk s' d q = disk s d q
by(auto simp add: Phase1or2Write-def hasRead-def)
with i2 i3 majority-nonempty
have ∀ d∈D. bal (dblock s' p) < mbal (disk s' d q) ∧ ¬hasRead s' p d q
by auto
with i2
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: ∀ D∈MajoritySet. ∀ q. ∃ d∈D. bal (dblock s' p) < mbal (disk s' d q)
→ hasRead s' p d q
with act have phase: phase s p = 2
by(auto simp add: Phase1or2Write-def)
show maxBalInp s' (bal (dblock s' p)) (imp (dblock s' p))
proof(rule HPhase1or2Write-HInv5-1[OF act, of p])
from HPhase1or2Write-HInv5-p2[OF act inv4 phase] inv i2 phase
show maxBalInp s (bal (dblock s p)) (imp (dblock s p))
by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

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

lemma HPhase1or2Write-HInv5-q2:
assumes act: HPhase1or2Write s s' p d
and pmq: 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)
95
proof
  from inv5-2
  obtain D qq
    where i1: isMajority D
    and i2: \( \forall d \in D. \ bal(dblock s q) < \ mbal(disk s d qq) \)
    and i3: \( \forall d \in D. \ \neg hasRead s q d qq \)
    by(auto simp add: MajoritySet-def)
  from act pnq
  — dblock and hasRead do not change
  have dblock': dblock s' = dblock s
  and hasread: \( \forall d. \ hasRead s' q d qq = hasRead s q d qq \)
  by(auto simp add: Phase1or2Write-def hasRead-def)
  have \( \forall d \in D. \ bal(dblock s d q) < mbal(disk s' d qq) \land \neg hasRead s' q d qq \)
  proof(cases qq=p)
    case True
    have bal(dblock s q) < mbal(dblock s p)
    proof
      from inv4a act i1
      have \( \exists d \in D. \ mbal(disk s d p) \leq mbal(dblock s p) \)
      by(auto simp add: MajoritySet-def HInv4a-def
        HInv4a2-def Phase1or2Write-def)
      with True i2
      show bal(dblock s q) < mbal(dblock s p)
      by auto
      qed
    with hasread dblock' True i1 i2 i3 act
    show ?thesis
    by(auto simp add: Phase1or2Write-def)
  next
  case False
  with act i2 i3
  show ?thesis
  by(auto simp add: Phase1or2Write-def hasRead-def)
  qed
  with i1
  show ?thesis
  by(auto simp add: MajoritySet-def)
  qed

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

theorem HPhase1or2Write-HInv5:
  [ HPhase1or2Write s s' p d; HInv5-inner s q; HInv4c s p; HInv4a s p ] \implies HInv5-inner s' q
by(blast dest: HPhase1or2Write-HInv5-q 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 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: \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)
  shows \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)
proof -
  from inv5-2
  obtain D q
    where i1: IsMajority D
    and i2: \forall d \in D. bal (dblock s p) < mbal (disk s d q)
    and i3: \forall d \in D. \neg hasRead s p d q
      by(auto simp add: MajoritySet-def)
  from inv2c phase
  have bal (dblock s p) = mbal (dblock s p)
    by(auto simp add: Inv2c-inner-def)
  moreover
  from act have mbal (disk s d r) < mbal (dblock s p)
    by(auto simp add: Phase1or2ReadThen-def)
  moreover
  from i2 have d \in D \implies bal (dblock s p) < mbal (disk s d q) by auto
  ultimately have pnq: d \in D \implies q \neq r by auto
  have pnq: p \neq q
proof -
from inv4c phase
obtain D1 where r1: IsMajority D1 ∧ (∀ d ∈ D1. mbal(disk s d p) = bal(dblock s p))
  by (auto simp add: Hinv4c-def MajoritySet-def)
with i1 majorities-intersect
have D ∩ D1 ≠ {} by auto
then obtain dd where dd ∈ D ∩ D1
  by auto
with i1 i2 r1
have bal(dblock s p) < mbal(disk s d q) ∧ mbal(disk s dd p) = bal(dblock s p)
  by auto
thus ?thesis by auto
qed

from pnr act
have hasRead': ∀ d ∈ D. hasRead s' p d q = hasRead s p d q
  by (auto simp add: Phase1or2ReadThen-def hasRead-def)
from act pnq
— dblock and disk do not change
have dblock s' = dblock s
  and ∀ d. disk s' = disk s
  by (auto simp add: Phase1or2ReadThen-def)
with i2 hasRead' i3
have ∀ d ∈ D. bal(dblock s' p) < mbal(disk s' d q) ∧ ¬ hasRead s' p d q
  by auto
with i1
show ?thesis
  by (auto simp add: MajoritySet-def)
qed

lemma HPhase1or2ReadThen-HInv5-p:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv5-inner s p
and inv4: Hinv4c s p
and inv2c: Inv2c s
shows HInv5-inner s' p
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase s' p = 2
  and i2: ∀ D ∈ MajoritySet. ∀ q. ∃ d ∈ D. bal(dblock s' p) < mbal(disk s' d q)
  —> hasRead s' p d q
with act have phase: phase s p = 2
  by (auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal(dblock s' p)) (inp(dblock s' p))
proof (rule HPhase1or2ReadThen-HInv5-1[OF act, of p])
  from inv2c
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  from HPhase1or2ReadThen-HInv5-p2[OF act inv4 this phase] inv i2 phase
  show maxBalInp s (bal(dblock s p)) (inp(dblock s p))
    by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

98
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 act pnq — dblock and hasRead do not change
  have dblock': dblock s' = dblock s
  and disk': disk s' = disk s
  and hasread: (∀ d. hasRead s' q d qq = hasRead s q d qq)
  by (auto simp add: Phase1or2ReadThen-def hasRead-def)
  with i2 i3
  have (∀ d ∈ D. bal(dblock s' q) < mbal(disk s' d qq) ∧ ¬ hasRead s' q d qq)
  by auto
  with i1
  show ?thesis
  by (auto simp add: MajoritySet-def)
qed

lemma HPhase1or2ReadThen-HInv5-q:
  assumes act: HPhase1or2ReadThen s s' p d r
  and inv: HInv5-inner s q
  and inv4a: HInv4a s p
  and pnq: p ≠ q
  shows HInv5-inner s' q
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
  assume phase': phase s' q = 2
  and i2: ∃ D ∈ MajoritySet. ∃ qa. (∀ d ∈ D. bal(dblock s' q) < mbal(disk s' d qa)
    ∧ ¬ hasRead s' q d qa)
  from phase' act have phase: phase s q = 2
by (auto simp add: Phase1or2ReadThen-def)

show maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
proof (rule HPhase1or2ReadThen-HInv5-1 [OF act, of q])
  from HPhase1or2ReadThen-HInv5-q2 [OF act pnq inv4a] inv i2 phase
  show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

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

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

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

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

lemma HEndPhase2-HInv5-q1:
  assumes act: HEndPhase2 s s' p
  and pnq: p != q
  and inv5-1: maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  shows maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
proof (auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk : allBlocks s'
  and bal: bal (dblock s' q) <= bal bk
  from act pnq
  have dblock': dblock s' q = dblock s q
  by (auto simp add: EndPhase2-def)
  from subsetD [OF HEndPhase2-allBlocks [OF act] bk] inv5-1 dblock' bal
  show inp bk = inp (dblock s' q)
  by (auto simp add: maxBalInp-def)
qed

lemma HEndPhase2-HInv5-q2:
  assumes act: HEndPhase2 s s' p
  and pnq: p != q

100
and \( \text{inv5-2}: \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\text{dblock } s \ q) < \text{mbal}(\text{disk } s \ d \ qq) \)
\[ \land \neg \text{hasRead } s \ q \ d \ qq \]
shows \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\text{dblock } s' \ q) < \text{mbal}(\text{disk } s' \ d \ qq) \)
\[ \land \neg \text{hasRead } s' \ q \ d \ qq \]

proof –
from act pnq
have disk: disk \( s' \) = disk \( s \)
and blocksRead: \( \forall d. \ \text{blocksRead } s' \ q \ d = \text{blocksRead } s \ q \ d \)
and dblock: dblock \( s' \ q = \text{dblock } s \ q \)
by(auto simp add: EndPhase2-def InitializePhase-def)
with inv5-2
show ?thesis
by(auto simp add: hasRead-def)
qed

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

theorem HEndPhase2-HInv5:
\[ \begin{array}{ll}
\text{HEndPhase2} & s \ s' \ p \ \\
\text{HInv5-inner} & s \ q
\end{array} \implies \text{HInv5-inner } s' \ q \]
by(blast dest: HEndPhase2-HInv5-q HEndPhase2-HInv5-p)

lemma HEndPhase1-HInv5-p:
assumes act: HEndPhase1 \( s \ s' \ p \)
and inv4: HInv4 \( s \)
and inv2a: Inv2a \( s \)
and inv2a': Inv2a \( s' \)
and inv2c: Inv2c \( s \)
and asm4: \( \neg \text{maxBalInp } s' (\text{bal}(\text{dblock } s' \ p)) (\text{inp}(\text{dblock } s' \ p)) \)
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 \text{hasRead } s' \ q \ d \ qq \)

proof –
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: maxBalInp-def)
then obtain \( q \) where p32: \( \bk \in \text{blocksOf } s' \ q \)
by(auto simp add: allBlocks-def)
from act
have dblock: \( p \neq q \implies \text{dblock } s' \ q = \text{dblock } s \ q \)
by(auto simp add: EndPhase1-def)
have \( bk \in \text{blocksOf } s \ q \)

proof (cases \( p=q \))

case True

with \( p32 \ p31 \ HEndPhase1\text{-blocksOf}[OF \ act] \)

show ?thesis

by auto

next

case False

from \( \text{dblock}[OF \ False] \ \text{subsetD}[OF \ HEndPhase1\text{-blocksOf}[OF \ act, \ of \ q] \ p32] \)

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 \( \exists q \in \text{UNIV} - \{p\}, \ bk \in \text{blocksOf } s \ q \)

proof -

from \( p22 \)

obtain \( q \) where \( bk: bk \in \text{blocksOf } s \ q \)

by (auto simp add: allBlocks-def)

from \( \text{act } p22 \)

have \( \text{mbal(dblock } s \ p) \leq \ \text{bal} bk \)

by (auto simp add: EndPhase1-def)

moreover

from \( \text{inv4} \)

have \( \text{HInv4b } s \ p \) by (auto simp add: HInv4-def)

ultimately

have \( p \neq q \)

using \( bk \)

by (auto simp add: HInv4-def HInv4b-def)

with \( bk \)

show ?thesis

by auto

qed

then obtain \( q \) where \( p23: q \in \text{UNIV} - \{p\} \ \land \ bk \in \text{blocksOf } s \ q \)

by auto

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

proof -

from \( p23 \ \text{inv4} \)

have \( \text{i4d}: \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal} bk \leq \ \text{mbal(disk } s \ d \ q) \)

by (auto simp add: HInv4-def HInv4d-def)

from \( \text{i4d } p22 \)

102
show \( \exists \)thesis
by force
qed
then obtain \( D \) where \( D_{maj} : D \in \text{MajoritySet} \) and \( p24 : (\forall d \in D. \ \text{bal}(\text{dblock} \ s' \ p) \leq \text{mbal}(\text{disk} \ s \ d \ q)) \)
by auto
have \( p25 : (\forall d \in D. \ \text{bal}(\text{dblock} \ s' \ p) < \text{mbal}(\text{disk} \ s \ d \ q)) \)
proof –
  from \( \text{inv2c} \)
  have \( \text{Inv2c-inner} \ s \ p \)
  by (auto simp add: \( \text{Inv2c-def} \))
  with \( \text{act} \)
  have \( \text{bal-pos} : 0 < \text{bal}(\text{dblock} \ s' \ p) \)
  by (auto simp add: \( \text{Inv2c-inner-def} \text{EndPhase1-def} \))
  with \( \text{inv2a'} \)
  have \( \text{bal}(\text{dblock} \ s' \ p) \in \text{Ballot} \ p \cup \{0\} \)
  by (auto simp add: \( \text{Inv2a-def} \text{Inv2a-inner-def} \text{Inv2a-innermost-def} \text{blocksOf-def} \))
  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: \( \text{Inv2a-def} \))
  hence \( (\forall d \in D. \ \text{mbal}(\text{disk} \ s \ d \ q) \in \text{Ballot} \ q \cup \{0\}) \)
  by (auto simp add: \( \text{Inv2a-def} \text{Inv2a-inner-def} \text{Inv2a-innermost-def} \text{blocksOf-def} \))
  with \( p24 \) \( \text{bal-pos} \)
  have \( (\forall d \in D. \ \text{mbal}(\text{disk} \ s \ d \ q) \in \text{Ballot} \ q) \)
  by force
  with \( \text{Ballot-disj} \ p23 \) \( \text{bal-in-p} \)
  have \( (\forall d \in D. \ \text{mbal}(\text{disk} \ s \ d \ q) \neq \text{bal}(\text{dblock} \ s' \ p)) \)
  by force
  with \( p23 \ p24 \)
  show \( \exists \)thesis
  by force
qed
with \( p23 \) \( \text{act} \)
have \( (\forall d \in D. \ \text{bal}(\text{dblock} \ s' \ p) < \text{mbal}(\text{disk} \ s' \ d \ q) \land \neg \text{hasRead} \ s' \ p \ d \ q) \)
by (auto simp add: \( \text{EndPhase1-def} \text{InitializePhase-def} \text{hasRead-def} \))
with \( D_{maj} \)
show \( \exists \)thesis
by blast
qed
lemma \( \text{union-inclusion} : \)
\[ [ A \subseteq A' ; B \subseteq B' ] \implies A \cup B \subseteq A \cup B' \]
by blast
lemma \( \text{HEndPhase1-blocksOf-q} : \)
assumes \( \text{act} : \text{HEndPhase1} \ s \ s' \ p \)
and \( \text{pnq} : p \neq q \)

103
shows $\text{blocksOf } s' \subseteq \text{blocksOf } s \ q$

proof –

from act \( pnq \)

have \( \text{dblock} : \{\text{dblock } s' q\} \subseteq \{\text{dblock } s \ q\} \)

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

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

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

from \( \text{disk} \)

have \( \text{disk}' : \{\text{disk } s' d q \mid d . \ d \in \text{UNIV}\} \subseteq \{\text{disk } s d q \mid d . \ d \in \text{UNIV}\} \) (is \( ?D' \subseteq ?D \))

by auto

from \( \text{pnq} \ \text{act} \)

have \( (\text{UN } \text{qq } d . \ \text{rdBy } s' q \ qq \ d) \subseteq (\text{UN } \text{qq } d . \ \text{rdBy } s \ qq \ d) \)

by (auto simp add: \text{EndPhase1-def} InitializePhase-def \text{rdBy-def} \text{split: split-if-asm, blast})

hence \( \{\text{block } br \mid br. \ br \in (\text{UN } \text{qq } d . \ \text{rdBy } s' q \ qq \ d)\} \subseteq \{\text{block } br \mid br. \ br \in (UN \text{qq } d . \ \text{rdBy } s \ qq \ d)\} \) (is \( ?R' \subseteq ?R \))

by blast

from union-inclusion[OF \text{dblock} \ \text{union-inclusion}\[OF \text{disk'} \ \text{this}\]]

show \( ?\text{thesis} \)

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

qed

lemma \text{HEndPhase1-allBlocks}:

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

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

proof (auto simp del: \text{HEndPhase1-def} simp add: \text{allBlocks-def} dest: \text{HEndPhase1-blocksOf-q}[OF act] \text{HEndPhase1-blocksOf}[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 \( x=d\text{block } s' \ p \)

proof (cases p=pa)

case True

from \( x-nblks \)

have \( x \notin \text{blocksOf } s \ p \)

by auto

with \( \text{True subsetD}[OF \text{HEndPhase1-blocksOf-q}[OF act] x-pa] \)

show \( ?\text{thesis} \)

by auto

next

case False

from \( x-nblks \ \text{subsetD}[OF \text{HEndPhase1-blocksOf-q}[OF act False] x-pa] \)

show \( ?\text{thesis} \)

by auto

qed

qed

lemma \text{HEndPhase1-Hrsv5-q}:
assumes act: HEndPhase1 s s' p
and inv: HInv5 s
and inv1: Inv1 s
and inv2a: Inv2a s'
and inv2a-q: Inv2a s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and phase': phase s' q = 2
and pnq: p ≠ q
and asm4: ¬ maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
shows (∃ D∈ MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s' q) < mbal(disk s' d qq) ∧ ¬ hasRead s' q d qq))

proof −
from act pnq
have phase s' q = phase s q
and phase-p: phase s p = 1
and disk: disk s' = disk s
and dblock: dblock s' q = dblock s q
and bal: bal(dblock s' p) = mbal(dblock s p)
by(auto simp add: EndPhase1-def InitializePhase-def)
with phase'
have phase: phase s q = 2 by auto
from phase inv2c
have bal-dblk-q: bal(dblock s q) ∈ Ballot q
by(auto simp add: Inv2c-def Inv2c-inner-def)
have ∃ D∈ MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s q) < mbal(disk s d qq) ∧ ¬ hasRead s q d qq)
proof(cases maxBalInp s (bal(dblock s q)) (inp(dblock s q)))
case True
have p21: bal(dblock s q) < bal(dblock s' p) ∧ inp(dblock s q) ≠ inp(dblock s' p)
proof −
from True asm4 dblock HEndPhase1-allBlocks[OF act]
have p32: bal(dblock s q) ≤ bal(dblock s' p) ∧ inp(dblock s q) ≠ inp(dblock s' p)
by(auto simp add: maxBalInp-def)
from inv2a
have bal(dblock s' p) ∈ Ballot p ∪ {0}
by(auto simp add: Inv2a-def Inv2a-inner-def)
moreover
from Ballot-disj Ballot-nzero pnq
have Ballot q ∩ (Ballot p ∪ {0}) = {}
by auto
ultimately
have bal(dblock s' p) ≠ bal(dblock s q)
using bal-dblk-q
by auto
with p32
show ?thesis
by auto
qed
have \( \exists D \in \text{MajoritySet} \cdot \forall d \in D. \ \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d p) \land \text{hasRead} s p d q \) proof -
  from act
  have \( \exists D \in \text{MajoritySet} \cdot \forall d \in D. \ d \in \text{disksWritten} s p \land (\forall q \in \text{UNIV} \setminus \{p\}. \ \text{hasRead} s p d q) \)
  by (auto simp add: EndPhase1-def MajoritySet-def)
then obtain \( D \)
where act1: \( \forall d \in D. \ d \in \text{disksWritten} s p \land (\forall q \in \text{UNIV} \setminus \{p\}. \ \text{hasRead} s p d q) \)
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 pq phase-p bal
have \( \forall d \in D. \ \text{bal}(\text{dblock} s') p = \text{mbal}(\text{disk} s d p) \land \text{hasRead} s p d q \)
by (auto simp add: Inv2b-def Inv2b-inner-def)
with p21 Dmaj
have \( \forall d \in D. \ \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d p) \land \text{hasRead} s p d q \)
by auto
with Dmaj
show ?thesis
by auto
qed
then obtain \( D \)
where p22: \( D \in \text{MajoritySet} \land (\forall d \in D. \ \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d p) \land \text{hasRead} s p d q) \)
by auto
have p23: \( \forall d \in D. \ \langle \text{block}=\text{dblock} s q, \ \text{proc}=q \rangle \notin \text{blocksRead} s p d \)
proof -
  have \( \text{dblock} s q \in \text{allBlocksRead} s p \longrightarrow \text{inp}(\text{dblock} s' p) = \text{inp}(\text{dblock} s q) \)
  proof auto
    assume dblock-q: \( \text{dblock} s q \in \text{allBlocksRead} s p \)
    from inv2a-q
    have \( (\text{bal}(\text{dblock} s q)=0) = (\text{inp}(\text{dblock} s q) = \text{NotAnInput}) \)
    by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def Inv2a-innermost-def)
  with bal-dblk-q Ballot-nzero dblock-q InputsOrNi
  have dblock-q-nib: \( \text{dblock} s q \in \text{nonInitBlks} s p \)
  by (auto simp add: nonInitBlks-def blocksSeen-def)
  with act
  have dblock-max: \( \text{inp}(\text{dblock} s' p)=\text{inp}(\text{maxBlk} s p) \)
  by (auto simp add: EndPhase1-def)
  from maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]
  have max-in-nib: \( \text{maxBlk} s p \in \text{nonInitBlks} s p \).

106
hence nonInitBlks s p ⊆ allBlocks s
  by (auto simp add: allBlocks-def nonInitBlks-def
     blocksSeen-def blocksOf-def rdBy-def
     allBlocksRead-def allRdBlks-def)

with True subsetD[OF this max-in-nib]
have bal (dblock s q) ≤ bal (maxBlk s p) --→ inp (maxBlk s p) = inp (dblock s q)
  by (auto simp add: maxBlk-in-nonInitBlks OF dblock-q-nib inv1]
    dblock-q-nib dblock-max
  show inp(dblock s' p) = inp(dblock s q)
  by auto
qed

have p24: ∃d∈D. ¬(∃br∈blocksRead s q d. bal(dblock s q) ≤ mbal (block br))
proof
  from inv2c phase
  have ∀d. ∃br∈blocksRead s q d. mbal(block br)<mbal(dblock s q)
    and bal(dblock s q) = mbal(dblock s q)
    by (auto simp add: Inv2c-def Inv2c-inner-def)
  thus ?thesis
    by force
qed

have p25: ∀d∈D. ¬hasRead s q d p
proof
  fix d
  assume d-in-D: d ∈ D
  and hasRead-qdp: hasRead s q d p
  have p31: (block=dblock s p, proc=p)∈blocksRead s q d
  proof
    from d-in-D p22
    have hasRead-pdq: hasRead s p d q by auto
      with hasRead-pdq 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]
  HEndPhase1-HInv5-q[OF act inv1 inv2a' inv2a inv2b inv2c inv3, of q]
  by (auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def)

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

lemma HFail-blocksOf-q:
  assumes act: HFail s s' p
  and pnq: p ≠ q
  shows blocksOf s' q ⊆ blocksOf s q
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\}
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 HFail-blocksOf[OF act] x-pa] show \?thesis by auto
next
  case False
  from x-nblks subsetD[OF HFail-blocksOf-q[OF act False] x-pa] show \?thesis by auto
qed

lemma HFail-HInv5-q1:
  assumes act: HFail s s' p
  and pnq: p\#q
  and inv2a: Inv2a-inner s' q
  and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
  shows maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
proof (auto simp add: maxBalInp-def)
fix bk
assume bk: bk \in allBlocks s'
  and bal: bal (dblock s' q) \leq bal bk
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 \in allBlocks s
  with inv5-1 dblock' bal
  show \?thesis by(auto simp add: maxBalInp-def)
next
assume bk: bk \in \{dblock s' p\}
  with act have bk-init: bk = InitDB
by(auto simp add: Fail-def)
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: \exists D\in MajoritySet. \exists qq. (\forall d\in D. bal(dblock s q) < mbal(disk s d qq)
\wedge \neg hasRead s q d qq)
shows \exists D\in MajoritySet. \exists qq. (\forall d\in D. bal(dblock s' q) < mbal(disk s' d qq)
\wedge \neg hasRead s' q d qq)
proof –
  from act pnq
  have disk: disk s' = disk s
  and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
  and dblock: dblock s' q = dblock s q
  by(auto simp add: Fail-def InitializePhase-def)
with inv5-2
show ?thesis
  by(auto simp add: hasRead-def)
qed

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

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

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

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

lemma HPhase0Read-HInv5-1:
  assumes act: HPhase0Read s s' p d
  and inv5-1: maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  shows maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
  using 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: \exists D \in MajoritySet. \exists qq. (\forall d \in D. bal (dblock s q) < mbal (disk s d qq)
  \land \neg hasRead s q d qq)
  \implies \exists D \in MajoritySet. \exists qq. (\forall d \in D. bal (dblock s' q) < mbal (disk s' d qq)
  \land \neg hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
    and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
    and dblock: dblock s' q = dblock s q
    by (auto simp add: Phase0Read-def InitializePhase-def)
  with inv5-2
  show ?thesis
    by (auto simp add: hasRead-def)
qed
lemma HPhase0Read-HInv5-q:
assumes act: HPhase0Read s s' p d
and inv: HInv5-inner s q
and pnq: p \neq q
shows HInv5-inner s' q
proof(auto simp add: HInv5-inner-def HInv5-inner-R-def)
  assume phase': phase s' q = 2
  and i2: \forall D\in MajoritySet. \forall qa. \exists d\in D. bal \(dblock s' q < mbal \(disk s' d qa)
  \rightarrow \{hasRead s' q d qa
  from phase' act have phase: phase s q = 2
  by(auto simp add: Phase0Read-def)
  show maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
  proof(rule HPhase0Read-HInv5-1[OF act, OF q])
  from HPhase0Read-HInv5-q2[OF act pnq] inv i2 phase
  show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

proof

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

lemma HEndPhase0-HInv5-p:
HEndPhase0 s s' p \Rightarrow 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 blocksOf s q
proof
  from act pnq
  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: EndPhase0-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 pnq act
  have (UN qq d. rdBy s' q qq d) \subseteq (UN qq d. rdBy s q qq d)
  by(auto simp add: EndPhase0-def InitializePhase-def
  rdBy-def split: split-if-asm, blast)
hence \{block br \ | \ br \in (UN qq d. rdBy s' q qq d)\} \subseteq
\{block br \ | \ br \in (UN qq d. rdBy s q qq d)\}

112
(is $\mathcal{R}' \subseteq \mathcal{R}$)
by blast
from union-inclusion[$\text{OF } \text{dblock union-inclusion}$[$\text{OF } \text{disk' this}$]]
show $\mathcal{R}$
by(auto simp add: blocksOf-def)
qed

lemma $\text{HEndPhase0-allBlocks}$:
assumes $\text{act}: \text{HEndPhase0 } s \ s' \ p$
shows $\text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{\text{dblock } s' \ p\}$
proof(auto simp del: $\text{HEndPhase0-def}$ simp add: allBlocks-def)
fix $x \ pa$
assume $x\text{-pa}$: $x \in \text{blocksOf } s \ s'$ and
$x\text{-nblks}$: $\forall xa. x \notin \text{blocksOf } s \ xa$
show $\mathcal{R}$
next
fix $bk$
assume $bk$: $bk \in \text{allBlocks } s'$
and $\text{bal}$: $\text{bal } (\text{dblock } s' \ q) \leq \text{bal } bk$
from $\text{act \ pnq}$
have $\text{dblock'}$: $\text{dblock } s' \ q = \text{dblock } s \ q$
by(auto simp add: $\text{EndPhase0-def}$)
from subsetD[$\text{OF } \text{HEndPhase0-allBlocks}[\text{OF } \text{act} ]$ $bk$]
show $\text{inp } bk = \text{inp } (\text{dblock } s' \ q)$
proof
assume $bk$: $bk \in \text{allBlocks } s'$
with $\text{inv5-1 } \text{dblock'}$ $\text{bal}$
show \textit{thesis} \\
by(auto simp add: maxBalInp-def) 
next 
\ass{bk}{bk \in \{\text{dblock } s' p\}} 
with HEndPhase0-some[OF act inv1] act 
\have\exists ba\in\text{allBlocksRead } s p. \bal ba = \bal (\text{dblock } s' p) \land \inp ba = \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: \bal ba = \bal (\text{dblock } s' p) \land \inp ba = \inp (\text{dblock } s' p) 
by auto 
\have allBlocksRead s p \subseteq \text{allBlocks } s 
by(auto simp add: allBlocksRead-def allRdBlks-def allBlocks-def blocksOf-def rdBy-def) 
from subsetD[OF this ba-blksread] ba-balinp bal bk dblock' inv5-1 
show \textit{thesis} 
\by(auto simp add: maxBalInp-def) 
qed 
\newpage 
\lemma HEndPhase0-HInv5-q2: 
\ass{act}{HEndPhase0 } s s' p 
\and pnq: p \neq q 
\and inv5-2: \exists D\in\text{MajoritySet. } \exists qq. \forall d\in D. \bal (\text{dblock } s q) < \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. \bal (\text{dblock } s' q) < \mbal (\text{disk } s' d qq) 
\land \neg \text{hasRead } s' q d qq) 
\proof 
from act pnq 
\have disk: disk s' = disk s 
\and blocksRead: \forall d. \text{blocksRead } s' q d = \text{blocksRead } s q d 
\and dblock: \text{dblock } s' q = \text{dblock } s q 
by(auto simp add: EndPhase0-def InitializePhase-def) 
with inv5-2 
\show \textit{thesis} 
\by(auto simp add: hasRead-def) 
qed 
\newpage 
\lemma HEndPhase0-HInv5-q: 
\ass{act}{HEndPhase0 } s s' p 
\and inv: HInv5-inner s q 
\and inv1: Inv1 s 
\and pnq: p \neq q 
\shows HInv5-inner s' q 
\using 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** \( H_{\text{EndPhase0-HInv5}} \):

\[
H_{\text{EndPhase0}} s s' \quad \text{and} \quad H_{\text{Inv5-inner}} s q \quad \text{and} \quad \text{Inv1} s \quad \Rightarrow \quad H_{\text{Inv5-inner}} s' q
\]

by (blast dest: \( H_{\text{EndPhase0-HInv5-q}} \) \( H_{\text{EndPhase0-HInv5-p}} \))

\( H_{\text{Inv1}} \land H_{\text{Inv2}} \land H_{\text{Inv3}} \land H_{\text{Inv4}} \land H_{\text{Inv5}} \) is an invariant of \( H_{\text{Next}} \).

**Lemma I2e:**

assumes \( \text{nxt:} H_{\text{Next}} s s' \)
and \( \text{inv:} H_{\text{Inv1}} s \land H_{\text{Inv2}} s \land H_{\text{Inv2}} s' \land H_{\text{Inv3}} s \land H_{\text{Inv4}} s \land H_{\text{Inv5}} s \)
shows \( H_{\text{Inv5}} s' \)
by (auto simp add: \( H_{\text{Inv5-def}} \) \( H_{\text{Next-def}} \) \( H_{\text{Next-def}} \),
auto simp add: \( H_{\text{Inv2-def intro: HStartBallot-HInv5}}, \)
auto intro: \( H_{\text{Phase0Read-HInv5}}, \)
auto simp add: \( H_{\text{Inv4-def intro: HPhase1or2Write-HInv5}}, \)
auto simp add: \( H_{\text{Phase1or2Read-def}} \)
intro: \( H_{\text{Phase1or2ReadThen-HInv5}}, \)
\( H_{\text{Phase1or2ReadElse-HInv5}}, \)
auto simp add: \( \text{EndPhase1or2-def HInv1-def HInv4-def HInv5-def} \)
intro: \( H_{\text{EndPhase1-HInv5}}, \)
\( H_{\text{EndPhase2-HInv5}}, \)
auto intro: \( H_{\text{Fail-HInv5}}, \)
auto intro: \( H_{\text{EndPhase0-HInv5 simp add: HInv1-def}} \))
end

**Theory** DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

C.6 Lemma I2f

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

**Constdefs**

\begin{align*}
\text{valueChosen} :: & \text{state} \Rightarrow \text{InputsOrNi} \Rightarrow \text{bool} \\
\text{valueChosen} s v & \equiv \\
& (\exists b \in (\cup \text{p. Ballot p}) . \text{maxBall} s b v) \\
& \land (\exists p. \exists D \in \text{MajoritySet}. (\forall d \in D. \ b \leq \text{bal}(\text{disk} s d p)) \\
& \land (\forall q. (\text{phase} s q = 1 \\
& \land b \leq \text{mbal}(\text{dblock} s q) \\
& \land \text{hasRead} s q d p) \\
& ) \rightarrow (\exists br \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} br))
\end{align*}
lemma $\text{HEndPhase1-valueChosen-inp}$:
assumes act: $\text{HEndPhase1} \ s \ s' \ q$
and inv2a: Inv2a $s$
and asm1: $b \in (\text{UN} \ p. \ \text{Ballot} \ p)$
and bk-blocksOf: $\text{bk}\in \text{blocksOf} \ s \ r$
and bk: $\text{bk}\in \text{blocksSeen} \ s \ q$
and b-bal: $b \leq \text{bal} \ bk$
and asm3: maxBallInp $s \ b \ v$
and inv1: Inv1 $s$
shows $\text{inp}(\text{dblock} \ s' \ q) = v$
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 $0 < \text{bal} \ bk$ by auto
with inv2a-bk
have inp bk $\neq \text{NotAnInput}$
  by (auto simp add: Inv2a-innermost-def)
with bk InputsOrNi
have bk-noninit: $\text{bk}\in \text{nonInitBlks} \ s \ q$
  by (auto simp add: nonInitBlks-def blocksSeen-def
       allBlocksRead-def allRdBlks-def)
with maxBlk-in-nonInitBlks[OF this inv1] b-bal
have maxBlk-b: $b \leq \text{bal} \ (\text{maxBlk} \ s \ q)$
  by auto
from maxBlk-in-nonInitBlks[OF bk-noninit inv1]
have $\exists \ p. \ \text{maxBlk} \ s \ q \in \text{blocksSeen} \ s \ p$
  by (auto simp add: nonInitBlks-def blocksSeen-def
       allBlocksRead-def allRdBlks-def rdBy-def, force)
with maxBlk-b asm3
have inp($\text{maxBlk} \ s \ q$) = $v$
  by (auto simp add: maxBallInp-def allBlocks-def)
with bk-noninit act
show ?thesis
  by (auto simp add: HEndPhase1-def)
qed

lemma $\text{HEndPhase1-maxBallInp}$:
assumes act: $\text{HEndPhase1} \ s \ s' \ q$
and asm1: $b \in (\text{UN} \ p. \ \text{Ballot} \ p)$
and asm2: $D\in \text{MajoritySet}$
and asm3: maxBallInp $s \ b \ v$
and asm4: \( \forall d \in D. \ b \leq \text{bal}(\text{disk s d p}) \)
\( \land (\forall q. (\text{phase s q} = 1 \land b \leq \text{mbal}(\text{dblock s q}) \land \text{hasRead s q d p} ) \rightarrow (\exists br \in \text{blocksRead s q d. } b \leq \text{bal(block br)})) \)

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

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

\( \text{case True} \)

show \( ?\text{thesis} \)

proof
\( \text{cases } p \neq q \)

assume pnq: \( p \neq q \)

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

proof

from act

have IsMajority\( (\{d. \ d \in \text{disksWritten s q} \land (\forall r \in \text{UNIV} - \{q\}. \ \text{hasRead s q d r})\}) \)

(is IsMajority\( (?M) \))

by (auto simp add: EndPhase1-def)

with majorities-intersect asm2

have \( D \cap ?M \neq {} \)

by (auto simp add: MajoritySet-def)

hence \( \exists d \in D. \ (\forall r \in \text{UNIV} - \{q\}. \ \text{hasRead s q d r}) \)

by auto

with pnq

show \( ?\text{thesis} \)

by auto

qed

then obtain \( d \) where \( p41: d \in D \land \text{hasRead s q d p} \) by auto

with asm4 asm3 act True

have \( p42: \exists br \in \text{blocksRead s q d. } b \leq \text{bal(block br)} \)

by (auto simp add: EndPhase1-def)

from True act

have thesis-L: \( b \leq \text{bal} (\text{dblock s' q}) \)

by (auto simp add: EndPhase1-def)

from p42

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

proof auto

fix \( br \)

assume \( br: br \in \text{blocksRead s q d} \)

and \( b-bal: \ b \leq \text{bal} (\text{block br}) \)

hence \( br-\text{rdBy}: br \in (\text{UN q d. } \text{rdBy s (proc br)} \ q d) \)

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

hence \( br-\text{blksOf}: \text{block br} \in \text{blocksOf s (proc br)} \)

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

from \( br \) have \( br-\text{bseen}: \text{block br} \in \text{blocksSeen s q} \)

by (auto simp add: \text{blocksSeen-def allBlocksRead-def allRdBlks-def})
from HEndPhase1-valueChosen-inp[OF act inv2a asm1 br-blksof br-bseen b-bal asm3 inv1]
  show ?thesis .
qed
  with asm3 HEndPhase1-allBlocks[OF act]
  show ?thesis
    by (auto simp add: maxBalInp-def)
next
case False
from asm4
  have p41: \(\forall d \in D. \; b \leq \text{bal}(\text{disk} \; s \; d \; p)\)
    by auto
  have p42: \(\exists d \in D. \; \text{disk} \; s \; d \; p = \text{dblock} \; s \; p\)
proof –
  from act
    have IsMajority \{d. \; d \in \text{disksWritten} \; s \; q \land (\forall p \in \text{UNIV} - \{q\}. \; \text{hasRead} \; s \; q \; d \; p)\} \; (\text{is} \; \text{IsMajority} \; ?S)
      by (auto simp add: EndPhase1-def)
    with majorities-intersect asm2
    have \(D \cap ?S \neq \{}\)
      by (auto simp add: MajoritySet-def)
    hence \(\exists d \in D. \; d \in \text{disksWritten} \; s \; q\)
      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: \(b \leq \text{bal}(\text{dblock} \; s \; q)\) by auto
    have db-blksof: (dblock s q) \in blocksOf s q
      by (auto simp add: blocksOf-def)
    have db-bseen: (dblock s q) \in blocksSeen s q
      by (auto simp add: blocksSeen-def)
    from HEndPhase1-valueChosen-inp[OF act inv2a asm1 db-blksof db-bseen b-bal asm3 inv1]
      show ?thesis .
qed
  with asm3 HEndPhase1-allBlocks[OF act]
  show ?thesis
    by (auto simp add: maxBalInp-def)
next
case False
  have dblock s' q \in allBlocks s'
    by (auto simp add: allBlocks-def blocksOf-def)
  show ?thesis
  proof
    (auto simp add: maxBalInp-def)
fix \( bk \)
assume \( bk: bk \in \text{allBlocks} \ s' \)
and \( b-bal: b \leq \text{bal} \ bk \)
from \( \text{subsetD[OF HEndPhase1-allBlocks[OF act] bk]} \)
show \( \text{inp bk} = v \)
proof
assume \( bk: bk \in \text{allBlocks} \ s \)
with \( \text{asm3 b-bal} \)
show \( \text{thesis} \)
  by\( (\text{auto simp add: maxBalInp-def}) \)
next
assume \( bk: bk \in \{ \text{dblock} \ s' \ q \} \)
from \( \text{act False} \)
have \( \neg \ b \leq \text{bal} \ (\text{dblock} \ s' \ q) \)
  by\( (\text{auto simp add: EndPhase1-def}) \)
with \( bk \ b-bal \)
show \( \text{thesis} \)
  by\( (\text{auto}) \)
qed
qed

lemma \( \text{HEndPhase1-valueChosen2} : \)
assumes \( \text{act: HEndPhase1} \ s \ s' \ q \)
and \( \text{asm4: } \forall \ d \in D. \ b \leq \text{bal} \ (\text{disk} \ s \ d \ p) \)
\( \land (\forall \ q. (\text{phase} \ s \ q = 1 \land \ b \leq \text{mbal} \ (\text{dblock} \ s \ q) \land \text{hasRead} \ s \ q \ d \ p) \longrightarrow (\exists \ br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal} \ (\text{block} \ br)) ) \) (is \( ?P \ s \))
shows \( ?P \ s' \)
proof\( (\text{auto}) \)
fix \( d \)
assume \( d: d \in D \)
with \( \text{act asm4} \)
show \( b \leq \text{bal} \ (\text{disk} \ s' \ d \ p) \)
  by\( (\text{auto simp add: EndPhase1-def}) \)
fix \( d \ q \)
assume \( d: d \in D \)
and \( \text{phase': phase} \ s' \ q = \text{Suc} \ 0 \)
and \( \text{dblk-mbal}: b \leq \text{mbal} \ (\text{dblock} \ s' \ q) \)
with \( \text{act} \)
have \( p\le1: \text{phase} \ s \ q = 1 \)
  and \( p\leq2: \text{dblock} \ s' \ q = \text{dblock} \ s \ q \)
  by\( (\text{auto simp add: EndPhase1-def split: split-if-asm}) \)
with \( \text{dblk-mbal} \)
have \( b \leq \text{mbal} \ (\text{dblock} \ s \ q) \) by \( \text{auto} \)
moreover
assume \( \text{hasRead: hasRead} \ s' \ q \ d \ p \)
with \( \text{act} \)
have \( \text{hasRead} \ s \ q \ d \ p \)
by (auto simp add: EndPhase1-def InitializePhase-def
hasRead-def split: split-if-asn)
ultimately
have \( \exists \br \in \text{blocksRead} \ s \ q \ d \ b \leq \text{bal}(\text{block} \ \br) \)
using p31 asm4 d
by blast
with \( \text{act} \ \text{hasRead} \)
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 \( \text{HEndPhase1-valueChosen} \):
assumes \( \text{act} \): \( \text{HEndPhase1} \ s \ s' \ q \)
and \( \text{vc} \): \( \text{valueChosen} \ s \ v \)
and \( \text{inv1} \): \( \text{Inv1} \ s \)
and \( \text{inv2a} \): \( \text{Inv2a} \ s \)
and \( \text{inv2b} \): \( \text{Inv2b} \ s \)
and \( \text{v-input} \): \( v \in \text{Inputs} \)
shows \( \text{valueChosen} \ s' \ v \)
proof –
from \( \text{vc} \)
obtain \( b \ p \ D \) where
\( \text{asm1} \): \( b \in (\bigcup \ p \ \text{Ballot} \ p) \)
and \( \text{asm2} \): \( D \in \text{MajoritySet} \)
and \( \text{asm3} \): \( \text{maxBalInp} \ s \ b \ v \)
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 \text{mbal}(\text{dblock} \ s \ q) \\
\land \ \text{hasRead} \ s \ q \ d \ p \\
) \rightarrow (\exists \br \in \text{blocksRead} \ s \ q \ d \ b \leq \text{bal}(\text{block} \ \br))) \)
by (auto simp add: valueChosen-def)
from \( \text{HEndPhase1-maxBalInp}\ [\OF \ \text{act} \ \text{asm1} \ \text{asm2} \ \text{asm3} \ \text{asm4} \ \text{inv1} \ \text{inv2a} \ \text{inv2b}] \)
have \( \text{maxBalInp} \ s' \ b \ v \).
with \( \text{HEndPhase1-valueChosen2}\ [\OF \ \text{act} \ \text{asm4}] \ \text{asm1} \ \text{asm2} \)
show ?thesis
by (auto simp add: valueChosen-def)
qed

lemma \( \text{HStartBallot-maxBalInp} \):
assumes \( \text{act} \): \( \text{HStartBallot} \ s \ s' \ q \)
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 \( b-bal \): \( b \leq \text{bal} \ \bk \)
from \( \text{subsetD}\ [\OF \ \text{HStartBallot-allBlocks}\ [\OF \ \text{act}] \ \bk] \)
show \( \text{inp} \ \bk = v \)

120
proof
  assume bk: bk ∈ allBlocks s
  with asm3 b-bal
  show ?thesis
    by (auto simp add: maxBalInp-def)
next
  assume bk: bk ∈ {dblock s' q}
  from asm3
  have b ≤ bal (dblock s q) ⟷ inp (dblock s q) = v
    by (auto simp add: maxBalInp-def allBlocks-def blocksOf-def)
  with act bk b-bal
  show ?thesis
    by (auto simp add: StartBallot-def)
qed
qed

lemma HStartBallot-valueChosen2:
  assumes act: HStartBallot s s' q
      and asm4: ∀ d ∈ D.  b ≤ bal (disk s d p)
           ∧ (∀ q. (phase s q = 1
                        ∧ b ≤ mbal (dblock s q)
                        ∧ hasRead s q d p
                    ) → (∃ br ∈ blocksRead s q d. b ≤ bal (block br))) (is ?P s)
  shows ?P s'
proof (auto)
  fix d
  assume d: d ∈ D
  with act asm4
  show b ≤ bal (disk s' d p)
    by (auto simp add: StartBallot-def)
  fix d q
  assume d: d ∈ D
      and phase': phase s' q = Suc 0
      and dblk-mbal: b ≤ mbal (dblock s' q)
      and hasRead: hasRead s' q d p
  from phase' act hasRead
  have p31: phase s q = 1
          and p32: dblock s' q = dblock s q
    by (auto simp add: StartBallot-def InitializePhase-def
                      hasRead-def split : split-if-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 : split-if-asm)
ultimately
  have ∃ br ∈ blocksRead s q d. b ≤ bal (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: 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 \in Inputs
  shows valueChosen s' v
proof –
  from vc
  obtain b p D where
    asm1: b \in (UN p. \text{Ballot} p)
    and asm2: D\in\text{MajoritySet}
    and asm3: maxBalInp s b v
    and asm4: \( \forall \, d \in D. \, b \leq \text{bal}(\text{disk} \, s \, d \, p) \)
      \( \wedge \forall \, q. ( \text{phase} \, s \, q = 1 \)
      \( \wedge \, b \leq \text{mbal}(\text{dblock} \, s \, q) \)
      \( \wedge \text{hasRead} \, s \, q \, d \, p \)
    ) \rightarrow (\exists \, br \in \text{blocksRead} \, s \, q \, d. \, b \leq \text{bal}(\text{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 \in 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: \( \forall \, d \in D. \, b \leq \text{bal}(\text{disk} \, s \, d \, p) \)

\(\forall q. (\text{phase } s \ q = 1 \wedge b \leq \text{mbal}(\text{dblock } s \ q) \wedge \text{hasRead } s \ q \ d \ p) \rightarrow (\exists br \in \text{blocksRead } s \ q \ d. b \leq \text{bal}(\text{block } br))\) (is \(\gamma P s\))

and \(\text{inv4}: H\text{inv4a } s \ pp\)

shows \(\gamma P s'\)

proof (auto)

fix \(d1\)

assume \(d: d1 \in D\)

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

proof (cases \(d1 = d \wedge pp = p\))

case True

with \(\text{inv4 } \text{act}\)

have \(H\text{inv4a2 } s \ p\)

by (auto simp add: \(\text{Phase1or2Write-def } H\text{inv4a-def}\))

with \(\text{asm2 } \text{majorities-intersect}\)

have \(\exists d1 \in D. \text{bal}(\text{disk } s \ dd \ p) \leq \text{bal}(\text{dblock } s \ p)\)

by (auto simp add: \(H\text{inv4a2-def } \text{MajoritySet-def}\))

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

by auto

from \(\text{asm4 } p41\)

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

by auto

with \(p41\)

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

by auto

from \(\text{act True}\)

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

by (auto simp add: \(\text{Phase1or2Write-def}\))

with \(p42 \ True\)

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

by auto

next

case False

with \(\text{act } \text{asm4 } d\)

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

by (auto simp add: \(\text{Phase1or2Write-def}\))

qed

next

fix \(d \ q\)

assume \(d: d \in D\)

and \(\text{phase' } \text{phase } s' \ q = \text{Suc } 0\)

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

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

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: \(\text{Phase1or2Write-def } \text{InitializePhase-def}\))

\(\text{hasRead-def } \text{split : split-if-asm}\)
with \( \text{dblk-mbal} \)

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

moreover
from act hasRead
have hasRead \( s \ q \ d \ p \)
by (auto simp add: \( \text{Phase1or2Write-def} \text{ InitializePhase-def} \text{ hasRead-def} \) split: split-if-asm)

ultimately
have \( \exists br \in \text{blocksRead } s \ q \ d. \ b \leq \text{bal}(\text{block } br) \)
using \( p \exists! \text{asm4} \ d \)
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: \( \text{Phase1or2Write-def} \text{ InitializePhase-def} \text{ hasRead-def} \))

qed

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

qed

lemma \( \text{HPhase1or2ReadThen-maxBalInp} \):
assumes act: \( \text{HPhase1or2ReadThen } s \ s' \ q \ d \ p \)
and \( \text{asm3: maxBalInp } s \ b \ v \)
shows \( \text{maxBalInp } s' \ b \ v \).
proof (auto simp add: maxBalInp-def)
fix \( bk \)
\[ \text{assume } bk : bk \in \text{allBlocks } s' \]
\[ \text{and } b{\text{-bal}} : b \leq \text{bal } bk \]
\[ \text{from } \text{subsetD[OF } H\text{Phase1or2ReadThen-allBlocks[OF } \text{act} ] bk] \text{ asm3 } b{\text{-bal}} \]
\[ \text{show } inp bk = v \]
\[ \text{by(auto simp add: maxBalInp-def)} \]
\[ \text{qed} \]

\text{lemma } H\text{Phase1or2ReadThen-valueChosen2} : 
\[ \text{assumes } \text{act}: H\text{Phase1or2ReadThen } s \ s' q \ d \ pp \]
\[ \text{and } \text{asm4}: \forall d \in D. \ b \leq \text{bal}(disk s \ d \ p) \]
\[ \land(\forall q.( \quad \text{phase } s \ q = 1 \]
\[ \land b \leq \text{mbal}(d\text{block } s \ q) \]
\[ \land \text{hasRead } s \ q \ d \ p \quad ) \rightarrow (\exists br \in \text{blocksRead } s \ q \ d. \ b \leq \text{bal}(\text{block } br))) \ (\text{is } \ ?P s) \]
\[ \text{shows } \ ?P s' \]
\[ \text{proof(auto)} \]
\[ \text{fix } dd \]
\[ \text{assume } d : dd \in D \]
\[ \text{with } \text{act } \text{asm4} \]
\[ \text{show } b \leq \text{bal}(\text{disk } s' \ dd \ p) \]
\[ \text{by(auto simp add: } H\text{Phase1or2ReadThen-def)} \]
\[ \text{fix } dd \ qq \]
\[ \text{assume } d : dd \in D \]
\[ \text{and } \text{phase': phase } s' \ qq = \text{Suc 0} \]
\[ \text{and } \text{dblk-mbal: } b \leq \text{mbal}(d\text{block } s' \ qq) \]
\[ \text{and } \text{hasRead: hasRead } s' \ qq \ dd \ p \]
\[ \text{show } \exists br \in \text{blocksRead } s' \ qq \ dd. \ b \leq \text{bal}(\text{block } br) \]
\[ \text{proof(cases } d=dd \land qq=q \land pp=p) \]
\[ \text{case True} \]
\[ \text{from } d \text{ asm4} \]
\[ \text{have } b \leq \text{bal}(\text{disk } s \ dd \ p) \]
\[ \text{by } \text{auto} \]
\[ \text{with } \text{act True} \]
\[ \text{show } \ ?\text{thesis} \]
\[ \text{by(auto simp add: } H\text{Phase1or2ReadThen-def)} \]
\[ \text{next} \]
\[ \text{case False} \]
\[ \text{with } \text{phase': act} \]
\[ \text{have } p31: \text{phase } s \ qq = 1 \]
\[ \text{and } p32: \text{dblock } s' \ qq = \text{dblock } s \ qq \]
\[ \text{by(auto simp add: } H\text{Phase1or2ReadThen-def)} \]
\[ \text{with } \text{dblk-mbal} \]
\[ \text{have } b \leq \text{mbal}(\text{dblock } s \ qq) \text{ by } \text{auto} \]
\[ \text{moreover} \]
\[ \text{from } \text{act hasRead False} \]
\[ \text{have hasRead } s \ qq \ dd \ p \]
\[ \text{by(auto simp add: } H\text{Phase1or2ReadThen-def} \]
\[ \text{hasRead-def split: split-if-as} \]
\[ \text{ultimately} \]
have $\exists b r \in \text{blocksRead} \ s \ q q \ d d. \ b \leq \text{bal(block} \ br)$

using p31 asm4 d

by blast

with act hasRead

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

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

qed

theorem HPhase1or2ReadThen-valueChosen:

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

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: $b \in \bigcup p. \text{Ballot} \ p$

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

and asm3: maxBalInp $s \ b \ v$

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

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

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

$\land \ hasRead \ s \ q \ d \ p)

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

by (auto simp add: valueChosen-def)

from HPhase1or2ReadThen-maxBalInp[OF act asm3]

have maxBalInp $s' \ b \ v$.

with HPhase1or2ReadThen-valueChosen2[OF act asm4] asm1 asm2

show ?thesis

by (auto simp add: valueChosen-def)

qed

theorem HPhase1or2ReadElse-valueChosen:

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

$\Rightarrow \text{valueChosen} \ s' \ v$

using HStartBallot-valueChosen

by (auto simp add: Phase1or2ReadElse-def)

lemma HEndPhase2-maxBalInp:

assumes act: HEndPhase2 $s \ s' \ q$

and asm3: maxBalInp $s \ b \ v$

shows maxBalInp $s' \ b \ v$

proof (auto simp add: maxBalInp-def)

fix bk

assume bk: bk $\in \text{allBlocks} \ s'$

and b-bal: $b \leq \text{bal} \ bk$

from subsetD[OF HEndPhase2-allBlocks[OF act] bk] asm3 b-bal

show inp bk = v
by (auto simp add: maxBalInp-def)
qed

lemma HEndPhase2-valueChosen2:
assumes act: HEndPhase2 s s' q
and asm4: \( \forall d \in D. \quad b \leq \text{bal}(\text{disk} \ s \ d \ p) \)
  \& (\forall q. (\text{phase} \ s \ q = 1
  \& b \leq \text{mbal}(\text{dblock} \ s \ q)
  \& \text{hasRead} \ s \ q \ d \ p)
) \longrightarrow (\exists br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ br))) \ (\text{is} \ ?P \ s)
shows ?P s'
proof (auto)
fix d
assume d: d \in D
with act asm4
show b \leq \text{bal} (\text{disk} \ s' \ d \ p)
  by (auto simp add: EndPhase2-def)
fix d q
assume d: d \in D
and phase': \text{phase} \ s' \ q = \text{Suc} \ 0
and dblk-mbal: b \leq \text{mbal} (\text{dblock} \ s' \ q)
and hasRead: \text{hasRead} \ s' \ q \ d \ p
from phase' act hasRead
have p31: \text{phase} \ s \ q = 1
  and p32: \text{dblock} \ s' \ q = \text{dblock} \ s \ q
  by (auto simp add: EndPhase2-def InitializePhase-def
       hasRead-def split: split-if-asm)
with dblk-mbal
have b \leq \text{mbal}(\text{dblock} \ s \ q)
  by auto
moreover
from act hasRead
have hasRead \ s \ q \ d \ p
  by (auto simp add: EndPhase2-def InitializePhase-def
       hasRead-def split: split-if-asm)
ultimately
have \exists \text{br} \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ \text{br})
  using p31 asm4 d
by blast
with act hasRead
show \exists \text{br} \in \text{blocksRead} \ s' \ q \ d. \ b \leq \text{bal}(\text{block} \ \text{br})
  by (auto simp add: EndPhase2-def InitializePhase-def
       hasRead-def)
qed

theorem HEndPhase2-valueChosen:
assumes act: HEndPhase2 s s' q
and vc: valueChosen s v
and v-input: v \in \text{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 HEndPhase2-maxBalInp[OF act asm3]
have maxBalInp s' b v.
with HEndPhase2-valueChosen2[OF act asm4] asm1 asm2
show ?thesis
by (auto simp add: valueChosen-def)
qed

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

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

theorem \textit{HFail-valueChosen}:
\begin{itemize}
\item assumes \texttt{act}: \textit{HFail} \texttt{s s' q}
\item and \texttt{vc}: \textit{valueChosen} \texttt{s v}
\item and \texttt{v-input}: \(v \in \text{Inputs}\)
\item shows \textit{valueChosen} \texttt{s' v}
\item proof --
\item from \texttt{vc}
\end{itemize}
obtain \( b \ p \ D \) where

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

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

from \(H\text{Fail-maxBalInp}[OF \text{act asm1 asm3}]\)

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

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

show \(?\text{thesis}\)

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

qed
and hasRead: hasRead $s'$ $q$ $d$ $p$

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

theorem HPhase0Read-valueChosen:
  assumes act: HPhase0Read $s$ $s'$ $q$ $d$
  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: $b \in (\text{UN} p. \text{Ballot} p)$
    and asm2: $D \in \text{MajoritySet}$
    and asm3: maxBallInp $s$ $b$ $v$
    and asm4: $\forall d \in D. \ b \leq \text{bal(disk} s d p)$
      $\land (\forall q. (\begin{array}{l} \text{phase} \ s \ q = 1 \\
        \land b \leq \text{mbal(dblock s q)} \\
        \land \text{hasRead} \ s \ q \ d \ p \\
        \end{array}) \rightarrow (\exists br \in \text{blocksRead} s q d. \ b \leq \text{bal(block br)})))$
  by (auto simp add: valueChosen-def)
from HPhase0Read-maxBallInp[of act asm3]
have maxBallInp $s'$ $b$ $v$.
with HPhase0Read-valueChosen2[of act asm4] asm1 asm2
show ?thesis

131
by (auto simp add: valueChosen-def)

qed

**lemma** HEndPhase0-maxBalInp:
**assumes** act: HEndPhase0 s s' q
and asm3 : maxBalInp s b v
and inv1 : Inv1 s
**shows** maxBalInp s' b v
**proof** (auto simp add: maxBalInp-def)
fix bk
assume bk : bk ∈ allBlocks s'
and b-bal : b ≤ bal bk
from subsetD [OF HEndPhase0-allBlocks [OF act] bk]
show inp bk = v
proof
assume bk : bk ∈ allBlocks s
with asm3 b-bal
show ?thesis
by (auto simp add: maxBalInp-def)
next
assume bk : bk ∈ {dblock s' q}
with HEndPhase0-some [OF act inv1] act
have ∃ ba ∈ allBlocksRead s q. bal ba = bal (dblock s' q) ∧ inp ba = inp (dblock s' q)
by (auto simp add: EndPhase0-def)
then obtain ba
where ba-blksread : ba ∈ allBlocksRead s q
and ba-balinp : bal ba = bal (dblock s' q) ∧ inp ba = inp (dblock s' q)
by auto
have allBlocksRead s q ⊆ allBlocks s
by (auto simp add: allBlocksRead-def allRdBlks-def allBlocks-def blocksOf-def rdBy-def)
from subsetD [OF this ba-blksread] ba-balinp bk b-bal asm3
show ?thesis
by (auto simp add: maxBalInp-def)
qed

**lemma** HEndPhase0-valueChosen2:
**assumes** act: HEndPhase0 s s' q
and asm4 : ∀ d ∈ D. b ≤ bal (disk s d p)
and (∀ 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 $d: d \in D$
with act asm4
show $b \leq \text{bal}(\text{disk } s' d p)$
  by (auto simp add: EndPhase0-def)
fix $d q$
assume $d: d \in D$
and phase': phase' $s' q = \text{Suc } 0$
and dblk-mbal: $b \leq \text{mbal}(\text{dblock } s' q)$
and hasRead: hasRead $s' q d p$
from phase' act hasRead
have $p\exists 1: \text{phase } s q = 1$
and $p\exists 2: \text{dblock } s' q = \text{dblock } s q$
  by (auto simp add: EndPhase0-def InitializePhase-def
       hasRead-def split : split-if-asm)
with dblk-mbal
have $b \leq \text{mbal}(\text{dblock } s q)$ by auto
moreover
from act hasRead
have hasRead $s q d p$
  by (auto simp add: EndPhase0-def InitializePhase-def
       hasRead-def split : split-if-asm)
ultimately
have $\exists \text{br}\in\text{blocksRead } s q d. b \leq \text{bal}(\text{block } \text{br})$
  using $p\exists 1$ asm4 $d$
  by blast
with act hasRead
show $\exists \text{br}\in\text{blocksRead } s' q d. b \leq \text{bal}(\text{block } \text{br})$
  by (auto simp add: EndPhase0-def InitializePhase-def
       hasRead-def)
qed

theorem HEndPhase0-valueChosen:
  assumes act: $\text{HEndPhase0 } s s' q$
  and vc: $\text{valueChosen } s v$
  and v-input: $v \in \text{Inputs}$
  and inv1: $\text{Inv1 } s$
  shows $\text{valueChosen } s' v$
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: $\text{maxBalInp } s b v$
    and asm4: $\forall d \in D. b \leq \text{bal}(\text{disk } s d p)$
       \& $(\forall q. (\text{phase } s q = 1$
       \& $b \leq \text{mbal}(\text{dblock } s q)$
       \& hasRead $s q d p$)$\rightarrow (\exists \text{br}\in\text{blocksRead } s q d. b \leq \text{bal}(\text{block } \text{br}))$
  by (auto simp add: valueChosen-def)
theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

constsdefs
  $HInv6 :: state \Rightarrow bool$
  $HInv6 s \equiv (chosen s \neq NotAnInput \rightarrow valueChosen s (chosen s))$
  $\land (\forall p. outpt s p \in \{chosen s, NotAnInput\})$

theorem $HInit-HInv6: HInit s \Rightarrow HInv6 s$
  by (auto simp add: $HInit$-def $Init$-def $InitDB$-def $HInv6$-def)

lemma $HEndPhase2-Inv6-1:
  assumes act: $HEndPhase2 s s' p$
  and inv: $HInv6 s$
  and inv2b: $Inv2b s$
  and inv2c: $Inv2c s$
  and inv3: $HInv3 s$
  and inv5: $HInv5-inner s p$
  and chosen': $chosen s' \neq NotAnInput$
  shows $valueChosen s' (chosen s')$
proof (cases chosen s\neq NotAnInput)
  from $\text{inv5 act}$
  have inv5R: $HInv5-inner-R s p$
    and phase: $phase s p = 2$
    and ep2-maj: $\text{IsMajority} \{d . \ d \in disksWritten s p$
    $\land (\forall q \in UNIV - \{p\}. \hasRead s p d q)\}$
    by (auto simp add: $EndPhase2$-def $HInv5$-inner-def)
  case True
  have $p$2: $maxBalInp s (bal(dblock s p)) (inp(dblock s p))$
  proof
    have $\neg (\exists D \in \text{MajoritySet}. \exists q. \ (\forall d \in D. \ bal (dblock s p) < \mbal (disk s d q) \land$
      $\neg \hasRead s p d q))$
      proof auto
  qed
end
fix $D$ $q$
assume $Dmaj': D \in \text{MajoritySet}$
from $ep2-maj$ $Dmaj$ $\text{majorities-intersect}$
have $\exists d \in D. \ d \in \text{disksWritten s p}$
  $\land (\forall q \in \text{UNIV} - \{p\}. \ hasRead s p d q)$
  by(auto simp add: MajoritySet-def, blast)
then obtain $d$
  where $dinD: d \in D$
    and $ddisk: d \in \text{disksWritten s p}$
    and $dhasR: \forall q \in \text{UNIV} - \{p\}. \ hasRead s p d q$
    by auto
from $inv2b$
have $\text{Inv2b-inner s p d}$
  by(auto simp add: $Inv2b-def$)
with $ddisk$
have $\text{disk s d p = dblock s p}$
  by(auto simp add: $Inv2b-inner-def$)
with $inv2c$ phase
have $\text{bal (dblock s p) = mbal (disk s d p)}$
  by(auto simp add: $Inv2c-def$ $Inv2c-inner-def$)
with $dhasR$ $dinD$
show $\exists d \in D. \ \text{bal (dblock s p) < mbal (disk s d q) \longrightarrow hasRead s p d q}$
  by auto
qed
with $inv5R$
show $\text{thesis}$
  by(auto simp add: $HInv5-inner-R-def$)
qed
have $p33: \text{maxBalInp s'} (\text{bal (dblock s' p)}) (\text{chosen s'})$
proof
  from $\text{act}$
  have $\text{outpt'}: \text{outpt s'} = (\text{outpt s}) (p := \text{inp (dblock s p)})$
    by(auto simp add: $EndPhase2-def$)
  have $\text{outpt'}-q: \forall q. \ p \neq q \longrightarrow \text{outpt s'} q = \text{NotAnInput}$
  proof auto
  qed
  from $\text{True act chosen'}$
  have $\text{chosen s'} = \text{inp (dblock s p)}$
  proof(auto simp add: $HNextPart-def$ $split: split-if-asm$)
  fix $pa$
  assume $\text{outpt'}-pa: \text{outpt s'} p a \neq \text{NotAnInput}$
  qed
from output\' - q
have someeq2: \( \bigwedge p.\ output s' p \neq NotAnInput \implies \) \( p = p\)
  by auto
with output\' - pa
have output s' p \neq NotAnInput
  by auto
from some-equality[of \( \lambda p.\ output s' p \neq NotAnInput\), OF this someeq2]
have (SOME p. output s' p \neq NotAnInput) = p .
with output'
show output s' (SOME p. output s' p \neq 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 \text{maxBalInp s (bal (dblock s' p)) (chosen s')}
  using p32
  by auto
with HEndPhase2-allBlocks[OF act]
show \text{thesis}
  by (auto simp add: maxBalInp-def)
qed
from ep2-maj inv2b majorities-intersect
have \( \exists D \in \text{MajoritySet}. \ (\forall d \in D.\ disk s d p = d\text{block s p}) \)
  \( \wedge (\forall q \in \text{UNIV} - \{p\}.\ hasRead s p d q)\)
  by (auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def)
then obtain D
where Dmaj: \( D \in \text{MajoritySet} \)
and p34: \( \forall d \in D.\ disk s d p = d\text{block s p} \)
  \( \wedge (\forall q \in \text{UNIV} - \{p\}.\ hasRead s p d q)\)
  by auto
have p35: \( \forall q.\forall d \in D.\ ( phase s q = 1 \wedge bal(d\text{block s p}) \leq mbal(d\text{block s p}) \wedge hasRead s q d p) \)
  \( \rightarrow (\langle \text{block}=d\text{block s p},\ proc=p\rangle) \in \text{blocksRead s q d} \)
proof auto
fix q d
assume dD: \( d \in D \) and phase-q: \( phase s q = Suc 0 \)
and bal-mbal: \( bal(d\text{block s p}) \leq mbal(d\text{block s q}) \) and hasRead: \( hasRead s q d p \)
from phase inv2c
have bal (dblock s p) = mbal (dblock s p)
  by (auto simp add: Inv2c-def Inv2c-inner-def)
moreover
from inv2c phase
have \( \forall br \in \text{blocksRead s p d}.\ mbal(block br) < mbal(d\text{block s p}) \)
  by (auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
have p41: \( \langle \text{block}=d\text{block s q},\ proc=q\rangle \notin \text{blocksRead s p d} \)
using \texttt{bal-mbal} by \texttt{auto}

from \texttt{phase q}

have \texttt{p\#q} by \texttt{auto}

with \texttt{p34 dD}

have \texttt{hasRead s p d q} by \texttt{auto}

with \texttt{phase q hasRead inv3 p41}

show \texttt{\{block = dblock s p, proc = p\} \in blocksRead s q d}

by (\texttt{auto simp add: HInv3-def HInv3-inner-def HInv3-L-def HInv3-R-def})

qed

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

\quad \longrightarrow (\exists br \in blocksRead s' q d. bal(block br) = bal(dblock s p))

proof (\texttt{auto})

\quad \texttt{fix q d}

\quad \texttt{assume dD: d \in D and phase-q: phase s' q = Suc 0}

\quad \quad and bal: bal (dblock s p) \leq mbal (dblock s' q)

\quad \quad and hasRead: hasRead s' q d p

\quad from \texttt{phase-q act}

\quad have \texttt{phase s' q = phase s q \land dblock s' q = dblock s q \land hasRead s' q d p = hasRead s q d p}

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

\quad with \texttt{p35 phase-q bal hasRead dD}

\quad have \texttt{\{block = dblock s p, proc = p\} \in blocksRead s' q d d}

\quad \quad by \texttt{auto}

\quad thus \exists br \in blocksRead s' q d. bal(block br) = bal(dblock s p)

\quad \quad by \texttt{force}

\quad qed

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

\quad \quad \longrightarrow (\exists br \in blocksRead s' q d. bal(dblock s p) \leq bal(block br))

\quad \quad by \texttt{force}

from \texttt{act}

\quad have \texttt{bal-dblock: bal(dblock s' p) = bal(dblock s p)}

\quad \quad and \texttt{disk: disk s' = disk s}

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

\quad from \texttt{bal-dblock p33}

\quad have \texttt{maxBalInp s' (bal(dblock s p)) (chosen s')}

\quad \quad by \texttt{auto}

\quad moreover

\quad from \texttt{disk p34}

\quad have \texttt{\forall d\in D. bal(dblock s p) \leq bal(disk s' d p)}

\quad \quad by \texttt{auto}

\quad ultimately

\quad have \texttt{maxBalInp s' (bal(dblock s p)) (chosen s')} 

\quad \quad \quad \quad (\exists D \in \texttt{MajoritySet.} 

\quad \quad \quad \quad \quad \forall d\in D. bal(dblock s p) \leq bal \ (disk \ s' \ d \ p) \land
\[ \forall q. \text{phase } s' q = \text{Suc } 0 \land \]
\[ \text{bal(dblock } s \ p) \leq \text{mbal(dblock } s' q) \land \text{hasRead } s' q d p \rightarrow \]
\[ (\exists br \in \text{blocksRead } s' q d. \text{bal(dblock } s \ p) \leq \text{bal(block } br))) \]

using \( p36-2 \text{ Dmaj } \)
by auto

moreover
from phase inv2c
have \( \text{bal(dblock } s \ p) \in \text{Ballot } p \)
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 (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: \( \forall d \in D. \ b \leq \text{bal (disk } s \ d \ p) \)
and max-w: \( \text{maxBalInp } s \ ba \ w \)
and Damaj: \( Da \in \text{MajoritySet} \)
and asm-w: \( \forall d \in Da. \ ba \leq \text{bal (disk } s \ d \ pa) \)
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 dinmaj: \( d \in D \cap Da \) and dab: \( \text{disk } s \ d \ pa \in \text{allBlocks } s \)
by auto
with asm-w
have ba: \( ba \leq \text{bal (disk } s \ d \ pa) \)
by auto
with b-ba
have \( b \leq \text{bal (disk } s \ d \ pa) \)
by auto

138
with max-v dab
have v-value: inp (disk s d pa) = v
  by (auto simp add: maxBalInp-def)
from ba max-w dab
have w-value: inp (disk s d pa) = w
  by (auto simp add: maxBalInp-def)
with v-value
show ?thesis by auto
qed

lemma valueChosen-equal:
  assumes v: valueChosen s v
  and w: valueChosen s w
  shows v = w
proof (auto)
fix a b aa ba p D pa Da
assume max-v: maxBalInp s b v
  and Dmaj: D ∈ MajoritySet
  and asm-v: ∀ d ∈ D. b ≤ bal (disk s d p) ∧
    (∃ q. phase s q = Suc 0 ∧
    b ≤ mbal (dblock s q) ∧ hasRead s q d p →
    (∃ br ∈ blocksRead s q d. b ≤ bal (block br))))
  and max-w: maxBalInp s ba w
  and Damaj: Da ∈ MajoritySet
  and asm-w: ∀ d ∈ Da. ba ≤ bal (disk s d pa) ∧
    (∃ q. phase s q = Suc 0 ∧
    ba ≤ mbal (dblock s q) ∧ hasRead s q d pa →
    (∃ br ∈ blocksRead s q d. ba ≤ bal (block br))))
from asm-v
have asm-v: ∀ d ∈ D. b ≤ bal (disk s d p) by auto
from asm-w
have asm-w: ∀ d ∈ Da. ba ≤ bal (disk s d pa) by auto
show v = w
proof (cases b ≤ ba)
  case True
    from valueChosen-equal-case [OF max-v Dmaj asm-v max-w Damaj asm-w True]
    show ?thesis .
next
  case False
    from valueChosen-equal-case [OF max-w Damaj asm-w max-v Dmaj asm-v]
    False
    show ?thesis
      by auto
qed

lemma HEndPhase2-Inv6-2:
  assumes act: HEndPhase2 s s′ p
  and inv: Hrwb6 s
and \texttt{inv2b}: Inv2b \; s
and \texttt{inv2c}: Inv2c \; s
and \texttt{inv3}: HInv3 \; s
and \texttt{inv5}: HInv5-inner \; s \; p
and \texttt{asm}: outpt \; s' \; r \neq \text{NotAnInput}

\textbf{shows} \; \text{outpt} \; s' \; r = \text{chosen} \; s'

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

\textbf{case} \; \text{True}
\textbf{with} \; \texttt{inv2c}
\begin{itemize}
\item \texttt{have} \; \forall \; q. \; \text{outpt} \; s \; q = \text{NotAnInput}
\begin{itemize}
\item \textbf{by} (\texttt{auto simp add: Inv2c-def Inv2c-inner-def})
\end{itemize}
\end{itemize}
\textbf{with} \; \texttt{True \; act \; asm}
\textbf{show} \; ?\texttt{thesis} (\texttt{auto simp add: EndPhase2-def HNextPart-def split: split-if-asm})

\textbf{next}
\begin{itemize}
\item \texttt{case} \; \text{False}
\textbf{with} \; \texttt{inv}
\end{itemize}
\begin{itemize}
\item \texttt{have} \; \texttt{p31: valueChosen} \; s \; (\text{chosen} \; s)
\begin{itemize}
\item \textbf{by} (\texttt{auto simp add: HInv6-def})
\end{itemize}
\end{itemize}
\textbf{with} \; \texttt{False \; act}
\begin{itemize}
\item \texttt{have} \; \texttt{chosen} \; s' \neq \text{NotAnInput}
\begin{itemize}
\item \textbf{by} (\texttt{auto simp add: HNextPart-def})
\end{itemize}
\end{itemize}
\textbf{from} \; \texttt{HEndPhase2-Inv6-1[OF \; act \; \texttt{inv} \; \texttt{inv2b} \; \texttt{inv2c} \; \texttt{inv3} \; \texttt{inv5} \; \texttt{this]}]
\texttt{have} \; \texttt{p32: valueChosen} \; s'(\text{chosen} \; s')
\item \textbf{from} \; \texttt{False \; InputsOrNi}
\texttt{have} \; \texttt{chosen} \; s \in \texttt{Inputs} \; \texttt{by} \; \texttt{auto}
\texttt{from} \; \texttt{valueChosen-equal[OF \; HEndPhase2-valueChosen[OF \; \texttt{act} \; \texttt{p31} \; \texttt{this}]} \; \texttt{p32]}
\texttt{have} \; \texttt{p33: chosen} \; s = \text{chosen} \; s'
\texttt{from} \; \texttt{act}
\begin{itemize}
\item \texttt{have} \; \texttt{maj: IsMajority} \; \{d . \; d \in \texttt{disksWritten} \; s \; p \}
\begin{itemize}
\item \texttt{and} \; \texttt{phase} \; \texttt{phase} \; s \; p = 2
\end{itemize}
\item \textbf{by} (\texttt{auto simp add: EndPhase2-def})
\end{itemize}
\textbf{show} \; ?\texttt{thesis} (\texttt{auto simp add: MajoritySet-def})
\textbf{proof} (\texttt{cases outpt} \; s \; r = \text{NotAnInput})
\texttt{case} \; \text{True}
\texttt{with} \; \texttt{asm \; act}
\begin{itemize}
\item \texttt{have} \; \texttt{p41: r=p}
\begin{itemize}
\item \textbf{by} (\texttt{auto simp add: EndPhase2-def split: split-if-asm})
\end{itemize}
\end{itemize}
\texttt{from} \; \texttt{maj}
\begin{itemize}
\item \texttt{have} \; \texttt{p42: \exists \; D \in \texttt{MajoritySet}. \; \forall \; d \in \texttt{D}. \; \forall \; q \in \texttt{UNIV} - \{p\}. \; \texttt{hasRead} \; s \; p \; d \; q}
\begin{itemize}
\item \textbf{by} (\texttt{auto simp add: MajoritySet-def})
\end{itemize}
\item \texttt{have} \; \texttt{p43: \neg(\exists \; D \in \texttt{MajoritySet}. \; \exists \; q. \; (\forall \; d \in \; D). \; \texttt{bal}\; (\texttt{dblock} \; s \; p) < \texttt{mbal}\; (\texttt{disk} \; s \; d \; q)}
\begin{itemize}
\item \texttt{and} \; \texttt{\neg\; hasRead} \; s \; p \; d \; q)
\end{itemize}
\item \textbf{proof} \; \texttt{auto}
\item \texttt{fix} \; D \; q
\item \texttt{assume} \; \texttt{Dmaj: \; D \in \texttt{MajoritySet}}
\end{itemize}
\[
\exists d \in D. \, \text{bal}(\text{dblock } s p) < \text{mbal}(\text{disk } s d q) \rightarrow \text{hasRead } s p d q
\]

**proof** (cases \( p=q \))
- assume \( pq : p=q \)
- thus \(?\)thesis

**proof** auto
- from \( \text{maj} \) \( \text{majorities-intersect } D \text{maj} \)
  - have \(!D \cap D \neq \{\} \)
    - by (auto simp add: \( \text{MajoritySet-def} \))
  - hence \( \exists d \in !D \cap D. \, d \in \text{disksWritten } s p \) by auto
- then obtain \( d \) where \( d \in \text{disksWritten } s p \) and \( d \in !D \cap D \)
  - by auto
- hence \( d \in D \) by auto
- have \( \text{disk } s d p = \text{dblock } s p \)
  - by (auto simp add: \( \text{Inv2b-def} \) \( \text{Inv2b-inner-def} \))
- with \( \text{inv2c phase} \)
  - have \( \text{bal}(\text{dblock } s p) = \text{mbal}(\text{disk } s d p) \)
    - by (auto simp add: \( \text{Inv2c-def} \) \( \text{Inv2c-inner-def} \))
  - show \( \exists d \in D. \, \text{bal}(\text{dblock } s q) < \text{mbal}(\text{disk } s d q) \rightarrow \text{hasRead } s q d q \)
    - 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
  - show \(?\)thesis
    - by (auto simp add: \( \text{MajoritySet-def} \), blast)

**qed**

**next**
- case \( False \)
- with \( p42 \)
  - have \( \exists b \in \text{allBlocks } s. \, \exists b \in (\text{UN } p. \, \text{Ballot } p). \, (\text{maxBalInp } s b (\text{chosen } s)) \land b \leq \text{bal } bk \)
    - proof
      - have \( \text{disk-allblks} : \forall d. \, \text{disk } s d p \in \text{allBlocks } s \)
        - by (auto simp add: \( \text{allBlocks-def} \) \( \text{blocksOf-def} \))
      - from \( p31 \)
        - have \( \exists b \in \text{allBlocks } s. \, \exists b \in (\text{UN } p. \, \text{Ballot } p). \, \text{maxBalInp } s b (\text{chosen } s) \land (\exists p. \, \exists D \in \text{MajoritySet}. (\forall d \in D. \, b \leq \text{bal}(\text{disk } s d p))) \)
          - by (auto simp add: \( \text{valueChosen-def} \), \( \text{force} \))
        - with \( \text{majority-nonempty} \)
          - obtain \( b \in D \) \( d \)
            - where \( \text{IsMajority } D \land b \in (\text{UN } p. \, \text{Ballot } p) \land \text{maxBalInp } s b (\text{chosen } s) \land d \in D \land b \leq \text{bal}(\text{disk } s d p) \)
              - by (auto simp add: \( \text{MajoritySet-def} \), blast)
with disk-allblks
show ?thesis
by(auto)
qed
then obtain bk b
where p45-bk: \( bk \in \text{allBlocks } s \land b \leq \text{bal } bk \)
and p45-b: \( b \in (\text{UN } p. \ \text{Ballot } p) \land (\text{maxBalInp } s b \ (\text{chosen } s)) \)
by auto
have p46: \( \text{inp}(\text{dblock } s p) = \text{chosen } s \)
proof(cases \( b \leq \text{bal}(\text{dblock } s p) \))
  case True
  have dblock s p \( \in \text{allBlocks } s \)
  by(auto simp add: allBlocks-def blocksOf-def)
  with p45-b True
  show ?thesis
  by(auto simp add: maxBalInp-def)
  next
  case False
  from p44 p45-bk False
  have inp bk = \( \text{inp}(\text{dblock } s p) \)
  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 = \( \text{outpt } s r \)
  by(auto simp add: Inv2c-inner-def EndPhase2-def split: split-if-asm)
  with inv p33 False
  show ?thesis
  by(auto simp False)
  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 \( \text{inv5}: \text{HInv5-inner s p} \)
shows \( \text{Hinv6 s'} \)
proof (auto simp add: HInv6-def)
assume chosen s' \( \neq \) NotAnInput
from \( \text{HEndPhase2-Inv6-1[OF act inv inv2b inv2c inv3 inv5 this]} \)
show valueChosen s' (chosen s') .
next
fix p
assume outpt s' p \( \neq \) NotAnInput
from \( \text{HEndPhase2-Inv6-2[OF act inv inv2b inv2c inv3 inv5 this]} \)
show outpt s' p = chosen s' .
qed

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

lemma outpt-Inv6:
[ outpt s = outpt s'; \( \forall \) p. outpt s p \( \in \) \{chosen s, NotAnInput\};
  Inv2c s; HNextPart s s' ] \( \longrightarrow \) \( \forall \) p. outpt s' p \( \in \) \{chosen s', NotAnInput\}
using outpt-chosen
by (auto!)

theorem HStartBallot-Inv6:
assumes act: HStartBallot s s' p
and inv: Hinv6 s
and inv2c: Inv2c s
shows Hinv6 s'
proof —
from outpt-chosen act inv2c inv
have chosen s' \( \neq \) NotAnInput \( \longrightarrow \) valueChosen s (chosen s')
by (auto simp add: StartBallot-def Hinv6-def)
from HStartBallot-valueChosen[OF act] this InputsOrNi
have t1: chosen s' \( \neq \) NotAnInput \( \longrightarrow \) valueChosen s' (chosen s')
by auto
from act
have outpt: outpt s = outpt s'
by (auto simp add: StartBallot-def)
from outpt-Inv6[OF outpt] act inv2c inv
have $\forall p. \text{outpt} s' p = \text{chosen} s' \lor \text{outpt} s' p = \text{NotAnInput}$
  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 chosen s' \neq NotAnInput \longrightarrow valueChosen s (chosen s')
  by(auto simp add: Phase1or2Write-def HInv6-def)
from HPhase1or2Write-valueChosen[OF act] inv4 this InputsOrNi
have t1: chosen s' \neq NotAnInput \longrightarrow valueChosen s' (chosen s')
  by auto
from act
have outpt: outpt s = outpt s'
  by(auto simp add: Phase1or2Write-def)
from outpt-Inv6[OF outpt] act inv2c inv
have $\forall p. \text{outpt} s' p = \text{chosen} s' \lor \text{outpt} s' p = \text{NotAnInput}$
  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 chosen s' \neq NotAnInput \longrightarrow valueChosen s (chosen s')
  by(auto simp add: Phase1or2ReadThen-def HInv6-def)
from HPhase1or2ReadThen-valueChosen[OF act] this InputsOrNi
have t1: chosen s' \neq NotAnInput \longrightarrow 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 \( t_1 \)

show ?thesis
  by (simp add: HInv6-def)

qed

theorem HPhase1or2ReadElse-Inv6:
  assumes act: HPhase1or2ReadElse \( s \ s' p \ d \ q \)
  and inv: HInv6 \( s \)
  and inv2: Inv2 \( s \)
  shows HInv6 \( s' \)
  using 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 \( s' \neq \text{NotAnInput} \to value\text{Chosen} \ s \ (\text{chosen} \ s') \)
    by (auto simp add: EndPhase1-def HInv6-def)
  from HEndPhase1-valueChosen[OF act] inv1 inv2a inv2b this InputsOrNi
  have \( t_1 \): chosen \( s' \neq \text{NotAnInput} \to value\text{Chosen} \ s' \ (\text{chosen} \ s') \)
    by auto
  from act
  have outpt: outpt \( s = \text{outpt} \ s' \)
    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 \( t_1 \)
  show ?thesis
    by (simp add: HInv6-def)

qed

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

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

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

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

theorem 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' \( \neq \) NotAnInput --- valueChosen s (chosen s')
by (auto simp add: Phase0Read-def HInv6-def)
from HPhase0Read-valueChosen[OF act] this InputsOrNi
have t1: chosen s' \neq \text{NotAnInput} \longrightarrow 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 = \text{NotAnInput}
  by (auto simp add: HInv6-def)
with t1
show \?thesis
  by (simp add: HInv6-def)
qed

theorem HEndPhase0-Inv6:
  assumes act: HEndPhase0 s s' p
  and inv: HInv6 s
  and inv1: Inv1 s
  and inv2c: Inv2c s
  shows HInv6 s'
proof -
  from outpt-chosen act inv2c inv
  have chosen s' \neq \text{NotAnInput} \longrightarrow valueChosen s (chosen s')
    by (auto simp add: EndPhase0-def HInv6-def)
  from HEndPhase0-valueChosen[OF act] inv1 this InputsOrNi
  have t1: chosen s' \neq \text{NotAnInput} \longrightarrow valueChosen s' (chosen s')
    by auto
  from act
  have outpt: outpt s = outpt s'
    by (auto simp add: EndPhase0-def)
  from outpt-Inv6[OF outpt] act inv2c inv
  have \forall p. outpt s' p = chosen s' \lor outpt s' p = \text{NotAnInput}
    by (auto simp add: HInv6-def)
  with t1
  show \?thesis
    by (simp add: HInv6-def)
qed

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

lemma I2f:
  assumes nxt: HNext s s'
  and inv: HInv1 s \land HInv2 s \land HInv2 s' \land HInv3 s \land HInv4 s \land HInv5 s \land HInv6 s
  shows HInv6 s'
by (auto simp add: HNext-def Next-def, auto simp add: HInv2-def intro: HStartBallot-Inv6, auto intro: HPhase0Read-Inv6,)

147
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

constdefs
HInv :: state ⇒ bool
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'
sows HInv s'
by(simp add: HInv-def)

end

theory DiskPaxos imports DiskPaxos-Invariant begin
C.9 Inner Module

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

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

IChoose :: Istate ⇒ Istate ⇒ Proc ⇒ bool
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

IFail :: Istate ⇒ Istate ⇒ Proc ⇒ bool
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

constdefs
INext :: Istate ⇒ Istate ⇒ bool
INext s s' ≡ ∃ p. IChoose s s' p ∨ IFail s s' p

constdefs
s2is :: state ⇒ Istate
s2is s ≡ (∣iinput = inpt s,
           ioutput = outpt s,
           ichosen=chosen s,
           iallInput = iallInput s)

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

theorem R2b:
assumes inv: HInv s
and inv': HInv s'
and nxt: HNext s s'

and \( \text{srel}: \text{is} = \text{s2is s} \land \text{is}' = \text{s2is s}' \)

shows \((\exists p. \text{IFail is is'} p \lor \text{IChoose is is'} p) \lor \text{is} = \text{is}'\)

proof(auto)

assume \(\text{chg-vars: is} \neq \text{is}'\)

with \(\text{srel}\)

have \(\text{s-change: inpt s \neq inpt s' \lor \text{outpt s} \neq \text{outpt s'} \lor \text{chosen s} \neq \text{chosen s'} \lor \text{allInput s} \neq \text{allInput s}'\)

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

from \(\text{inv}\)

have \(\text{inv2c5}: \forall p. \text{inpt s} p \in \text{allInput s} \land (\text{chosen s} = \text{NotAnInput} \rightarrow \text{outpt s} p = \text{NotAnInput})\)

by(auto simp add: \(\text{HInv-def} \ \text{HInv2-def} \ \text{Inv2c-def} \ \text{Inv2c-inner-def}\))

from \(\text{nxt s-change inv2c5}\)

have \(\text{inpt s'} \neq \text{inpt s} \lor \text{outpt s'} \neq \text{outpt s} \lor \text{chosen s'} \neq \text{chosen s} \lor \text{allInput s} \neq \text{allInput s}'\)

by(auto simp add: \(\text{HNext-def} \ \text{Next-def} \ \text{HNextPart-def}\))

with \(\text{nxt}\)

have \(\exists p. \text{Fail s s'} p \lor \text{EndPhase2 s s'} p\)

by(auto simp add: \(\text{StartBallot-def} \ \text{Phase0Read-def} \ \text{Phase1or2Write-def} \ \text{Phase1or2Read-def} \ \text{Phase1or2ReadThen-def} \ \text{Phase1or2ReadElse-def} \ \text{EndPhase1or2-def} \ \text{EndPhase1-def} \ \text{EndPhase0-def}\))

then obtain \(p\) where fail-or-endphase2: \(\text{Fail s s'} p \lor \text{EndPhase2 s s'} p\)

by auto

from \(\text{inv}\)

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

by(auto simp add: \(\text{HInv-def} \ \text{HInv2-def} \ \text{Inv2c-def} \ \text{Inv2c-inner-def}\))

from fail-or-endphase2 have IFail is is' p \lor IChoose is is' p

proof

assume fail: \(\text{Fail s s'} p\)

hence phase': phase s' p = 0

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

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

have IFail is is' p

proof

from fail \(\text{srel}\)

have ioutput is' = (ioutput is) (p:= NotAnInput)

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

moreover

from \(\text{nxt}\)

have all-nxt: allInput s' = allInput s \cup (\text{range (inpt s')})

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

from fail \(\text{srel}\)

have \(\exists ip \in \text{Inputs}. \text{input is'} = (\text{input is})(p:= \text{ip})\)

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

then obtain \(ip\) where input-Input: \(ip \in \text{Inputs} \ \text{and} \ \text{input is'} = (\text{input is})(p:= \text{ip})\)

by auto

with \(\text{inv2c5 srel all-nxt}\)

have input is' = (input is)(p:= ip)
\[ \text{iallInput is'} = \text{iallInput is} \cup \{ip\} \]

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: inp (dblock s p) \in allInput s

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

from inv

have allInput s \subseteq Inputs

by (auto simp add: HInv-def HInv1-def)

with p22 NotAnInput endphase2

have outpt-nni: outpt s' p \neq NotAnInput

by (auto simp add: EndPhase2-def)

show ?thesis

proof (cases chosen s = NotAnInput)

case True

with inv2c5

have p31: \(\forall q\). outpt s q = NotAnInput

by auto

with endphase2

have p32: \(\forall q \in UNIV - \{p\}\). outpt s' q = NotAnInput

by (auto simp add: EndPhase2-def)

hence some-eq: (\(\exists x. outpt s' x \neq NotAnInput \Rightarrow x = p\))

by auto

from p32 True nxt some-equality[of \(\lambda p. outpt s' p \neq NotAnInput\), 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) ∨ outpt s' = (outpt s)(p:=inp(dblock s p))
by(auto simp add: EndPhase2-def)
with True p22
have if (chosen s = NotAnInput)
then (∃ ip ∈ allInput s. chosen s' = ip
∧ outpt s' = (outpt s) (p := ip))
else ( outpt s' = (outpt s) (p:= chosen s)
∧ chosen s' = chosen s)
by auto
moreover
from endphase2 inv2c5 nxt
have inp t s' = inp t ∧ 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 ∈ {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
thus ∃p. IFail is is' p ∨ IChoose is is' p
  by auto
qed
end