Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff    Stephan Merz

October 27, 2022

Abstract

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

Contents

1 Introduction 2

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

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

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

5 Conclusion 11

A TLA+ correctness specification 12

B Disk Paxos Algorithm Specification 13
1 Introduction

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

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

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

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

2 The Disk Paxos Algorithm

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

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

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

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

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

\textbf{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.

\textbf{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:

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

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

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

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

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

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

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

$$HDiskSynodSpec \triangleq HInit \land \Box[HNext](vars, chosen, allInput)$$

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

Analogously, we have the specification of the Inner module:

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

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

**THEOREM R1**
$$HInit \Rightarrow IInit$$

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>TLA$^+$</th>
<th>Isabelle/HOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists d \in D: \text{disk}(d,</td>
<td>( \exists d \in D. \ \text{bal}(\text{disk}(s, d, q)) = bk )</td>
</tr>
<tr>
<td>\text{bal} = bk</td>
<td></td>
</tr>
<tr>
<td>\text{phase}' = [\text{phase} \ \text{EXCEPT} \ !p = 1]</td>
<td>\text{phase}' = (\text{phase}(s))(p := 1)</td>
</tr>
<tr>
<td>\text{UN} \ p. \ \text{blocksOf} \ s \ p</td>
<td></td>
</tr>
<tr>
<td>\text{UNCHANGED} v</td>
<td>\text{v s}' = v s</td>
</tr>
</tbody>
</table>

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

**Theorem 11** \( H\text{Init} \Rightarrow H\text{Inv} \)

**Theorem 12** \( H\text{Inv} \land H\text{Next} \Rightarrow H\text{Inv}' \)

Again, we have TLA proof rules that say that I1 and I2 imply R2a. In summary, R2b, I1, and I2 together imply \( H\text{DiskSynodSpec} \Rightarrow I\text{Spec} \).

Finding a predicate \( H\text{Inv} \) that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present \( H\text{Inv} \) as a conjunction of 6 predicates \( H\text{Inv}_1, \ldots, H\text{Inv}_6 \), where \( H\text{Inv}_1 \) is a simple “type invariant” and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of \( H\text{Inv}_i \) by the algorithm’s next-state relation relies on all \( H\text{Inv}_j \) (for \( j \leq i \)) being true in the state before the transition. In our proofs we are going to use exactly the same tactic.

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

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

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

#### 3.1 Typed vs. Untyped

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

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

CONSTANT Inputs

\[\begin{align*} 
\text{NotAnInput} & \triangleq \text{choose } c : c \notin \text{Inputs} \\
\text{DiskBlock} & \triangleq \{ \text{mbal} : (\text{union Ballot}(p) : p \in \text{Proc}) \cup \{0\}, \\
& \quad \text{bal} : (\text{union Ballot}(p) : p \in \text{Proc}) \cup \{0\}, \\
& \quad \text{inp} : \text{Inputs} \cup \{\text{NotAnInput}\}\} 
\end{align*}\]

Isabelle/HOL:

typedef InputsOrNi

consts

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

axioms

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

record

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

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

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

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

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

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

Isabelle/HOL:

\[ \text{Phase1or2Write} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \]
\[ \text{Phase1or2Write} \ s \ s' \ p \ d \equiv \]
\[ \land \ \text{phase} \ s \ p \in \{1, 2\} \]
\[ \land \ \text{disk}' = (\text{disk} \ s) (d := (\text{disk} \ s \ d) (p := \text{dblock} \ s \ p)) \]
\[ \land \ \text{disksWritten}' = (\text{disksWritten} \ s) (p := (\text{disksWritten} \ s \ p) \cup \{d\}) \]
\[ \land \ \text{inpt}' = \text{inpt} \ s \land \text{outpt}' = \text{outpt} \ s \]
\[ \land \ \text{phase}' = \text{phase} \ s \land \text{dblock}' = \text{dblock} \ s \]
\[ \land \ \text{blocksRead}' = \text{blocksRead} \ s \]

Figure 3: Translation of an action

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

3.2 Primed Variables

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

3.3 Restructuring the specification

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

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

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

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

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

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

Another example is \( \text{HInv2} \), 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 \( \text{Inv2a} \), 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:

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

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 \( \text{Inv2a} \), 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 $I2f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate $I$ was an invariant of $Next$, we preferred proving the invariance of $I$ for each action, rather than a big theorem proving the invariance of $I$ for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $Hinv_3$ for the $EndPhase_0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

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

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

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

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

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

References


A TLA\(^+\) correctness specification

--- MODULE Synod ---

EXTENDS Naturals
CONSTANT N, Inputs
ASSUME (N ∈ Nat) ∧ (N > 0)
Proc \(\triangleq\) 1..N
NotAnInput \(\triangleq\) \(\text{choose } c : c \notin\) Inputs
VAR inputs, output

--- MODULE Inner ---

VAR allInput, chosen

Init \(\triangleq\) \(\land\) input \(\in\) [Proc → Inputs]
\land output = [p \in Proc ↦ NotAnInput]
\land chosen = NotAnInput
\land allInput = input[p] : p \in Proc

IChoose(p) \(\triangleq\)
\land output[p] = NotAnInput
\land IF chosen = NotAnInput
THEN ip \in allInput : \land chosen' = ip
\land output' = [output except ![p] = ip]
ELSE \land output' = [output except ![p] = chosen]
\land UNCHANGED chosen
\land UNCHANGED ⟨input, allInput⟩

IFail(p) \(\triangleq\)
\land output' = [output except ![p] = NotAnInput]
\land \exists ip \in Inputs : \land input' = [input except ![p] = ip]
\land allInput' = allInput ∪ {ip}

INext \(\triangleq\) \(\exists p \in\) Proc : IChoose(p) \lor IFail(p)
ISpec \(\triangleq\) Init \land □[INext]⟨input, output, chosen, allInput⟩

IS(chosen, allInput) \(\triangleq\) INSTANCE Inner
SynodSpec \(\triangleq\) \(\exists\) chosen, allInput : IS(chosen, allInput) \& ISpec

---

B Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedec Input

typedec Disk

typedec Proc

axiomatization

Inputs :: Input ∪ {NotAnInput} and
NotAnInput :: Input ∪ {NotAnInput} and
Ballot :: Proc ⇒ nat set and
IsMajority :: Disk set ⇒ bool

where

NotAnInput: NotAnInput ∉ Input and
Input-union: (UNIV :: Input ∪ {NotAnInput}) = Input ∪ {NotAnInput} and
Ballot-ncero: ∀ p. p ∉ Ballot p and
Ballot-disj: ∀ p q. p ≠ q −→ (Ballot p) ∩ (Ballot q) = {} and
Dis-isMajority: IsMajority(UNIV) and
majorities-intersect:
∀ S T. IsMajority(S) ∧ IsMajority(T) → S ∩ T ≠ {}

lemma ballots-not-zero [simp]:

proof (rule contr)

assume b: b ∈ Ballot p

and contr: ¬ (b ≠ p)

from Ballot-ncero

have b ∉ Ballot p

thus False

by auto

qed

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

proof (auto)

from majorities-intersect

have IsMajority({}) ∧ IsMajority({}) → {} ∩ {} ≠ {} 

by auto

thus IsMajority {} → False

by auto

qed

definition AllBallots :: nat set

where AllBallots = (UN p. Ballot p)

record

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

**definition** InitDB :: DiskBlock
where InitDB = ⟨ mbal = 0, bal = 0, inp = NotAnInput ⟩

**record**
BlockProc =
  block :: DiskBlock
  proc :: Proc

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

allInput :: InputsOrNi set
chosen :: InputsOrNi

**definition** hasRead :: state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool
where hasRead s p d q = (∃ br ∈ blocksRead s p d. proc br = q)

**definition** allRdBlks :: state ⇒ Proc ⇒ BlockProc set
where allRdBlks s p = (UN d. blocksRead s p d)

**definition** allBlocksRead :: state ⇒ Proc ⇒ DiskBlock set
where allBlocksRead s p = block ' (allRdBlks s p)

**definition** Init :: state ⇒ bool
where
  Init s =
    (range (inpt s) ⊆ Inputs
    & outpt s = (λp. NotAnInput)
    & disk s = (λd p. InitDB)
    & phase s = (λp. 0)
    & dblock s = (λp. InitDB)
    & disksWritten s = (λp. {}))
    & blocksRead s = (λp d. {}))

**definition** InitializePhase :: state ⇒ state ⇒ Proc ⇒ bool
where
  InitializePhase s s' p =

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

\textbf{definition} \texttt{StartBallot :: state ⇒ state ⇒ Proc ⇒ bool}
\textbf{where}
\texttt{StartBallot } s\ s'\ p =
\begin{align*}
& (\text{phase } s\ p \in \{1, 2\}) \\
& \& (\exists b \in \text{Ballot } p, \\
& \quad \text{mbal}(\text{dblock } s\ p) < b) \\
& \& (\exists b \in \text{Ballot } p, \\
& \quad \text{mbal'}(\text{dblock } s\ p)(p := (\text{dblock } s p)(\text{mbal} := b))) \\
& \& \text{InitializePhase } s\ s'\ p \\
& \& \text{inpt } s' = \text{inpt } s \& \text{outpt } s' = \text{outpt } s \& \text{disk } s' = \text{disk } s)
\end{align*}

\textbf{definition} \texttt{Phase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool}
\textbf{where}
\texttt{Phase1or2Write } s\ s'\ p\ d =
\begin{align*}
& (\text{phase } s\ p \in \{1, 2\}) \\
& \& (\exists b \in \text{disksWritten } s\ p, \\
& \quad \text{mbal}(\text{block } s\ p) < b) \\
& \& \text{blocksRead } s' = (\text{blocksRead } s)(p := (\text{blocksRead } s p)(\text{mbal} := b))) \\
& \& \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{blocksRead } s' = \text{blocksRead } s)
\end{align*}

\textbf{definition} \texttt{Phase1or2ReadThen :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ Proc ⇒ bool}
\textbf{where}
\texttt{Phase1or2ReadThen } s\ s'\ p\ d\ q =
\begin{align*}
& (d \in \text{disksWritten } s\ p) \\
& \& \text{mbal}(\text{disk } s\ d\ q) < \text{mbal}(\text{block } s\ p) \\
& \& \text{blocksRead } s' = (\text{blocksRead } s)(p := (\text{blocksRead } s p)(d := \\
& \quad (\text{blocksRead } s p d)(\text{proc} := q))) \\
& \& \text{inpt } s' = \text{inpt } s \& \text{outpt } s' = \text{outpt } s \\
& \& \text{disk } s' = \text{disk } s \& \text{phase } s' = \text{phase } s \\
& \& \text{blocksRead } s' = \text{blocksRead } s \& \text{disksWritten } s' = \text{disksWritten } s)
\end{align*}

\textbf{definition} \texttt{Phase1or2ReadElse :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ Proc ⇒ bool}
\textbf{where}
\texttt{Phase1or2ReadElse } s\ s'\ p\ d\ q =
\begin{align*}
& (d \in \text{disksWritten } s\ p) \\
& \& \text{StartBallot } s\ s'\ p)
\end{align*}

\textbf{definition} \texttt{Phase1or2Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ Proc ⇒ bool}
\textbf{where}
\texttt{Phase1or2Read } s\ s'\ p\ d\ q =
\begin{align*}
& (\text{Phase1or2ReadThen } s\ s'\ p\ d\ q) \\
& \lor \text{Phase1or2ReadElse } s\ s'\ p\ d\ q)
\end{align*}

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

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

definition \( \text{maxBlk} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \)
where
\[
\text{maxBlk } s p = \\
(SOME \ b . b \in \text{nonInitBlks } s p \land (\forall c \in \text{nonInitBlks } s p . \text{bal } c \leq \text{bal } b))
\]
definition \( \text{EndPhase1} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
where
\[
\text{EndPhase1 } s s' p = \\
(\text{IsMajority } \{ d . d \in \text{disksWritten } s p \} \\
\land (\forall q \in \text{UNIV} - \{ p \} . \text{hasRead } s p d q)) \\
\land \text{phase } s p = 1 \\
\land \text{dblock } s' = (\text{dblock } s) (p := \text{dblock } s p) \\
\quad \{ \text{bal } := \text{mbal}(\text{dblock } s p), \\
\quad \text{inp } := \\
\quad \quad \{ \text{if nonInitBlks } s p = \{ \}
\quad \quad \text{then inp } s p \\
\quad \quad \text{else inp } (\text{maxBlk } s p) \\
\quad \} \\
\land \text{outpt } s' = \text{outpt } s \\
\land \text{phase } s' = (\text{phase } s) (p := \text{phase } s p + 1) \\
\land \text{InitializePhase } s s' p \\
\land \text{inpt } s' = \text{inpt } s \land \text{disk } s' = \text{disk } s)
\]
definition \( \text{EndPhase2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
where
\[
\text{EndPhase2 } s s' p = \\
(\text{IsMajority } \{ d . d \in \text{disksWritten } s p \} \\
\land (\forall q \in \text{UNIV} - \{ p \} . \text{hasRead } s p d q)) \\
\land \text{phase } s p = 2 \\
\land \text{outpt } s' = (\text{outpt } s) (p := \text{inp } (\text{dblock } s p)) \\
\land \text{dblock } s' = \text{dblock } s \\
\land \text{phase } s' = (\text{phase } s) (p := \text{phase } s p + 1) \\
\land \text{InitializePhase } s s' p \\
\land \text{inpt } s' = \text{inpt } s \land \text{disk } s' = \text{disk } s)
\]
definition \( \text{EndPhase1or2} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
where \( \text{EndPhase1or2 } s s' p = (\text{EndPhase1 } s s' p \lor \text{EndPhase2 } s s' p) \)
definition \( \text{Fail} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
where
\[
\text{Fail } s s' p = \\
(\exists ip \in \text{Inputs}. \text{inpt } s' = (\text{inpt } s) (p := ip) \\
\land \text{phase } s' = (\text{phase } s) (p := 0) \\
\land \text{dblock } s' = (\text{dblock } s) (p := \text{InitDB})
\]

16
\[ \begin{align*}
\land outpt s' &= (outpt s) \ (p := \text{NotAnInput}) \\
\land InitializePhase s \ s' \ p \\
\land disk s' &= disk s)
\end{align*} \]

**definition** \( \text{Phase0Read} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \)

**where**

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

**definition** \( \text{EndPhase0} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)

**where**

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

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

**where**

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

In the following, for each action or state name we name \( Hname \) the corresponding action that includes the history part of the \( \text{HNext} \) action or state predicate that includes history variables.

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

**where**

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

**Definition**

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

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

**Definition**

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

where
\[
\text{HNext} \ s \ s' = \\
( \text{Next } s \ s' \ \\
\quad \land \text{HNextPart } s \ s')
\]

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

where
\[
\text{HFail } s \ s' = (\text{Fail } s \ s' \ p \land \text{HNextPart } s \ s')
\]
definition

HPhase0Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool where
HPhase0Read s s' p d = (Phase0Read s s' p d ∧ HNextPart s s')

definition

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

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

declare HPhase1or2ReadThen-def [simp]
declare HPhase1or2ReadElse-def [simp]
declare HEndPhase1-def [simp]
declare HStartBallot-def [simp]
declare HPhase1or2Write-def [simp]
declare HEndPhase2-def [simp]
declare HFail-def [simp]
declare HPhase0Read-def [simp]
declare HEndPhase0-def [simp]

C Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1 Invariant 1

This is just a type Invariant.

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

definition HInv1 :: state ⇒ bool where
HInv1 s = (Inv1 s ∧ allInput s ⊆ Inputs)

declare HInv1-def [simp]

We added the assertion that the set allRdBlksp is finite for every process p;
one may therefore choose a block with a maximum ballot number in action
EndPhase1.
With the following the lemma, it will be enough to prove \( \text{Inv}_1 s' \) for every action, without taking the history variables in account.

**lemma** \( \text{HNextPart-Inv}_1 : \left[ \text{HInv}_1 s; \text{HNextPart} s s' ; \text{Inv}_1 s' \right] \implies \text{HInv}_1 s' \)

by \((\text{auto simp add: HNextPart-def Inv1-def})\)

**theorem** \( \text{HInit-HInv}_1 : \text{HInit} s \implies \text{HInv}_1 s \)

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

**lemma** \( \text{allRdBlks-finite} : \)

- **assumes** \( \text{inv} : \text{HInv}_1 s \)
- and \( \text{asm} : \forall p. \text{allRdBlks} s' p \subseteq \text{insert} \ bk \ (\text{allRdBlks} \ s \ p) \)
- **shows** \( \forall p. \text{finite} (\text{allRdBlks} \ s' \ p) \)

**proof**

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

qed

**theorem** \( \text{HPhase1or2ReadThen-HInv}_1 : \)

- **assumes** \( \text{inv1} : \text{HInv}_1 s \)
- and \( \text{act} : \text{HPhase1or2ReadThen} s s' p d q \)
- **shows** \( \text{HInv}_1 s' \)

**proof**

- we focus on the last conjunct of \( \text{Inv}_1 \)
- from \( \text{act} \)
- have \( \forall p. \text{allRdBlks} s' p \subseteq \text{allRdBlks} s \ p \cup \{ (\text{block} = \text{disk} \ s \ d \ q, \text{proc} = q) \} \)
  - by \((\text{auto simp add: Phase1or2ReadThen-def allRdBlks-def split: if-split_asm})\)
- with \( \text{inv1} \)
- have \( \forall p. \text{finite} (\text{allRdBlks} \ s' \ p) \)
  - by \((\text{blast dest: allRdBlks-finite})\)
  - the others conjuncts are trivial
- with \( \text{inv1 act} \)
- show \( \text{?thesis} \)
  - by \((\text{auto simp add: Inv1-def Phase1or2ReadThen-def HNextPart-def})\)

qed

**theorem** \( \text{HEndPhase1-HInv}_1 : \)

- **assumes** \( \text{inv1} : \text{HInv}_1 s \)
- and \( \text{act} : \text{HEndPhase1} s s' p \)
- **shows** \( \text{HInv}_1 s' \)

**proof**

- from \( \text{inv1 act} \)
have \texttt{Inv1} \; s' \\
by\texttt{(auto simp add: Inv1-def EndPhase1-def InitializePhase-def allRdBlks-def)} \\
with \texttt{inv1} \; \texttt{act} \\
show \; \texttt{?thesis} \\
by\texttt{(auto simp del: HInv1-def dest: HNextPart-Inv1)}

\textbf{qed}

\textbf{theorem} \texttt{HStartBallot-HInv1}: \\
\texttt{assumes} \; \texttt{inv1: HInv1} \; \texttt{s} \\
\texttt{and act: HStartBallot} \; \texttt{s} \; \texttt{s'} \; \texttt{p} \\
\texttt{shows} \; \texttt{HInv1} \; \texttt{s'}

\textbf{proof} – \\
\texttt{from} \; \texttt{inv1} \; \texttt{act} \\
\texttt{have} \; \texttt{Inv1} \; \texttt{s'} \\
\texttt{by\texttt{(auto simp add: Inv1-def StartBallot-def InitializePhase-def allRdBlks-def)}} \\
\texttt{with} \; \texttt{inv1} \; \texttt{act} \\
\texttt{show} \; \texttt{?thesis} \\
\texttt{by\texttt{(auto simp del: HInv1-def elim: HNextPart-Inv1)}}

\textbf{qed}

\textbf{theorem} \texttt{HPhase1or2Write-HInv1}: \\
\texttt{assumes} \; \texttt{inv1: HInv1} \; \texttt{s} \\
\texttt{and act: HPhase1or2Write} \; \texttt{s} \; \texttt{s'} \; \texttt{p} \; \texttt{d} \\
\texttt{shows} \; \texttt{HInv1} \; \texttt{s'}

\textbf{proof} – \\
\texttt{from} \; \texttt{inv1} \; \texttt{act} \\
\texttt{have} \; \texttt{Inv1} \; \texttt{s'} \\
\texttt{by\texttt{(auto simp add: Inv1-def Phase1or2Write-def allRdBlks-def)}} \\
\texttt{with} \; \texttt{inv1} \; \texttt{act} \\
\texttt{show} \; \texttt{?thesis} \\
\texttt{by\texttt{(auto simp del: HInv1-def elim: HNextPart-Inv1)}}

\textbf{qed}

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

\textbf{theorem} \texttt{HEndPhase2-HInv1}: \\
\texttt{assumes} \; \texttt{inv1: HInv1} \; \texttt{s} \\
\texttt{and act: HEndPhase2} \; \texttt{s} \; \texttt{s'} \; \texttt{p} \\
\texttt{shows} \; \texttt{HInv1} \; \texttt{s'}

\textbf{proof} – \\
\texttt{from} \; \texttt{inv1} \; \texttt{act} \\
\texttt{have} \; \texttt{Inv1} \; \texttt{s'} \\
\texttt{by\texttt{(auto simp add: Inv1-def EndPhase2-def InitializePhase-def allRdBlks-def)}} \\
\texttt{with} \; \texttt{inv1} \; \texttt{act}
show \( \text{thesis} \)
   by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

theorem HFail-HInv1:
  assumes inv1: \( HInv1 s \)
  and \( \text{act}: HFail s s' p \)
  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 \( \text{thesis} \)
    by (auto simp del: HInv1-def elim: HNextPart-Inv1)
qed

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

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

\textit{HInv1} is an invariant of \textit{HNext}

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

\textbf{end}

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

\textbf{C.2 Invariant 2}

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

\textbf{definition} \textit{rdBy} :: \textit{state} $\Rightarrow$ \textit{Proc} $\Rightarrow$ \textit{Proc} $\Rightarrow$ \textit{Disk} $\Rightarrow$ \textit{BlockProc set}
\textbf{where}
\begin{itemize}
  \item \textit{rdBy s p q d} = \{ \textit{br} . \textit{br} \in \textit{blocksRead s q d} \land \textit{proc br = p} \}
\end{itemize}

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

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

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

definition Inv2a-inner :: state ⇒ Proc ⇒ bool
where Inv2a-inner s p = (∀ bk ∈ blocksOf s p. Inv2a-innermost s p bk)

definition Inv2a :: state ⇒ bool
where Inv2a s = (∀ p. Inv2a-inner s p)

definition Inv2b-inner :: state ⇒ Proc ⇒ Disk ⇒ bool
where
Inv2b-inner s p d =
  ((d ∈ disksWritten s p) →
   (phase s p ∈ {1,2} ∧ disk s d p = dblock s p))
∧ (phase s p ∈ {1,2} →
   (blocksRead s p d ≠ {} → d ∈ disksWritten s p)
    ∧ ¬ hasRead s p d p))

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

definition Inv2c-inner :: state ⇒ Proc ⇒ bool
where
Inv2c-inner s p =
  ((phase s p = 0 →
    (dblock s p = InitDB
     ∧ disksWritten s p = {})
    ∧ (∀ d. ∀ br ∈ blocksRead s p d.
        proc br = p ∧ block br = disk s d p)))
∧ (phase s p ≠ 0 →
    (mbal(dblock s p) ∈ Ballot p
     ∧ bal(dblock s p) ∈ Ballot p ∪ {0})
    ∧ (∀ d. ∀ br ∈ blocksRead s p d.
        mbal(block br) < mbal(dblock s p)))
∧ (phase s p ∈ {2,3} → bal(dblock s p) = mbal(dblock s p))
∧ output s p = (if phase s p = 3 then inp(dblock s p) else NotAnInput)
∧ chosen s ∈ allInput s ∪ {NotAnInput}
∧ (∀ p. input s p ∈ allInput s
    ∧ (chosen s = NotAnInput → output s p = NotAnInput))

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

definition \( H\text{Inv2} :: \text{state} \Rightarrow \text{bool} \)
where \( H\text{Inv2} \ s = (\text{Inv2a} \ s \land \text{Inv2b} \ s \land \text{Inv2c} \ s) \)

C.2.1 Proofs of Invariant 2 a

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

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

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

theorem \( H\text{Phase1or2ReadThen-Inv2a} \)
assumes \( \text{inv} \): \( \text{Inv2a} \ s \)
and \( \text{act} \): \( \text{HPhase1or2ReadThen} \ s \ s' p d q \)
shows \( \text{Inv2a} \ s' \)
proof (clarsimp simp add: \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \) \( \text{Inv2a-innermost-def} \) \( \text{HNextPart-def} \))
fix \( pp \ bk \)
assume \( bk \): \( bk \in \text{blocksOf} \ s' pp \)
with \( \text{inv} \) \( \text{HPhase1or2ReadThen-blocksOf} [OF \text{act}] \)
have \( \text{Inv2a-innermost} \ s pp bk \)
  by (auto simp add: \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \) \( \text{Inv2a-innermost-def} \) \( \text{HNextPart-def} \) \( \text{HNextPart-def} \))
with \( \text{act} \)
show \( \text{Inv2a-innermost} \ s' pp bk \)
  by (auto simp add: \( \text{Inv2a-innermost-def} \) \( \text{HNextPart-def} \) \( \text{HNextPart-def} \))
qed

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

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

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

proof –
from act
have $\text{mbal'}: \text{mbal} \ (\text{dblock} \ s' \ p) \in \text{Ballot} \ p$
  by (auto simp add: \text{StartBallot-def})
from act
have $\text{bal'}: \text{bal} \ (\text{dblock} \ s' \ p) = \text{bal} \ (\text{dblock} \ s \ p)$
  by (auto simp add: \text{StartBallot-def})
with act
have $\text{inp'}: \text{inp} \ (\text{dblock} \ s' \ p) = \text{inp} \ (\text{dblock} \ s \ p)$
  by (auto simp add: \text{StartBallot-def})
from act
have $\text{mbal} (\text{dblock} \ s \ p) \leq \text{mbal} (\text{dblock} \ s' \ p)$
  by (auto simp add: \text{StartBallot-def})
with $\text{bal'} \ \text{inv2a}$
have $\text{bal-mbal}: \text{bal} \ (\text{dblock} \ s' \ p) \leq \text{mbal} (\text{dblock} \ s' \ p)$
  by (auto simp add: \text{Inv2a-innermost-def})
from act
have $\text{allInput} \ s \subseteq \text{allInput} \ s'$
  by (auto simp add: \text{HNextPart-def} \text{InitializePhase-def} \text{Inv2a-innermost-def})
with $\text{mbal'} \ \text{bal'} \ \text{inp'} \ \text{bal-mbal} \ \text{act} \ \text{inv2a}$
show $?\text{thesis}$
by (auto simp add: \text{Inv2a-innermost-def})
qed

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

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

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

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

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

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

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

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

lemma HFail-Inv2a-dblock-q:
  assumes act: HFail s s' p
  and inv: Inv2a-innermost s q (dblock s q)
  shows Inv2a-innermost s' q (dblock s' q)
proof (cases p=q)
  case True
  with act
  have dblock s' q = InitDB
    by (simp add: Fail-def)
  with True
  show ?thesis
    by (auto simp add: InitDB-def Inv2a-innermost-def)
next
  case False
  with inv act
show ?thesis
  by(auto simp add: Fail-def HNextPart-def
      InitializePhase-def Inv2a-innermost-def)
qed

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

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

theorem HPhase0Read-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase0Read s s' p d
  shows Inv2a s'
proof(clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ blocksOf s' q
  from inv bk HPhase0Read-blocksOf[OF act]
have inp-q-bk: Inv2a-innermost s q bk
  by (auto simp add: Inv2a-def Inv2a-inner-def)
with act
show Inv2a-innermost s' q bk
  by (auto simp add: Inv2a-innermost-def HNextPart-def)
qed

lemma HEndPhase0-blocksOf:
  HEndPhase0 s s' p ⟹ blocksOf s' q ⊆ blocksOf s q ∪ {dblock s' q}
by (auto simp add: EndPhase0-def blocksOf-def
dest: subset[OF InitializePhase-rdBy])

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

EndPhase0 has the additional difficulty of having a choose expression. We
prove that there exists an x such that the predicate of the choose expression

lemma HEndPhase0-some:
  assumes act: HEndPhase0 s s' p
  and inv1: Inv1 s
  shows (SOME b. b ∈ allBlocksRead s p
          ∧ (∀ t∈allBlocksRead s p. bal t ≤ bal b)
          ) ∈ allBlocksRead s p
          ∧ (∀ t∈allBlocksRead s p. bal t ≤ bal (SOME b. b ∈ allBlocksRead s p
          ∧ (∀ t∈allBlocksRead s p. bal t ≤ bal b)))
proof —
  from inv1 have finite (bal ∘ allBlocksRead s p) (is finite ?S)
     by (simp add: Inv1-def allBlocksRead-def)
  moreover
  from HEndPhase0-blocksRead[OF act]
  have ?S ≠ {}
     by (auto simp add: allBlocksRead-def allRdBlks-def)
  ultimately
  have Max ?S ∈ ?S and ∀ t ∈ ?S. t ≤ Max ?S by auto
  hence ∃ r ∈ ?S. ∀ t ∈ ?S. t ≤ r ..
  then obtain mblk
    where mblk ∈ allBlocksRead s p
          ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk) (is ?P mblk)
lemma \texttt{HEndPhase0-dblock-allBlocksRead}:
  \begin{itemize}
  \item \textbf{assumes} act: \texttt{HEndPhase0 s s' p}
  \item \textbf{and} \texttt{inv1: Inv1 s}
  \item \textbf{shows} \texttt{dblock s' p} \in (\lambda x. x (\texttt{mbal := mbal(dblock s' p)})) \cdot \texttt{allBlocksRead s p}
  \end{itemize}
using act \texttt{HEndPhase0-some[OF act inv1]}
by (auto simp add: EndPhase0-def)
\qedsymbol

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

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

with \text{allproc-p} \text{ have} \text{ proc } \text{bk} = p \text{ by auto}

with \text{bk-in-blocksRead inv2a-bk}

show \text{Inv2a-innermost } s \text{ p} \text{ (block bk)} \text{ by blast}

\text{qed}

from \text{act}

have \text{mbal'}-gt: \forall b k \in \text{allBlocksRead } s \text{ p}. \text{mbal } b k \leq \text{mbal } (\text{dblock } s' \text{ p})

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

with \text{mbal'} \text{ allBlocks-inv2a}

show \text{thesis}

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

fix \ t

assume \ t-blocksRead: \ t \in \text{allBlocksRead } s \text{ p}

with \text{allBlocks-inv2a}

have \text{bal } t \leq \text{mbal } t \text{ by (auto simp add: Inv2a-innermost-def)}

moreover

from \ t-blocksRead \text{mbal'}-gt

have \text{mbal } t \leq \text{mbal } (\text{dblock } s' \text{ p}) \text{ by blast}

ultimately show \text{bal } t \leq \text{mbal } (\text{dblock } s' \text{ p})

by \text{auto}

\text{qed}

\text{qed}

\text{lemma HEndPhase0-Inv2a-dblock:}

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

\text{and inv1: Inv1 } s

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

\text{and inv2c: Inv2c-inner } s \text{ p}

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

\text{proof --}

from \text{act inv2a inv2c}

have \text{t1: } \forall \ t \in (\lambda x. \ x (\ {\text{mbal}}:= \text{mbal } (\text{dblock } s' \text{ p})) \text{) \text{ allBlocksRead } s \text{ p}.}

\text{Inv2a-innermost } s \text{ p} \ t

by(blast dest: HEndPhase0-Inv2a-allBlocksRead)

from \text{act inv1}

have \text{dblock'} \text{ s' } \text{ p} \in (\lambda x. \ x (\ {\text{mbal}}:= \text{mbal(dblock } s' \text{ p})) \text{) \text{ allBlocksRead } s \text{ p}

by(simp, blast dest: HEndPhase0-dblock-allBlocksRead)

with \text{t1}

have \text{inv2-dblock: Inv2a-innermost } s \text{ p} \text{ (dblock } s' \text{ p)} \text{ by auto}

with \text{act}

have \text{inp (dblock } s' \text{ p)} \in \text{allInput } s' \text{ } \text{∪ } \text{NotAnInput}

by(auto dest: HNextPart-allInput-or-NotAnInput)

with \text{inv2-dblock}

show \text{thesis}

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

\text{qed}

\text{lemma HEndPhase0-Inv2a-dblock-q:}

\text{assumes act: HEndPhase0 } s \text{ s' p}
and \( \text{inv1: Inv1 } s \)
and \( \text{inv2a: Inv2a-inner } s \ q \)
and \( \text{inv2c: Inv2c-inner } s \ p \)
shows \( \text{Inv2a-innermost } s' q (\text{dblock } s' q) \)

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

qed

theorem \( \text{HEndPhase0-Inv2a} \):
assumes \( \text{inv: Inv2a } s \)
and \( \text{act: HEndPhase0 } s \ s' \ p \)
and \( \text{inv1: Inv1 } s \)
and \( \text{inv2c: Inv2c-inner } s \ p \)
shows \( \text{Inv2a } s' \)

proof (clarsimp simp add: \( \text{Inv2a-def} \ \text{Inv2a-inner-def} \))
  fix \( q \ bk \)
  assume \( bk: bk \in \text{blocksOf } s' q \)
  with \( \text{HEndPhase0-blocksOf[OF act]} \)
  have \( \text{dblock-blocks: bk} \in \{\text{dblock } s' q\} \cup \text{blocksOf } s \)
    by blast
  thus \( \text{Inv2a-innermost } s' q \ bk \)

proof
  from inv
  have \( \text{inv-q: Inv2a-inner } s \ q \)
    by (auto simp add: \( \text{Inv2a-def} \))
  assume \( bk \in \{\text{dblock } s' q\} \)
  with act \( \text{inv1} \ \text{inv2c} \ \text{inv-q} \)
  show \( ?\text{thesis} \)
    by (blast dest: \( \text{HEndPhase0-Inv2a-dblock-q} \))

next
  assume \( bk-in-blocks: bk \in \text{blocksOf } s \ q \)
  with inv
  have \( \text{Inv2a-innermost } s \ q \ bk \)
    by (auto simp add: \( \text{Inv2a-def} \ \text{Inv2a-inner-def} \))
  with act show \( ?\text{thesis} \)
    by (auto simp add: \( \text{EndPhase0-def} \ \text{HNextPart-def} \ \text{InitializePhase-def} \ \text{Inv2a-innermost-def} \))
lemma HEndPhase1-blocksOf:
HEndPhase1 s s' p \implies blocksOf s' q \subseteq blocksOf s q \cup \{dblock s' q\}
by (auto simp add: EndPhase1-def blocksOf-def dest: subsetI[OF InitializePhase-rdBy])

lemma maxBlk-in-nonInitBlks:
assumes b: b \in nonInitBlks s p
and inv1: Inv1 s
shows maxBlk s p \in nonInitBlks s p
\land (\forall c \in nonInitBlks s p. bal c \leq bal (maxBlk s p))
proof -
have nibals-finite: finite (bal \cdot (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 \cdot nonInitBlks s p \neq \{}
  by auto
with nibals-finite
have Max ?S \in ?S and \forall bb \in ?S. bb \leq Max ?S by auto
hence \exists mb \in ?S. \forall bb \in ?S. bb \leq mb ..
then obtain mblk
  where mblk \in nonInitBlks s p
  \land (\forall c \in nonInitBlks s p. bal c \leq 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:
(\forall p bk: bk \in blocksOf s p \rightarrow P bk)
\implies bk \in nonInitBlks s p \rightarrow 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
  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'

35
by (blast dest: HEndPhase1-HInv1)
from
have inv2a: Inv2a-innermost s p (dblock s p)
  by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
from
have mbal': mbal (dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
moreover
from
have bal': bal (dblock s' p) = mbal (dblock s p)
  by (auto simp add: EndPhase1-def)
moreover
from
have inp': inp (dblock s' p) ∈ allInput s'
  by (blast dest: HEndPhase1-dblock-allInput)
moreover
with
have inp (dblock s' p) ≠ NotAnInput
  by (auto simp add: HInv1-def)
ultimately show ?thesis
  using act inv2a
  by (auto simp add: Inv2a-innermost-def EndPhase1-def)
qed

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

theorem HEndPhase1-Inv2a:
  assumes act: HEndPhase1 s s' p
  and inv1: HInv1 s

36
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
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'

37
by (auto simp add: Phase1or2Write-def Inv2b-def Inv2b-inner-def hasRead-def)

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

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

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

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

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

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

C.2.3 Proofs of Invariant 2 c

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

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

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

qed

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

qed

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

theorem HPhase1or2ReadThen-Inv2c:
  assumes inv: Inv2c s
  and act: HPhase1or2ReadThen s s' p d q
  and inv2a: Inv2a s
  shows Inv2c s'
proof
  from inv2a act
  have inv2a': Inv2a s'
    by (blast dest: HPhase1or2ReadThen-Inv2a)
  from act inv
  have outpt': ∀ p. outpt s' p = (if phase s' p = 3
      then inpt (dblock s' p)
      else NotAnInput)
    by (auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: ∀ p. inpt (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 \( \forall p. \text{ inp} s' p \in \text{ allInput} s' \land (\text{ chosen } s' = \text{ NotAnInput} \rightarrow \text{ outpt } s' p = \text{ NotAnInput}) \)
by (auto dest: HNextPart-chosen HNextPart-allInput)
with \text{ outpt'} \text{ chosen' } \text{ act inv}
show \( ?\text{thesis} \)
by (auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)
qed

theorem HStartBallot-Inv2c:
assumes \( \text{ inv: Inv2c } s \) and \( \text{ act: HStartBallot } s s' p \) and \( \text{ inv2a: Inv2a } s \)
shows \( \text{ Inv2c } s' \)
proof –
  from act
have phase': \( \text{ phase } s' p = 1 \)
by (simp add: StartBallot-def)
  from act
have phase: \( \text{ phase } s p \in \{1,2\} \)
by (simp add: StartBallot-def)
  from act inv
have mbal': \( \text{ mbal}(\text{ dblock } s' p) \in \text{ Ballot } p \)
by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
  from inv phase
have bal(\text{ dblock } s p) \in \text{ Ballot } p \cup \{0\}
by (auto simp add: Inv2c-def Inv2c-inner-def)
  with act
have bal': \( \text{ bal}(\text{ dblock } s' p) \in \text{ Ballot } p \cup \{0\} \)
by (auto simp add: StartBallot-def)
  from act inv phase phase'
have blks': \( \forall d. \forall \text{ br } \in \text{ blocksRead } s' p \ d. \)
\( \text{ mbal}(\text{ block } \text{ br}) < \text{ mbal}(\text{ dblock } s' p) \)
by (auto simp add: StartBallot-def InitializePhase-def Inv2c-def Inv2c-inner-def)
  from inv2a act
have inv2a': \( \text{ Inv2a } s' \)
by (blast dest: HStartBallot-Inv2a)
  from act inv
have outpt': \( \forall p. \text{ outpt } s' p = (\text{ if } \text{ phase } s' p = 3 \)
then \( \text{ inp}(\text{ dblock } s' p) \)
else \( \text{ NotAnInput} \)\)
by (auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
  from inv2a'
have dblk: \( \forall p. \text{ inp } (\text{ dblock } s' p) \in \text{ allInput } s' \cup \{\text{ NotAnInput}\} \)
by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
have chosen': \( \text{ chosen } s' \in \text{ allInput } s' \cup \{\text{ NotAnInput}\} \)
by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \( \forall p. \) inpt \( s \) \( p \) \( \in \) allInput \( s \)
\( \land (\)chosen \( s \) = NotAnInput
\( \rightarrow \) 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 inv2a: Inv2a \( s \)
shows Inv2a \( s' \)
proof
from inv2a act
have inv2a': Inv2a \( s' \)
by(blast dest: HPhase1or2Write-Inv2a)
from act inv
have outpt': \( \forall p. \) outpt \( s \) \( p \) = (if phase \( s \) \( p \) = 3
then inp(dblock \( s \) \( p \))
else NotAnInput)
by(auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: \( \forall p. \) inp (dblock \( s \) \( p \)) \( \in \) allInput \( s \) \( \cup \) \{NotAnInput\}
by(auto simp add: Inv2a-def Inv2a-inner-def
Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen \( s \) \( \in \) allInput \( s \) \( \cup \) \{NotAnInput\}
by(auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \( \forall p. \) inpt \( s \) \( p \) \( \in \) allInput \( s \) \( \land (\)chosen \( s \) = NotAnInput
\( \rightarrow \) outpt \( s \) \( p \) = NotAnInput\)
by(auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
by(auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
qed

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

theorem HEndPhase1-Inv2c:
assumes inv: Inv2c \( s \)
and act: HEndPhase1 \( s \) \( s' \) \( p \)
and inv2a: Inv2a \( s \)
and inv1: HInv1 s
shows Inv2c s'
proof –
  from inv
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  with inv2a act inv1
  have inv2a': Inv2a s'
    by (blast dest: HEndPhase1-Inv2a)
  from act inv
  have mbal': mbal(dblock s' p) ∈ Ballot p
    by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
  from act
  have bal': bal(dblock s' p) = mbal(dblock s' p)
    by (auto simp add: EndPhase1-def)
  from act inv
  have blks': (∀ d. ∀ br ∈ blocksRead s' p d.
    mbal(block br) < mbal(dblock s' p))
    by (auto simp add: EndPhase1-def InitializePhase-def
      Inv2c-def Inv2c-inner-def)
  from act inv
  have outpt': ∀ p. outpt s' p = (if phase s' p = 3
    then inp(dblock s' p)
    else NotAnInput)
    by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: ∀ p. inp(dblock s' p) ∈ allInput s' ∪ {NotAnInput}
    by (auto simp add: Inv2a-def Inv2a-inner-def
      Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
    by (auto dest: HNextPart-Inv2c-chosen)
  from act inv
  have allinp: ∀ p. inp' s' p ∈ allInput s'
    ∧ (chosen' s' = NotAnInput
      −→ outpt s' p = NotAnInput)
    by (auto dest: HNextPart-chosen HNextPart-allInput)
  with mbal' bal' blks' outpt' chosen' act inv
  show ?thesis
    by (auto simp add: EndPhase1-def InitializePhase-def
      Inv2c-def Inv2c-inner-def)
qed

theorem HEndPhase2-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase2 s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
proof –
  from inv2a act

42
have \( \text{inv2a': Inv2a s'} \)
by (blast dest: \text{HEndPhase2-Inv2a})
from act inv
have \( \text{outpt': } \forall p. \text{ outpt } s' p = (\text{if phase } s' p = 3 \text{ then inp(dblock } s' p) \text{ else NotAnInput}) \)
by (auto simp add: \text{EndPhase2-def Inv2c-def Inv2c-inner-def})
from \( \text{inv2a'} \)
have \( \text{dblk: } \forall p. \text{ inp (dblock } s' p) \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
by (auto simp add: \text{Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def})
with act inv outpt'
have \( \text{chosen': chosen } s' \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
by (auto dest: \text{HNextPart-Inv2c-chosen})
from act inv
have \( \text{allinp: } \forall p. \text{ inpt } s' p \in \text{allInput } s' \)
\( \wedge (\text{chosen } s' = \text{NotAnInput} \rightarrow \text{outpt } s' p = \text{NotAnInput}) \)
by (auto dest: \text{HNextPart-chosen HNextPart-allInput})
with outpt' chosen' act inv
show \( \text{thesis} \)
by (auto simp add: \text{EndPhase2-def InitializePhase-def Inv2c-def Inv2c-inner-def})
qed

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

theorem HEndPhase0-Inv2c:
  assumes inv: Inv2c s
  and act: HEndPhase0 s s' p
  and inv2a: Inv2a s
  and inv1: Inv1 s
  shows Inv2c s'
proof
  from inv
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  with inv2a act inv1

44
have inv2a': Inv2a s'
  by (blast dest: HEndPhase0-Inv2a)
hence bal': bal(dblock s' p) ∈ Ballot p ∪ {0}
  by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
from act inv
have mbal': mbal(dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase0-def Inv2c-def Inv2c-inner-def)
from act inv
have blks': (∀ d. ∀ br ∈ blocksRead s' p d.
        mbal(block br) < mbal(dblock s' p))
  by (auto simp add: EndPhase0-def InitializePhase-def Inv2c-def Inv2c-inner-def)
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: EndPhase0-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)
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' blks' outpt' chosen' act inv
show ?thesis
  by (auto simp add: EndPhase0-def InitializePhase-def Inv2c-def Inv2c-inner-def)
qed

theorem HInit-HInv2:
  HInit s ⇒ HInv2 s
using HInit-Inv2a HInit-Inv2b HInit-Inv2c
by (auto simp add: HInv2-def)
HInv1 ∧ HInv2 is an invariant of HNext.

lemma I2b:
  assumes nxt: HNext s s'
  and inv: HInv1 s ∧ HInv2 s
  shows HInv2 s'
proof (auto simp add: HInv2-def)
  show Inv2a s' using assms
    by (auto simp add: HInv2-def HNext-def Next-def, auto intro: HStartBallot-Inv2a,
theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

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

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

**definition** \( HInv3-R :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \)

where

\[ HInv3-R \ s \ p \ q \ d = ( (\langle \text{block} = \text{dblock} \ s \ q, \text{proc} = q \rangle \in \text{blocksRead} \ s \ p \ d \ \lor \ (\langle \text{block} = \text{dblock} \ s \ p, \text{proc} = p \rangle \in \text{blocksRead} \ s \ q \ d) ) \]

**definition** \( HInv3-inner :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \)

where

\[ HInv3-inner \ s \ p \ q \ d = ( HInv3-L \ s \ p \ q \ d \rightarrow HInv3-R \ s \ p \ q \ d ) \]

**definition** \( HInv3 :: \text{state} \Rightarrow \text{bool} \)

where

\[ HInv3 \ s = (\forall \ p \ q \ d. \ HInv3-inner \ s \ p \ q \ d) \]

### C.3.1 Proofs of Invariant 3

**theorem** \( HInit-HInv3 :: HInit \ s \Rightarrow HInv3 \ s \)

by (simp add: HInit-def Init-def HInv3-def HInv3-inner-def HInv3-L-def HInv3-R-def)

**lemma** \( InitPhase-HInv3-p :: [\text{InitializePhase} \ s \ s' \ p; \ HInv3-L \ s' \ p \ q \ d] \Rightarrow HInv3-R \ s' \ p \ q \ d \)

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

**lemma** \( InitPhase-HInv3-q :: [\text{InitializePhase} \ s \ s' \ q; \ HInv3-L \ s' \ p \ q \ d] \Rightarrow HInv3-R \ s' \ p \ q \ d \)

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

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

by (auto simp add: HInv3-L-def)

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

by (auto simp add: HInv3-R-def)

**lemma** \( Phase1or2ReadThen-HInv3-pq :: \)

assumes act: \( \text{Phase1or2ReadThen} \ s \ s' \ p \ d \ q \)

and inv-L': \( HInv3-L \ s' \ p \ q \ d \)

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

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

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

**proof**

from \( \text{inv-L'} \) \( \text{act pq} \)

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

by (auto simp add: Phase1or2ReadThen-def HInv3-L-def hasRead-def split: if-split-asm)
with inv2b
have disk s d q = dblock s q
  by (auto simp add: Inv2b-def Inv2b-inner-def
       hasRead-def)
with act
show ?thesis
  by (auto simp add: Phase1or2ReadThen-def HInv3-def
       HInv3-inner-def HInv3-R-def)
qed

lemma Phase1or2ReadThen-HInv3-hasRead:
  [¬hasRead s pp dd qq; Phase1or2ReadThen s s′ p d q;
   pp ≠ p ∨ qq ≠ q ∨ dd ≠ d] ⇒ ¬hasRead s′ pp dd qq
  by (auto simp add: hasRead-def Phase1or2ReadThen-def)

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

48
∧ hasRead s' pp dd qq ∧ hasRead s' qq dd pp
by (auto simp add: HInv3-L-def Phase1or2ReadThen-def)
with act False
show ?thesis
by (auto dest: Phase1or2ReadThen-HInv3-hasRead)
qed
qed
qed

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

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

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

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

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

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

show ?thesis
  by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-t)

qed

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

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

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

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

lemma EndPhase1-HInv3-t:
\[
[ \text{EndPhase1} \ s \ s' \ t; \ \text{HInv3-inner} \ s \ p \ q \ d; \ t \neq p; \ t \neq q ]
\implies \text{HInv3-inner} \ s' \ p \ q \ d
\]
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:
\[
[ \text{HEndPhase1} \ s \ s' \ p; \ \text{HInv3} \ s ] \implies \text{HInv3} \ s'
\]
by (auto simp add: HInv3-def dest: EndPhase1-HInv3)

lemma EndPhase2-HInv3-p:
\[
[ \text{EndPhase2} \ s \ s' \ p; \ \text{HInv3-L} \ s' \ p \ q \ d ] \implies \text{HInv3-R} \ s' \ p \ q \ d
\]
by (auto simp add: EndPhase2-def dest: InitPhase-HInv3-p)

lemma EndPhase2-HInv3-q:
\[
[ \text{EndPhase2} \ s \ s' \ q; \ \text{HInv3-L} \ s' \ p \ q \ d ] \implies \text{HInv3-R} \ s' \ p \ q \ d
\]
by (auto simp add: EndPhase2-def dest: InitPhase-HInv3-q)

lemma EndPhase2-HInv3-nL:
\[
[ \text{EndPhase2} \ s \ s' \ t; \ \neg \text{HInv3-L} \ s \ p \ q \ d; \ t \neq p; \ t \neq q ]
\implies \neg \text{HInv3-L} \ s' \ p \ q \ d
\]
by (auto simp add: EndPhase2-def InitializePhase-def HInv3-L-def hasRead-def)

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

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

lemma EndPhase2-HInv3:
assumes act: EndPhase2 \ s \\ s' \ t
and inv: HInv3-inner \ s \\ p \\ q \\ d
shows HInv3-inner \ s' \\ p \\ q \\ d
proof (cases \ t = p \lor 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:
\[
\text{HEndPhase2 } s \\ s' \\ p; \\ HInv3 s
\implies HInv3 s'
\]
by (auto simp add: HInv3-def dest: EndPhase2-HInv3)

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

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

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

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

lemma Fail-HInv3: 
\begin{align*}
\text{assumes } & \text{act: Fail } s \ s' \ t \\
\text{and } & \text{inv: HInv3-inner } s \ p \ q \ d \\
\text{shows } & HInv3-inner \ s' \ p \ q \ d \\
\text{proof}\text{(cases } t = p \lor t = q) \\
\text{case False} & \\
\text{with } & \text{inv act} \\
\text{show } & \text{?thesis} \\
& \text{by(auto simp add: HInv3-inner-def dest: Fail-HInv3-p Fail-HInv3-q)}
\end{align*}

next
\begin{align*}
\text{case True} & \\
\text{with } & \text{inv act} \\
\text{show } & \text{?thesis} \\
& \text{by(auto simp add: HInv3-inner-def dest: Fail-HInv3-t)}
\end{align*}
qed

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

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

lemma EndPhase0-HInv3-p:
  [ EndPhase0 s s' p; HInv3-L s' p q d ]
  \implies 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 ]
  \implies 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 \neq p; t \neq q ]
  \implies ¬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 \neq p; t \neq q ]
  \implies 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 \neq p; t \neq q ]
  \implies 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 \lor t=q)
  case True
  with act inv
  show ?thesis
    by (auto simp add: HInv3-inner-def
      dest: EndPhase0-HInv3-p EndPhase0-HInv3-q)
next
theorem HEndPhase0-HInv3:
  \[ H\text{EndPhase0} s s' \Rightarrow H\text{Inv3} s' \]
by (auto simp add: HInv3-def dest: EndPhase0-HInv3)

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

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

theory DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

C.4 Invariant 4

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

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

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

definition $HInv4a2 :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$
where
$HInv4a2 \ s \ p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal(\text{disk} \ s \ d \ p) \le mbal(\text{dblock} \ s \ p))$
\wedge bal(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$

definition $HInv4a :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$
where
$HInv4a \ s \ p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$
\wedge \text{bal}(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$

definition $HInv4b :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$
where
$HInv4b \ s \ p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$
\wedge \text{bal}(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$

definition $HInv4c :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$
where
$HInv4c \ s \ p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal(\text{disk} \ s \ d \ p) = \text{bal}(\text{dblock} \ s \ p))$
\wedge \text{bal}(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$

definition $HInv4d :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$
where
$HInv4d \ s \ p = (\forall D \in \text{MajoritySet}. (\exists d \in D. \ mbal(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$
\wedge \text{bal}(\text{disk} \ s \ d \ p) \le \text{bal}(\text{dblock} \ s \ p))$

definition $HInv4 :: \text{state} \Rightarrow \text{bool}$
where
$HInv4 \ s = (\forall p. \ HInv4a \ s \ p \wedge HInv4b \ s \ p \wedge HInv4c \ s \ p \wedge HInv4d \ s \ p)$

The initial state implies Invariant 4.

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

using $\text{Disk-isMajority}$

by (auto simp add: $HInit$-def $Init$-def $HInv4$-def $HInv4a1$-def $HInv4a2$-def $HInv4b$-def $HInv4c$-def $HInv4d$-def $\text{MajoritySet}$-def $\text{blocksOf}$-def $\text{InitDB}$-def $\text{rdBy}$-def)

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

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

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

**Lemma** $HStartBallot-HInv4a1$:

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

shows $HInv4a1 \ s' \ p$

proof (auto simp add: $HInv4a1$-def)

fix $bk$
assume bk ∈ blocksOf s' p
with HStartBallot-blocksOf[OF act]
have bk ∈ {dblock s' p} ∪ blocksOf s p
  by blast
thus bal bk ≤ mbal (dblock s' p)
proof
  assume bk ∈ {dblock s' p}
  with inv2a
  show ?thesis
    by(auto simp add: Inv2a-innermost-def Inv2a-inner-def blocksOf-def)
next
  assume bk ∈ blocksOf s p
  with inv act
  show ?thesis
    by(auto simp add: StartBallot-def HInv4a1-def)
qed
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 \( act: HStartBallot \ s \ s' \ p \)
  and \( inv: HInv4a \ s \ q \)
  and \( pnq: p\neq q \)
  shows \( HInv4a \ s' \ q \)
proof
  from \( act \ pnq \)
  have blocksOf \( s' \ q \subseteq \) blocksOf \( s \ q \)
    by(auto simp add: StartBallot-def InitializePhase-def
        blocksOf-def rdBy-def)
  with \( act \ inv \ pnq \)
  show \( ?thesis \)
    by(auto simp add: StartBallot-def HInv4a-def
        HInv4a1-def HInv4a2-def)
qed

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

lemma Phase1or2Write-HInv4a1:
lemma Phase1or2Write-HInv4a1:
\[\begin{align*}
\text{Phase1or2Write } s & \quad s' \\
HInv4a1 & \quad s \\
\Rightarrow & \quad HInv4a1 \\
\end{align*}\]
by (auto simp add: Phase1or2Write-def HInv4a1-def blocksOf-def rdBy-def)

lemma Phase1or2Write-HInv4a2:
\[\begin{align*}
\text{Phase1or2Write } s & \quad s' \\
HInv4a2 & \quad s \\
\Rightarrow & \quad HInv4a2 \\
\end{align*}\]
by (auto simp add: Phase1or2Write-def HInv4a2-def)

theorem HPhase1or2Write-HInv4a:
assumes act: HPhase1or2Write s s' p d
and inv: HInv4a s q
shows HInv4a s' q
proof -
  from act have phase': phase s = phase s'
    by (simp add: Phase1or2Write-def)
  show ?thesis
  proof (cases phase s q = 0)
    case True
    with phase' act show ?thesis
      by (auto simp add: HInv4a-def)
  next
    case False
    with phase' act inv show ?thesis
    proof (auto simp add: HInv4a-def dest: Phase1or2Write-HInv4a1 Phase1or2Write-HInv4a2)
  qed
qed

lemma HPhase1or2ReadThen-HInv4a1-p:
assumes act: HPhase1or2ReadThen s s' p d q
and inv: HInv4a1 s p
shows HInv4a1 s' p
proof (auto simp: HInv4a1-def)
  fix bk
  assume bk: bk \in blocksOf s' p
  with HPhase1or2ReadThen-blocksOf[OF act] have bk \in blocksOf s p by auto
  with inv act show bal bk \leq mbal (dblock s' p)
    by (auto simp add: HInv4a1-def Phase1or2ReadThen-def)
qed

lemma HPhase1or2ReadThen-HInv4a2:
\[\begin{align*}
\text{HPhase1or2ReadThen } s & \quad s' \\
HInv4a2 & \quad s \\
\Rightarrow & \quad HInv4a2 \\
\end{align*}\]
by (auto simp add: Phase1or2ReadThen-def HInv4a2-def)
lemma HPhase1or2ReadThen-HInv4a-p:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv4a s p
and inv2b: Inv2b s
shows HInv4a s' p
proof –
from act inv2b
have phase s p ∈ {1, 2}
  by(auto simp add: Phase1or2ReadThen-def Inv2b-def Inv2b-inner-def)
with act inv
show ?thesis
  by(auto simp del: HPhase1or2ReadThen-def simp add: HInv4a-def
      HInv4a1-def HInv4a2-def)
qed

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

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

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

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

proof(auto simp add: HInv4a1-def)
fix bk
assume bk: $bk \in \text{blocksOf} s' p$
from bk $H_{EndPhase1-blocksOf}[OF \text{act}]$
have bk $\in \{\text{dblock s' p}\} \cup \text{blocksOf} s p$
  by blast
with act inv
show $bal bk \leq mbal (\text{dblock s' p})$
  by(auto simp add: HInv4a-def HInv4a1-def EndPhase1-def)
qed

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

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

proof –
  from act
  have phase: $0 < \text{phase} \ s \ p$
  by(auto simp add: EndPhase1-def)
  with act inv inv2a
  show ?thesis
  by(auto simp del: HEndPhase1-def simp add: HInv4a-def
    elim: HEndPhase1-HInv4a1 HEndPhase1-HInv4a2)
qed

lemma HEndPhase1-HInv4a-q:
  assumes act: HEndPhase1 $s \ s' \ p$
  and inv: $HInv4a \ s \ q$
  and pnq: $p \neq q$
  shows $HInv4a \ s' \ q$
proof –
  from act pnq
  have dblock $s' \ q = \text{dblock} \ s \ q \land \text{disk} \ s' = \text{disk} \ s$
  by(auto simp add: EndPhase1-def)
  moreover
  from act pnq
  have $\forall p \ d. \ \text{rdBy} \ s' \ q \ p \ d \subseteq \text{rdBy} \ s \ q \ p \ d$
  by(auto simp add: EndPhase1-def InitializePhase-def rdBy-def)
  hence $(\bigcup p \ d. \ \text{rdBy} \ s' \ q \ p \ d) \subseteq (\bigcup p \ d. \ \text{rdBy} \ s \ q \ p \ d)$
  by(auto, blast)
  ultimately
  have blocksOf $s' \ q \subseteq \text{blocksOf} \ s \ q$
  by(auto simp add: blocksOf-def, blast)
  with act inv pnq
  show ?thesis
  by(auto simp add: EndPhase1-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

theorem HEndPhase1-HInv4a:
  $[ \text{HEndPhase1} \ s \ s' \ p; \ HInv4a \ s \ q; \ Inv2a \ s ] \Longrightarrow HInv4a \ s' \ q$
by(blast dest: HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q)

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

theorem HPhase0Read-HInv4a:
  $[ \text{HPhase0Read} \ s \ s' \ p \ d; \ HInv4a \ s \ q ] \Longrightarrow HInv4a \ s' \ q$
by(auto simp add: Phase0Read-def HInv4a-def HInv4a1-def
    HInv4a2-def InitializePhase-def
    blocksOf-def rdBy-def)
**Theorem** $H_{\text{EndPhase2-4a}}$:

\[
[H_{\text{EndPhase2}} s s' p; H_{\text{Inv4a}} s q] \implies H_{\text{Inv4a}} s' q
\]

by (auto simp add: EndPhase2-def $H_{\text{Inv4a-def}}$ $H_{\text{Inv4a1-def}}$ $H_{\text{Inv4a2-def}}$ InitializePhase-def blocksOf-def rdBy-def)

**Lemma** $\text{allSet}$:

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

shows $Q rb$

**Proof**

- from $aPQ$ have $\forall r \in P d. Q r$ by auto
- with $rb$
- show $\text{thesis}$ by auto

qed

**Lemma** $\text{EndPhase0-44}$:

assumes $act$: $\text{EndPhase0} s s' p$
and $bk$: $bk \in \text{blocksOf} s p$
and $inv4d$: $H_{\text{Inv4d}} s p$
and $inv2c$: $\text{Inv2c-inner} s p$

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

**Proof**

- from $bk$ $inv4d$
- have $\exists D1 \in \text{MajoritySet}. \forall d \in D1. \text{bal} bk \leq \text{mbal} (\text{disk} s d p)$ — 4.2
  - by (auto simp add: $H_{\text{Inv4d-def}}$)
- with $\text{majorities-intersect}$
- have $p43$: $\forall D \in \text{MajoritySet}. \exists d \in D. \text{bal} bk \leq \text{mbal} (\text{disk} s d p)$
  - by (simp add: MajoritySet-def, blast)
- from $act$
- have $\forall d. \forall rb \in \text{blocksRead} s p d. \text{block} rb = \text{disk} s d p$ — 5.1
  - by (simp add: Inv2c-inner-def)
- hence $\forall d. \exists \text{hasRead} s p d p$
  - $\implies (\exists \text{rb}\in\text{blocksRead} s p d. \text{block} rb = \text{disk} s d p)$ — 5.2
  - (is $\forall d. \exists H d \implies \exists P d$)
  - by (auto simp add: hasRead-def)
- with $act$
- have $p53$: $\exists D \in \text{MajoritySet}. \forall d \in D. \exists P d$
  - by (auto simp add: MajoritySet-def EndPhase0-def)
- show $\text{thesis}$

**Proof**

- from $p43$ $p53$
- have $\exists D \in \text{MajoritySet}. (\exists d \in D. \text{bal} bk \leq \text{mbal} (\text{disk} s d p))$
  - $\wedge (\forall d \in D. \exists P d)$
  - by auto
- thus $\text{thesis}$
  - by force

qed
lemma $H_{\text{EndPhase0}}-H_{\text{Inv4a1}}$-
p
  assumes $\text{act} \colon H_{\text{EndPhase0}} \ s \ s' \ p$
  and $\text{inv2a}' \colon \text{Inv2a} \ s'$
  and $\text{inv2c} \colon \text{Inv2c-inner} \ s \ p$
  and $\text{inv4d} \colon H_{\text{Inv4d}} \ s \ p$
  shows $H_{\text{Inv4a1}} \ s' \ p$
proof
  (auto simp add: $H_{\text{Inv4a1}}$-def)
  fix $bk$
  assume $bk \in \text{blocksOf} \ s' \ p$
  with $H_{\text{EndPhase0}}$-blocksOf[OF $\text{act}$]
  have $bk \in \{\text{dblock} \ s' \ p\} \cup \text{blocksOf} \ s \ p$
  by auto
  thus $\text{bal} \ bk \leq \text{mbal} \ (\text{dblock} \ s' \ p)$
  proof
    assume $bk \in \{\text{dblock} \ s' \ p\}$
    with $\text{inv2a}'$
    have $\text{Inv2a-innermost} \ s' \ p \ bk$
    by (auto simp add: $\text{Inv2a-def}$ $\text{Inv2a-inner-def}$ blocksOf-def)
    with $bk$ show $?\text{thesis}$
    by (auto simp add: $\text{Inv2a-innermost-def}$)
  next
    assume $bk \in \text{blocksOf} \ s \ p$
    from $\text{act}$
    have $f1 \colon \forall \ r \in \text{allBlocksRead} \ s \ p. \ \text{mbal} \ r < \text{mbal} \ (\text{dblock} \ s' \ p)$
    by (auto simp add: $H_{\text{EndPhase0}}$-def)
    with act $\text{inv4d}$ $\text{inv2c}$ $bk$
    have $\exists \ d. \ \exists\ \text{rb} \in \text{blocksRead} \ s \ p \ d. \ \text{bal} \ bk \leq \text{mbal} (\text{block} \ \text{rb})$
    by (auto dest: $H_{\text{EndPhase0}}$-44)
    with $f1$
    show $?\text{thesis}$
    by (auto simp add: $H_{\text{EndPhase0}}$-def allBlocksRead-def
      allRdBiks-def dest: allSet)
qed

lemma $\text{hasRead-allBlks}$:
  assumes $\text{inv2c} \colon \text{Inv2c-inner} \ s \ p$
  and $\text{phase} \colon \text{phase} \ s \ p = 0$
  shows $\forall d \in \{d. \ \text{hasRead} \ s \ p \ d \ p\}. \ \text{disk} \ s \ d \ p \in \text{allBlocksRead} \ s \ p$
proof
  fix $d$
  assume $d \in \{d. \ \text{hasRead} \ s \ p \ d \ p\}$
  is $d \in \text{?D}$
  hence $\text{br-ne} : \text{blocksRead} \ s \ p \ d \neq \{\}$
  by (auto simp add: $\text{hasRead-def}$)
  from $\text{inv2c}$ $\text{phase}$
  have $\forall \ \text{br} \in \text{blocksRead} \ s \ p \ d. \ \text{block} \ \text{br} = \text{disk} \ s \ d \ p$
  by (auto simp add: $\text{Inv2c-inner-def}$)
  with $\text{br-ne}$
have \( \text{disk } s \ d \ p \in \text{block } \) \( \text{blocksRead s p d} \)
   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 inv1: Inv1 s
and inv2c: Inv2c-inner s p
shows \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{mbal(disk s d p)} \leq \text{mbal(dblock s' p)} \)
   \( \land \text{bal(disk s d p)} \leq \text{bal(dblock s' p)} \)
proof
  from act HEndPhase0-some[OF act inv1]
  have p51: \( \forall br \in \text{allBlocksRead s p}. \ \text{mbal br} < \text{mbal(dblock s' p)} \)
   \( \land \text{bal br} \leq \text{bal(dblock s' p)} \)
   and a: IsMajority(\{d. hasRead s p d p\})
   and phase: phase s p = 0
   by (auto simp add: EndPhase0-def)+
  from inv2c phase
  have \( \forall d \in \{d. \text{hasRead s p d p}\}. \text{disk s d p} \in \text{allBlocksRead s p} \)
   by (auto dest: hasRead-allBlks)
with p51
  have \( \forall d \in \{d. \text{hasRead s p d p}\}. \ \text{mbal(disk s d p)} \leq \text{mbal(dblock s' p)} \)
   \( \land \text{bal(disk s d p)} \leq \text{bal(dblock s' p)} \)
   by force
with a show ?thesis
   by (auto simp add: MajoritySet-def)
qed

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

65
and \( \text{inv2c: Inv2c-inner s p} \)
shows \( \text{HInv4a2 s' p} \)

**proof** (*simp add: HInv4a2-def*)

from act
have disk': disk s' = disk s
  by (*simp add: EndPhase0-def*)
from act inv1 inv2c
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \)  
  \( \text{mbal(disk s d p) \leq mbal(dblock s' p)} \)
\( \wedge \text{bal(disk s d p) \leq bal(dblock s' p)} \)
  by (*blast dest: HEndPhase0-41*)
from Majority-exQ[OF this]
have \( \forall D \in \text{MajoritySet}. \exists d \in D. \)  
  \( \text{mbal(disk s d p) \leq mbal(dblock s' p)} \)
\( \wedge \text{bal(disk s d p) \leq bal(dblock s' p)} \)
(is ?P (disk s)).
from ssubst[OF disk', of ?P, OF this]
show \( \forall D \in \text{MajoritySet}. \exists d \in D. \)  
  \( \text{mbal(disk s' d p) \leq mbal(dblock s' p)} \)
\( \wedge \text{bal(disk s' d p) \leq bal(dblock s' p)} \).

qed

**lemma** \( \text{HEndPhase0-HInv4a-p:} \)

assumes act: \( \text{HEndPhase0 s s' p} \)
and \( \text{inv2a: Inv2a s} \)
and \( \text{inv2: Inv2c s} \)
and \( \text{inv4d: HInv4d s p} \)
and \( \text{inv1: Inv1 s} \)
and \( \text{inv: HInv4a s p} \)
sows \( \text{HInv4a s' p} \)

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

qed

**lemma** \( \text{HEndPhase0-HInv4a-q:} \)

assumes act: \( \text{HEndPhase0 s s' p} \)
and \( \text{inv: HInv4a s q} \)
and \( \text{pnq: p\#q} \)
sows \( \text{HInv4a s' q} \)

**proof** –

66
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)
with act inv pnq
show ?thesis
  by (auto simp add: EndPhase0-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

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

C.4.2 Proofs of Invariant 4b

lemma blocksRead-allBlocksRead:
rb ∈ blocksRead s p d ⇒ block rb ∈ allBlocksRead s p
by (auto simp add: allBlocksRead-def allRdBlks-def)

lemma HEndPhase0-dblock-mbal:
[ [ HEndPhase0 s s' p ] ]
⇒ ∀br ∈ allBlocksRead s p. mbal br < mbal(dblock s' p)
by (auto simp add: EndPhase0-def)

lemma HEndPhase0-HInv4b-p-dblock:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a s
and inv2c: Inv2c-inner s p
shows bal(dblock s' p) < mbal(dblock s' p)
proof
  from act have phase s p = 0 by (auto simp add: EndPhase0-def)
  with inv2c
  have ∀d.∀br ∈ blocksRead s p d. proc br = p ∧ block br = disk s d p
    by (auto simp add: Inv2c-inner-def)
hence allBlks-in-blocksOf: allBlocksRead s p ⊆ blocksOf s p
    by (auto simp add: allBlocksRead-def allRdBlks-def blocksOf-def)
from act HEndPhase0-some[OF act inv1]
have \( p53 : \exists \, br \in \text{allBlocksRead} s \, p. \, \text{bal}(\text{dblock} s' p) = \text{bal} br \)

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

from \( \text{inv2a} \)

have \( i2 : \forall \, p. \, \forall \, bk \in \text{blocksOf} s \, p. \, \text{bal} bk \leq \text{mbal} bk \)

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

with \( \text{allBlks-in-blocksOf} \)

have \( \forall \, bk \in \text{allBlocksRead} s \, p. \, \text{bal} bk \leq \text{mbal} bk \)

\( \text{by auto} \)

with \( p53 \)

have \( \exists \, br \in \text{allBlocksRead} s \, p. \, \text{bal}(\text{dblock} s' p) \leq \text{mbal} br \)

\( \text{by force} \)

with \( \text{HEndPhase0-dblock-mbal[OF act]} \)

show \( ?\text{thesis} \)

\( \text{by auto} \)

qed

lemma \( \text{HEndPhase0-HInv4b-p-blocksOf}: \)

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

and \( \text{inv4d : HInv4d s p} \)

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

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

shows \( \text{bal} bk < \text{mbal}(\text{dblock} s' p) \)

proof –

from \( \text{inv4d \, majorities-intersect \, bk} \)

have \( p43 : \forall \, D \in \text{MajoritySet}. \exists \, d \in D. \, \text{bal} bk \leq \text{mbal}(\text{disk} s \, d \, p) \)

\( \text{by (auto simp add: HInv4d-def MajoritySet-def Majority-exQ)} \)

have \( \exists \, br \in \text{allBlocksRead} s \, p. \, \text{bal}(\text{dblock} s' p) \leq \text{mbal} br \)

\( \text{by auto} \)

proof –

from \( \text{act} \)

have \( \text{maj: IsMajority}\{\, d. \, \text{hasRead} s \, p \, d \, p\}\} (\text{is IsMajority(?D)}) \)

and \( \text{phase : phase s p = 0} \)

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

have \( \text{br-ne: \forall \, d \in ?D. \, \text{blocksRead} s \, p \, d \neq \{\}} \)

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

from \( \text{phase \, inv2c} \)

have \( \forall \, d \in ?D. \exists \, br \in \text{blocksRead} s \, p \, d. \, \text{block} \, br = \text{disk} s \, d \, p \)

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

with \( \text{br-ne} \)

have \( \forall \, d \in ?D. \ \exists \, \text{br} \in \text{allBlocksRead} s \, p. \, \text{br} = \text{disk} s \, d \, p \)

\( \text{by (blast dest: blocksRead-allBlocksRead)} \)

with \( p43 \, \text{maj} \)

show \( ?\text{thesis} \)

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

qed

with \( \text{HEndPhase0-dblock-mbal[OF act]} \)

show \( ?\text{thesis} \)

\( \text{by auto} \)

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

lemma HEndPhase0-HInv4b-q:
  assumes act: HEndPhase0 s s' p
  and pq: p≠q
  and inv: HInv4b s q
  shows HInv4b s' q
proof –
from act pq have disk': disk s'=disk s
  and dblock': dblock s' q= dblock s q
  and phase': phase s' q = phase s q
  by(auto simp add: EndPhase0-def)
from act pq have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
  by(auto simp add: EndPhase0-def InitializePhase-def allRdBlks-def)
with disk' dblock'
have blocksOf s' q ⊆ blocksOf s q
  by(auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
with inv phase' dblock'
show \(?thesis\)
  by (auto simp add: HInv4b-def)
qed

theorem HEndPhase0-HInv4b:
  assumes act: HEndPhase0 s s' p
  and inv: HInv4b s q
  and inv4d: HInv4d s p
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2c: Inv2c-inner s p
  shows HInv4b s' q
proof (cases p = q)
  case True
  with HEndPhase0-HInv4b-p[OF act inv4d inv1 inv2a inv2c]
  show \(?thesis\) by 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 \in blocksOf s' p
  from act
  have phase': phase s' p = 1
    and phase: phase s p \in \{1, 2\}
    by (auto simp add: StartBallot-def)
  from act
  have p42: mbal (dblock s p) < mbal (dblock s' p)
    \land bal(dblock s p) = bal(dblock s' p)
    by (auto simp add: StartBallot-def)
  from HStartBallot-blocksOf[OF act] bk
  have bk \in {dblock s' p} \cup blocksOf s p
    by blast
  thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk \in {dblock s' p}
  from inv2a
  have bal (dblock s p) \leq mbal (dblock s p)
    by (auto simp add: Inv2a-innermost-def)
  with p42 bk
show \(?thesis\) by \textit{auto}

next
already assumed \(bk\): \(bk \in \text{blocksOf} \, s \, p\)
from \textit{phase inv4a}
have \(p41: \text{HInv4a1} \, s \, p\)
\hspace{1em} by\,(\textit{auto simp add: HInv4a-def})
with \(p42\) \(bk\)
show \(?thesis\)
\hspace{1em} by\,(\textit{auto simp add: HInv4a1-def})

qed

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

\textbf{proof} –

\hspace{1em} \textbf{from} \text{act} \, \text{pnq}
\hspace{1em} \textbf{have} \(\text{disk}'\): \(\text{disk} \, s' = \text{disk} \, s\)
\hspace{1em} \textbf{and} \(\text{dblock}'\): \(\text{dblock} \, s' \, q = \text{dblock} \, s \, q\)
\hspace{1em} \textbf{and} \(\text{phase}'\): \(\text{phase} \, s' \, q = \text{phase} \, s \, q\)
\hspace{1em} \textbf{by}(\textit{auto simp add: StartBallot-def})

\hspace{1em} \textbf{from} \text{act} \, \text{pnq}
\hspace{1em} \textbf{have} \(\text{blocksRead}'\): \(\forall \, q. \, \text{allRdBlks} \, s' \, q \subseteq \text{allRdBlks} \, s \, q\)
\hspace{1em} \textbf{by}(\textit{auto simp add: StartBallot-def InitializePhase-def allRdBlks-def})

\hspace{1em} \textbf{with} \(\text{disk}'\) \(\text{dblock}'\)
\hspace{1em} \textbf{have} \(\text{blocksOf} \, s' \, q \subseteq \text{blocksOf} \, s \, q\)
\hspace{1em} \textbf{by}(\textit{auto simp add: blocksOf-def})

\hspace{1em} \textbf{with} \(\text{inv phase}' \, \text{dblock}'\)
\hspace{1em} \textbf{show} \(?thesis\)
\hspace{1em} \textbf{by}(\textit{auto simp add: HInv4b-def})

\textbf{qed}

\textbf{theorem} \(\text{HStartBallot-HInv4b}\):
\textbf{assumes} \(\text{act}: \text{HStartBallot} \, s \, s' \, p\)
\textbf{and} \(\text{inv2a}: \text{Inv2a} \, s\)
\textbf{and} \(\text{inv4b}: \text{HInv4b} \, s \, q\)
\textbf{and} \(\text{inv4a}: \text{HInv4a} \, s \, p\)
\textbf{shows} \(\text{HInv4b} \, s \, s' \, q\)

\textbf{using} \(\text{act} \, \text{inv2a} \, \text{inv4b} \, \text{inv4a}\)

\textbf{proof} (cases \(p=q\))
\hspace{1em} \textbf{case} \(\text{True}\)
\hspace{1em} \textbf{from} \(\text{inv2a}\)
\hspace{1em} \textbf{have} \(\text{inv2a-innermost} \, s \, p \, (\text{dblock} \, s \, p)\)
\hspace{1em} \textbf{by}(\textit{auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def})

\hspace{1em} \textbf{with} \(\text{act} \, \text{True} \, \text{inv4b} \, \text{inv4a}\)
\hspace{1em} \textbf{show} \(?thesis\)
\hspace{1em} \textbf{by}(\textit{blast dest: HStartBallot-HInv4b-p})
next
  case False
  with act inv4b
  show ?thesis
    by (blast dest: HStartBallot-HInv4b-q)
qed

theorem HPhase1or2Write-HInv4b:
  [ HPhase1or2Write s s' p d; HInv4b s q ] =⇒ HInv4b s' q
by (auto simp add: Phase1or2Write-def HInv4b-def
     blocksOf-def rdBy-def)

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

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

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

theorem HPhase1or2ReadElse-HInv4b:
  [ HPhase1or2ReadElse s s' p d q; HInv4b s r; Inv2a s; HInv4a s p ]
  =⇒ HInv4b s' r
using HStartBallot-HInv4b
by (auto simp add: Phase1or2ReadElse-def)

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

lemma HEndPhase1-HInv4b-q:
  assumes act: HEndPhase1 s s' p
  and pnq: p ≠ q
and inv: HInv4b s q
shows HInv4b s' q

proof –
  from act pnq
  have disk': disk s'=disk s
    and dblock': dblock s' q=dblock s q
    and phase': phase s' q=phase s q
    by(auto simp add: EndPhase1-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s q ⊆ allRdBlks s q
    by(auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q ⊆ blocksOf s q
    by(auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
  show ?thesis
    by(auto simp add: HInv4b-def)
qed

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

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

lemma HEndPhase2-HInv4b-q:
  assumes act: HEndPhase2 s s' p
  and pnq: p≠q
  and inv: HInv4b s q
  shows HInv4b s' q
proof –
  from act pnq
  have disk': disk s'=disk s
    and dblock': dblock s' q=dblock s q
    and phase': phase s' q=phase s q
    by(auto simp add: EndPhase2-def)
  from act pnq
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 [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: Fail-def)
from act pnq have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
by (auto simp add: Fail-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 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 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: Fail-def)
from act pnq have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
by (auto simp add: Fail-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 HFail-HInv4b:
assumes act: HFail s s' p
and inv: HInv4b s q
shows HInv4b s' q
proof (cases p=q)
case True
with HFail-HInv4b-p[OF act]
show ?thesis by simp
next
case False
from HFail-HInv4b-q[OF act False inv]
show ?thesis.
qed

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

lemma HPhase0Read-HInv4b-q:
assumes act: HPhase0Read s s' p d
and pmq: p\neq q
and inv: HInv4b s q
shows HInv4b s' q
proof 
from act pmq
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:

\[
\begin{align*}
HStartBallot \ s \ s' \ p; \ HInv4c \ s \ p
\rightarrow \ HInv4c \ s' \ p
\end{align*}
\]

by (auto simp add: StartBallot-def HInv4c-def)

Lemma HStartBallot-HInv4c-q:

assumes act: HStartBallot \ s \ s' \ p
and inv: HInv4c \ s \ q
and pnq: p \neq q

shows HInv4c \ s' \ q

proof -

from act pnq
have phase: 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 \ ?thesis
by (auto simp add: HInv4c-def)

qed

Theorem HStartBallot-HInv4c:

\[
\begin{align*}
HStartBallot \ s \ s' \ p; \ HInv4c \ s \ q
\rightarrow \ HInv4c \ s' \ q
\end{align*}
\]

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 \ ?thesis

proof (auto simp add: HInv4c-def phase'-MajoritySet-def)

from act phase'
have bal: bal (dblock \ s' \ p) = 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) = bal (dblock \ s \ p)

by (auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)

with bal
have bal (dblock \ s' \ p) = mbal (disk \ s' \ d \ p)

by auto

with inv phase act
show \ \exists \ D. \ \ IsMajority \ D
\land (\forall d \in D. \ mbal \ (disk \ s' \ d \ p) = bal \ (dblock \ s' \ p))

by (auto simp add: HInv4c-def Phase1or2Write-def MajoritySet-def)

qed
next
  case False
  with act
  show ?thesis
    by (auto simp add: HInv4c-def Phase1or2Write-def)
qed

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

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

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

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

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

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

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

\textbf{theorem} HEndPhase1-HInv4c:
\[
[ \ HEndPhase1 \ s \ s' \ p; \ HInv4c \ s \ q; \ Inv2b \ s ] \implies HInv4c \ s' \ q
\]
\textbf{by}(\textbf{blast dest:} HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q)
lemma \( H\text{EndPhase}2-\text{HInv}4c-p: \)
\[
[ H\text{EndPhase}2 \ s \ s' \ p; \ H\text{Inv}4c \ s \ p ] \implies \ H\text{Inv}4c \ s' \ p
\]
by (auto simp add: \text{EndPhase}2-def H\text{Inv}4c-def)

lemma \( H\text{EndPhase}2-\text{HInv}4c-q: \)
assumes act: \( H\text{EndPhase}2 \ s \ s' \ p \)
and inv: \( H\text{Inv}4c \ s \ q \)
and pnq: \( p \neq q \)
shows \( H\text{Inv}4c \ s' \ q \)
proof -
from act pnq
have phase: phase \( s' \ q = \text{phase} \ s \ q \)
and dblock: dblock \( s \ q = \text{dblock} \ s' \ q \)
and disk: disk \( s' = \text{disk} \ s \)
by (auto simp add: \text{EndPhase}2-def)
with inv
show \( ?\text{thesis} \)
by (auto simp add: H\text{Inv}4c-def)
qed

theorem \( H\text{EndPhase}2-\text{HInv}4c: \)
\[
[ H\text{EndPhase}2 \ s \ s' \ p; \ H\text{Inv}4c \ s \ q ] \implies \ H\text{Inv}4c \ s' \ q
\]
by (blast dest: H\text{EndPhase}2-\text{HInv}4c-p H\text{EndPhase}2-\text{HInv}4c-q)

lemma \( H\text{Fail}-\text{HInv}4c-p: \)
\[
[ H\text{Fail} \ s \ s' \ p; \ H\text{Inv}4c \ s \ p ] \implies \ H\text{Inv}4c \ s' \ p
\]
by (auto simp add: \text{Fail-def} H\text{Inv}4c-def)

lemma \( H\text{Fail}-\text{HInv}4c-q: \)
assumes act: \( H\text{Fail} \ s \ s' \ p \)
and inv: \( H\text{Inv}4c \ s \ q \)
and pnq: \( p \neq q \)
shows \( H\text{Inv}4c \ s' \ q \)
proof -
from act pnq
have phase: phase \( s' \ q = \text{phase} \ s \ q \)
and dblock: dblock \( s \ q = \text{dblock} \ s' \ q \)
and disk: disk \( s' = \text{disk} \ s \)
by (auto simp add: \text{Fail-def})
with inv
show \( ?\text{thesis} \)
by (auto simp add: H\text{Inv}4c-def)
qed

theorem \( H\text{Fail}-\text{HInv}4c: \)
\[
[ H\text{Fail} \ s \ s' \ p; \ H\text{Inv}4c \ s \ q ] \implies \ H\text{Inv}4c \ s' \ q
\]
by (blast dest: H\text{Fail}-\text{HInv}4c-p H\text{Fail}-\text{HInv}4c-q)

lemma \( H\text{Phase}0\text{Read}-\text{HInv}4c-p: \)
lemma \textit{HPhase0Read-HInv4c-q}:
assumes \textit{act}: \textit{HPhase0Read \ s s' p d}
and \textit{inv}: \textit{HInv4c \ s q}
and \textit{pnq}: \textit{p \neq q}
shows \textit{HInv4c \ s' q}
proof  
from \textit{act pnq} 
have \textit{phase}: \textit{phase \ s' q = phase \ s q}
and \textit{dblock}: \textit{dblock \ s \ q = dblock \ s' q}
and \textit{disk}: \textit{disk \ s' = disk \ s}
by\textit{(auto simp add: Phase0Read-def)}
with \textit{inv} 
show \textit{?thesis}
by\textit{(auto simp add: HInv4c-def)}
qed

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

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

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

theorem \textit{HEndPhase0-HInv4c}:
\[ \textit{HEndPhase0 \ s s' p; HInv4c \ s q} \implies \textit{HInv4c \ s' q} \]
by\textit{(blast dest: HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q)}

80
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] 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
with act
show ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s' d p)
  by (auto simp add: StartBallot-def)
qed

lemma HStartBallot-HInv4d-q:
  assumes act: HStartBallot s s' p
  and inv: HInv4d s q
  and p≠q: p ≠ q
  shows HInv4d s' q
proof
  from act p≠q
  have disk': disk s' = disk s
    by (auto simp add: StartBallot-def)
  from act p≠q
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by (auto simp add: 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)

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

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

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

lemma HPhase1or2Write-HInv4d-q: 
assumes act: HPhase1or2Write s s' p d 
and inv: HInv4d s q 
and pnq: p\neq q 
shows HInv4d s' q 
proof
from act pnq have disk': \forall d. disk s' d q = disk s d q
  by(auto simp add: Phase1or2Write-def)
from act pnq have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
  by(auto simp add: Phase1or2Write-def)
  InitializePhase-def allRdBlks-def)
with act pnq have blocksOf s' q \subseteq 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:
\[ \begin{array}{l}
\text{HPhase1or2Write } s \ s' \ p \ d ; \ HInv4d \ s \ q ; \ HInv4a \ s \ p
\implies \ HInv4d \ s' \ q
\end{array} \]
by(blast dest: HPhase1or2Write-HInv4d-p HPhase1or2Write-HInv4d-q)

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

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

83
by (auto simp add: HInv4d-def)

with disk'
show ?thesis
by (auto simp add: HInv4d-def)

qed

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

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

lemma HEndPhase1-HInv4d-p:
assumes act: HEndPhase1 \ s \ s' \ p
and inv: HInv4d \ s \ p
and inv2b: Inv2b \ s
and inv4c: HInv4c \ s \ p
shows HInv4d \ s' \ p
proof (clarsimp simp add: HInv4d-def)
  fix bk
  assume bk: bk \in blocksOf \ s' \ p
  from HEndPhase1-HInv4c[OF act inv4c inv2b]
  have HInv4c \ s' \ p .
  with act
  have p31: \ \exists D \in MajoritySet.
              \ \forall d \in D. \ mbal (disk s' \ d \ p) = bal (dblock s' \ p)
        and disk': disk s' = disk s
        by (auto simp add: EndPhase1-def HInv4c-def)
  from subset[OF HEndPhase1-blocksOf[OF act] bk]
  show \ \exists D \in MajoritySet. \ \forall d \in D. \ bal \ bk \leq \ mbal (disk s' \ d \ p)
proof
  assume bk: bk \in blocksOf \ s \ p
  with inv disk'
  show ?thesis
  by (auto simp add: HInv4d-def)
next
  assume bk: bk \in \{dblock s' \ p\}
  with p31
  show ?thesis
  by force
qed

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

**proof**

from act \( \text{pq} \)

have \( \text{disk} : \text{disk} s' = \text{disk} s \)
and \( \text{dblock} : \text{dblock} s' = \text{dblock} s \ q \)
by (auto simp add: EndPhase1-def)

from act \( \text{pq} \)

have \( \text{blocksRead} : \forall \ q. \ \text{allRdBlks} s' \ q \subseteq \text{allRdBlks} s \ q \)
by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)

with \( \text{disk} \ q' \)

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

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

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 \( \text{disk} q' \)

show \( \exists \ D \in \text{MajoritySet}. \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} s' d q) \)
by (auto simp add: EndPhase2-def)

qed

**theorem** \( \text{HEndPhase1-HInv4d} : \)

\[
[ \begin{array}{l}
\text{HEndPhase1} \ s \ s' \ p ; \\
HInv4d \ s \ q ; \\
\text{Inv2b} \ s ; \\
HInv4c \ s \ p
\end{array} ] \implies HInv4d \ s' \ q \]

by (blast dest: HEndPhase1-HInv4d-p HEndPhase1-HInv4d-q)

**lemma** \( \text{HEndPhase2-HInv4d-p} : \)

assumes \( \text{act} : \text{HEndPhase2} \ s \ s' \ p \)
and \( \text{inv} : \text{HInv4d} \ s \ p \)
shows \( \text{HInv4d} \ s' \ p \)

**proof** (clarsimp simp add: HInv4d-def)

fix \( bk \)

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

from \( \text{act} \)

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

from \( \text{subsetD}[\text{OF HEndPhase2-blocksOf}[\text{OF \ act}] \ bk] \)

have \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} s d p) \)
by (auto simp add: HInv4d-def)

with \( \text{act} \)

show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal} \ bk \leq \text{mbal} (\text{disk} s' d p) \)
by (auto simp add: EndPhase2-def)

qed

**lemma** \( \text{HEndPhase2-HInv4d-q} : \)

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

proof –
from \( \text{act pq} \)
have \( \text{disk'}: \text{disk } s' = \text{disk } s \)
  by(auto simp add: EndPhase2-def)
from \( \text{act pq} \)
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] \( \text{inv} \)
have \( \forall \ bk \in \text{blocksOf } s' \ q. \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal } \text{disk } s \ d \ q \)
  by(auto simp add: HInv4d-def)
with \( \text{disk'} \)
show \( \ ?\text{thesis} \)
by(auto simp add: HInv4d-def)
qed

theorem \( HEndPhase2-HInv4d: \)
\[ [ HEndPhase2 \ s \ s' \ p; HInv4d \ s \ q ] \Rightarrow HInv4d \ s' \ q \]
by(blast dest: HEndPhase2-HInv4d-p HEndPhase2-HInv4d-q)

lemma \( HFail-HInv4d-p: \)
assumes \( \text{act}: HFail \ s \ s' \ p \)
and \( \text{inv}: HInv4d \ s \ p \)
shows \( HInv4d \ s' \ p \)
proof(clarsimp simp add: HInv4d-def)
fix \( \ bk \)
assume \( \ bk: \ bk \in \text{blocksOf } s' \ p \)
from \( \text{act} \)
have \( \text{disk'}: \text{disk } s' = \text{disk } s \)
  by(auto simp add: Fail-def)
from subsetD[OF HFail-blocksOf[OF \( \text{act} \)] \( \text{bk} \)]
show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal } (\text{disk } s' \ d \ p) \)
proof
assume \( \ bk: \ bk \in \{ \text{dblock } s' \ p \} \)
with \( \text{inv disk'} \)
show \( \ ?\text{thesis} \)
  by(auto simp add: HInv4d-def)
next
assume \( \ bk: \ bk \in \{ \text{dblock } s' \ p \} \)
with \( \text{act} \)
have \( \text{bal } bk = 0 \)
  by(auto simp add: Fail-def InitDB-def
with \( \text{Disk-isMajority} \)
show \( \ ?\text{thesis} \)
  by(auto simp add: MajoritySet-def)
lemma HFail-HInv4d-q:
assumes act: HFail s s' p
and inv: HInv4d s q
and pq: p≠q
shows HInv4d s' q

proof –
from act pq
have disk': disk s'=disk s
and dblock': dblock s' q=dblock s q
by(auto simp add: Fail-def)
from act pq
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 p≠q
shows HInv4d s' q
proof –
  from act p≠q
  have disk: disk s'=disk s
    by(auto simp add: Phase0Read-def)
  from act p≠q
  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 ∀ bk∈blocksOf s' q.
    ∃ D∈MajoritySet. ∀ d∈D. bal bk ≤ mbal(disk s d q)
    by(auto simp add: HInv4d-def)
  thus ?thesis
  by(auto simp add: HInv4d-def)
qed

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

lemma HEndPhase0-blocksOf2:
assumes act: HEndPhase0 s s' p
and inv2c: Inv2c-inner s p
shows allBlocksRead s p ⊆ blocksOf s p
proof –
  from act inv2c
  have ∀ d.∀ br ∈ blocksRead s p d. proc br =p
    ∧ block br = disk s d p
    by(auto simp add: EndPhase0-def Inv2c-inner-def)
  thus ?thesis
  by(auto simp add: allBlocksRead-def allRdBlks-def
    blocksOf-def)
qed

lemma HEndPhase0-HInv4d-p:
assumes act: HEndPhase0 s s' p
and inv: HInv4d s p
and inv2c: Inv2c s
and inv1: Inv1 s
shows HInv4d s' p
proof(clarsimp simp add: HInv4d-def)
  fix bk
assume \( bk : bk \in \text{blocksOf} \ s \ s' \ p \)

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

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

proof

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

with inv

show \( ?\text{thesis} \)

by (auto simp add: \text{HInv4d-def})

next

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

from \text{inv2c}

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

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

from \( bk \ \text{HEndPhase0-some} \{ \text{OF} \ \text{act} \ \text{inv1} \} \)

\( \text{HEndPhase0-blocksOf2} \{ \text{OF} \ \text{act} \ \text{inv2c-inner} \} \ \text{act} \)

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

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

with inv

show \( ?\text{thesis} \)

by (auto simp add: \text{HInv4d-def})

qed

with \text{act}

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

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

qed

lemma \( \text{HEndPhase0-HInv4d-q} \):

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

and \text{inv}: \( \text{HInv4d} s \ q \)

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

shows \( \text{HInv4d} s' \ q \)

proof –

from \text{act pnq}

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

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

moreover

from \text{act pnq}

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

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

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

by (auto, blast)

ultimately

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

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

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

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

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

by (auto simp add: \text{HInv4d-def})

with \text{act}
show thesis 
by(auto simp add: EndPhase0-def HInv4d-def)
qed

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

Since we have already proved HInv2 is an invariant of HNext, HInv1 ∧ HInv2 ∧ HInv4 is also an invariant of HNext.

lemma I2d:
  assumes nxt: HNext s s' 
  and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv4 s 
  shows HInv4 s' 
proof(auto simp add: HInv4-def)
  fix p 
  show HInv4a s' p using assms 
    by(auto simp add: HInv4-def HNext-def Next-def, 
      auto simp add: HInv2-def intro: HStartBallot-HInv4a, 
      auto intro: HPhase0Read-HInv4a, 
      auto intro: HPhase1or2Write-HInv4a, 
      auto simp add: Phase1or2Read-def 
        intro: HPhase1or2ReadThen-HInv4a 
        HPhase1or2ReadElse-HInv4a, 
      auto simp add: EndPhase1or2-def 
        intro: HEndPhase1-HInv4a 
        HEndPhase2-HInv4a, 
      auto intro: HFail-HInv4a, 
      auto intro: HEndPhase0-HInv4a simp add: HInv1-def) 
  show HInv4b s' p using assms 
    by(auto simp add: HInv4-def HNext-def Next-def, 
      auto simp add: HInv2-def 
        intro: HStartBallot-HInv4b, 
      auto intro: HPhase0Read-HInv4b, 
      auto intro: HPhase1or2Write-HInv4b, 
      auto simp add: Phase1or2Read-def 
        intro: HPhase1or2ReadThen-HInv4b 
        HPhase1or2ReadElse-HInv4b, 
      auto simp add: EndPhase1or2-def 
        intro: HEndPhase1-HInv4b 
        HEndPhase2-HInv4b, 
      auto intro: HFail-HInv4b, 
      auto intro: HEndPhase0-HInv4b simp add: HInv1-def Inv2c-def) 
  show HInv4c s' p using assms 
    by(auto simp add: HInv4-def HNext-def Next-def, 
      auto simp add: HInv2-def 
        intro: HStartBallot-HInv4c, 
      auto intro: HPhase0Read-HInv4c,
auto intro: HPhase1or2Write-HInv4c,
auto simp add: Phase1or2Read-def
  intro: HPhase1or2ReadThen-HInv4c
  HPhase1or2ReadElse-HInv4c,
auto simp add: EndPhase1or2-def
  intro: HEndPhase1-HInv4c
  HEndPhase2-HInv4c,
auto intro: HFail-HInv4c,
auto intro: HEndPhase0-HInv4c simp add: HInv1-def)
show HInv4d s' p using assms
  by (auto simp add: HInv4-def HNext-def Next-def,
    auto simp add: HInv2-def
    intro: HStartBallot-HInv4d,
    auto intro: HPhase0Read-HInv4d,
    auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-HInv4d
    HPhase1or2ReadElse-HInv4d,
    auto simp add: EndPhase1or2-def
    intro: HEndPhase1-HInv4d
    HEndPhase2-HInv4d,
    auto intro: HFail-HInv4d,
    auto intro: HEndPhase0-HInv4d simp add: HInv1-def)
qed
end

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

C.5 Invariant 5

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

definition maxBalInp :: state ⇒ nat ⇒ InputsOrNi ⇒ bool
  where maxBalInp s b v = (\( \forall bk \in \text{allBlocks } s \). b ≤ bal bk ↦ inp bk = v)

definition Hinv5-inner-R :: state ⇒ Proc ⇒ bool
  where
  Hinv5-inner-R s p = 
    (maxBalInp s (bal(dblock s p)) (inp(dblock s p))
    ∨ (∃ D∈ MajoritySet. ∃ q. (∀ d∈D. bal(dblock s p) < mbal(disk s d q)
    ∧ ¬ hasRead s p d q)))

definition Hinv5-inner :: state ⇒ Proc ⇒ bool
where $\text{HInv5-inner } s \ p = (\text{phase } s \ p = 2 \implies \text{HInv5-inner-R } s \ p)$

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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

**Theorem** $\text{HInit-HInv5}: \text{HInit } s \implies \text{HInv5 } s$

**Using** Disk-isMajority

**By** \((\text{auto simp add: HInit-def Init-def HInv5-def HInv5-inner-def})\)

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 $\text{HInv2}$. Now we prove that for each action the new **allBlocks** are included in the old **allBlocks** or, in some cases, included in the old **allBlocks** union the new **dblock**.

**Lemma** $\text{HStartBallot-HInv5-p}$:

**Assumes** act: $\text{HStartBallot } s \ s' \ p$

**And** inv: $\text{HInv5-inner } s \ p$

**Shows** $\text{HInv5-inner } s' \ p$ **using** assms

**By** \((\text{auto simp add: StartBallot-def HInv5-inner-def})\)

**Lemma** $\text{HStartBallot-blocksOf-q}$:

**Assumes** act: $\text{HStartBallot } s \ s' \ p$

**And** $p \neq q$

**Shows** $\text{blocksOf } s' \ q \subseteq \text{blocksOf } s \ q$ **using** assms

**By** \((\text{auto simp add: StartBallot-def InitializePhase-def blocksOf-def rdBy-def})\)

**Lemma** $\text{HStartBallot-allBlocks}$:

**Assumes** act: $\text{HStartBallot } s \ s' \ p$

**Shows** $\text{allBlocks } s' \subseteq \text{allBlocks } s \cup \{ \text{dblock } s' \ p\}$

**Proof** \((\text{auto simp del: HStartBallot-def simp add: allBlocks-def dest: HStartBallot-blocksOf-q[OF act])}\)

**Fix** $x \ pa$

**Assume** $x-pa$: $x \in \text{blocksOf } s' \ pa$ **and**

$x-nblks$: $\forall \ xa. \ x \notin \text{blocksOf } s \ xa$

**Show** $x=\text{dblock } s' \ p$

**Proof** \((\text{cases p=pa})\)

**Case** True

**From** $x-nblks$

**Have** $x \notin \text{blocksOf } s \ p$

**By** auto

**With** True subsetD|OF $\text{HStartBallot-blocksOf}|OF$ act| $x-pa$
show \(?thesis\)
   by auto
next
case False
  from x-nblks subsetD[OF HStartBallot-blocksOf-q[OF act False] x-pa]
  show \(?thesis\)
     by auto
qed
qed

lemma HStartBallot-HInv5-q1:
  assumes act: HStartBallot s s' p
  and pnq: p\#q
  and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
  shows maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
proof(auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk \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: StartBallot-def)
  from subsetD[OF HStartBallot-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\}
  have dblock s p \in 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: \exists D \in MajoritySet. \exists qq. \(\forall d \in D. \) bal(dblock s q) < mbal(disk s d qq)
     \land \neg hasRead s q d qq)
  shows \exists D \in MajoritySet. \exists qq. \(\forall d \in D. \) bal(dblock s' q) < mbal(disk s' d qq)
     \land \neg hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
and \( \text{blocksRead} \): \( \forall d. \text{blocksRead} s' q d = \text{blocksRead} s q d \)
and \( \text{dblock} \): \( \text{dblock} s' q = \text{dblock} s q \)
by (auto simp add: \text{StartBallot-def} \text{InitializePhase-def})

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

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

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

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

lemma \( \text{HPhase1or2Write-HInv5-p2} \):
assumes \( \text{act:} \) \( \text{HPhase1or2Write} s s' p d \)
and \( \text{inv4c:} \) \( \text{HInv4c} s p \)
and \( \text{phase:} \) \( \text{phase} s p = 2 \)
and \( \text{inv5-2:} \) \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock} s p) < \text{mbal}(\text{disk} s d q)) \cap \neg \text{hasRead} s p d q \)
shows \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock} s' p) < \text{mbal}(\text{disk} s' d q)) \cap \neg \text{hasRead} s' p d q \)
proof –
from \( \text{inv5-2} \)
obtain \( D q \)
where \( i1: \) \( \text{IsMajority} D \)
and \( i2: \) \( \forall d \in D. \text{bal}(\text{dblock} s p) < \text{mbal}(\text{disk} s d q) \)
and \( i3: \) \( \forall d \in D. \neg \text{hasRead} s p d q \)
by (auto simp add: \text{MajoritySet-def})
have \( \text{pnq:} \) \( p \neq q \)
proof –
from \( \text{inv4c} \) \( \text{phase} \)
obtain \( D1 \) where \( r1: \) \( \text{IsMajority} D1 \cap (\forall d \in D1. \text{mbal}(\text{disk} s d p) = \text{bal} \ (\text{dblock} s p)) \)

94
by(auto simp add: HInv4c-def MajoritySet-def)
with i1 majorities-intersect
have $D \cap D_1 \neq \{\}$ by auto
then obtain $dd$ where $dd \in D \cap D_1$
  by auto
thus ?thesis by auto
qed
from act pnq — dblock and hasRead do not change
have dblock $s' = dblock s$
  and $\forall d. hasRead s' p d q = hasRead s p d q$
  — In all disks $q$ blocks don’t change
  and $\forall d. disk s' d q = disk s d q$
  by(auto simp add: Phase1or2Write-def hasRead-def)
with i2 i1 i3 majority-nonempty
have $\forall d \in D. (dblock s p) < mbal (disk s dd q) \land \neg hasRead s' p d q$
  by(auto)
with i1
show ?thesis
  by(auto simp add: MajoritySet-def)
qed
lemma HPhase1or2Write-HInv5-p:
assumes act: HPhase1or2Write s s' p d
and inv: HInv5-inner s p
and inv4: HInv4c s p
shows HInv5-inner s' p
proof(auto simp add: HInv5-inner-def HInv5-inner-R-def)
  assume phase': phase $s' = 2$
  and i2: $\forall D \in MajoritySet. \forall q. \exists d \in D. (dblock s' p) < mbal (disk s' d q) \longrightarrow hasRead s' p d q$
with act have phase: phase $s p = 2$
  by(auto simp add: Phase1or2Write-def)
show maxBalInp s' (bal (dblock s' p)) (inp (dblock s' p))
proof(rule HPhase1or2Write-HInv5-1[OF act, of p])
  from HPhase1or2Write-HInv5-p2[OF act inv4 phase] inv i2 phase
  show maxBalInp s (bal (dblock s p)) (inp (dblock s p))
    by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed
lemma HPhase1or2Write-allBlocks:
assumes act: HPhase1or2Write s s' p d
shows allBlocks s' $\subseteq$ allBlocks s
using HPhase1or2Write-blocksOf[OF act]
by(auto simp add: allBlocks-def)

lemma $H\text{Phase1or2Write-HInv5-q2}$:

assumes act: $H\text{Phase1or2Write s s'} p d$

and $p\neq q$

and inv4a: $H\text{Inv4a s p}$

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

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

proof

from inv5-2

obtain $D q q'$

where i1: $\text{IsMajority D}$

and i2: $\forall d \in D. \text{bal}(\text{dblock s q}) < \text{mbal}(\text{disk s d q})$

and i3: $\forall d \in D. \neg \text{hasRead s q d q'}$

by (auto simp add: MajoritySet-def)

from act $p \neq q$

— dblock and hasRead do not change

have dblock: $\text{dblock s'} = \text{dblock s}$

and hasread: $\forall d. \text{hasRead s' q d q} = \text{hasRead s q d q}$

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

have $\forall d \in D. \text{bal}(\text{dblock s' q'}) < \text{mbal}(\text{disk s' d q'}) \land \neg \text{hasRead s' q d q}$

proof (cases $q = p$)

case True

have $\text{bal}(\text{dblock s q}) < \text{mbal}(\text{dblock s p})$

proof —

from inv4a act i1

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

by (auto simp add: MajoritySet-def $H\text{Inv4a-def}$ $H\text{Inv4a2-def}$ $H\text{Phase1or2Write-def}$)

with True i2

show $\text{bal}(\text{dblock s q}) < \text{mbal}(\text{dblock s p})$

by auto

qed

with hasread dblock' True i1 i2 i3 act

show $?\text{thesis}$

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

next

case False

with act i2 i3

show $?\text{thesis}$

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

qed

with i1

show $?\text{thesis}$

by (auto simp add: MajoritySet-def)

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

theorem HPhase1or2Write-HInv5:
[ HPhase1or2Write s s′ p d; HInv5-inner s q; HInv4c s p; HInv4a s p ] → HInv5-inner s′ q
by (auto simp add: MajoritySet-def)

lemma HPhase1or2ReadThen-HInv5-1:
assumes act: HPhase1or2ReadThen s s′ p d r
and inv5-1: maxBalInp s (bal (dblock s q)) (inp (dblock s q))
shows maxBalInp s′ (bal (dblock s′ q)) (inp (dblock s′ q))
using assms and HPhase1or2ReadThen-blocksOf
by (auto simp add: Phase1or2ReadThen-def maxBalInp-def allBlocks-def)

lemma HPhase1or2ReadThen-HInv5-p2:
assumes act: HPhase1or2ReadThen s s′ p d r
and inv4c: HInv4c s p
and inv2c: Inv2c-inner s p
and phase: phase s p = 2
and inv5-2: ∃ D∈ MajoritySet. ∃ q. (∀ d∈ D. bal (dblock s p) < mbal (disk s d q)
  ∧ ¬ hasRead s p d q)
shows ∃ D∈ MajoritySet. ∃ q. (∀ d∈ D. bal (dblock s′ p) < mbal (disk s′ d q)
  ∧ ¬ hasRead s′ p d q)
proof
  from inv5-2
  obtain D q
    where i1: IsMajority D
    and i2: ∀ d∈ D. bal (dblock s p) < mbal (disk s d q)
    and i3: ∀ d∈ D. ¬ hasRead s p d q
    by (auto simp add: MajoritySet-def)
  from inv2c phase
  qed
have \( \text{bal}(\text{dblock } s p) = \text{mbal}(\text{dblock } s p) \)
by \((\text{auto simp add: Inv2c-inner-def})\)

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

moreover
from \( \text{i2} \) have \( d \in D \to \text{bal}(\text{dblock } s p) < \text{mbal}(\text{disk } s d q) \) by \text{auto}
ultimately have \(\text{pnr}: d \in D \to q \neq r \) by \text{auto}

have \( \text{pnr}: p \neq q \)
proof
from \text{act} have \( \text{bal}(\text{dblock } s p) < \text{mbal}(\text{disk } s d q) \) by \text{auto}

proof
from \text{inv4c phase}
obtain \( D1 \) where
\( r1 : \text{IsMajority } D1 \land (\forall d \in D1. \text{mbal}(\text{disk } s d p) = \text{bal}(\text{dblock } s p)) \)
by \((\text{auto simp add: HInv4c-def MajoritySet-def})\)
with \( i1 \) \text{majorities-intersect}
have \( D \cap D1 \neq \{\} \) by \text{auto}
then obtain \( dd \) where
\( dd \in D \cap D1 \)
by \text{auto}
with \( i1 \ i2 \ r1 \)
have \( \text{bal}(\text{dblock } s p) < \text{mbal}(\text{disk } s dd q) \land \text{mbal}(\text{disk } s dd p) = \text{bal}(\text{dblock } s p) \)
by \text{auto}
thus \( ?\text{thesis} \) by \text{auto}
qed

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

from \text{act} \text{pnr}
— \text{dblock and disk do not change}
have \( \text{dblock } s' = \text{dblock } s \)
and \( \forall d. \text{disk } s' = \text{disk } s \)
by \((\text{auto simp add: Phase1or2ReadThen-def})\)
with \( \text{i2} \) \text{hasRead'} \text{i3}
have \( \forall d \in D. \text{bal}(\text{dblock } s' p) < \text{mbal}(\text{disk } s' d q) \land \neg \text{hasRead'} s' p d q \)
by \text{auto}
with \( i1 \)
show \( ?\text{thesis} \)
by \((\text{auto simp add: MajoritySet-def})\)
qed

lemma \( H\text{Phase1or2ReadThen-HInv5-p} : \)
assumes \( \text{act}: H\text{Phase1or2ReadThen} s s' p d r \)
and \( \text{inv}: H\text{Inv5-inner } s p \)
and \( \text{inv4}: H\text{Inv4c } s p \)
and \( \text{inv2c}: H\text{Inv2c } s \)
shows \( H\text{Inv5-inner } s' p \)
proof\((\text{auto simp add: HInv5-inner-def HInv5-inner-R-def})\)
assume \( \text{phase'}: \text{phase } s' p = 2 \)
and \( \text{i2}: \forall D \in \text{MajoritySet}. \forall q. \exists d \in D. \text{bal}(\text{dblock } s' p) < \text{mbal}(\text{disk } s' d q) \to \text{hasRead } s' p d q \)

98
with \( act \) have phase: phase \( s \) \( p \) = 2 
by (auto simp add: Phase1or2ReadThen-def)

show \( \text{maxBalInp} \ s' \ (\text{bal} \ (\text{dblock} \ s' \ p)) \ (\text{inp} \ (\text{dblock} \ s' \ p)) \)
proof (rule HPhase1or2ReadThen-HInv5-i [OF \( \text{act} \) \( \text{of} \) \( \text{p} \)])
from inv2c
have Inv2c-inner \( s \) \( p \) by (auto simp add: Inv2c-def)
from HPhase1or2ReadThen-HInv5-p2 [OF \( \text{act} \) inv4 phase]
show \( \text{maxBalInp} \ s \ (\text{bal} \ (\text{dblock} \ s \ p)) \ (\text{inp} \ (\text{dblock} \ s \ p)) \)
by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

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

lemma HPhase1or2ReadThen-HInv5-q2:
assumes \( \text{act} \): HPhase1or2ReadThen \( s \) \( s' \) \( p \) \( d \) \( r \)
and pnq: \( p \neq q \)
and inv4a: HInv4a \( s \) \( p \)
and inv5-2: \( \exists \text{MajoritySet} \. \exists qq. \ (\forall d \in \text{MajoritySet} \. \text{bal}(\text{dblock} \ s \ q) < \text{mbal}(\text{disk} \ s \ d \ qq) \)
\( \wedge \neg \text{hasRead} \ s \ q \ d \ qq \)
shows \( \exists \text{MajoritySet} \. \exists qq. \ (\forall d \in \text{MajoritySet} \. \text{bal}(\text{dblock} \ s' \ q) < \text{mbal}(\text{disk} \ s' \ d \ qq) \)
\( \wedge \neg \text{hasRead} \ s' \ q \ d \ qq \)
proof (from inv5-2
obtain \( D \) \( qq \)
where i1: IsMajority \( D \)
and i2: \( \forall d \in \text{MajoritySet} \. \text{bal}(\text{dblock} \ s \ q) < \text{mbal}(\text{disk} \ s \ d \ qq) \)
and i3: \( \forall d \in \text{MajoritySet} \. \neg \text{hasRead} \ s \ q \ d \ qq \)
by (auto simp add: MajoritySet-def)
from \( \text{act} \) \( \text{pnq} \)
— dblock and hasRead do not change
have dblock': dblock \( s' = \) dblock \( s \)
and disk': disk \( s' = \) disk \( s \)
and hasread: \( \forall d \in \text{MajoritySet} \. \text{hasRead} \ s' \ q \ d \ qq = \text{hasRead} \ s \ q \ d \ qq \)
by (auto simp add: Phase1or2ReadThen-def hasRead-def)
with i2 i3
have \( \forall d \in \text{MajoritySet} \. \text{bal}(\text{dblock} \ s' \ q) < \text{mbal}(\text{disk} \ s' \ d \ qq) \wedge \neg \text{hasRead} \ s' \ q \ d \ qq \)
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)
  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-q 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 HEndPhase2-2-Hinv5-p:
  HEndPhase2 s s' p ⇒ HInv5-inner s' q
by(auto simp add: EndPhase2-def HInv5-inner-def)

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

lemma HEndPhase2-HInv5-q1:
  assumes act: HEndPhase2 s s' p
  and pnq: p ≠ q
  and inv5-1: maxBalInp s (bal (dblock s q)) (inp (dblock s q))
  shows maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
proof(auto simp add: maxBalInp-def)
  fix bk
  assume bk: bk ∈ allBlocks s'
  and bal: bal (dblock s' q) ≤ bal bk
from act pnq
have 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\neq q
  and inv5-2: \exists D\in MajoritySet. \exists qq. (\forall d\in D. \ bal(dblock s q) < mbal(disk s d qq)
                                  \land \neg hasRead s q d qq)
  shows \exists D\in MajoritySet. \exists qq. (\forall d\in D. \ bal(dblock s' q) < mbal(disk s' d qq)
                                \land \neg hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
    and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
    and dblock: dblock s' q = dblock s q
    by (auto simp add: EndPhase2-def InitializePhase-def)
  with inv5-2
  show \?thesis
    by (auto simp add: hasRead-def)
qed

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

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

lemma HEndPhase1-HInv5-p:
  assumes act: HEndPhase1 s s' p
  and inv4: Hinv4 s
  and inv2a: Inv2a s
  and inv2a': Inv2a' s'
  and inv2c: Inv2c s
  and asm4: \neg maxBalInp s' (bal(dblock s' p)) (inp(dblock s' p))
  shows (\exists D\in MajoritySet. \exists qq. (\forall d\in D. \ bal(dblock s' p) < mbal(disk s' d qq)
                                    \land \neg hasRead s' p d qq))
proof

have \( \exists bk \in \text{allBlocks } s \). 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 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 \( \text{dblock}: p \neq q \implies \text{dblock } s' \, q = \text{dblock } s \, q \)

by(auto simp add: EndPhase1-def)

have \( bk \in \text{blocksOf } s \, q \)

proof(cases \( p=q \))

case True

with p32 p31 HEndPhase1-blocksOf[OF act]

show \(?thesis \)

by auto

next

case False

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

show \(?thesis \)

by(auto simp add: blocksOf-def)

qed

with p31

show \(?thesis \)

by(auto simp add: allBlocks-def)

qed

then obtain bk where p22: \( bk \in \text{allBlocks } s \land bal(\text{dblock } s' \, p) \leq \text{bal } bk \land bk \neq \text{dblock } s' \, p \)

by(auto simp add: maxBalInp-def)

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 act p22

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

by(auto simp add: EndPhase1-def)

moreover

from act

have \( \text{phase } s \, p = 1 \)

by(auto simp add: EndPhase1-def)

moreover

from inv4

have \( H\text{inv}_4 \, b \, s \, p \)

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 ∈ UNIV − {p} ∧ bk ∈ blocksOf s q
    by auto
have ∃ D ∈ MajoritySet. ∀ d ∈ D. bal (dblock s' p) ≤ mbal (disk s d q)
    proof
      from p23 inv4
      have i4d: ∃ D ∈ MajoritySet. ∀ d ∈ D. bal bk ≤ mbal (disk s d q)
        by (auto simp add: HInv4-def HInv4d-def)
      from i4d p22
      show ?thesis
        by force
    qed

then obtain D where Dmaj: D ∈ MajoritySet and p24: (∀ d ∈ D. bal (dblock s' p) ≤ mbal (disk s d q))
    by auto
have p25: ∀ d ∈ D. bal (dblock s' p) < mbal (disk s d q)
    proof
      from inv2c
      have Inv2c-inner s p
        by (auto simp add: Inv2c-def)
      with act
      have bal-pos: 0 < bal (dblock s' p)
        by (auto simp add: Inv2c-inner-def EndPhase1-def)
      with inv2a'
      have bal (dblock s' p) ∈ Ballot p ∪ {0}
        by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
      with bal-pos have bal-in-p: bal (dblock s' p) ∈ Ballot p
        by auto
      from inv2a have Inv2a-inner s q
        by (auto simp add: Inv2a-def)
      hence ∀ d ∈ D. mbal (disk s d q) ∈ Ballot q ∪ {0}
        by (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
      with p24 bal-pos
      have ∀ d ∈ D. mbal (disk s d q) ∈ Ballot q
        by force
      with Ballot-disj p23 bal-in-p
      have ∀ d ∈ D. mbal (disk s d q) ≠ bal (dblock s' p)
        by force
      with p23 p24
      show ?thesis
        by force
    qed

with p23 act
have ∀ d ∈ D. bal (dblock s' p) < mbal (disk s d q) ∧ ¬ hasRead s' p d q
    by (auto simp add: EndPhase1-def InitializePhase-def hasRead-def)
with Dmaj
show ?thesis
by blast
qed

lemma union-inclusion:
\[ A \subseteq A'; B \subseteq B' \] \implies A \cup B \subseteq A' \cup B'
by blast

lemma HEndPhase1-blocksOf-q:
assumes act: HEndPhase1 s s' p
and 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 blk: blocksRead s' q = blocksRead s q
by(auto simp add: EndPhase1-def InitializePhase-def)
from disk
have disk': \{ disk s' d q \mid d . d \in UNIV \} \subseteq \{ disk s d q \mid d . d \in UNIV \} (is \ ?D'
\subseteq \ ?D)
by auto
from pnq act
have (UN qq d. rdBy s' q qq d) \subseteq \{ disk s d q \mid d . d \in UNIV \} \subseteq \{ disk s d q \mid d . d \in UNIV \} (is \ ?D' \subseteq \ ?R)
by(auto simp add: EndPhase1-def InitializePhase-def rdBy-def split: if-split-asm, blast)
hence \{ block br \mid br . br \in \{ UN qq d. rdBy s' q qq d \} \} \subseteq \{ block br \mid br . br \in \{ UN qq d. rdBy s q qq d \} \} (is \ ?R' \subseteq \ ?R)
by auto blast
from union-inclusion[OF dblock union-inclusion[OF disk' this]]
show ?thesis
by(auto simp add: blocksOf-def)
qed

lemma HEndPhase1-allBlocks:
assumes act: HEndPhase1 s s' p
shows allBlocks s' \subseteq allBlocks s \cup \{ dblock s' p \}
proof(auto simp del: HEndPhase1-def simp add: allBlocks-def
dest: HEndPhase1-blocksOf-q[OF act!])
fix x pa
assume x-pa: x \in blocksOf s' pa and
x-nblks: \forall xa. x \notin blocksOf s xa
show x=dblock s' p
proof(cases p=pa)
case True
from x-nblks
have x \notin blocksOf s p
by auto
with True subsetD[OF HEndPhase1-blocksOf[OF act] x-pa]
show ?thesis
by auto
next
case False
from x-nblks subsetD[OF HEndPhase1-blocksOf-q[OF False] x-pa]
show ?thesis
by auto
qed
qed

lemma HEndPhase1-HInv5-q:
assumes act: HEndPhase1 s s' p
and inv: HInv5 s
and inv1: Inv1 s
and inv2a: Inv2a s'
and inv2a-q: Inv2a s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HInv3 s
and phase': phase s' q = 2
and pnq: p\neq q
and asm4': \neg \text{maxBalInp}(s' \cdot \text{bal}(\text{dblock} s' q)) \cdot \text{inp}(\text{dblock} s' q)
shows \(\exists D \in \text{MajoritySet. } \exists qq. (\forall d \in D. \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s d qq) \land \neg \text{hasRead} s' q d qq)\)

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: \text{bal}(\text{dblock} s q) \in \text{Ballot} q
  by(auto simp add: Inv2c-def Inv2c-inner-def)
have \(\exists D \in \text{MajoritySet. } \exists qq. (\forall d \in D. \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d qq) \land \neg \text{hasRead} s q d qq)\)
proof(cases maxBalInp \text{p} \cdot (\text{bal}(\text{dblock} s q)) \cdot \text{inp}(\text{dblock} s q))
case True
have p21: \text{bal}(\text{dblock} s q) < \text{bal}(\text{dblock} s' p) \land \text{inp}(\text{dblock} s q) \neq \text{inp}(\text{dblock} s' p)
proof –
from True asm4\text{d} \text{dblock} HEndPhase1-allBlocks[OF act]
have p32: \text{bal}(\text{dblock} s q) \leq \text{bal}(\text{dblock} s' p) \
  \land \text{inp}(\text{dblock} s q) \neq \text{inp}(\text{dblock} s' p)
  by(auto simp add: maxBalInp-def)
from inv2a
have \( \text{bal}(\text{dblock } s' \ p) \in \text{Ballot } p \cup \{ \emptyset \} \)
\[ \text{by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def) \] 

moreover
\[ \text{from Ballot-disj Ballot-nzero pnq} \]
\[ \text{have Ballot } q \cap (\text{Ballot } p \cup \{ \emptyset \}) = \{ \} \]
\[ \text{by auto} \]

ultimately
\[ \text{have } \text{bal}(\text{dblock } s' \ p) \neq \text{bal}(\text{dblock } s \ q) \]
\[ \text{using bal-dblk-q} \]
\[ \text{by auto} \]

with \( p32 \)
\[ \text{show } ? \text{thesis} \]
\[ \text{by auto} \]

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

proof –
\[ \text{from act} \]
\[ \text{have } \exists D \in \text{MajoritySet}. \forall d \in D. d \in \text{disksWritten } s \ p \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s \ p \ d \ q) \]
\[ \text{by (auto simp add: EndPhase1-def MajoritySet-def) \] 

then obtain \( D \)
\[ \text{where act1: } \forall d \in D. d \in \text{disksWritten } s \ p \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead } s \ p \ d \ q) \]

and \( D\text{maj}: D \in \text{MajoritySet} \)
\[ \text{by auto} \]

from \( \text{inv2b} \)
\[ \text{have } \forall d. \text{Inv2b-inner } s \ p \ d \text{ by (auto simp add: Inv2b-def) \] 

with \( \text{act1 pnq phase-p bal} \)
\[ \text{have } \forall d \in D. \text{bal}(\text{dblock } s' \ p) = \text{mbal}(\text{disk } s \ d \ p) \land \text{hasRead } s \ p \ d \ q \]
\[ \text{by (auto simp add: Inv2b-def Inv2b-inner-def) \] 

with \( p21 \ D\text{maj} \)
\[ \text{have } \forall d \in D. \text{bal}(\text{dblock } s \ q) < \text{mbal}(\text{disk } s \ d \ p) \land \text{hasRead } s \ p \ d \ q \]
\[ \text{by auto} \]

with \( \text{Dmaj} \)
\[ \text{show } ? \text{thesis} \]
\[ \text{by auto} \]

qed

then obtain \( D \)
\[ \text{where p22: } D \in \text{MajoritySet} \land (\forall d \in D. \text{bal}(\text{dblock } s \ q) < \text{mbal}(\text{disk } s \ d \ p) \land \text{hasRead } s \ p \ d \ q) \]
\[ \text{by auto} \]

have \( p23: \forall d \in D. (\text{block}=\text{dblock } s \ q, \text{proc}=q) \notin \text{blocksRead } s \ p \ d \)

proof –
\[ \text{have } \text{dblock } s \ q \in \text{allBlocksRead } s \ p \longrightarrow \text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{dblock } s \ q) \]

proof auto
\[ \text{assume dblock-q: } \text{dblock } s \ q \in \text{allBlocksRead } s \ p \]

from \( \text{inv2a-q} \)
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 \(\text{bal-dblk-q Ballot-nzero dblock-q InputsOrNi}\)
have \(\text{dblock-q-nib: dblock s q} \in \text{nonInitBlks } s \ p\)
   by(auto simp add: nonInitBlks-def blocksSeen-def)

with \(\text{act}\)
have \(\text{dblock-max: inp}(\text{dblock } s' \ p) = \text{inp}(\text{maxBlk } s \ p)\)
   by(auto simp add: EndPhase1-def)
from \(\text{maxBlk-in-nonInitBlks}[\text{OF dblock-q-nib inv1}]\)
have \(\text{max-in-nib: maxBlk } s \ p \in \text{nonInitBlks } s \ p\ ..\)

hence \(\text{nonInitBlks } s \ p \subseteq \text{allBlocks } s\)
   by(auto simp add: allBlocks-def nonInitBlks-def blocksSeen-def blocksOf-def rdBy-def allBlocksRead-def allRdBlks-def)
with \(\text{True subsetD[OF this max-in-nib]}\)

have \(\text{bal}(\text{dblock } s \ q) \leq \text{bal}(\text{maxBlk } s \ p) \rightarrow \text{inp}(\text{maxBlk } s \ p) = \text{inp}(\text{dblock } s \ q)\)
   by(auto simp add: maxBalInp-def)
with \(\text{maxBlk-in-nonInitBlks}[\text{OF dblock-q-nib inv1}]\)

show \(\text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{dblock } s \ q)\)
   by auto
qed
with \(\text{p21}\)

have \(\text{dblock } s \ q \notin \text{block ' allRdBlks } s \ p\)
   by(auto simp add: allBlocksRead-def)

hence \(\forall \ d. \ \text{dblock } s \ q \notin \text{block ' blocksRead } s \ p \ d\)
   by(auto simp add: allRdBlks-def)
thus \(\text{?thesis}\)
   by force
qed

have \(\text{p24: } \forall \ d \in D. \neg (\exists b r \in \text{blocksRead } s \ q \ d. \ \text{bal}(\text{dblock } s \ q) \leq \text{mbal}(\text{block } b r))\)
proof
   from \(\text{inv2c phase}\)
   have \(\forall \ d. \ \forall b r \in \text{blocksRead } s \ q \ d. \ \text{mbal}(\text{block } b r) < \text{mbal}(\text{dblock } s \ q)\)
   and \(\text{bal}(\text{dblock } s \ q) = \text{mbal}(\text{dblock } s \ q)\)
   by(auto simp add: Inv2c-def Inv2c-inner-def)

thus \(\text{?thesis}\)
   by force
qed

have \(\text{p25: } \forall \ d \in D. (\neg \text{hasRead } s \ q \ d \ p)\)
proof auto
fix \(\ d\)
assume \(\ d-in-D: \ d \in D\)
and \(\text{hasRead-qdp: hasRead } s \ q \ d \ p\)

have \(\text{p31: } (\text{block= } \text{dblock } s \ p, \text{proc=} p) \in \text{blocksRead } s \ q \ d\)
proof
   from \(\text{d-in-D p22}\)

107
have hasRead-pdq: hasRead s p d q by auto
with hasRead-qdp phase phase-p inv3
have HInv3-R s q p d
  by(auto simp add: HInv3-def HInv3-inner-def HInv3-L-def)
with p23 d-in-D
show ?thesis
  by(auto simp add: HInv3-R-def)
qed

from p21 act
have p32: bal(dblock s q) < mbal(dblock s p)
  by(auto simp add: EndPhase1-def)
with p31 d-in-D hasRead-qdp p24
show False
  by(force)
qed

with p22
show ?thesis
  by auto
next
  case False
  with inv phase
  show ?thesis
    by(auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def)
  qed
then obtain D qq where
  D ∈ MajoritySet ∧ (∀d ∈ D. bal(dblock s q) < mbal(disk s d qq)
  ∧ ¬hasRead s q d qq)
  by auto
moreover
from act pnq
have ∀d. hasRead s' q d qq = hasRead s q d qq
  by(auto simp add: EndPhase1-def InitializePhase-def hasRead-def)
ultimately show ?thesis
  using disk dblock
  by auto
qed

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

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

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

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

109
from act pnq
have \( \text{dblock} s' q = \text{dblock} s q \) \( \text{by} \) (auto simp add: Fail-def)
from subsetD[OF HFail-allBlocks[OF act] bk]
show \( \text{inp} \ bk = \text{inp} \ (\text{dblock} s' q) \)
proof
  assume \( bk: bk \in \text{allBlocks} s \)
  with inv5-1 \( \text{dblock} s' \) bal
  show \( \text{thesis} \)
    \( \text{by} \) (auto simp add: maxBalInp-def)
next
  assume \( bk: bk \in \{ \text{dblock} s' p \} \)
  with act have \( \text{bk-init: bk} = \text{InitDB} \)
  with bal
  have \( \text{bal} (\text{dblock} s' q) = 0 \)
    \( \text{by} \) (auto simp add: InitDB-def)
  with inv2a
  have \( \text{inp} (\text{dblock} s' q) = \text{NotAnInput} \)
    \( \text{by} \) (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with \( \text{bk-init} \)
  show \( \text{thesis} \)
    \( \text{by} \) (auto simp add: InitDB-def)
qed
qed

lemma HFail-HInv5-q2:
  assumes act: HFail s s' p
and pnq: \( p \neq q \)
and inv5-2: \( \exists D \in \text{MajoritySet}. \exists q q. \quad (\forall d \in D. \quad \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} s d q q)) \quad \land \quad \neg \text{hasRead} s q d q q) \)
shows \( \exists D \in \text{MajoritySet}. \exists q q. \quad (\forall d \in D. \quad \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s' d q q)) \quad \land \quad \neg \text{hasRead} s' q d q q) \)
proof
  from act pnq
  have \( \text{disk: disk} s' = \text{disk} s \)
  and blocksRead: \( \forall d. \text{blocksRead} s' q d = \text{blocksRead} s q d \)
  and \( \text{dblock: dblock} s' q = \text{dblock} s q \)
    \( \text{by} \) (auto simp add: Fail-def InitializePhase-def)
  with inv5-2
  show \( \text{thesis} \)
    \( \text{by} \) (auto simp add: hasRead-def)
qed

lemma HFail-HInv5-q:
  assumes act: HFail s s' p
and inv: HInv5-inner s q
and pnq: \( p \neq q \)
and inv2a: Inv2a-inner s'
shows $\text{HInv5-inner } s' q$

proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase $s' q = 2$
and nR2: $\forall D \in \text{MajoritySet}$. 
\hspace{1cm} $\forall qa. \exists d \in D. \text{bal} (\text{dblock } s' q) < \text{mbal} (\text{disk } d \ qa) \longrightarrow 
\hspace{1cm} \text{hasRead } s' d qa (\text{is } ?P s')$
from HFail-HInv5-q2[OF act pnq]
have $\neg (?P s) \Longrightarrow \neg (?P s')$
  by auto
  with nR2
  have P: $?P s$
  by blast
from inv HFail-HInv5-q1[OF act pnq inv2a]
show maxBalInp s $\langle$ bal (dblock $s' q)$ $(\text{inp (dblock } s' q)$ $\rangle$
  by (auto simp add: HInv5-inner-def HInv5-inner-R-def)
qed

theorem HFail-HInv5:
  $\{ H\text{Fail } s \ s' p; H\text{Inv5-inner } s \ q; H\text{Inv5-inner } s' q \} 
\Longrightarrow H\text{Inv5-inner } s' q$
by (blast dest: HFail-HInv5-q HFail-HInv5-p)

lemma HPhase0Read-HInv5-p:
  $H\text{Phase0Read } s \ s' p \ d 
\Longrightarrow H\text{Inv5-inner } s' p$
by (auto simp add: Phase0Read-def HInv5-inner-def)

lemma HPhase0Read-allBlocks:
  assumes act: $H\text{Phase0Read } s \ s' p \ d$
  shows allBlocks $s' \subseteq \text{allBlocks } s$
  using $H\text{Phase0Read-blocksOf} [OF \text{act}]$
  by (auto simp add: allBlocks-def)

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

lemma HPhase0Read-HInv5-q2:
  assumes act: $H\text{Phase0Read } s \ s' p \ d$
  and pnq: $p \neq q$
  and inv5-2: $\exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \ \text{bal (dblock } s q) < \text{mbal (disk } d \ qq)$
  \hspace{1cm} $\land \neg \text{hasRead } s d qq)$

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

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: Phase0Read-def InitializePhase-def)
with inv5-2
show ?thesis
by(auto simp add: hasRead-def)
qed

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

qed

theorem HPhase0Read-HInv5:
[ HPhase0Read $s, s'$ $p, d; HInv5-inner s, q ] \Rightarrow HInv5-inner s', q$
by(blind 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 $s' q \subseteq \text{blocksOf } s q$
proof –
from act pnq
have dblock: $\{ \text{dblock } s' q \} \subseteq \{ \text{dblock } s q \}$
and disk: disk $s' = disk s$

112
and blks:: blocksRead s' q = blocksRead s q 
by(auto simp add: EndPhase0-def InitializePhase-def)
from disk have disk' q = blocksRead s q 
by(auto simp add: EndPhase0-def InitializePhase-def)
from disk
have disk': {disk s d q | d . d∈ UNIV} ⊆ {disk s d q | d . d∈ UNIV} (is ?D' ⊆ ?D)
  by auto
from pnq act have (UN qq d. rdBy s' q qq d) ⊆ (UN qq d. rdBy s q qq d)
  by(auto simp add: EndPhase0-def InitializePhase-def
      rdBy-def split: if-split_asm, blast)
hence {block br | br. br ∈ (UN qq d. rdBy s' q qq d)} ⊆ {block br | br. br ∈ (UN qq d. rdBy s q qq d)}
(is ?R' ⊆ ?R)
  by auto blast
from union-inclusion[OF dblock union-inclusion[OF disk' this]] show ?thesis
  by(auto simp add: blocksOf-def)
qed

lemma HEndPhase0-allBlocks:
assumes act: HEndPhase0 s s' p
shows allBlocks s' ⊆ allBlocks s ∪ {dblock s' p}
proof(auto simp del: HEndPhase0-def simp add: allBlocks-def
      dest: HEndPhase0-blocksOf-q[OF act])
fix x pa
assume x-pa: x∈ blocksOf s' pa and
  x-nblks: ∨ xa. x /∈ blocksOf s xa
show x=dblock s' p
proof(cases p=pa)
case True
  from x-nblks
  have x /∈ blocksOf s p
    by auto
  with True subsetD[OF HEndPhase0-blocksOf[OF act] x-pa]
  show ?thesis
    by auto
next
case False
  from x-nblks subsetD[OF HEndPhase0-blocksOf-q[OF act False] x-pa]
  show ?thesis
    by auto
qed

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

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

fix \( bk \)

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

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

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

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

from \( \text{subsetD} \ (\text{OF} \ \text{HEndPhase0-allBlocks} \ (\text{OF} \ \text{act} \ bk) \)

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

proof

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

with \( \text{inv5-1 dblock} \)

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

next

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

with \( \text{HEndPhase0-some} \ (\text{OF} \ \text{inv1} \ \text{act} \)

have \( \exists \ ba \in \text{allBlocksRead} \ s \ p. \ \text{bal} \ ba = \text{bal} (\text{dblock} \ s \ p) \land \text{inp} \ ba = \text{inp} (\text{dblock} \ s \ p) \)

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

then obtain \( ba \)

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

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

by auto

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

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

from \( \text{subsetD} \ (\text{OF} \ \text{this} \ \text{ba-blksread} \ \text{ba-balinp} \ \text{bal} \ bk \ \text{dblock'} \ \text{inv5-1} \)

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

qed

qed

lemma \( \text{HEndPhase0-Hinv5-q2} \):

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

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

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

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

proof –

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

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

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

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

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

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

show \( ?\text{thesis} \)

 qed
by (auto simp add: hasRead-def)

qed

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

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

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

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

end

theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

C.6 Lemma I2f

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

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

show \(?\text{thesis}\)

by (auto simp add: EndPhase1-def)

qed

lemma \( H\text{EndPhase1-maxBalInp} \):

assumes \( \text{act}: H\text{EndPhase1} s s' q \)
and \( \text{asm1}: b \in (\text{UN} \ p. \ \text{Ballot} p) \)
and \( \text{asm2}: \text{D} \in \text{MajoritySet} \)
and \( \text{asm3}: \text{maxBalInp} s b v \)
and \( \text{asm4}: \forall d \in D. \ b \leq \text{bal}(\text{disk} s d p) \)
\( \land (\forall q. (\text{phase} s q = 1 \land b \leq \text{mbal} (\text{dblock} s q) \land \text{hasRead} s q d p)) \)
\( \rightarrow (\exists \text{br} \in \text{blocksRead} s q d. b \leq \text{bal}(\text{block} \text{br})) \)

and \( \text{inv1}: \text{Inv1} s \)
and \( \text{inv2a}: \text{Inv2a} s \)
and \( \text{inv2b}: \text{Inv2b} s \)
shows \( \text{maxBalInp} s' b v \)

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

case True

show \(?\text{thesis}\)

proof (cases \( p \neq q \))

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

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

proof (auto simp add: EndPhase1-def)

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 \( \text{IsMajority}(?M) \))

by (auto simp add: EndPhase1-def)

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

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 \( \text{pnq} \)

show \(?\text{thesis}\)

by auto

qed

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

with \( \text{asm4} \) \( \text{asm3} \) \( \text{act True} \)

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

by (auto simp add: EndPhase1-def)

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

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

by (auto simp add: EndPhase1-def)

from \( p42 \)

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

fix br

assume br: br ∈ blocksRead s q d

and b-bal: b ≤ bal (block br)

hence br-rdBy: br ∈ (UN q d. rdBy s (proc br) q d)

by(auto simp add: rdBy-def)

hence br-blksof: block br ∈ blocksOf s (proc br)

by(auto simp add: blocksOf-def)

from br have br-bseen: block br ∈ blocksSeen s q

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

from HEndPhase1-valueChosen-inp[OF act inv2a asm1 br-blksof br-bseen b-bal
asm3 inv1]

  show ?thesis .

qed

next

case False

from asm4

have p41: ∀ d∈D. b ≤ bal(disk s d p)

by auto

have p42: ∃ d∈D. disk s d p = dblock s p

proof –

from act

have IsMajority {d. d∈disksWritten s q ∧ (∀ p∈UNIV−{q}. hasRead s q d p)} (is IsMajority ?S)

by(auto simp add: EndPhase1-def)

with majorities-intersect asm2

have D ∩ ?S ≠ {}

by(auto simp add: MajoritySet-def)

hence ∃ d∈D. d∈disksWritten s q

by auto

with inv2b False

show ?thesis

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

qed

have inp(dblock s' q) = v

proof –

from p42 p41 False

have b-bal: b ≤ bal(dblock s q) by auto

have db-blksof: (dblock s q) ∈ blocksOf s q

by(auto simp add: blocksOf-def)

have db-bseen: (dblock s q) ∈ blocksSeen s q

by(auto simp add: blocksSeen-def)

from HEndPhase1-valueChosen-inp[OF act inv2a asm1 db-blksof db-bseen b-bal
asm3 inv1]

  show ?thesis .

qed
\begin{verbatim}

with asm3 HEndPhase1-allBlocks[OF act]

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

qed

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 allBlocks s
      and b-bal: b \leq bal bk
      from subsetD[OF HEndPhase1-allBlocks[OF act] bk]
      show inp bk = v
        proof(auto simp add: maxBalInp-def)

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

      qed

    qed

lemma HEndPhase1-valueChosen2:

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

  shows \?P s'

proof(auto)

  fix d
  assume d: d \in D
  with act asm4
  show b \leq bal (disk s' d p)
    by(auto simp add: EndPhase1-def)
  fix d q
  assume d: d \in D
  and phase': phase s' q = Suc 0

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

with \( \text{act} \)

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

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

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

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

ultimately
have \( \exists \ br \in \text{blocksRead } s \ q \ d. \ b \leq \text{bal} (\text{block } br) \)
using \( p31 \) asm4 d
by blast

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

qed

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

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

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

\textbf{proof} –

from \( \text{vc} \)

obtain \( b \ p \ D \) where
\( \text{asm1: } b \in (\text{UN } p. \ \text{Ballot } p) \)
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: \( \text{valueChosen-def} \))

from \( \text{HEndPhase1-maxBalInp}[\text{OF } \text{asm1 } \text{asm2 } \text{asm3 } \text{asm4 } \text{inv1 } \text{inv2a } \text{inv2b}] \)

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

with \( \text{HEndPhase1-valueChosen2}[\text{OF } \text{asm4}] \) \( \text{asm1 } \text{asm2} \)

\textbf{show} ?thesis

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

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

lemma HStartBallot-valueChosen2:
assumes act: HStartBallot s s' q
and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
   ∧ (∀ q. (phase s q = 1
      ∧ b ≤ mbal(dblock s q)
      ∧ hasRead s q d p
   ) *** (∃ br ∈ blocksRead s q d. b ≤ bal(block br))) (is ?P s)
shows ?P s'
proof (auto)
fix d
assume d: d ∈ D
with act asm4
show b ≤ bal (disk s' d p)
  by (auto simp add: StartBallot-def)
fix d q
assume d: d ∈ D
and phase': phase s' q = Suc 0
and dblk-mbal: b ≤ mbal (dblock s' q)
and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
and p32: dblock s' q = dblock s q
  by (auto simp add: StartBallot-def InitializePhase-def)
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: \text{StartBallot-def} \text{InitializePhase-def} hasRead-def split: if-split-asm)
ultimately
have \( \exists \ br \in \text{blocksRead} \ s \ q \ d \ b \leq \text{bal}(\text{block } br) \)
  using p\(\tilde{\text{3}}\) asm\(\tilde{\text{4}}\) 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{StartBallot-def} \text{InitializePhase-def} hasRead-def)
qed

theorem HStartBallot-valueChosen:
assumes \text{act}: HStartBallot \( s \ s' \ q \)
and \text{vc}: valueChosen \( s \ v \)
and \text{v-input}: \( v \in \text{Inputs} \)
shows valueChosen \( s' \ v \)
proof
  from \text{vc}
  obtain \( b \ p \ D \) where
    asm\(\tilde{1}\): \( b \in (\bigcup \ p. \text{Ballot } p) \)
    and asm\(\tilde{2}\): \( D \in \text{MajoritySet} \)
    and asm\(\tilde{3}\): maxBalInp \( s \ b \ v \)
    and asm\(\tilde{4}\): \( \forall \ d \in D. \ b \leq \text{bal}(\text{disk } s \ d \ p) \)
    \( \land (\forall \ q. \ (\text{phase } s \ q = 1 \land b \leq \text{mbal}(\text{dblock } s \ q) \land \text{hasRead } s \ q \ d \ p) \implies (\exists \ br \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block } br))) \)
    by(auto simp add: valueChosen-def)
from HStartBallot-maxBalInp[\text{OF act asm\(\tilde{3}\)}]
have maxBalInp \( s' \ b \ v \).
with HStartBallot-valueChosen2[\text{OF act asm\(\tilde{4}\)}] asm\(\tilde{1}\) asm\(\tilde{2}\)
show ?thesis
  by(auto simp add: valueChosen-def)
qed

lemma HPhase1or2Write-maxBalInp:
assumes \text{act}: HPhase1or2Write \( s' \ q \ d \)
and asm\(\tilde{3}\): 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 \( b - \text{bal} \): \( b \leq \text{bal} \ bk \)

from \( \text{subsetD[OF HPhase1or2Write-allBlocks[OF act] bk]} \) \( asm3 \) \( b - \text{bal} \)

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

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

qed


lemma \( HPhase1or2Write-valueChosen2: \)

assumes \( act: HPhase1or2Write \ s s' \ pp \ d \)

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

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

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

and \( inv4: HInv4a \ s pp \)

shows \( ?P s' \)

proof (auto)

fix \( d1 \)

assume \( d: d1 \in D \)

show \( b \leq \text{bal(disk} s d \ pp) \)

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

  case True

  with \( inv4 \) \( act \)

  have \( HInv4a2 \ s p \)

  by (auto simp add: \text{Phase1or2Write-def HInv4a-def})

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

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

  by (auto simp add: \text{HInv4a2-def MajoritySet-def})

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

  by \( \text{auto} \)

from \( asm4 \) \( p41 \)

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

by \( \text{auto} \)

with \( p41 \)

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

by \( \text{auto} \)

from \( act \) \( True \)

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

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

with \( p42 \) \( True \)

show \( \text{thesis} \)

by \( \text{auto} \)

next

  case False

  with \( act \) \( asm4 \) \( d \)

  show \( \text{thesis} \)

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

qed

next
fix d q
assume d : d ∈ D
  and phase': phase s q = Suc 0
  and dblk-mbal: b ≤ mbal (dblock s q)
  and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
  and p32: dblock s' q = dblock s q
  by (auto simp add: Phase1or2Write-def InitializePhase-def
                  hasRead-def split : if-split-asm)
with dblk-mbal
have b ≤ mbal (dblock s q) by auto
moreover
from act hasRead
have hasRead s q d p
  by (auto simp add: Phase1or2Write-def InitializePhase-def
                hasRead-def split : if-split-asm)
ultimately
have ∃ br ∈ blocksRead s q d. b ≤ bal (block br)
  using p31 asm4 d
  by blast
with act hasRead
show ∃ br ∈ blocksRead s' q d. b ≤ bal (block br)
  by (auto simp add: Phase1or2Write-def InitializePhase-def
                     hasRead-def)
qed

theorem HPhase1or2Write-valueChosen:
assumes act: HPhase1or2Write s s' q d
and vc: valueChosen s v
and v-input: v ∈ Inputs
and inv4: HInv4a s q
shows valueChosen s' v
proof –
from vc
obtain b p D where
  asm1: b ∈ (UN p. Ballot p)
  and asm2: D ∈ MajoritySet
  and asm3: maxBallInp s b v
  and asm4: ∀ d ∈ D. b ≤ bal (disk s d p)
    ∧ (∀ q. (phase s q = 1
      ∧ b ≤ mbal (dblock s q)
      ∧ hasRead s q d p
    ) → (∃ br ∈ blocksRead s q d. b ≤ bal (block br))))
  by (auto simp add: valueChosen-def)
from HPhase1or2Write-maxBallInp[OF act asm3]
have maxBallInp s' b v .
with HPhase1or2Write-valueChosen2[OF act asm2 asm4 inv4] asm1 asm2
show ?thesis
by (auto simp add: valueChosen-def)
qed

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

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

\[ \text{theorem } HPhase1or2ReadThen-valueChosen: \]
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 \( b \ p \ D \ where \) 
\[ \text{asm1: } b \in (\text{UN } p. \ \text{Ballot } p) \] 
\[ \text{asm2: } D \in \text{MajoritySet} \] 
\[ \text{asm3: } \text{maxBalInp } s \ b \ v \] 
\[ \text{asm4: } \forall d \in D. \ b \leq \text{bal}(\text{disk } s \ d \ p) \] 
\[ \wedge (\forall q. (\text{phase } s \ q = 1 \) 
\[ \wedge b \leq \text{mbal}(\text{dblock } s \ q) \] 
\[ \wedge \text{hasRead } s \ q \ dd \ p \) 
\[ ) \longrightarrow (\exists \ br \in \text{blocksRead } s \ q \ dd. \ b \leq \text{bal}(\text{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 

\[ \text{theorem } HPhase1or2ReadElse-valueChosen: \]
[ HPhase1or2ReadElse \( s \ s' \ p \ d \ r \); valueChosen \( s \ v \); v\in \text{Inputs} ] 

\[ \Longrightarrow \text{valueChosen } s' \ v \] 
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 ∈ allBlocks s'
    and b-bal: b ≤ bal bk
  from subsetD[OF HEndPhase2-allBlocks[OF act] bk] asm3 b-bal
  show inp bk = v
    by(auto simp add: maxBalInp-def)
qed

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

qed

theorem HEndPhase2-valueChosen:
assumes act: HEndPhase2 s s' q
and vc: valueChosen s v
and v-input: \( v \in \text{Inputs} \)
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: \( \text{maxBalInp } s \ b \ v \)
and asm4: \( \forall d \in D. \ b \leq \text{bal(disk } s \ d \ p) \)
\( \wedge \forall q. (\text{phase } s \ q = 1 \)
\( \wedge b \leq \text{mbal(dblock } s \ q) \)
\( \wedge \text{hasRead } s \ q \ d \ p \)
\( \rightarrow (\exists b r \in \text{blocksRead } s \ q \ d. \ b \leq \text{bal(block br)}) \)
by(auto simp add: valueChosen-def)
from HEndPhase2-maxBalInp[OF act asm3]
have maxBalInp s' b v .
with HEndPhase2-valueChosen2[OF act asm4] asm1 asm2
show ?thesis
by(auto simp add: valueChosen-def)
qed

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

moreover
from Ballot-nzero asm1
have \(0 < b\)
  by auto

ultimately
show \(\text{thesis}\)
  using b-bal
  by auto

qed

qed

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

shows \(\neg P \; s'\)

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

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

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

lemma HPhase0Read-valueChosen2:
  assumes act: HPhase0Read s s' q q d d
  and asm4: ∀ d ∈ D. b ≤ bal(disk s d p)
    ∧ (∀ q. (phase s q = 1
       ∧ b ≤ mbal(dblock s q)
       ∧ hasRead s q d p
       ) → (∃ br ∈ blocksRead s q d. b ≤ bal(block br))))
  shows ?P s'
proof (auto)
fix d
assume d: d∈D
with act asm4
  show b ≤ bal (disk s' d p)
    by(auto simp add: Phase0Read-def)
next
fix d q
assume d: d∈D
  and phase': phase s' q = Suc 0
  and dblk-mbal: b ≤ mbal (dblock s' q)
  and hasRead: hasRead s' q d p
from phase' act
  have qqnq: qq≠q
    by(auto simp add: Phase0Read-def)
  show ∃ br∈blocksRead s' q d. b ≤ bal (block br)
proof —
  from phase' act hasRead
  have p31: phase s q = 1
    and p32: dblock s' q = dblock s q
    by(auto simp add: Phase0Read-def hasRead-def)
  with dblk-mbal
  have b ≤ mbal (dblock s q)
    by auto
moreover
  from act hasRead qqnq
  have hasRead s q d p
    by(auto simp add: Phase0Read-def hasRead-def
      split: if-split-asm)
ultimately
  have ∃ br∈blocksRead s q d. b ≤ bal (block br)
    using p31 asm4 d
    by blast
  with act hasRead
  show ∃ br∈blocksRead s' q d. b ≤ bal (block br)
    by(auto simp add: 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 ∈ Inputs
  shows valueChosen s' v
proof —
  from vc
  obtain b p D where
    asm1: b ∈ (UN p. Ballot p)
    and asm2: D∈MajoritySet
    and asm3: maxBalInp s b v
and \( asm4 \): \( \forall d \in D. \ b \leq \text{bal}(\text{disk} s d p) \)
\( \wedge (\forall q. ( \text{phase} s q = 1 \wedge b \leq \text{mbal}(\text{dblock} s q) \wedge \text{hasRead} s q d p) \rightarrow (\exists br \in \text{blocksRead} s q d. \ b \leq \text{bal}(\text{block} br))) \)

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

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

with \( \text{HPhase0Read-valueChosen2[OF act asm4]} \) \( \text{asm1 asm2} \)
show \( \text{thesis} \)
by \( \text{(auto simp add: valueChosen-def)} \)

qed

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

qed
lemma HEndPhase0-valueChosen2:
assumes act: HEndPhase0 s s' q and asm4: \(\forall d \in D. \quad b \leq bal(disk s d p)\)
\(\land (\forall q. (\quad phase s q = 1 \land b \leq mbal(dblock s q)) \land hasRead s q d p)\)
\(\quad \rightarrow (\exists br \in blocksRead s q d. b \leq bal(block br)))\)
(is \(?P s)\)
shows \(?P s'\)
proof (auto)
fix d assume d: \(d \in D\)
with act asm4
show \(b \leq bal(disk s' d p)\)
by (auto simp add: EndPhase0-def)
fix d q assume d: \(d \in D\)
and phase': phase s' q = Suc 0
and dblk-mbal: \(b \leq mbal(dblock s' q)\)
and hasRead: hasRead s' q d p
from phase' act hasRead
have p31: phase s q = 1
and p32: dblock s' q = dblock s q
by (auto simp add: EndPhase0-def InitializePhase-def hasRead-def split : if-split-asm)
with dblk-mbal
have \(b \leq mbal(dblock s q)\) by auto
moreover
from act hasRead
have hasRead s q d p
by (auto simp add: EndPhase0-def InitializePhase-def hasRead-def split : if-split-asm)
ultimately
have \(\exists br \in blocksRead s q d. b \leq bal(block br)\)
using p31 asm4 d
by blast
with act hasRead
show \(\exists br \in blocksRead s' q d. b \leq bal(block br)\)
by (auto simp add: EndPhase0-def InitializePhase-def hasRead-def)
qed

theorem HEndPhase0-valueChosen:
assumes act: HEndPhase0 s s' q
and vc: valueChosen s v
and v-input: \(v \in Inputs\)
and inv1: Inv1 s
shows valueChosen s' v
proof
from vc

133
obtain $b p D$ where

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

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

and asm3: $\maxBalInp s b v$

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

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

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

$\wedge \text{hasRead s q d p})$

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

by (auto simp add: valueChosen-def)

from HEndPhase0-maxBalInp[OF act asm3 inv1]

have $\maxBalInp s' b v$.

with HEndPhase0-valueChosen2[OF act asm4] asm1 asm2

show $\text{thesis}$

by (auto simp add: valueChosen-def)

qed

end

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

definition $HInv6 :: \text{state }\Rightarrow\text{ bool}$

where

$HInv6 s = ((\text{chosen s} \neq \text{NotAnInput} \rightarrow \text{valueChosen s (chosen s)})$

$\wedge (\forall p. \text{outpt s p} \in \{\text{chosen s}, \text{NotAnInput}\})$)

theorem $HInit-HInv6$: $HInit s \Rightarrow HInv6 s$

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

lemma HEndPhase2-Inv6-1:

assumes act: $HEndPhase2 s s' p$

and inv: $HInv6 s$

and inv2b: $\text{Inv2b s}$

and inv2c: $\text{Inv2c s}$

and inv3: $HInv3 s$

and inv5: $HInv5-inner s p$

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

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

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

from inv5 act

have inv5R: $HInv5-inner-R s p$

and phase: phase s p = 2

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

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

case True

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

proof -

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

  proof auto

  fix D q

  assume Dmaj: D \in \text{MajoritySet}

  from ep2-maj Dmaj majorities-intersect

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

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

  then obtain d

  where dinD: d \in D

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

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

  by auto

  from inv2b

  have Inv2b-inner s p d

  by (auto simp add: Inv2b-def)

  with ddisk

  have disk s d p = dblock s p

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

  with inv2c phase

  have bal (dblock s p) = mbal(disk s d p)

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

  with dhasR dinD

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

  by auto

  qed

  with inv5R

  show \?thesis

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

  qed

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

proof -

from act

have outpt': outpt s' = (outpt s) (p:= inp (dblock s p))

by (auto simp add: EndPhase2-def)

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

proof auto

fix q

assume pnq: p \neq q

from outpt' pnq

have outpt s' q = outpt s q

by (auto simp add: EndPhase2-def)

with True inv2c

135
show outpt s' q= NotAnInput
  by(auto simp add: Inv2c-def Inv2c-inner-def)
qed

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

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

from ep2-maj inv2b majorities-intersect
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \) disk s d p = dblock s p
  \& (\( \forall q \in \text{UNIV} - \{p\}. \) 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 = dblock s p
    \& (\( \forall q \in \text{UNIV} - \{p\}. \) hasRead s p d q)
  by auto
have p35: \forall q. \forall d \in D. ( phase s q=1 \& bal(dblock s p)≤mbal(dblock s q)\& hasRead s q d p)
  \( \implies \) \( \{\text{block}=dblock s p, \text{proc}=p\}\in\text{blocksRead s q d} \)
proof auto
  fix q d
  assume dD: \( d \in D \) and phase-q: phase s q= Suc 0
  and bal-mbal: bal(dblock s p)≤mbal(dblock s q) and hasRead: hasRead s q d p
  from phase inv2c

136
have \( \text{bal}(\text{dblock } s \ p) = \text{mbal}(\text{dblock } s \ p) \)
\hspace{1em} by (auto simp add: Inv2c-def Inv2c-inner-def)
moresover
from inv2c phase
have \( \forall \text{br} \in \text{blocksRead } s \ p \ d . \ \text{mbal} (\text{block } \text{br}) < \text{mbal}(\text{dblock } s \ p) \)
\hspace{1em} by (auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
have \( p41 : \{ \text{block} = \text{dblock } s \ q, \ \text{proc} = q \} \not\in \text{blocksRead } s \ p \ d \)
\hspace{1em} using bal-mbal
\hspace{1em} by auto
from phase phase-q
have \( p \neq q \) by auto
with \( p34 \) \( dD \)
have \( \text{hasRead } s \ p \ d \ q \)
\hspace{1em} by auto
with \( \text{phase phase-q hasRead } \) inv3 \( p41 \)
show \( \{ \text{block} = \text{dblock } s \ q, \ \text{proc} = p \} \not\in \text{blocksRead } s \ q \ d \)
\hspace{1em} by (auto simp add: HInv3-def HInv3-inner-def HInv3-L-def HInv3-R-def)
qed
have \( p36 : \forall \ q . \ \forall \ d \in D . \ \text{phase } s' \ q = 1 \wedge \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' \ q) \wedge \text{hasRead } s' \ q \ d \ p \)
\hspace{1em} \rightarrow (\exists \text{br} \in \text{blocksRead } s' \ q \ d . \ \text{bal}(\text{block } \text{br}) = \text{bal}(\text{dblock } s \ p))
proof (auto)
fix \ q \ d
assume \( dD : d \in D \) and phase-q: phase \( s' \ q = \text{Suc } 0 \)
and bal: \( \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' \ q) \)
and hasRead: \( \text{hasRead } s' \ q \ d \ p \)
from phase-q act
have phase \( s' \ q = \text{phase } s \ q \wedge \text{dblock } s' \ q = \text{dblock } s \ q \wedge \text{hasRead } s' \ q \ d \ p = \text{hasRead } s \ q \ d \ p \wedge \text{blocksRead } s' \ q \ d = \text{blocksRead } s \ q \ d \)
\hspace{1em} by (auto simp add: EndPhase2-def hasRead-def InitializePhase-def)
with \( p35 \) phase-q bal hasRead \( dD \)
have \( \{ \text{block} = \text{dblock } s \ p, \ \text{proc} = p \} \in \text{blocksRead } s' \ q \ d \)
\hspace{1em} by auto
thus \( \exists \text{br} \in \text{blocksRead } s' \ q \ d . \ \text{bal}(\text{block } \text{br}) = \text{bal}(\text{dblock } s \ p) \)
\hspace{1em} by force
qed
hence \( p36-2 : \forall \ q . \ \forall \ d \in D . \ \text{phase } s' \ q = 1 \wedge \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' \ q) \wedge \text{hasRead } s' \ q \ d \ p \)
\hspace{1em} \rightarrow (\exists \text{br} \in \text{blocksRead } s' \ q \ d . \ \text{bal}(\text{block } \text{br}) \leq \text{bal}(\text{block } \text{br}))
by force
from act
have bal-dblock: \( \text{bal}(\text{dblock } s' \ p) = \text{bal}(\text{dblock } s \ p) \)
\hspace{1em} and disk: \( \text{disk } s' = \text{disk } s \)
\hspace{1em} by (auto simp add: EndPhase2-def)
from bal-dblock p33
have maxBalInp \( s' (\text{bal}(\text{dblock } s \ p)) (\text{chosen } s') \)
\hspace{1em} by auto
moreover
from disk p34
have \( \forall d \in D. \text{bal}(\text{dblock } s \ p) \leq \text{bal}(\text{disk } s' \ d \ p) \)
  by auto
ultimately
have 
\[ \text{maxBalInp } s' \ (\text{bal}(\text{dblock } s \ p)) \ (\text{chosen } s') \land \]
\( (\exists D \in \text{MajoritySet}. \)
\( \forall d \in D. \text{bal}(\text{dblock } s \ p) \leq \text{bal} (\text{disk } s' \ d \ p) \land \)
\( (\forall q. \text{phase } s' q = \text{Suc } 0 \land \)
\( \text{bal}(\text{dblock } s \ p) \leq \text{mbal} (\text{dblock } s' q) \land \text{hasRead } s' q \ d \ p \longrightarrow \)
\( (\exists \text{br}\in\text{blocksRead } s' q \ d. \text{bal}(\text{dblock } s \ p) \leq \text{bal} (\text{block } \text{br}))) \)
using p36-2 Dmaj
by auto
moreover
from phase inv2c
have \( \text{bal}(\text{dblock } s \ p) \in \text{Ballot } p \)
  by (auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
show \(?\text{thesis} \)
  by (auto simp add: valueChosen-def)
next
\text{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: Hviv6-def)
from HEndPhase2-valueChosen[OF act this] p31 False InputsOrNi
show \( ?\text{thesis} \)
  by auto
qed

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

lemma \(\text{valueChosen-equal}:\)
assumes \(v: \text{valueChosen s v}\)
and \(w: \text{valueChosen s w}\)
shows \(v = w\) using \(\text{assms}\)
proof (auto simp add: \(\text{valueChosen-def}\))
fix \(a\ b\ aa\ ba\ p\ D\ pa\ Da\)
assume \(\text{max-v: maxBalInp s b v}\)
and \(\text{Dmaj: D \in MajoritySet}\)
and \(\text{asm-v: \forall d \in D. b \leq \text{bal (disk s d p)}}\) \(\land\)
\(\forall q. \text{phase s q} = \text{Suc 0} \land\)
\(b \leq \text{mbal (dblock s q) \land hasRead s q d p} \longrightarrow\)
\(\text{hasRead s q d} \longrightarrow\)
\(\exists br \in \text{blocksRead s q d. b \leq} \text{bal (block br)}))\)
and \(\text{max-w: maxBalInp s ba w}\)
and \(\text{Damaj: Da \in MajoritySet}\)
and \(\text{asm-w: \forall d \in Da. ba \leq} \text{bal (disk s d pa)}\) \(\land\)
\(\forall q. \text{phase s q} = \text{Suc 0} \land\)
\(ba \leq \text{mbal (dblock s q) \land hasRead s q d pa} \longrightarrow\)
\(\exists br \in \text{blocksRead s q d. ba \leq} \text{bal (block br)}))\)
from \(\text{asm-v}\)
have \(\text{asm-v: \forall d \in D. b \leq} \text{bal (disk s d p)}\) by auto
from \(\text{asm-w}\)
have \(\text{asm-w: \forall d \in Da. ba \leq} \text{bal (disk s d pa)}\) by auto
show \(v = w\)
proof (cases \(b \leq ba\))
case \(\text{True}\)
from \(\text{valueChosen-equal-case[OF max-v Dmaj asm-v max-w Damaj asm-w True]}\)
show \(\text{thesis},\)
next
case \(\text{False}\)
from \(\text{valueChosen-equal-case[OF max-w Damaj asm-w max-v Dmaj asm-v]}\) \(\text{False}\)
show \(\text{thesis}\)
by auto
qed
qed

lemma HEndPhase2-Inv6-2:
assumes act: HEndPhase2 s s' p
and inv: HHinv6 s
and inv2b: Inv2b s
and inv2c: Inv2c s
and inv3: HHinv3 s
and inv5: HHinv5-inner s p
and asm: outpt s' r ≠ NotAnInput
shows outpt s' r = chosen s'
proof (cases chosen s = NotAnInput)
case True
with inv2c
have ∀ q. outpt s q = NotAnInput
  by (auto simp add: Inv2c-def Inv2c-inner-def)
with True act asm
show ?thesis
  by (auto simp add: EndPhase2-def HNextPart-def split: if-split-asm)
next
case False
with inv
have p31: valueChosen s (chosen s)
  by (auto simp add: HHinv6-def)
with False act
have chosen s' ≠ NotAnInput
  by (auto simp add: HNextPart-def)
from HEndPhase2-Inv6-1[OF act inv inv2b inv2c inv3 inv5 this]
have p32: valueChosen s' (chosen s')
from False InputsOrNi
have chosen s ∈ Inputs by auto
from valueChosen-equal[OF HEndPhase2-valueChosen[OF act p31 this] p32]
have p33: chosen s = chosen s'
from act
have maj: IsMajority {d . d ∈ disksWritten s p
  ∧ (∀ q ∈ UNIV – {p}. hasRead s p d q}) (ls IsMajority ?D)
  and phase: phase s p = 2
  by (auto simp add: EndPhase2-def)
show ?thesis
proof (cases outpt s r = NotAnInput)
case True
with asm act
have p41: r = p
  by (auto simp add: EndPhase2-def split: if-split-asm)
from maj
have p42: ∃ D ∈ MajoritySet. ∀ d ∈ D. ∀ q ∈ UNIV – {p}. hasRead s p d q
by (auto simp add: MajoritySet-def)

have p43: \(\neg (\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \ \text{bal}(\text{dblock s p}) < \text{mbal}(\text{disk s d q}) \land \neg \text{hasRead s p d q})\))

proof auto

fix D q

assume Dmaj: D \in MajoritySet

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

proof (cases p=q)

assume pq: p=q

thus ?thesis

proof auto

from maj majorities-intersect Dmaj

have ?D \cap D \neq {}

by (auto simp add: MajoritySet-def)

hence \(\exists d \in ?D \cap D. d \in \text{disksWritten s p}\) by auto

then obtain d where d: d \in disksWritten s p and d \in ?D \cap D

by auto

hence dD: d \in D by auto

from d inv2b

have disk s d p = dblock s p

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

with inv2c phase

have bal(dblock s p) = mbal(disk s d p)

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

with dD pq

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

by auto

qed

next

case False

with p42

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

by auto

with majorities-intersect Dmaj

show ?thesis

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

qed

qed

with inv5 act

have p44: maxBalInp s (bal(dblock s p)) (inp(dblock s p))

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

have \(\exists bk \in \text{allBlocks s}. \exists b \in (\text{UN p. Ballot p}). (\text{maxBalInp s b (chosen s)}) \land b \leq \text{bal bk}\)

proof -

have disk-allblks: \(\forall d p. \text{disk s d p} \in \text{allBlocks s}\)

by (auto simp add: allBlocks-def blocksOf-def)

from p31

141
have \( \exists b \in (\text{UN } p. \text{Ballot } p), \maxBalInp s b (\text{chosen } s) \) \( \wedge \)

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

by(auto simp add: valueChosen-def, force)

with majority-nonempty obtain \( b \) \( D \) \( d \)

where IsMajority \( D \) \( b \in (\text{UN } p. \text{Ballot } p) \) \( \wedge \)
maxBalInp \( s \) \( b \) (chosen \( s \)) \( \wedge \) \( d \in D \) \( b \leq \text{bal}(\text{disk } s d p) \)

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

with disk-allblks

show \( \text{thesis} \)

by(auto)

qed

then obtain \( bk \) \( b \)

where \( p45-bk: bk \in \text{allBlocks } s \wedge b \leq \text{bal } bk \)

and \( p45-b: bc(UN p. \text{Ballot } p) \wedge (\maxBalInp s b (\text{chosen } s)) \)

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 \( \text{dblock } s p \in \text{allBlocks } s \)

by(auto simp add: allBlocks-def blocksOf-def)

with \( p45-b True \)

show \( \text{thesis} \)

by(auto simp add: maxBalInp-def)

next
case False

from \( p44 p45-bk False \)

have \( \text{inp } bk = \text{inp}(\text{dblock } s p) \)

by(auto simp add: maxBalInp-def)

with \( p45-b p45-bk \)

show \( \text{thesis} \)

by(auto simp add: maxBalInp-def)

qed

with \( p41 p33 act \)

show \( \text{thesis} \)

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

by(auto simp add: Inv2c-inner-def EndPhase2-def split: if-split-asm)

with \( inv p33 False \)

show \( \text{thesis} \)

by(auto simp add: HInv6-def)

qed
**Theorem:** $H_{EndPhase2}^\text{Inv6}$:

- **Assumes** $\text{act} : H_{EndPhase2} s s'$
- and $\text{inv} : H_{inv6} s$
- and $\text{inv2b} : H_{inv2b} s$
- and $\text{inv2c} : H_{inv2c} s$
- and $\text{inv3} : H_{inv3} s$
- and $\text{inv5} : H_{inv5-inner} s p$

- **Shows** $H_{inv6} s'$

**Proof:** (auto simp add: $H_{inv6-def}$)

Assume $\text{chosen } s' \neq \text{NotAnInput}$

From $H_{EndPhase2}^\text{Inv6-1}[OF \text{act } \text{inv } \text{inv2b } \text{inv2c } \text{inv3 } \text{inv5 } \text{this}]$

Show $\text{valueChosen } s' (\text{chosen } s')$.

**Next**

Fix $p$

Assume $\text{outpt } s' p \neq \text{NotAnInput}$

From $H_{EndPhase2}^\text{Inv6-2}[OF \text{act } \text{inv } \text{inv2b } \text{inv2c } \text{inv3 } \text{inv5 } \text{this}]$

Show $\text{outpt } s' p = \text{chosen } s'$.

**QED**

**Lemma:** $\text{outpt-chosen}$:

- **Assumes** $\text{outpt} : \text{outpt } s = \text{outpt } s'$
- and $\text{inv2c} : H_{inv2c} s$
- and $\text{nextp} : H_{nextPart} s s'$

- **Shows** $\text{chosen } s' = \text{chosen } s$

**Proof** –

From $\text{inv2c}$

Have $\text{chosen } s = \text{NotAnInput} \implies (\forall p. \text{outpt } s p = \text{NotAnInput})$

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

With $\text{outpt nextp}$

Show $?thesis$

By (auto simp add: $H_{nextPart-def}$)

**QED**

**Lemma:** $\text{outpt-Inv6}$:

- [$\text{outpt } s = \text{outpt } s'; \forall p. \text{outpt } s p \in \{\text{chosen } s, \text{NotAnInput}\}$;
  $\text{Inv2c } s; H_{nextPart} s s' \implies \forall p. \text{outpt } s' p \in \{\text{chosen } s', \text{NotAnInput}\}$]

Using $\text{outpt-chosen}$

By auto

**Theorem:** $H_{StartBallot}^\text{Inv6}$:

- **Assumes** $\text{act} : H_{StartBallot} s s'$
- and $\text{inv} : H_{inv6} s$
- and $\text{inv2c} : H_{inv2c} s$

- **Shows** $H_{inv6} s'$

**Proof** –

From $\text{outpt-chosen } \text{act } \text{inv2c } \text{inv}$

Have $\text{chosen } s' \neq \text{NotAnInput} \implies \text{valueChosen } s (\text{chosen } s')$

By (auto simp add: $\text{StartBallot-def} \ H_{inv6-def}$)

143
from HStartBallot-valueChosen[OF act] this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
  by auto
from act have outpt: outpt s = outpt s'
  by(auto simp add: StartBallot-def)
from outpt-Inv6[OF outpt] act inv2c inv
have ∀p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
  by(auto simp add: Hinv6-def)
with t1 show ?thesis
  by(simp add: Hinv6-def)
qed

theorem HPhase1or2Write-Inv6:
assumes act: HPhase1or2Write s s' p d and inv: HInv6 s
and inv4: HInv4a s p and inv2c: Inv2c s
shows HInv6 s'
proof –
from outpt-chosen act inv2c inv
have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
  by(auto simp add: Phase1or2Write-def HInv6-def)
from HPhase1or2Write-valueChosen[OF act] this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
  by auto
from act have outpt: outpt s = outpt s'
  by(auto simp add: Phase1or2Write-def)
from outpt-Inv6[OF outpt] act inv2c inv
have ∀p. outpt s' p = chosen s' ∨ outpt s' p = NotAnInput
  by(auto simp add: Hinv6-def)
with t1 show ?thesis
  by(simp add: Hinv6-def)
qed

theorem HPhase1or2ReadThen-Inv6:
assumes act: HPhase1or2ReadThen s s' p d q and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'
proof –
from outpt-chosen act inv2c inv
have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
  by(auto simp add: Phase1or2ReadThen-def HInv6-def)
from HPhase1or2ReadThen-valueChosen[OF act] this InputsOrNi
have t1: chosen s' ≠ NotAnInput → valueChosen s' (chosen s')
by auto
from act
have outpt: outpt s = outpt s'
  by (auto simp add: Phase1or2ReadThen-def)
from outpt-Inv6[OF outpt] act inv2c inv
have \( \forall p. \) outpt s' \( p = \) chosen s' \( \lor \) outpt s' \( p = \) NotAnInput
  by (auto simp add: Hinv6-def)
with t1
show \( ? \)thesis
  by (simp add: Hinv6-def)
qed

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

theorem HEndPhase1-Inv6:
  assumes act: HEndPhase1 s s' p
  and inv: HInv6 s
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  shows HInv6 s'
proof –
from outpt-chosen act inv2c inv
have chosen s' \( \neq \) NotAnInput \( \rightarrow \) valueChosen s (chosen s')
  by (auto simp add: EndPhase1-def Hinv6-def)
from HEndPhase1-valueChosen[OF act] inv1 inv2a inv2b this InputsOrNi
have t1: chosen s' \( \neq \) NotAnInput \( \rightarrow \) valueChosen s' (chosen s')
  by auto
from act
have outpt: outpt s = outpt s'
  by (auto simp add: EndPhase1-def)
from outpt-Inv6[OF outpt] act inv2c inv
have \( \forall p. \) outpt s' \( p = \) chosen s' \( \lor \) outpt s' \( p = \) NotAnInput
  by (auto simp add: Hinv6-def)
with t1
show \( ? \)thesis
  by (simp add: Hinv6-def)
qed

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

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

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

theorem HPhase0Read-Inv6:
assumes act: HPhase0Read s s' p d
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'

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

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

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

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

lemma I2f:
assumes \textit{nxt}: \textit{HNext} \( s \ s' \)
and \textit{inv}: \textit{HInv1} \( s \wedge \textit{HInv2} \) \( s \wedge \textit{HInv3} \) \( s \wedge \textit{HInv4} \) \( s \wedge \textit{HInv5} \) \( s \wedge \textit{HInv6} \) \( s \)
shows \( \textit{HInv6} \) \( s' \) using \textit{assms}
by\((\text{auto simp add: \textit{HNext-def} \textit{Next-def},}
\text{auto simp add: \textit{HInv2-def} intro: \textit{HStartBallot-Inv6},}
\text{auto simp add: \textit{Phase1or2Read-def} intro: \textit{HPhase1or2ReadThen-Inv6}
\textit{HPhase1or2ReadElse-Inv6},}
\text{auto simp add: \textit{HInv4-def} intro: \textit{HPhase1or2Write-Inv6},}
\text{auto simp add: \textit{Phase1or2Read-def intro: \textit{HPhase1or2ReadThen-Inv6}
\textit{HPhase1or2ReadElse-Inv6},}
\text{auto simp add: \textit{Phase1or2Read-def} \textit{HInv1-def} \textit{HInv5-def intro: \textit{HEndPhase1-Inv6}
\textit{HEndPhase2-Inv6},}
\text{auto intro: \textit{HEndPhase0-Inv6}})\)
end

theory \textit{DiskPaxos-Invariant} imports \textit{DiskPaxos-Inv6} begin

C.8 \textbf{The Complete Invariant}

definition \textit{HInv} :: state \Rightarrow bool
where
\[ \textit{HInv} \ s = (\textit{HInv1} \ s \wedge \textit{HInv2} \ s \wedge \textit{HInv3} \ s \wedge \textit{HInv4} \ s \wedge \textit{HInv5} \ s \wedge \textit{HInv6} \ s) \]

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

theorem \textit{I2}:
assumes \textit{inv}: \textit{HInv} \( s \)
and \textit{nxt}: \textit{HNext} \( s \ s' \)
shows \textit{HInv} \( s' \)
using \textit{inv} \( l2a[\text{OF } \textit{nxt} \] \( l2b[\text{OF } \textit{nxt} \] \( l2c[\text{OF } \textit{nxt} \] \( l2d[\text{OF } \textit{nxt} \] \( l2e[\text{OF } \textit{nxt} \] \( l2f[\text{OF } \textit{nxt} \]
by\((\text{simp add: \textit{HInv-def}})\)
end
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

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

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

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

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

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

theorem R1: 
[ IInit s; is = s2is s ] ⇒ IInit is
theorem R2b:
assumes inv: HInv s
and inv': HInv s'
and nxt: HNext s s'
and srel: is = s2is s ∧ is' = s2is s'
shows (∃ p. IFail is is' p ∨ IChoose is is' p) ∨ is = is'
proof (auto)
  assume chg-vars: is ≠ is'
  with srel
  have s-change: inpt s ≠ inpt s' ∨ outpt s ≠ outpt s'
           ∨ chosen s ≠ chosen s' ∨ allInput s ≠ allInput s'
    by (auto simp add: s2is-def)
  from inv
  have inv2c5: ∀ p. inpt s p ∈ allInput s
       ∧ (chosen s = NotAnInput ⇒ outpt s p = NotAnInput)
    by (auto simp add: HInv-def HInv2-def Inv2c-def Inv2c-inner-def)
  from nxt s-change inv2c5
  have inpt s' ≠ inpt s ∨ outpt s' ≠ outpt s
    by (auto simp add: HNext-def HNextPart-def)
  with nxt
  have ∃ p. Fail s s' p ∨ EndPhase2 s s' p
    by (auto simp add: HNext-def HNextPart-def
      StartBallot-def Phase0Read-def Phase1or2Write-def
      Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def
      EndPhase1or2-def EndPhase1-def EndPhase0-def)
then obtain p where fail-or-endphase2: Fail s s' p ∨ EndPhase2 s s' p
by auto
from inv
have inv2c: Inv2c-inner s p
  by (auto simp add: HInv-def HInv2-def Inv2c-def)
from fail-or-endphase2 have IFail is is' p ∨ IChoose is is' p
proof
  assume fail: Fail s s' p
  hence phase': phase s' p = 0
  and outpt: outpt s' = (outpt s) (p := NotAnInput)
      by (auto simp add: Fail-def)
  have IFail is is' p
proof –
  from fail srel
  have ioutput is' = (ioutput is) (p := NotAnInput)
        by (auto simp add: Fail-def s2is-def)
  moreover
  from nxt
  have all-nxt: allInput s' = allInput s ∪ (range (inpt s'))
      by (auto simp add: HNext-def HNextPart-def)
  from fail srel
  have ∃ ip ∈ Inputs. iinput is' = (iinput is) (p := ip)

by(auto simp add: Fail-def s2is-def)
then obtain ip where ip-Input: ip∈Inputs and iinput is′ = (iinput is)(p:=
ip)
  by auto
with inv2c5 srel all-nxt
have 'iinput is′ = (iinput is)(p:= ip)
  ∧ iallInput is′ = iallInput is ∪ {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) ≠ 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) ∈ allInput s
  by(auto simp add: Inv2a-innermost-def)
from inv
have allInput s ⊆ Inputs
  by(auto simp add: HInv-def HInv1-def)
with p22 NotAnInput endphase2
have outpt-nni: outpt s′ p ≠ NotAnInput
  by(auto simp add: EndPhase2-def)
show ?thesis
proof(cases chosen s = NotAnInput)
case True
with inv2c5
have p31: ∀ q. outpt s q = NotAnInput
  by auto
with endphase2
have p32: ∀ q ∈ UNIV −{p}. outpt s′ q = NotAnInput
  by(auto simp add: EndPhase2-def)
hence some-eq: (\( \forall x. \text{outpt } s' \neq \text{NotAnInput} \implies x = p \))
  by auto
from p32 True nxt some-equality[of \( \lambda p. \text{outpt } s' \neq \text{NotAnInput} \), OF outpt-nni some-eq]
  have p33: chosen \( s' = \text{outpt } s' \) by (auto simp add: HNext-def HNextPart-def)
  with endphase2
  have chosen \( s' = \text{inp(dblock s p)} \land \text{outpt } s' = (\text{outpt } s)(p:=\text{inp(dblock s p)}) \)
    by (auto simp add: EndPhase2-def)
  with True p22
  have if (chosen \( s = \text{NotAnInput} \))
    then (\( \exists ip \in \text{allInput } s \). chosen \( s' = ip \)
      \land \text{outpt } s' = (\text{outpt } s)(p := ip) \))
    else ( \text{outpt } s' = (\text{outpt } s)(p:= \text{chosen } s) 
      \land \text{chosen } s' = \text{chosen } s \)
      by auto
moreover
from endphase2 inv2c5 nxt
  have inpt \( s' = \text{inpt } s \land \text{allInput } s' = \text{allInput } s \)
    by (auto simp add: EndPhase2-def HNext-def HNextPart-def)
ultimately
  show ?thesis
    using srel p31
    by (auto simp add: IChoose-def s2is-def)
next
  case False
  with nxt
  have p31: chosen \( s' = \text{chosen } s \)
    by (auto simp add: HNext-def HNextPart-def)
from inv'
  have inv6: HInv6 \( s' \)
    by (auto simp add: HInv-def)
  have p32: \text{outpt } s' p = \text{chosen } s
    proof
      from endphase2
      have \text{outpt } s' p = \text{inp(dblock s p)}
        by (auto simp add: EndPhase2-def)
      moreover
      from inv6 p31
      have \text{outpt } s' p \in \{ \text{chosen } s, \text{NotAnInput} \}
        by (auto simp add: HInv6-def)
      ultimately
      show ?thesis
        using outpt-nni
        by auto
    qed
  from srel False
  have IChoose is is' p
    proof(clarsimp simp add: IChoose-def s2is-def)
from endphase2 inv2c
have outpt s p = NotAnInput
  by (auto simp add: EndPhase2-def Inv2c-inner-def)
moreover
from endphase2 p31 p32 False
have outpt s' = (outpt s) (p:= chosen s) ∧ chosen s' = chosen s
  by (auto simp add: EndPhase2-def)
moreover
from endphase2 nxt inv2c5
have inpt s' = inpt s ∧ allInput s' = allInput s
  by (auto simp add: EndPhase2-def HNext-def HNextPart-def)
ultimately
show outpt s p = NotAnInput
  ∧ outpt s' = (outpt s)(p := chosen s) ∧ chosen s' = chosen s
  ∧ inpt s' = inpt s ∧ allInput s' = allInput s
  by auto
qed
thus ?thesis
  by auto
qed
thus ∃p. IFail is is' p ∨ IChoose is is' p
  by auto
qed
end