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.
C Proof of Disk Paxos’ Invariant 19

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Invariant 1</td>
<td>19</td>
</tr>
<tr>
<td>C.2</td>
<td>Invariant 2</td>
<td>23</td>
</tr>
<tr>
<td>C.2.1</td>
<td>Proofs of Invariant 2 a</td>
<td>25</td>
</tr>
<tr>
<td>C.2.2</td>
<td>Proofs of Invariant 2 b</td>
<td>37</td>
</tr>
<tr>
<td>C.2.3</td>
<td>Proofs of Invariant 2 c</td>
<td>38</td>
</tr>
<tr>
<td>C.3</td>
<td>Invariant 3</td>
<td>46</td>
</tr>
<tr>
<td>C.3.1</td>
<td>Proofs of Invariant 3</td>
<td>47</td>
</tr>
<tr>
<td>C.4</td>
<td>Invariant 4</td>
<td>55</td>
</tr>
<tr>
<td>C.4.1</td>
<td>Proofs of Invariant 4a</td>
<td>56</td>
</tr>
<tr>
<td>C.4.2</td>
<td>Proofs of Invariant 4b</td>
<td>67</td>
</tr>
<tr>
<td>C.4.3</td>
<td>Proofs of Invariant 4c</td>
<td>76</td>
</tr>
<tr>
<td>C.4.4</td>
<td>Proofs of Invariant 4d</td>
<td>81</td>
</tr>
<tr>
<td>C.5</td>
<td>Invariant 5</td>
<td>91</td>
</tr>
<tr>
<td>C.5.1</td>
<td>Proof of Invariant 5</td>
<td>92</td>
</tr>
<tr>
<td>C.6</td>
<td>Lemma I2f</td>
<td>115</td>
</tr>
<tr>
<td>C.7</td>
<td>Invariant 6</td>
<td>134</td>
</tr>
<tr>
<td>C.8</td>
<td>The Complete Invariant</td>
<td>148</td>
</tr>
<tr>
<td>C.9</td>
<td>Inner Module</td>
<td>149</td>
</tr>
</tbody>
</table>

1 Introduction

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

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

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

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

2 The Disk Paxos Algorithm

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

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

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

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

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

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

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

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

In each block, processors maintain three values:

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

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

2.2 Disk Paxos and its TLA+ Specification

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

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

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

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

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

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

Analogously, we have the specification of the Inner module:

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

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

**THEOREM R1**  \(HInit \Rightarrow IInit\)

**THEOREM R2**  \(HInit \land \square[HNext]_{\langle\text{vars},\text{chosen},\text{allInput}\rangle} \Rightarrow \square[INext]_{ivars}\)

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**  \(HInit \land \square[HNext]_{\langle\text{vars},\text{chosen},\text{allInput}\rangle} \Rightarrow \square[HInv]\)

**THEOREM R2b**  \(HInv \land HInv' \land HNext \Rightarrow INext \lor (\text{UNCHANGED} \ ivars)\)

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

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

\[
\text{choose } \ v x. P x \epsilon x. P x
\]

\[
\text{phase}' = [\text{phase except } !p = 1]
\]

\[
\text{UN } p. \text{blocksOf } s p
\]

\[
\text{UN}\ p.\ \text{blocksOf}\ s\ p
\]

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

\[
\text{UNCHANGED } v
\]

\[
v s' = v s
\]

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

\textbf{Theorem I1} \quad HInit ⇒ HInv

\textbf{Theorem I2} \quad HInv \land HNext ⇒ HInv'

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

Finding a predicate HInv that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present HInv as a conjunction of 6 predicates HInv1, ..., HInv6, where HInv1 is a simple “type invariant” and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of HInv\(i\) by the algorithm’s next-state relation relies on all HInv\(j\) (for \(j \leq i\)) 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.

\section{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\(^1\).

\subsection{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**

\[ \text{NotAnInput} \triangleq \text{CHOOSE } c : c \notin \text{Inputs} \]

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

Isabelle/HOL:

**typedef** InputsOrNi

**consts**

\[ \text{Inputs} :: \text{InputsOrNi set} \]
\[ \text{NotAnInput} :: \text{InputsOrNi} \]

**axioms**

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

**record**

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

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

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

For example, see figure 2. The type 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 \cup \{NotAnInput\}, as nothing prevents the InputsOrNi type from having more values. Consequently, we add the axiom InputsOrNi to establish that the only values of this type are the ones in Inputs and NotAnInput.

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

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

Isabelle/HOL:

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

Figure 3: Translation of an action

3.2 Primed Variables

In TLA⁺, to denote the value of a variable in the state resulting from an action, there exists the concept of primed variables. In Isabelle/HOL, there is no built-in notion of state, so we define the state as a record of the variables involved. Instead of defining a “priming” operator, we just define actions as predicates that take any two states, intended to be the previous state and the next state. In this way, \( P s s' \) will be true iff executing an action \( P \) in the \( s \) state could result in the \( s' \) state. In figure 3 we can see how the action \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 \text{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, Phase1or2Read is mainly a big if-then-else. We break it down into two simpler actions:

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

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

Another example is HInv2, which we break down into:

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

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

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

we write:

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

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

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

Now we can express that we want to obtain the fact

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

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

4 Structure of the Correctness Proof

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

There are informal proofs for invariants $HInv3$-$HInv6$ and for theorem $R2b$ in [GL00]. These informal proofs are written in the structured-proof style that Lamport advocates [Lam95], and are rigorous and quite detailed. We based our formal proofs on these informal proofs, but in many cases a higher level of detail was needed. In some cases, the steps where too big for Isabelle to solve them automatically, and intermediate steps had to be proved first; in other cases, some of the facts relevant to the proofs were omitted in the informal proofs.

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv4$ and $HInv5$ 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 $HInv3$ for the $EndPhase0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

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

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

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

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

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

References


A TLA+ correctness specification

---

**MODULE Synod**

EXTENDS Naturals

CONSTANT N, Inputs

ASSUME (N ∈ Nat) ∧ (N > 0)

\[ \text{Proc} \triangleq 1..N \]

\[ \text{NotAnInput} \triangleq \text{choose } c : c \notin \text{Inputs} \]

VARIABLES inputs, output

---

**MODULE Inner**

VARIABLES allInput, chosen

\[ \text{IInit} \triangleq \quad \land \text{input} \in [\text{Proc} \rightarrow \text{Inputs}]
\land \text{output} = [p \in \text{Proc} \mapsto \text{NotAnInput}]
\land \text{chosen} = \text{NotAnInput}
\land \text{allInput} = \text{input}[p] : p \in \text{Proc} \]

\[ \text{IChoose}(p) \triangleq \quad \land \text{output}[p] = \text{NotAnInput}
\land \text{if } \text{chosen} = \text{NotAnInput}
\text{then } \land \text{ip} \in \text{allInput} : \land \text{chosen}' = \text{ip}
\land \text{output}' = [\text{output} \text{ except } ![p] = \text{ip}]
\text{else } \land \text{output}' = [\text{output} \text{ except } ![p] = \text{chosen}]
\land \text{UNCHANGED chosen}
\land \text{UNCHANGED } \langle \text{input}, \text{allInput} \rangle \]

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

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

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

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

\[ \text{SynodSpec} \triangleq \exists \text{chosen, allInput} : \text{IS}(\text{chosen, allInput})!\text{ISpec} \]
B Disk Paxos Algorithm Specification

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedecl InputsOrNi
typedecl Disk
typedecl Proc

axiomatization

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

where

NotAnInput: NotAnInput \notin Inputs and
InputsOrNi: (UNIV :: InputsOrNi set) = Inputs \cup \{NotAnInput\} and
Ballot-nzero: \forall p. 0 \notin Ballot p and
Ballot-disj: \forall p q. p \neq q \rightarrow (Ballot p) \cap (Ballot q) = {} and
Disk-isMajority: IsMajority(UNIV) and

majorities-intersect:
\forall S T. IsMajority(S) \land IsMajority(T) \rightarrow S \cap T \neq {}

lemma ballots-not-zero [simp]:
\( b \in \text{Ballot} p \Rightarrow 0 < b \)

proof (rule ccontr)
assume b: \( b \in \text{Ballot} p \)
and contr: \( \neg (0 < b) \)
from Ballot-nzero
have 0 \notin Ballot p ..
with b contr
show False
  by auto
qed

lemma majority-nonempty [simp]: IsMajority(S) \implies S \neq {}

proof(auto)
from majorities-intersect
have IsMajority(\{\}) \land IsMajority(\{\}) \rightarrow \{\} \cap \{\} \neq {}
  by auto
thus IsMajority \{\} \implies 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
\[
\begin{align*}
\text{(disksWritten } s' &= \text{ (disksWritten } s)(p \coloneqq \{\}) \\
\& \text{ blocksRead } s' &= \text{ (blocksRead } s)(p \coloneqq (\lambda d. \{\}))
\end{align*}
\]

**definition** StartBallot :: state ⇒ state ⇒ Proc ⇒ bool

**where**

\begin{align*}
\text{StartBallot } s \ s' \ p = \\
(\text{phase } s \ p \in \{1, 2\}) \\
\& \text{ phase } s' = (\text{phase } s)(p \coloneqq 1) \\
\& (\exists b \in \text{Ballot } p. \\
\quad mbal(\text{dblock } s \ p) < b \\
\& \text{ dblock } s' = (\text{dblock } s)(p \coloneqq (\text{dblock } s \ p)(mbal \coloneqq 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*}

**definition** Phase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool

**where**

\begin{align*}
\text{Phase1or2Write } s \ s' \ p \ d = \\
(\text{phase } s \ p \in \{1, 2\}) \\
\& \text{ disk } s' = (\text{disk } s)(d \coloneqq (\text{disk } s \ d)(p \coloneqq (\text{dblock } s \ p))) \\
\& \text{ disksWritten } s' = (\text{disksWritten } s)(p \coloneqq (\text{disksWritten } s \ p) \cup \{d\}) \\
\& \text{ inpt } s' = \text{inpt } s \& \text{ outpt } s' = \text{outpt } s \\
\& \text{ phase } s' = \text{phase } s \& \text{ dblock } s' = \text{dblock } s \\
\& \text{ blocksRead } s' = \text{blocksRead } s)
\end{align*}

**definition** Phase1or2ReadThen :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool

**where**

\begin{align*}
\text{Phase1or2ReadThen } s \ s' \ p \ d \ q = \\
(d \in \text{disksWritten } s \ p) \\
\& \text{ mbal}(\text{disk } s \ d \ q) < \text{mbal}(\text{dblock } s \ p) \\
\& \text{ blocksRead } s' = (\text{blocksRead } s)(p \coloneqq (\text{blocksRead } s \ p)(d \coloneqq \\
\quad (\text{blocksRead } s \ p \ d) \cup \{d \coloneqq \text{block} = \text{disk } s \ d \ q, \\
\quad \text{proc} = q \ |})) \\
\& \text{ inpt } s' = \text{inpt } s \& \text{ outpt } s' = \text{outpt } s \\
\& \text{ disk } s' = \text{disk } s \& \text{ phase } s' = \text{phase } s \\
\& \text{ dblock } s' = \text{dblock } s \& \text{ disksWritten } s' = \text{disksWritten } s)
\end{align*}

**definition** Phase1or2ReadElse :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool

**where**

\begin{align*}
\text{Phase1or2ReadElse } s \ s' \ p \ d \ q = \\
(d \in \text{disksWritten } s \ p) \\
\& \text{ StartBallot } s \ s' \ p)
\end{align*}

**definition** Phase1or2Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool

**where**

\begin{align*}
\text{Phase1or2Read } s \ s' \ p \ d \ q = \\
(\text{Phase1or2ReadThen } s \ s' \ p \ d \ q \\
\lor \text{Phase1or2ReadElse } s \ s' \ p \ d \ q)
\end{align*}

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

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

definition maxBlk :: state ⇒ Proc ⇒ DiskBlock
where
\[
\text{maxBlk} \ s \ p = \\
(SOME \ b . \ b \ ∈ \text{nonInitBlks} \ s \ p \land (\forall \ c \ ∈ \text{nonInitBlks} \ s \ p . \ bal \ c \ ≤ \ bal \ b))
\]
definition EndPhase1 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase1 \ s \ s' \ p = \\
(\text{IsMajority} \ \{d . d ∈ \text{disksWritten} \ s \ p \land (\forall q ∈ \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) \\
\{ bal := \text{mbal} (\text{dblock} \ s \ p), \\
\text{inp} := \\
\{ \text{if nonInitBlks} \ s \ p = \{} \\
\text{then inp} \ s \ p \\
\text{else inp} \ (\text{maxBlk} \ s \ p) \} \\
\} \\
\land \text{outpt} \ s' = \text{outpt} \ s \\
\land \text{phase} \ s' = (\text{phase} \ s) \ (p := \text{phase} \ s \ p + 1) \\
\land \text{InitializePhase} \ s \ s' \ p \\
\land \text{inpt} \ s' = \text{inpt} \ s \land \text{disk} \ s' = \text{disk} \ s 

definition EndPhase2 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase2 \ s \ s' \ p = \\
(\text{IsMajority} \ \{d . d ∈ \text{disksWritten} \ s \ p \land (\forall q ∈ \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 EndPhase1or2 :: state ⇒ state ⇒ Proc ⇒ bool
where
EndPhase1or2 \ s \ s' \ p = (\text{EndPhase1} \ s \ s' \ p \lor \text{EndPhase2} \ s \ s' \ p) 

definition Fail :: state ⇒ state ⇒ Proc ⇒ bool
where
Fail \ s \ s' \ p = \\
(\exists ip ∈ \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})
\[ \text{outpt} \ s' = (\text{outpt} \ s) \ (p := \text{NotAnInput}) \]

\[ \text{InitializePhase} \ s \ s' \ p \]

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

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

**where**

\[ \text{Phase0Read} \ s \ s' \ p \ d = \]

\[ (\text{phase} \ s \ p = 0) \]

\[ \land \text{blocksRead} \ s' = (\text{blocksRead} \ s) \ (p := (\text{blocksRead} \ s \ p) \ (d := \text{blocksRead} \ s \ p \ d) \ \cup \ \{ \{ \text{block} = \text{disk} \ s \ d \ p, \ \text{proc} = p \ [] \} \}) \]

\[ \land \text{inpt} \ s' = \text{inpt} \ s \ \land \ \text{outpt} \ s' = \text{outpt} \ s \]

\[ \land \text{disk} \ s' = \text{disk} \ s \ \land \ \text{phase} \ s' = \text{phase} \ s \]

\[ \land \text{dblock} \ s' = \text{dblock} \ s \ \land \ \text{disksWritten} \ s' = \text{disksWritten} \ s \]

**definition** EndPhase0 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool

**where**

\[ \text{EndPhase0} \ s \ s' \ p = \]

\[ (\text{phase} \ s \ p = 0) \]

\[ \land \text{IsMajority} \ (\{ d. \ \text{hasRead} \ s \ p \ d \}) \]

\[ \land (\exists b \in \text{Ballot} \ p. \]

\[ (\forall r \in \text{allBlocksRead} \ s \ p. \ \text{mbal} \ r < b) \]

\[ \land \text{dblock} \ s' = (\text{dblock} \ s) \ (p:= \]

\[ \text{SOME} \ r. \ r \in \text{allBlocksRead} \ s \ p \]

\[ \land (\forall s \in \text{allBlocksRead} \ s \ p. \ \text{bal} \ s \leq \ \text{bal} \ r) \ (\| \text{mbal} := b \ |) \}) \]

\[ \land \text{InitializePhase} \ s \ s' \]

\[ \land \text{inpt} \ s' = \text{inpt} \ s \ \land \ \text{outpt} \ s' = \text{outpt} \ s \ \land \ \text{disk} \ s' = \text{disk} \ s \]

**definition** Next :: state \(\Rightarrow\) state \(\Rightarrow\) bool

**where**

\[ \text{Next} \ s \ s' = (\exists p. \]

\[ \text{StartBallot} \ s \ s' \ p \]

\[ \lor (\exists d. \ \text{Phase0Read} \ s \ s' \ p \ d) \]

\[ \lor \text{Phase1or2Write} \ s \ s' \ p \ d \]

\[ \lor (\exists q. \ q \neq p \ \land \ \text{Phase1or2Read} \ s \ s' \ p \ d \ q)) \]

\[ \lor \text{EndPhase1or2} \ s \ s' \ p \]

\[ \lor \text{Fail} \ s \ s' \ p \]

\[ \lor \text{EndPhase0} \ s \ s' \ p \]

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

**definition** HInit :: state \(\Rightarrow\) bool

**where**

\[ \text{HInit} \ s = \]

\[ (\text{Init} \ s \]

\[ & \ \text{chosen} \ s = \text{NotAnInput} \]

\[ & \ \text{allInput} \ s = \text{range} \ (\text{inpt} \ s)) \]

17
HNextPart is the part of the Next action that is concerned with history variables.

definition HNextPart :: state ⇒ state ⇒ bool
where
HNextPart s s′ =
  (chosen s′ =
    (if chosen s ≠ NotAnInput ∨ (∀ p. outpt s′ p = NotAnInput)
    then chosen s
    else outpt s′ (SOME p. outpt s′ p ≠ NotAnInput))
∧ allInput s′ = allInput s ∪ (range (inpt s′)))

definition HNext :: state ⇒ state ⇒ bool
where
HNext s s′ =
  (Next s s′
∧ HNextPart s s′)

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

definition HPhase1or2ReadThen :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool where
HPhase1or2ReadThen s s′ p d q = (Phase1or2ReadThen s s′ p d q ∧ HNextPart s s′)

definition HEndPhase1 :: state ⇒ state ⇒ Proc ⇒ bool where
HEndPhase1 s s′ p = (EndPhase1 s s′ p ∧ HNextPart s s′)

definition HStartBallot :: state ⇒ state ⇒ Proc ⇒ bool where
HStartBallot s s′ p = (StartBallot s s′ p ∧ HNextPart s s′)

definition HPhase1or2Write :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool where
HPhase1or2Write s s′ p d = (Phase1or2Write s s′ p d ∧ HNextPart s s′)

definition HPhase1or2ReadElse :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ Proc ⇒ bool where
HPhase1or2ReadElse s s′ p d q = (Phase1or2ReadElse s s′ p d q ∧ HNextPart s s′)

definition HEndPhase2 :: state ⇒ state ⇒ Proc ⇒ bool where
HEndPhase2 s s′ p = (EndPhase2 s s′ p ∧ HNextPart s s′)

definition HFail :: state ⇒ state ⇒ Proc ⇒ bool where
HFail s s′ p = (Fail s s′ p ∧ HNextPart s s′)
definition
HPhase0Read :: state ⇒ state ⇒ Proc ⇒ Disk ⇒ bool where
HPhase0Read s s′ p d = (Phase0Read s s′ p d ∧ HNextPart s s′)

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

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

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

end

C  Proof of Disk Paxos’ Invariant

theory DiskPaxos-Inv1 imports DiskPaxos-Model begin

C.1  Invariant 1

This is just a type Invariant.

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

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

declare HInv1-def [simp]

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

**lemma** \( H_{\text{NextPart-Inv1}}: [ H_{\text{Inv1}} s; H_{\text{NextPart}} s s' ; \text{Inv1} s'] \rightarrow H_{\text{Inv1}} s' \)

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

**theorem** \( H_{\text{Init-HInv1}}: H_{\text{Init}} s \rightarrow H_{\text{Inv1}} s \)

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

**lemma** allRdBlks-finite:
- assumes inv: \( H_{\text{Inv1}} s \)
- and asm: \( \forall p. \text{allRdBlks} s p' \subseteq \text{insert} bk (\text{allRdBlks} s p) \)
- shows \( \forall p. \text{finite} (\text{allRdBlks} s p') \)

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

**qed**

**theorem** \( H_{\text{Phase1or2ReadThen-HInv1}}: \)
- assumes inv1: \( H_{\text{Inv1}} s \)
- and act: \( H_{\text{Phase1or2ReadThen}} s s' p d q \)
- shows \( H_{\text{Inv1}} s' \)

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

**qed**

**theorem** \( H_{\text{EndPhase1-HInv1}}: \)
- assumes inv1: \( H_{\text{Inv1}} s \)
- and act: \( H_{\text{EndPhase1}} s s' p \)
- shows \( H_{\text{Inv1}} s' \)

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

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

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

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

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

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

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

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

qed

declare \texttt{HInv1-def [simp del]}

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

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

end

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

\textbf{C.2 Invariant 2}

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

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

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

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

definition \text{Inv2a-innermost} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \Rightarrow \text{bool} \\
\text{where} \\
\text{Inv2a-innermost} s p bk = \\
(\text{mbal} bk \in (\text{Ballot} p) \cup \{0\}) \\
\land (\text{bal} bk \in (\text{Ballot} p) \cup \{0\}) \\
\land (\text{bal} bk = 0) = (\text{inp} bk = \text{NotAnInput}) \\
\land \text{bal} bk \leq \text{mbal} bk \\
\land \text{inp} bk \in (\text{allInput} s) \cup \{\text{NotAnInput}\})

definition \text{Inv2a-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \text{Inv2a-inner} s p = (\forall bk \in \text{blocksOf} s p. \text{Inv2a-innermost} s p bk)

definition \text{Inv2a} :: \text{state} \Rightarrow \text{bool} \\
\text{where} \text{Inv2a} s = (\forall p. \text{Inv2a-inner} s p)

definition \text{Inv2b-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \\
\text{where} \\
\text{Inv2b-inner} s p d = \\
((d \in \text{disksWritten} s p \rightarrow \\
(\text{phase} s p \in \{1,2\} \land \text{disk} s d p = \text{dblock} s p)) \\
\land (\text{phase} s p \in \{1,2\} \rightarrow \\
(\text{blocksRead} s p d \neq \{} \rightarrow d \in \text{disksWritten} s p) \\
\land \neg \text{hasRead} s p d p))

definition \text{Inv2b} :: \text{state} \Rightarrow \text{bool} \\
\text{where} \text{Inv2b} s = (\forall p d. \text{Inv2b-inner} s p d)

definition \text{Inv2c-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\text{where} \\
\text{Inv2c-inner} s p = \\
((\text{phase} s p = 0 \rightarrow \\
(\text{dblock} s p = \text{InitDB}) \\
\land \text{disksWritten} s p = \{\} \\
\land (\forall d. \forall br \in \text{blocksRead} s p d. \\
\text{proc} br = p \land \text{block} br = \text{disk} s d p)) \\
\land (\text{phase} s p \neq 0 \rightarrow \\
(\text{mbal} (\text{dblock} s p) \in \text{Ballot} p \\
\land \text{bal} (\text{dblock} s p) \in \text{Ballot} p \cup \{0\}) \\
\land (\forall d. \forall br \in \text{blocksRead} s p d. \\
\text{mbal} (\text{block} br) < \text{mbal} (\text{dblock} s p))) \\
\land (\text{phase} s p \in \{2,3\} \rightarrow \text{bal} (\text{dblock} s p) = \text{mbal} (\text{dblock} s p)) \\
\land \text{outpt} s p = (\text{if phase} s p = 3 \text{ then inp} (\text{dblock} s p) \text{ else NotAnInput}) \\
\land \text{chosen} s \in \text{allInput} s \cup \{\text{NotAnInput}\} \\
\land (\forall p. \text{inpt} s p \in \text{allInput} s \\
\land (\text{chosen} s = \text{NotAnInput} \rightarrow \text{outpt} s p = \text{NotAnInput})))

definition \text{Inv2c} :: \text{state} \Rightarrow \text{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** \( H\text{Init-Inv2a}: \) \( H\text{Init} \ s \longrightarrow \text{Inv2a} \ s \)
by (auto simp add: \( H\text{Init-def} \) \( Init-def \) \( \text{Inv2a-def} \) \( \text{Inv2a-inner-def} \) \( \text{rdBy-def} \) \( \text{blocksOf-def} \) \( \text{InitDB-def} \))

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

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

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

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

**lemma** \( H\text{StartBallot-Inv2a-dblock}: \)
- assumes \( \text{act} \): \( H\text{StartBallot} \ s \ s' p \)
- and \( \text{inv2a} \): \( \text{Inv2a-innermost} \ s p \) \( \text{(dblock} \ s \ p) \)
shows \textit{Inv2a-innermost} \(s' \, p \) (\textit{dblock} \(s' \, p \))

\textbf{proof} –

\textbf{from} act

\textbf{have} \(mbal': \, mbal \ (\textit{dblock} \, s' \, p \) \in \textit{Ballot} \, p \)

\textbf{by}(auto \, simp \, add: \textit{StartBallot-def})

\textbf{from} act

\textbf{have} \(bal': \, bal \ (\textit{dblock} \, s' \, p \) = bal \ (\textit{dblock} \, s \, p \)

\textbf{by}(auto \, simp \, add: \textit{StartBallot-def})

\textbf{with} act

\textbf{have} \(inp': \, inp \ (\textit{dblock} \, s' \, p \) = inp \ (\textit{dblock} \, s \, p \)

\textbf{by}(auto \, simp \, add: \textit{StartBallot-def})

\textbf{from} act

\textbf{have} \(mbal \ (\textit{dblock} \, s \, p \) \leq \, mbal \ (\textit{dblock} \, s' \, p \)

\textbf{by}(auto \, simp \, add: \textit{Inv2a-innermost-def})

\textbf{with} \textit{bal}' \, \textit{inv2a}

\textbf{have} \(bal-mbal: \, bal \ (\textit{dblock} \, s' \, p \) \leq \, mbal \ (\textit{dblock} \, s' \, p \)

\textbf{by}(auto \, simp \, add: \textit{Inv2a-innermost-def})

\textbf{from} act

\textbf{have} \(allInput \, s \subseteq allInput \, s' \)

\textbf{by}(auto \, simp \, add: \textit{HNextPart-def} \, \textit{InitializePhase-def} \, \textit{Inv2a-innermost-def})

\textbf{with} \textit{mbal}' \, \textit{bal}' \, \textit{inp}' \, \textit{bal-mbal} \, \textit{act} \, \textit{inv2a}

\textbf{show} \, ?\textit{thesis}

\textbf{by}(auto \, simp \, add: \textit{Inv2a-innermost-def})

\textbf{qed}

\textbf{lemma} \textit{HStartBallot-Inv2a-dblock-q}: 

\textbf{assumes} act: \textit{HStartBallot} \, s \, s' \, p 

\textbf{and} \textit{inv2a}: \textit{Inv2a-innermost} \, s \, q \ (\textit{dblock} \, s \, q) 

\textbf{shows} \textit{Inv2a-innermost} \, s' \, q \ (\textit{dblock} \, s' \, q)

\textbf{proof}(cases \, p=q)

\textbf{case} True

\textbf{with} act \, \textit{inv2a}

\textbf{show} \, ?\textit{thesis}

\textbf{by}(blast \, dest: \textit{HStartBallot-Inv2a-dblock})

\textbf{next}

\textbf{case} False

\textbf{with} act \, \textit{inv2a}

\textbf{show} \, ?\textit{thesis}

\textbf{by}(clarsimp \, simp \, add: \textit{StartBallot-def} \, \textit{HNextPart-def} \, \textit{InitializePhase-def} \, \textit{Inv2a-innermost-def})

\textbf{qed}

\textbf{theorem} \textit{HStartBallot-Inv2a}: 

\textbf{assumes} inv: \textit{Inv2a} \, s 

\textbf{and} act: \textit{HStartBallot} \, s \, s' \, p 

\textbf{shows} \textit{Inv2a} \, s'

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

\textbf{fix} \, q \, bk

\textbf{assume} \, bk: \, bk \in \textit{blocksOf} \, s' \, q

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

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

theorem HPhase1or2Write-Inv2a:
  assumes inv: Inv2a s
  and act: HPhase1or2Write s s’ p d
  shows Inv2a s’
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ blocksOf s’ q
  from inv bk HPhase1or2Write-blocksOf[OF act]
  have inp-q-bk: Inv2a-innermost s q bk
    by (auto simp add: Inv2a-def Inv2a-inner-def)
  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 blocksOf s' q \subseteq blocksOf s q
by (auto simp add: EndPhase2-def blocksOf-def
    dest: subsetD[OF InitializePhase-rdBy])

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

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

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

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

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

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

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

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

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

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

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

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

lemma \text{HEndPhase0-Inv2a-allBlocksRead}:  
assumes act: \text{HEndPhase0 } s s' p
and inv2a: Inv2a-inner s p
and inv2c: Inv2c-inner s p
shows \( \forall t \in (\lambda x. \{mbal:= mbal(dblock s' p)\}) \,' \text{allBlocksRead } s p. \)  
Inv2a-innermost s p t

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

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

lemma HEndPhase0-Inv2a-dblock:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a-inner s p
and inv2c: Inv2c-inner s p
shows Inv2a-innermost s' p (dblock s' p)
proof –
from act inv2a inv2c
have t1: \( \forall t \in (\lambda x. x \{(\text{mbal:= mbal (dblock s' p)})\} \land \text{allBlocksRead s p}. \text{Inv2a-innermost s p t} \)
by(blast dest: HEndPhase0-Inv2a-allBlocksRead)
from act inv1
have dblock s' p \in (\lambda x. x \{(\text{mbal:= mbal (dblock s' p)})\} \land \text{allBlocksRead s p}
by(simp, blast dest: HEndPhase0-dblock-allBlocksRead)
with t1
have inv2-dblock: Inv2a-innermost s p (dblock s' p) by auto
with act
have inp (dblock s' p) \in allInput s' \cup \{\text{NotAnInput}\}
by(auto dest: HNextPart-allInput-or-NotAnInput)
with inv2-dblock
show \?thesis
by(auto simp add: Inv2a-innermost-def)
qed

lemma HEndPhase0-Inv2a-dblock-q:
assumes act: HEndPhase0 s s' p
and inv1: Inv1 s
and inv2a: Inv2a-inner s q
and inv2c: Inv2c-inner s p
shows Inv2a-innermost s’ q (dblock s’ q)

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

qed

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

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

lemma maxBlk-in-nonInitBlks:
assumes b: b ∈ nonInitBlks s p
and inv1: Inv1 s
shows maxBlk s p ∈ nonInitBlks s p ∧ (∀c ∈ nonInitBlks s p. bal c ≤ bal (maxBlk s p))
proof –
have nibals-finite: finite (bal ' (nonInitBlks s p)) (is finite ?S)
proof (rule finite-imageI)
from inv1
have finite (allRdBlks s p)
by (auto simp add: Inv1-def)
hence finite (allBlocksRead s p)
by (auto simp add: allBlocksRead-def)
hence finite (blocksSeen s p)
by (simp add: blocksSeen-def)
thus finite (nonInitBlks s p)
by (auto simp add: nonInitBlks-def intro: finite-subset)
qed
from b have bal ' nonInitBlks s p ≠ {}
by auto
with nibals-finite
have Max ?S ∈ ?S and ∀bb ∈ ?S. bb ≤ Max ?S by auto
hence ∃mb ∈ ?S. ∀bb ∈ ?S. bb ≤ mb ..
then obtain mbblk
where mbblk ∈ nonInitBlks s p
∧ (∀c ∈ nonInitBlks s p. bal c ≤ bal mbblk)
(is ?P mbblk)
by auto
hence ?P (SOME b. ?P b)
by (rule someI)
thus ?thesis
by (simp add: maxBlk-def)
qed

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

34
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'

by (blast dest: HEndPhase1-HInv1)

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

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

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

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

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

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

lemma HEndPhase1-Inv2a-dblock-q:
  assumes act: HEndPhase1 s s' p
  and inv1: HInv1 s
  and inv: Inv2a s
  and inv2c: Inv2c-inner s p
  shows Inv2a-innermost s' q (dblock s' q)
proof (cases p = q)
  case True
  with act inv inv2c inv1
  show ?thesis
  by (blast dest: HEndPhase1-Inv2a-dblock)
next
  case False
  from inv
  have inv-q-dblock: Inv2a-innermost s q (dblock s q)
    by (auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
  with False act
  show ?thesis
    by (clarsimp simp add: EndPhase1-def HNextPart-def
      InitializePhase-def Inv2a-innermost-def)
qed

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

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

C.2.2 Proofs of Invariant 2 b

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

theorem HInit-Inv2b: HInit s −→ Inv2b s
by (auto simp add: HInit-def Init-def Inv2b-def
    Inv2b-inner-def InitDB-def)

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

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

theorem HPhase1or2Write-Inv2b:
[ Inv2b s; HPhase1or2Write s s’ p d ]
⇒ Inv2b s’
by (auto simp add: Phase1or2Write-def Inv2b-def
    Inv2b-inner-def hasRead-def)

**Theorem** HPhase1or2ReadElse-Inv2b:
\[
[\text{Inv2b } s \land \text{HPhase1or2ReadElse } s \ s' \ p \ q \ ] 
\implies \text{Inv2b } s'
\]
by (auto simp add: Phase1or2ReadElse-def StartBallot-def hasRead-def
    InitializePhase-def Inv2b-def Inv2b-inner-def)

**Theorem** HEndPhase1-Inv2b:
\[
[\text{Inv2b } s \land \text{HEndPhase1 } s \ s' \ p \ ] 
\implies \text{Inv2b } s'
\]
by (auto simp add: EndPhase1-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

**Theorem** HFail-Inv2b:
\[
[\text{Inv2b } s \land \text{HFail } s \ s' \ p \ ] 
\implies \text{Inv2b } s'
\]
by (auto simp add: Fail-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

**Theorem** HEndPhase2-Inv2b:
\[
[\text{Inv2b } s \land \text{HEndPhase2 } s \ s' \ p \ ] 
\implies \text{Inv2b } s'
\]
by (auto simp add: EndPhase2-def InitializePhase-def
    Inv2b-def Inv2b-inner-def hasRead-def)

**Theorem** HPhase0Read-Inv2b:
\[
[\text{Inv2b } s \land \text{HPhase0Read } s \ s' \ p \ d \ ] 
\implies \text{Inv2b } s'
\]
by (auto simp add: Phase0Read-def Inv2b-def
    Inv2b-inner-def hasRead-def)

**Theorem** HEndPhase0-Inv2b:
\[
[\text{Inv2b } s \land \text{HEndPhase0 } s \ s' \ p \ ] 
\implies \text{Inv2b } s'
\]
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 \rightarrow Inv2c s
by (auto simp add: HInit-def Init-def Inv2c-def Inv2c-inner-def)

**Lemma** HNextPart-Inv2c-chosen:
assumes  \( hnp: \text{HNextPart } s \ s' \)
and  \( inv2c: \text{Inv2c } s \)
and  \( outpt': \forall p. \text{outpt } s' \ p = (\text{if } \text{phase } s' \ p = 3 \ \text{then } \text{inp}(\text{dblock } s' \ p) \ \\
\text{else } \text{NotAnInput}) \)
and  \( inp-dblk: \forall p. \text{inp } (\text{dblock } s' \ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\} \)
sows chosen s' \in allInput s' \cup \{\text{NotAnInput}\}

38
using hnp outpt' inp-dblk inv2c
proof(auto simp add: HNextPart-dec Inv2c-dec Inv2c-inner-dec
   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-dec 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. inp s' p ∈ allInput s'
  by(auto simp add: HNextPart-dec Inv2c-dec Inv2c-inner-dec)

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

with outpt' chosen' act inv

show ?thesis
by (auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)

qed

theorem HStartBallot-Inv2c:
assumes inv: Inv2c s
and act: HStartBallot s s' p
and inv2a: Inv2a s
shows Inv2c s'

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

40
from act inv
have allinp: \( \forall p. \ inpt s' p \in allInput s' \)
\( \land (chosen s' = NotAnInput \implies outpt s' p = NotAnInput) \)
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 Inv2c s'
proof –
from inv2a act
have inv2a': Inv2a s'
  by (blast dest: HPhase1or2Write-Inv2a)
from act inv
have outpt': \( \forall p. outpt s' p = (\text{if phase } s' p = 3 \text{ then inp(dblock } s' p) \)
\( \text{ else NotAnInput}) \)
by (auto simp add: Phase1or2Write-def Inv2c-def Inv2c-inner-def)
from inv2a'
have blk: \( \forall p. inp (dblock s' p) \in allInput s' \cup \{NotAnInput\} \)
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: \( \forall p. inpt s' p \in allInput s' \land (chosen s' = NotAnInput \implies outpt s' p = NotAnInput) \)
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 ] \implies Inv2c s'
by (auto simp add: Phase1or2ReadElse-def dest: HStartBallot-Inv2c)

theorem HEndPhase1-Inv2c:
assumes inv: Inv2c s
and act: HEndPhase1 s s' p
and inv2a: Inv2a s
and \( \text{inv1} : H\text{Inv1} s \)
shows \( \text{Inv2c} s' \)

proof –

from \( \text{inv} \)
have \( \text{Inv2c-inner} s \ p \) by (auto simp add: \( \text{Inv2c-def} \))
with \( \text{inv2a act inv1} \)
have \( \text{inv2a}' : \text{Inv2a} s' \)
  by (blast dest: \( H\text{EndPhase1-Inv2a} \))
from \( \text{act inv} \)
have \( \text{mbal}' : \text{mbal}(\text{dblock} s' \ p) \in \text{Ballot} \ p \)
  by (auto simp add: \( \text{EndPhase1-def Inv2c-def Inv2c-inner-def} \))
from \( \text{act} \)
have \( \text{bal}' : \text{bal}(\text{dblock} s' \ p) = \text{mbal}(\text{dblock} s' \ p) \)
  by (auto simp add: \( \text{EndPhase1-def} \))
from \( \text{act inv} \)
have \( \text{blks}' : (\forall d. \forall br \in \text{blocksRead} s' p d. \text{mbal}(\text{block} br) < \text{mbal}(\text{dblock} s' \ p)) \)
  by (auto simp add: \( \text{EndPhase1-def InitializePhase-def Inv2c-def Inv2c-inner-def} \))
from \( \text{act inv} \)
have \( \text{outpt}' : \forall p. \text{outpt} s' p = (\text{if phase} s' p = 3 \text{ then inp}(\text{dblock} s' \ p) \text{ else NotAnInput}) \)
  by (auto simp add: \( \text{EndPhase1-def Inv2c-def Inv2c-inner-def} \))
from \( \text{inv2a}' \)
have \( \text{dblk} : \forall p. \text{inp}(\text{dblock} s' \ p) \in \text{allInput} s' \cup \{\text{NotAnInput}\} \)
  by (auto simp add: \( \text{Inv2a-def Inv2a-inner-def blocksOf-def} \))
with \( \text{act inv outpt}' \)
have \( \text{chosen}' : \text{chosen} s' \in \text{allInput} s' \cup \{\text{NotAnInput}\} \)
  by (auto dest: \( H\text{NextPart-Inv2c-chosen} \))
from \( \text{act inv} \)
have \( \text{allinp} : \forall p. \text{inpt} s' p \in \text{allInput} s' \)
\( \wedge (\text{chosen} s' = \text{NotAnInput} \rightarrow \text{outpt} s' p = \text{NotAnInput}) \)
  by (auto dest: \( H\text{NextPart-chosen HNextPart-allInput} \))
with \( \text{mbal}' \text{ bal}' \text{ blks}' \text{ outpt}' \text{ chosen}' \text{ act inv} \)
show \( \text{thesis} \)
  by (auto simp add: \( \text{EndPhase1-def InitializePhase-def Inv2c-def Inv2c-inner-def} \))

qed

theorem \( H\text{EndPhase2-Inv2c} : \)
assumes \( \text{inv} : \text{Inv2c} s \)
and \( \text{act} : H\text{EndPhase2} s \ s' \ p \)
and \( \text{inv2a} : \text{Inv2a} s \)
shows \( \text{Inv2c} s' \)

proof –
from \( \text{inv2a act} \)
have \( \text{inv2a} \): Inv2a \( s \)
  by (blast dest: HEndPhase2-inv2a)
from act inv
have \( \text{outpt'} \) : \( \forall p. \text{outpt} \ s' \ p = (\text{if phase} \ s' \ p = 3 \ \
  \text{then inp}(\text{dblock} \ s' \ p) \ \text{else NotAnInput}) \)
  by (auto simp add: EndPhase2-def Inv2c-def Inv2c-inner-def)
from inv2a'
have \( \text{dblk} \) : \( \forall p. \text{inp}(\text{dblock} \ s' \ p) \in \text{allInput} \ s' \cup \{\text{NotAnInput}\} \)
  by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have \( \text{chosen}' \) : \( \text{chosen} \ \ s' \in \text{allInput} \ s' \cup \{\text{NotAnInput}\} \)
  by (auto dest: HNextPart-inv2c-chosen)
from act inv
have \( \text{allinp} \) : \( \forall p. \ \text{inpt} \ s' \ p \in \text{allInput} \ s' \ \
  \land (\text{chosen} \ s' = \text{NotAnInput} \ \
  \longrightarrow \text{outpt} \ s' \ p = \text{NotAnInput}) \)
  by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show \(? \text{thesis}\)
  by (auto simp add: EndPhase2-def InitializePhase-def Inv2c-def Inv2c-inner-def)
qed

theorem HFail-inv2c:
  assumes \( \text{inv} \): Inv2c \( s \) and \( \text{act} \): HFail \( \ s \ s' \ p \) and \( \text{inv2a} \): Inv2a \( s \)
  shows Inv2c \( s' \)
proof --
  from inv2a act
  have \( \text{inv2a'} \): Inv2a \( s' \)
    by (blast dest: HFail-inv2a)
  from act inv
  have \( \text{outpt'} \) : \( \forall p. \text{outpt} \ s' \ p = (\text{if phase} \ s' \ p = 3 \ \
  \text{then inp}(\text{dblock} \ s' \ p) \ \text{else NotAnInput}) \)
    by (auto simp add: Fail-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have \( \text{dblk} \) : \( \forall p. \text{inp}(\text{dblock} \ s' \ p) \in \text{allInput} \ s' \cup \{\text{NotAnInput}\} \)
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have \( \text{chosen}' \) : \( \text{chosen} \ \ s' \in \text{allInput} \ s' \cup \{\text{NotAnInput}\} \)
    by (auto dest: HNextPart-inv2c-chosen)
  from act inv
  have \( \text{allinp} \) : \( \forall p. \ \text{inpt} \ s' \ p \in \text{allInput} \ s' \ \
  \land (\text{chosen} \ s' = \text{NotAnInput} \ \
  \longrightarrow \text{outpt} \ s' \ p = \text{NotAnInput}) \)
  qed
by\,(\texttt{auto dest: HNextPart-chosen HNextPart-allInput})

with\ \texttt{outpt\' chosen\' act inv}

show\ ?thesis
  \quad by\,(\texttt{auto simp add: Fail-def InitializePhase-def Inv2c-def Inv2c-inner-def})

qed

theorem \texttt{HPhase0Read-Inv2c}:
  assumes\ inv: Inv2c\ s
  and\ act: HPhase0Read\ s\ s'\ p\ d
  and\ inv2a: Inv2a\ s
  shows\ Inv2c\ s'

proof
  \quad from\ inv2a
  \quad have\ inv2a': Inv2a\ s'
    \quad by\,(\texttt{blast dest: HPhase0Read-Inv2a})
  \quad from\ act\ inv
  \quad have\ outpt': \forall \ p.\ outpt\ s'\ p\ =\ (\textit{if}\ phase\ s'\ p\ =\ 3
  \quad \quad \textit{then}\ inp\ (dblock\ s'\ p)
  \quad \quad \textit{else}\ NotAnInput)
    \quad by\,(\texttt{auto simp add: Phase0Read-def Inv2c-def Inv2c-inner-def})
  \quad from\ inv2a'
  \quad have\ blk: \forall \ p.\ inp\ (dblock\ s'\ p)\ \in\ allInput\ s'\ \cup\ \{NotAnInput\}
    \quad by\,(\texttt{auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def})
  \quad with\ act\ inv\ outpt'
  \quad have\ chosen': chosen\ s'\ \in\ allInput\ s'\ \cup\ \{NotAnInput\}
    \quad by\,(\texttt{auto dest: HNextPart-Inv2c-chosen})
  \quad from\ act\ inv
  \quad have\ allinp: \forall \ p.\ inpt\ s'\ p\ \in\ allInput\ s'
    \quad \land\ (chosen\ s'\ =\ NotAnInput
    \quad \quad \rightarrow\ outpt\ s'\ p\ =\ NotAnInput)
    \quad by\,(\texttt{auto dest: HNextPart-chosen HNextPart-allInput})
  with\ outpt' chosein\ act inv
  show\ ?thesis
    \quad by\,(\texttt{auto simp add: Phase0Read-def Inv2c-def Inv2c-inner-def})

qed

theorem \texttt{HEndPhase0-Inv2c}:
  assumes\ inv: Inv2c\ s
  and\ act: HEndPhase0\ s\ s'\ p
  and\ inv2a: Inv2a\ s
  and\ inv1: Inv1\ s
  shows\ Inv2c\ s'

proof
  \quad from\ inv
  \quad have\ Inv2c-inner\ s\ p\ by\ (\texttt{auto simp add: Inv2c-def})
  with\ inv2a\ act\ inv1

44
have \( inv2a \): \( \text{Inv2a } s' \)
  by (blast dest: HEndPhase0-Inv2a)

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

from act inv
have \( mbal' \): \( \text{mbal}(\text{dblock } s' p) \in \text{Ballot } p \)
  by (auto simp add: EndPhase0-def Inv2c-def Inv2c-inner-def)

from act inv
have \( blks' \): \( \forall d. \forall br \in \text{blocksRead } s' p d. \)
  \( \text{mbal}(\text{block } br) < \text{mbal}(\text{dblock } s' p) \)
  by (auto simp add: EndPhase0-def InitializePhase-def Inv2c-def Inv2c-inner-def)

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

from inv2a'

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

with act inv outpt'

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

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

with \( mbal' bal' blks' outpt' chosen' \) act inv

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

qed

theorem \( HInit-\text{Inv2} \):
  \( HInit s \Rightarrow \text{HInv2 } s \)

using \( HInit-\text{Inv2a } HInit-\text{Inv2b } HInit-\text{Inv2c} \)
by (auto simp add: HInv2-def)

\( H\text{Inv1} \wedge H\text{Inv2} \) is an invariant of \( H\text{Next} \).

lemma \( I2b \):
  assumes \( \text{nxt} \): \( \text{HNext } s s' \)
  and \( \text{inv} \): \( \text{HInv1 } s \wedge \text{HInv2 } s \)
  shows \( \text{HInv2 } s' \)

proof
  (auto simp add: HInv2-def)

show \( \text{Inv2a } s' \) using \( \text{assms} \)
  by (auto simp add: HInv2-def HNext-def Next-def, auto intro: HStartBallot-Inv2a, auto simp add: HDefEndPhase0)
auto intro: HPhase1or2Write-Inv2a,
auto simp add: Phase1or2Read-def
  intro: HPhase1or2ReadThen-Inv2a
  HPhase1or2ReadElse-Inv2a,
auto intro: HPhase0Read-Inv2a,
auto simp add: EndPhase1or2-2-def Inv2c-def
  intro: HEndPhase1-Inv2a
  HEndPhase2-Inv2a,
auto intro: HFail-Inv2a,
auto simp add: HInv1-def
  intro: HEndPhase0-Inv2a)
show Inv2b s' using assms
  by(auto simp add: HInv2-def HNext-def Next-def,
  auto intro: HStartBallot-Inv2b,
  auto intro: HPhase0Read-Inv2b,
  auto intro: HPhase1or2Write-Inv2b,
  auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2b
    HPhase1or2ReadElse-Inv2b,
  auto simp add: EndPhase1or2-2-def
    intro: HEndPhase1-Inv2b HEndPhase2-Inv2b,
  auto intro: HFail-Inv2b HEndPhase0-Inv2b)
show Inv2c s' using assms
  by(auto simp add: HInv2-def HNext-def Next-def,
  auto intro: HStartBallot-Inv2c,
  auto intro: HPhase0Read-Inv2c,
  auto intro: HPhase1or2Write-Inv2c,
  auto simp add: Phase1or2Read-def
    intro: HPhase1or2ReadThen-Inv2c
    HPhase1or2ReadElse-Inv2c,
  auto simp add: EndPhase1or2-2-def
    intro: HEndPhase1-Inv2c
    HEndPhase2-Inv2c,
  auto intro: HFail-Inv2c,
  auto simp add: HInv1-def intro: HEndPhase0-Inv2c)
qed

end

theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

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

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

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

**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** \(InitPhase-HInv3-p::\)
\(\begin{array}{l}
[\text{\{InitializePhase } s \ s' \ p; \ HInv3-L \ s' \ p \ q \ d \} \Rightarrow HInv3-R \ s' \ p \ q \ d]
\end{array}\)
by(auto simp add: \(InitializePhase\)-def \(HInv3-inner\)-def hasRead-def \(HInv3-L\)-def \(HInv3-R\)-def)

**lemma** \(InitPhase-HInv3-q::\)
\(\begin{array}{l}
[\text{\{InitializePhase } s \ s' \ q; \ HInv3-L \ s' \ p \ q \ d \} \Rightarrow HInv3-R \ s' \ p \ q \ d]
\end{array}\)
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 \(pq: \ p\neq q\)
and \(inv2b: \ Inv2b \ s\)
shows \(HInv3-R \ s' \ p \ q \ d\)
proof –
from \(inv-L'\) 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
    from nh3l h3l' act
    have (∼hasRead s pp dd q ∨ ∼hasRead s qq dd pp)
∧ hasRead s' pp dd qq ∧ hasRead s' qq dd pp
by (auto simp add: Hinv3-L-def Phase1or2ReadThen-def)
with act False
show ?thesis
by (auto dest: Phase1or2ReadThen-Hinv3-hasRead)
qed
qed
qed

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

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

lemma StartBallot-Hinv3-nL:
[ StartBallot s s' t; ¬Hinv3-L s p q d; t≠p; t≠ q ]
⇒ ¬Hinv3-L s' p q d
by (auto simp add: StartBallot-def InitializePhase-def
Hinv3-L-def hasRead-def)

lemma StartBallot-Hinv3-R:
[ StartBallot s s' t; Hinv3-R s p q d; t≠p; t≠ q ]
⇒ Hinv3-R s' p q d
by (auto simp add: StartBallot-def InitializePhase-def
Hinv3-R-def hasRead-def)

lemma StartBallot-Hinv3-t:
[ StartBallot s s' t; Hinv3-inner s p q d; t≠p; t≠ q ]
⇒ Hinv3-inner s' p q d
by (auto simp add: Hinv3-inner-def
dest: StartBallot-Hinv3-nL StartBallot-Hinv3-R)

lemma StartBallot-Hinv3:
assumes act: StartBallot s s' t
and inv: Hinv3-inner s p q d
shows Hinv3-inner s' p q d
proof (cases t=p ∨ t=q)
case True
with act inv
show ?thesis
by (auto simp add: Hinv3-inner-def
dest: StartBallot-Hinv3-p StartBallot-Hinv3-q)
next
case False
with inv act
show \(?thesis
  by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-t)
qed


definition 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

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

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

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

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

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

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

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

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

lemma EndPhase2-HInv3:
  assumes act: EndPhase2 s s' t
  and inv: HInv3-inner s p q d
  shows HInv3-inner s' p q d
proof (cases t=p \or t=q)
  case True
  with act inv
  show \?thesis
    by (auto simp add: HInv3-inner-def
                     dest: EndPhase2-HInv3-p EndPhase2-HInv3-q)
next
  case False
  with inv act
  show \?thesis
    by (auto simp add: HInv3-inner-def dest: EndPhase2-HInv3-t)
qed

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

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

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

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

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

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

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

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

53
next
  case False
  with act
  have ¬HInv3-L \(s\)' pp qq dd
    by (auto simp add: HInv3-L-def hasRead-def Phase0Read-def)
  thus ?thesis
    by (simp add: HInv3-inner-def)
qed

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

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

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

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

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

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

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

HInv1 \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' using assms
  by (auto simp add: HNext-def Next-def,
    auto intro: HStartBallot-HInv3,
    auto intro: HPhase0Read-HInv3,
    auto intro: HPhase1or2Write-HInv3,
    auto simp add: Phase1or2Read-def HInv2-def
    intro: HPhase1or2ReadThen-HInv3
    HPhase1or2ReadElse-HInv3,
    auto simp add: EndPhase1or2-def
    intro: HEndPhase1-HInv3
    HEndPhase2-HInv3,
    auto intro: HFail-HInv3,
    auto intro: HEndPhase0-HInv3)

end

theory DiskPaxos-Inv4 imports DiskPaxos-Inv2 begin

C.4 Invariant 4

This invariant expresses relations among \( mbal \) 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 $HInv4a1\ s\ p = (\forall bk \in blocksOf\ s\ p.\ bal\ bk \leq mbal\ (dblock\ s\ p))$

definition $HInv4a2 :: state \Rightarrow Proc \Rightarrow bool$

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

and $bal\ (disk\ s\ d\ p) \leq bal\ (dblock\ s\ p))$

definition $HInv4a :: state \Rightarrow Proc \Rightarrow bool$

where $HInv4a\ s\ p = (\forall D \in MajoritySet. (\exists d \in D.\ mbal\ (disk\ s\ d\ p) \leq mbal\ (dblock\ s\ p))$

and $bal\ (disk\ s\ d\ p) \leq bal\ (dblock\ s\ p))$

definition $HInv4b :: state \Rightarrow Proc \Rightarrow bool$

where $HInv4b\ s\ p = (\forall D \in MajoritySet. (\exists d \in D.\ mbal\ (disk\ s\ d\ p) \leq mbal\ (dblock\ s\ p))$

and $bal\ (disk\ s\ d\ p) \leq bal\ (dblock\ s\ p))$

definition $HInv4c :: state \Rightarrow Proc \Rightarrow bool$

where $HInv4c\ s\ p = (\forall D \in MajoritySet. (\exists d \in D.\ mbal\ (disk\ s\ d\ p) \leq mbal\ (dblock\ s\ p))$

and $bal\ (disk\ s\ d\ p) \leq bal\ (dblock\ s\ p))$

definition $HInv4d :: state \Rightarrow Proc \Rightarrow bool$

where $HInv4d\ s\ p = (\forall D \in MajoritySet. (\exists d \in D.\ mbal\ (disk\ s\ d\ p) \leq mbal\ (dblock\ s\ p))$

and $bal\ (disk\ s\ d\ p) \leq bal\ (dblock\ s\ p))$

definition $HInv4 :: state \Rightarrow bool$

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

The initial state implies Invariant 4.

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

using $Disk-isMajority$

by (auto simp add: $HInit-def$ $Init-def$ $HInv4-def$ $HInv4a1-def$

and $HInv4a2-def$ $HInv4b-def$ $HInv4c-def$ $HInv4d-def$

MajoritySet-def blocksOfDef InitDBDef rdByDef)

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 \( bk \in blocksOf s' p \)
with \( HStartBallot \)-blocksOf[OF act]
have \( bk \in \{dblock s' p\} \cup blocksOf s p \)
  by blast
thus \( bal bk \leq mbal (dblock s' p) \)
proof
  assume \( bk \in \{dblock s' p\} \)
  with inv2a
  show \(?thesis\)
    by(auto simp add: Inv2a-innermost-def Inv2a-inner-def blocksOf-def)
next
  assume \( bk \in blocksOf s p \)
  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: \( D \in MajoritySet \)
from inv Dmaj
have \( \exists d \in D. \ mbal (disk s d p) \leq mbal (dblock s p) \)
  \( \land \ bal (disk s d p) \leq bal (dblock s p) \)
  by(auto simp add: HInv4a2-def)
then obtain \( d \)
where \( d \in D \)
  \( \land \ mbal (disk s d p) \leq mbal (dblock s p) \)
  \( \land \ bal (disk s d p) \leq bal (dblock s p) \)
  by auto
with act
have \( d \in D \)
  \( \land \ mbal (disk s' d p) \leq mbal (dblock s' p) \)
  \( \land \ bal (disk s' d p) \leq bal (dblock s' p) \)
  by(auto simp add: StartBallot-def)
with Dmaj
show \( \exists d \in D. \ mbal (disk s' d p) \leq mbal (dblock s' p) \)
  \( \land \ bal (disk s' d p) \leq bal (dblock s' p) \)
  by auto
qed

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


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


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


lemma \text{Phase1or2Write-HInv4a1}:  

lemma Phase1or2Write-HInv4a2:

\[
\begin{align*}
\text{HPhase1or2Write} \ s \ s' \ p \ d \quad \Rightarrow \quad \text{HInv4a} \ s \ q \\
\text{HPhase1or2Write} \ s' \ p \ d \quad \Rightarrow \quad \text{HInv4a} \ s' \ q
\end{align*}
\]
by \((auto \ simp \ add: \text{Phase1or2Write-def} \ \text{HInv4a-def} \ \text{blocksOf-def} \ \text{rdBy-def})\)

theorem HPhase1or2Write-HInv4a:

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

lemma HPhase1or2ReadThen-HInv4a1-p:

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

lemma HPhase1or2ReadThen-HInv4a2:

\[
\begin{align*}
\text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ r \quad \Rightarrow \quad \text{HInv4a2} \ s' \ q
\end{align*}
\]
by \((auto \ simp \ add: \text{Phase1or2ReadThen-def} \ \text{HInv4a2-def})\)
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 add: HPhase1or2ReadThen-def simp del: HPhase1or2ReadThen-HInv4a1-p HPhase1or2ReadThen-HInv4a2)
qed

lemma HPhase1or2ReadThen-HInv4a-q:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: Hinv4a s q
and pnq: p ≠ q
shows Hinv4a s' q

proof –
from act pnq
have blocksOf s' q ⊆ blocksOf s q
  by (auto simp add: Phase1or2ReadThen-def InitializePhase-def blocksOf-def rdBy-def)
with act inv pnq
show ?thesis
  by (auto simp add: Phase1or2ReadThen-def HInv4a-def HInv4a1-def HInv4a2-def)
qed

theorem HPhase1or2ReadThen-HInv4a:
[ HPhase1or2ReadThen s s' p d r; Hinv4a s q; Inv2b s ] ⇒ Hinv4a s' q
by (blast dest: HPhase1or2ReadThen-HInv4a-p HPhase1or2ReadThen-HInv4a-q)

theorem HPhase1or2ReadElse-HInv4a:
assumes act: HPhase1or2ReadElse s s' p d r
and inv: Hinv4a s q and inv2a: Inv2a s'
shows Hinv4a s' q

proof –
from act have HStartBallot s s' p
  by (simp add: Phase1or2ReadElse-def)
with inv inv2a show ?thesis
  by (blast dest: HStartBallot-HInv4a)
qed

lemma HEndPhase1-HInv4a1:
assumes act: HEndPhase1 s s' p
and inv: Hinv4a1 s p

shows $HInv4a1$ s' p
proof(auto simp add: HInv4a1-def)
fix bk
assume bk: $bk \in \text{blocksOf } s'$ p
from bk $HEndPhase1$-blocksOf[OF act]
have bk $\in \{dblock s' p\} \cup \text{blocksOf } s$ p
  by blast
with act inv
show bal bk $\leq$ mbal (dblock s' p)
  by(auto simp add: HInv4a-def HInv4a1-def EndPhase1-def)
qed

lemma $HEndPhase1$-$HInv4a2$:
assumes act: $HEndPhase1$ s s' p
and inv: $HInv4a2$ s p
and inv2a: Inv2a s
shows $HInv4a2$ 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 (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 $HEndPhase1$-$HInv4a-p$:
assumes act: $HEndPhase1$ s s' p
and inv: $HInv4a$ s p
and inv2a: Inv2a s

61
shows \( HInv4a \ s \ s' \ p \)

proof –

from act

have phase: \( 0 < \text{phase} \ s \ p \)

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

with act inv inv2a

show ?thesis

\[ \text{by (auto simp del: HEndPhase1-def simp add: HInv4a-def}} \]

\[ \text{elim: HEndPhase1-HInv4a1 HEndPhase1-HInv4a2)} \]

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

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

moreover

from act pnq

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

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

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

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

ultimately

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

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

with act inv pnq

show ?thesis

\[ \text{by (auto simp add: EndPhase1-def HInv4a-def HInv4a1-def HInv4a2-def)} \]

qed

theorem HEndPhase1-HInv4a:

\[ [ HEndPhase1 \ s \ s' \ p; HInv4a \ s \ q ]; \text{Inv2a} \ s ]; \Rightarrow HInv4a s' q \]

\[ \text{by (blast dest: HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q)} \]

theorem HFail-HInv4a:

\[ [ HFail \ s \ s' \ p; HInv4a \ s \ q ]; \Rightarrow HInv4a s' q \]

\[ \text{by (auto simp add: Fail-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)} \]

theorem HPhase0Read-HInv4a:

\[ [ HPhase0Read \ s \ s' \ p \ d; HInv4a \ s \ q ]; \Rightarrow HInv4a s' q \]

\[ \text{by (auto simp add: Phase0Read-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)} \]
theorem HEndPhase2-HInv4a:
\[
\begin{align*}
& \text{HEndPhase2 } s \ s' \ p. \ \text{HInv4a } s \ q \ \Rightarrow \ \text{HInv4a } s' \ q \\
& \text{by (auto simp add: EndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def blocksOf-def rdBy-def)}
\end{align*}
\]

lemma 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 ?thesis by auto
qed

lemma EndPhase0-44:
 assumes act: EndPhase0 s s' p
 and bk: \( bk \in \text{blocksOf } s \ p \)
 and inv4d: \( \text{HInv4d } s \ p \)
 and inv2c: \( \text{Inv2c-inner } s \ p \)
shows \( \exists d. \exists \text{rb } \in \text{blocksRead } s \ p \ d. \ \text{bal } bk \leq \text{mbal } (\text{block } rb) \)
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: HInv4d-def)
  with 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 phase s p = 0 by (simp add: EndPhase0-def)
  with inv2c
  have \( \forall \text{rb } \in \text{blocksRead } s \ p \ d. \ \text{block } rb \text{ = disk } s \ d \ p \) — 5.1
    by (simp add: Inv2c-inner-def)
  hence \( \forall d. \ \text{hasRead } s \ p \ d \)
    \( \rightarrow (\exists \text{rb } \in \text{blocksRead } s \ p \ d. \ \text{block } rb \text{ = disk } s \ d \ p) \) — 5.2
    (is \( \forall d. \ ?H \ d \rightarrow ?P \ d \)
    by (auto simp add: hasRead-def)
  with act
  have p53: \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ ?P \ d \)
    by (auto simp add: MajoritySet-def EndPhase0-def)
  show ?thesis
proof
  from p43 p53
  have \( \exists D \in \text{MajoritySet}. \ \ (\exists d \in D. \ \text{bal } bk \leq \text{mbal } (\text{disk } s \ d \ p)) \)
    \( \wedge (\forall d \in D. \ ?P \ d) \)
    by auto
  thus ?thesis
    by force
qed
qed

lemma HEndPhase0-HInv4a1-p;
  assumes act: HEndPhase0 s s' p
  and inv2a': Inv2a s'
  and inv2c: Inv2c-inner s p
  and inv4d: HInv4d s p
  shows HInv4a1 s' p
proof(auto simp add: HInv4a1-def)
  fix bk
  assume bk ∈ blocksOf s' p
  with HEndPhase0-blocksOf[OF act]
  have bk ∈ {dblock s' p} ∪ blocksOf s p by auto
  thus bal bk ≤ mbal (dblock s' p)
  proof
    assume bk: bk ∈ {dblock s' p}
    with inv2a'
    have Inv2a-innermost s' p bk
      by(auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def)
    with bk show ?thesis
      by(auto simp add: Inv2a-innermost-def)
  next
    assume bk: bk ∈ blocksOf s p
    from act
    have f1: ∀r ∈ allBlocksRead s p. mbal r < mbal (dblock s' p)
      by(auto simp add: EndPhase0-def)
    with act inv4d inv2c bk
    have ∃d. ∃rb ∈ blocksRead s p d. bal bk ≤ mbal(block rb)
      by(auto dest: EndPhase0-44)
    with f1
    show ?thesis
      by(auto simp add: EndPhase0-def allBlocksRead-def allRdBlks-def dest: allSet)
  qed

qed

lemma hasRead-allBlks:
  assumes inv2c: Inv2c-inner s p
  and phase: phase s p = 0
  shows (∀d∈{d. hasRead s p d p}. disk s d p ∈ allBlocksRead s p)
proof
  fix d
  assume d: d∈{d. hasRead s p d p} (is d∈ ?D)
  hence br-ne: blocksRead s p d≠{d}
    by (auto simp add: hasRead-def)
  from inv2c phase
  have ∀br ∈ blocksRead s p d. block br = disk s d p
    by(auto simp add: Inv2c-inner-def)
  with br-ne
have disk s d p ∈ block "blocksRead s p d" by force
thus disk s d p ∈ allBlocksRead s p by (auto simp add: allBlocksRead-def allRdBlks-def)
qed

lemma HEndPhase0-41:
assumes act: HEndPhase0 s s' p and inv1: Inv1 s and inv2c: Inv2c-inner s p
shows ∃ D∈MajoritySet. ∀ d∈D. mbal(disk s d p) ≤ mbal(dblock s' p) ∧ bal(disk s d p) ≤ bal(dblock s' p)

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

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

lemma HEndPhase0-HInv4a2-p:
assumes act: HEndPhase0 s s' p and inv1: Inv1 s
and \( \text{inv2c: Inv2c-inner s p} \)

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 \( \exists D \in \text{MajoritySet}. \forall d \in D. \ mbal(disk s d p) \leq \ mbal(dblock s' d p) \)
  \( \wedge \ bal(disk s d p) \leq \ bal(dblock s' d p) \)
  by (blast dest: HEndPhase0-41)

from Majority-exQ[OF this]
have \( \forall D \in \text{MajoritySet}. \exists d \in D. \ mbal(disk s' d p) \leq \ mbal(dblock s' d p) \)
  \( \wedge \ bal(disk s' d p) \leq \ bal(dblock s' d p) \).

qed

lemma HEndPhase0-HInv4a-p:

assumes act: HEndPhase0 s s' p

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

and \( \text{inv2: Inv2c s} \)

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

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

and \( \text{inv: HInv4a s p} \)

shows \( \text{HInv4a s' p} \)

proof –

from inv2
have inv2c: Inv2c-inner s p
  by (auto simp add: Inv2c-def)

with inv1 inv2a act
have inv2a': Inv2a s'
  by (blast dest: HEndPhase0-Inv2a)

from act
have phase s' p = 1
  by (auto simp add: EndPhase0-def)

with act inv inv2c inv4d inv2a' inv1

show \( ?\text{thesis} \)
  by (auto simp add: HInv4a-def simp del: HEndPhase0-def
elem: HEndPhase0-HInv4a1-p HEndPhase0-HInv4a2-p)

qed

lemma HEndPhase0-HInv4a-q:

assumes act: HEndPhase0 s s' p

and \( \text{inv: HInv4a s q} \)

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

shows \( \text{HInv4a s' q} \)

proof –
from act pnq
have dblock s' q = dblock s q ∧ 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 \( p53: \exists \, \text{br} \in \text{allBlocksRead} \, s \, p \). \( \text{bal}(\text{dblock} \ s \ s') = \text{bal} \, \text{br} \)
by (auto simp add: EndPhase0-def)
from \( \text{inv2a} \)
have \( i2: \forall \, p. \, \forall \, \text{bk} \in \text{blocksOf} \, s \, p \). \( \text{bal} \, \text{bk} \leq \text{mbal} \, \text{bk} \)
by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
with \( \text{allBlks-in-blocksOf} \)
have \( \forall \, \text{bk} \in \text{allBlocksRead} \, s \, p \). \( \text{bal} \, \text{bk} \leq \text{mbal} \, \text{bk} \)
by auto
with \( p53 \)
have \( \exists \, \text{br} \in \text{allBlocksRead} \, s \, p \). \( \text{bal} \, (\text{dblock} \ s \ s') \leq \text{mbal} \, \text{br} \)
by force
with \( \text{HEndPhase0-dblock-mbal[OF act]} \)
show \(?\)thesis
by auto
qed

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

68
lemma HEndPhase0-HInv4b-p:
  assumes act: HEndPhase0 s s' p
  and inv4d: HInv4d s p
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2c: Inv2c-inner s p
  shows HInv4b s' p
proof (clarsimp simp add: HInv4b-def)
  from act have phase: phase s p = 0
    by (auto simp add: EndPhase0-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
  with HEndPhase0-blocksOf[OF act]
  have bk ∈ {dblock s' p } ∨ bk ∈ blocksOf s p
    by blast
  thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk ∈ {dblock s' p }
  with act inv1 inv2a inv2c
  show ?thesis
    by (auto simp del: HEndPhase0-def
        dest: HEndPhase0-HInv4b-p-dblock)
next
  assume bk: bk ∈ blocksOf s p
  with act inv2c inv4d
  show ?thesis
    by (blast dest: HEndPhase0-HInv4b-p-blocksOf)
qed
qed

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

next

assume bk: bk \in blocksOf s p

from phase inv4a

have p41: HInv4a1 s p

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

with p42 bk

show \textit{thesis}

\by (auto simp add: \textit{HInv4a1}-def)

qed

lemma \textit{HStartBallot-HInv4b-q}:

assumes act: \textit{HStartBallot} s s' p

and pnq: p \neq q

and inv: \textit{HInv4b} s q

shows \textit{HInv4b} s' q

proof –

from act pnq

have disk': disk s' = disk s

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

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

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

from act pnq

have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q

\by (auto simp add: \textit{StartBallot-def InitializePhase-def allRdBlks-def})

with

have blocksOf s' q \subseteq blocksOf s q

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

with inv phase' dblock'

show \textit{thesis}

\by (auto simp add: \textit{HInv4b-def})

qed

theorem \textit{HStartBallot-HInv4b}:

assumes act: \textit{HStartBallot} s s' p

and inv2a: Inv2a s

and inv4b: \textit{HInv4b} s q

and inv4a: \textit{HInv4a} s p

shows \textit{HInv4b} s' q

using act inv2a inv4b inv4a

proof (cases p=q)

case True

from inv2a

have inv2a-innermost s p (dblock s p)

\by (auto simp add: Inv2a-def inv2a-inner-def blocksOf-def)

with act True inv4b inv4a

show \textit{thesis}

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

theorem HPhase1or2Write-HInv4b:
  [ HPhase1or2Write s s' p d; HInv4b s q ] \implies HInv4b s' q
by (auto simp add: Phase1or2Write-def HInv4b-def
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 \neq q
  shows HInv4b s' q
using assms HPhase1or2ReadThen-blocksOf[OF act]
by (auto simp add: Phase1or2ReadThen-def HInv4b-def)

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

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

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

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

shows $HInv4b\ s'\ q$

proof –

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

qed

theorem HEndPhase1-HInv4b:
  assumes act: $HEndPhase1\ s\ s'\ p$
  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:
$HEndPhase2\ s\ s'\ p \implies HInv4b\ s'\ p$
by (auto simp add: EndPhase2-def HInv4b-def)

lemma HEndPhase2-HInv4b-q:
  assumes act: $HEndPhase2\ s\ s'\ p$
  and pnq: $p \neq q$
  and inv: $HInv4b\ s\ q$
  shows $HInv4b\ s'\ q$
proof –
from act pnq
have disk': disk s' = disk s
and dblock': dblock s' q = dblock s q
and phase': phase s' q = phase s q
  by (auto simp add: EndPhase2-def)
from act pnq
have \( \text{blocksRead}' \): \( \forall q. \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
by (auto simp add: EndPhase2-def InitializePhase-def allRdBlks-def)
with disk' dblock'
have \( \text{blocksOf} s' q \subseteq \text{blocksOf} s q \)
  by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
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 False inv
  have disk': disk \( s' = \text{disk} s \)
    and dblock': dblock \( s' q = \text{dblock} s q \)
    and phase': phase \( s' q = \text{phase} s q \)
    by (auto simp add: Fail-def)
  from act False inv
  have blocksRead': \( \forall q. \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
    by (auto simp add: Fail-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have \( \text{blocksOf} s' q \subseteq \text{blocksOf} s q \)
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
  show ?thesis
    by (auto simp add: HInv4b-def)
qed

lemma HFail-HInv4b-p:
  HFail \( s \ s' p \Rightarrow HInv4b s' p \)
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 \( s' = \text{disk} s \)
    and dblock': dblock \( s' q = \text{dblock} s q \)
    and phase': phase \( s' q = \text{phase} s q \)
    by (auto simp add: Fail-def)
  from act pnq
  have blocksRead': \( \forall q. \text{allRdBlks} s' q \subseteq \text{allRdBlks} s q \)
    by (auto simp add: Fail-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have \( \text{blocksOf} s' q \subseteq \text{blocksOf} s q \)
    by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  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 ⟹ HInv4b s' p
by (auto simp add: HPhase0Read-def HInv4b-def)

lemma HPhase0Read-HInv4b-q:
  assumes act: HPhase0Read s s' p d
  and pq: p≠q
  and inv: HInv4b s q
  shows HInv4b s' q
proof (cases)
  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

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

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

lemma HStartBallot-HInv4c-q:
assumes act: HStartBallot s s' p
and inv: HInv4c s q
and pnq: p \neq q
shows HInv4c s' q
proof
from act pnq
have phase: 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:
\([\ HStartBallot s s' p; HInv4c s q \implies HInv4c s' q \]
by (blast dest: HStartBallot-HInv4c-p HStartBallot-HInv4c-q)

lemma HPhase1or2Write-HInv4c-p:
assumes act: HPhase1or2Write s s' p d
and inv: HInv4c s p
and inv2c: Inv2c s
shows HInv4c s' p
proof (cases phase s' p = 2)
assume phase': phase s' p = 2
show ?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
theorem HPhase1or2ReadThen-HInv4c:
\[ [HPhase1or2ReadThen \ s \ s' \ p \ d \ r; \ HInv4c \ s \ q] \Rightarrow HInv4c \ s' \ q \]
by (blast dest: HPhase1or2ReadThen-HInv4c-p HPhase1or2ReadThen-HInv4c-q)

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

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

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

theorem HEndPhase1-HInv4c:
\[ [HEndPhase1 \ s \ s' \ p; \ HInv4c \ s \ q; \ Inv2b \ s] \Rightarrow HInv4c \ s' \ q \]
by (blast dest: HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q)
lemma  HEndPhase2-HInv4c-p:
[ HEndPhase2 s s' p; HInv4c s p ] \implies HInv4c s' p
by(auto simp add: EndPhase2-def HInv4c-def)

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

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

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

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

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

lemma  HPhase0Read-HInv4c-p:
**lemma** HPhase0Read-HInv4c-q:

assumes act: HPhase0Read $s$ $s'$ $p$ $d$

and inv: HInv4c $s$ $q$

and pn: $p 
eq q$

shows HInv4c $s'$ $q$

**proof**

from act pnq

have phase: phase $s'$ $q = phase s$ $q$

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

and disk: disk $s'$ = disk $s$

by (auto simp add: Phase0Read-def)

with inv

show thesis

by (auto simp add: HInv4c-def)

qed

**theorem** HPhase0Read-HInv4c

[ [ HPhase0Read $s$ $s'$ $p$ $d$; HInv4c $s$ $p$ ] ] \Rightarrow HInv4c $s'$ $q$

by (blast dest: HPhase0Read-HInv4c-p HPhase0Read-HInv4c-q)

**lemma** HEndPhase0-HInv4c-p:

[ [ HEndPhase0 $s$ $s'$ $p$ ] ] \Rightarrow HInv4c $s'$ $p$

by (auto simp add: EndPhase0-def HInv4c-def)

**lemma** HEndPhase0-HInv4c-q:

assumes act: HEndPhase0 $s$ $s'$ $p$

and inv: HInv4c $s$ $q$

and pnq: $p \neq q$

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: EndPhase0-def)

with inv

show thesis

by (auto simp add: HInv4c-def)

qed

**theorem** HEndPhase0-HInv4c

[ [ HEndPhase0 $s$ $s'$ $p$; HInv4c $s$ $p$ ] ] \Rightarrow HInv4c $s'$ $q$

by (blast dest: HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q)
C.4.4 Proofs of Invariant 4d

lemma HStartBallot-HInv4d-p:
  assumes act: HStartBallot s s' p
  and inv: HInv4d s p
  shows HInv4d s' p
proof (clarsimp simp add: HInv4d-def)
  fix bk
  assume bk: bk ∈ blocksOf s' p
  from act have bal': bal (dblock' s' p) = bal (dblock s p)
    by (auto simp add: StartBallot-def)
  from subsetD[OF HStartBallot-blocksOf[OF act] 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 pnq: p ≠ q
  shows HInv4d s' q
proof
  from act pnq have disk': disk s' = disk s
    and dblock': dblock s' q = dblock s q
    by (auto simp add: StartBallot-def)
  from act pnq have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by (auto simp add: StartBallot-def InitializePhase-def allRdBlks-def)
  with disk' 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 HStartBallot-HInv4d:
[ HStartBallot s s′ p; HInv4d s q ] ⊢ HInv4d s′ q
by(blast dest: HStartBallot-HInv4d-p HStartBallot-HInv4d-q)

lemma HPhase1or2Write-HInv4d-p:
assumes act: HPhase1or2Write s s′ p d
and inv: HInv4d s p
and inv4a: HInv4a s p
shows HInv4d s′ p
proof(clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk ∈ blocksOf s′ p
from act have ddisk: ∀dd. disk s′ dd p = (if d = dd
then dblock s p 
else disk s dd p)

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

lemma HPhase1or2Write-HInv4d-q:
assumes act: HPhase1or2Write s s′ p d
and inv: HInv4d s q
and pq: p≠q
shows HInv4d s′ q
proof –
from act pq
have disk′: ∀d. disk s′ d q = disk s d q 
by(auto simp add: Phase1or2Write-def)
from act pq
have blocksRead′: ∀q. allRdBlks s′ q ⊆ allRdBlks s q 
by(auto simp add: Phase1or2Write-def
InitializePhase-def allRdBlks-def)
with act pq
have blocksOf s′ q ⊆ blocksOf s q

82
by (auto simp add: Phase1or2Write-def allRdBlks-def blocksOf-def rdBy-def)

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

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

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

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

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' d 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 inv: HInv4d s q
and p ≠ q
shows HInv4d s' q

proof –
  from act pnq
  have disk': disk s' = disk s
     and dblock': dblock s' q = dblock s q
     by (auto simp add: EndPhase1-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
     by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
  with disk'
  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 HEndPhase1-HInv4d:
  [ HEndPhase1 s s' p; HInv4d s q; Inv2b s; HInv4c s p ]
  ⇒ HInv4d s' q
by (blast dest: HEndPhase1-HInv4d-p HEndPhase1-HInv4d-q)

lemma HEndPhase2-HInv4d-p:
  assumes act: HEndPhase2 s s' p
  and inv: HInv4d s p
  shows HInv4d s' p
proof (clar simp 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: EndPhase2-def)
  from subsetD[OF HEndPhase2-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: EndPhase2-def)
  qed

lemma HEndPhase2-HInv4d-q:
  assumes act: HEndPhase2 s s' p
and \( \text{inv: } H\text{inv}_4 d \ s \ q \)
and \( \text{pnq: } p \neq q \)
shows \( H\text{inv}_4 d \ s' \ q \)

**proof** –

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

**theorem** HEndPhase2-HInv4d:
[ 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 act: HFail s s' p
and inv: HInv4d s p
shows HInv4d s' p
**proof**(clarsimp simp add: HInv4d-def)
fix bk
assume bk: bk \in \text{blocksOf s'} p
from act
have disk': disk' = disk s
  by (auto simp add: Fail-def)
from subsetD[OF HFail-blocksOf[OF act] bk]
show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ bal bk \leq mbal(\text{disk s d p}) \)
proof
assume bk: bk \in \{d\text{block s'} p\}
with inv disk'
show \(?thesis\)
  by (auto simp add: HInv4d-def)
next
assume bk: bk \in \{d\text{block s'} p\}
with act
have bal bk = 0
  by (auto simp add: Fail-def InitDB-def)
with Disk-isMajority
show \(?thesis\)
  by (auto simp add: MajoritySet-def)

86
lemma $H\text{Fail-}H\text{Inv4d-q}$:
    assumes act: $H\text{Fail } s \ s' \ p$
    and inv: $H\text{Inv4d } s \ q$
    and pnq: $p \neq q$
    shows $H\text{Inv4d } s' \ q$
proof (auto simp add: Fail-def)
  from act pnq have $\exists \ D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal (disk } s \ d \ q)$
    by (auto simp add: $H\text{Inv4d-def}$)
  with act show $\exists \ D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal (disk } s' \ d \ q)$
    by (auto simp add: $H\text{Inv4d-def}$)
qed

 theorem $H\text{Fail-}H\text{Inv4d}$:
  $[ H\text{Fail } s \ s' \ p; \ H\text{Inv4d } s \ q ] \implies H\text{Inv4d } s' \ q$
by (blast dest: $H\text{Fail-}H\text{Inv4d-p}$)

lemma $H\text{Phase0Read-}H\text{Inv4d-p}$:
    assumes act: $H\text{Phase0Read } s \ s' \ p \ d$
    and inv: $H\text{Inv4d } s \ p$
    shows $H\text{Inv4d } s' \ p$
proof (clarsimp simp add: $H\text{Inv4d-def}$)
  assume bk: $bk \in \text{blocksOf } s' \ p$
  from act have $\text{bal'': } \text{bal (dblock } s' \ p) = \text{bal (dblock } s \ p)$
    by (auto simp add: $H\text{Phase0Read-def}$)
  from $\exists \ D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal (disk } s \ d \ p)$
  have $\exists \ D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal (disk } s' \ d \ p)$
    by (auto simp add: $H\text{Inv4d-def}$)
  with act show $\exists \ D \in \text{MajoritySet}. \ \forall d \in D. \ \text{bal } bk \leq \text{mbal (disk } s' \ d \ p)$
    by (auto simp add: $H\text{Phase0Read-def}$)
qed
lemma $H_{\text{Phase0Read-HInv4d-q}}$:
  assumes act: $H_{\text{Phase0Read}} s s' p d$
  and inv: $H_{\text{Inv4d}} s q$
  and pnq: $p \neq q$
  shows $H_{\text{Inv4d}} s' q$
proof –
  from act pnq
  have disk': disk $s' = \text{disk } s$
    by (auto simp add: $\text{Phase0Read-def}$)
  from act pnq
  have blocksOf $s' q \subseteq \text{blocksOf } s q$
    by (auto simp add: $\text{Phase0Read-def allRdBlks-def}$
                   blocksOf-def rdBy-def)
  from subsetD[OF this] inv
  have $\forall bk \in \text{blocksOf } s' q$.
      $\exists D \in \text{MajoritySet}. \forall d \in D. \text{bal } bk \leq \text{mbal(disk } s d q)$
    by (auto simp add: $\text{HInv4d-def}$)
  with disk'
  show $?\text{thesis}$
    by (auto simp add: $\text{HInv4d-def}$)
qed

theorem $H_{\text{Phase0Read-HInv4d}}$:
  $[ H_{\text{Phase0Read }} s s' p d; H_{\text{Inv4d } s q} ] \Rightarrow H_{\text{Inv4d } s' q}$
by (blast dest: $H_{\text{Phase0Read-HInv4d-p}}$ $H_{\text{Phase0Read-HInv4d-q}}$)

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

lemma $H_{\text{EndPhase0-HInv4d-p}}$:
  assumes act: $H_{\text{EndPhase0 }} s s' p$
  and inv: $H_{\text{Inv4d } s p}$
  and inv2c: $\text{Inv2c } s$
  and inv1: $\text{Inv1 } s$
  shows $H_{\text{Inv4d } s' p}$
proof(clarsimp simp add: $\text{HInv4d-def}$)
  fix $bk$

assume \( bk: bk \in \text{blocksOf } s' p \)
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 \( \text{inv} \)
  show \( \text{thesis} \)
  by (auto simp add: \text{HInv4d-def})

next

- assume \( bk: bk \in \{ \text{dblock } s' p \} \)
  from \( \text{inv2c} \)
  have \( \text{inv2c-inner}: \text{Inv2c-inner } s \ p \)
  by (auto simp add: \text{Inv2c-def})
  from \( bk \ \text{HEndPhase0-some}[\text{OF } \text{act inv1}] \)
  \( \text{HEndPhase0-blocksOf2}[\text{OF } \text{act inv2c-inner} \ text{act} \)
  have \( \text{bal } bk \in \text{bal '}(\text{blocksOf } s' p) \)
  by (auto simp add: \text{EndPhase0-def})
  with \( \text{inv} \)
  show \( \text{thesis} \)
  by (auto simp add: \text{HInv4d-def})

qed

with \( \text{act} \)
show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ \text{bal } bk \leq \text{mbal } (\text{disk } s' d p) \)
by (auto simp add: \text{EndPhase0-def})

qed

lemma \( \text{HEndPhase0-HInv4d-q} \):
assumes \( \text{act}: \text{HEndPhase0 } s s' p \)
and \( \text{inv}: \text{HInv4d } s q \)
and \( \text{pnq}: p \neq q \)
shows \( \text{HInv4d } s' q \)
proof

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

moreover

- from \( \text{act pnq} \)
  have \( \forall p d. \ \text{rdBy } s' q p d \subseteq \text{rdBy } s q p d \)
  by (auto simp add: \text{EndPhase0-def} \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 \( \text{act} \)

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

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

Since we have already proved \( HInv2 \) is an invariant of \( HNext \), \( HInv1 \wedge HInv2 \wedge HInv4 \) is also an invariant of \( HNext \).

lemma I2d:
assumes nxt: HNext s s'
and inv: HInv1 s \wedge HInv2 s \wedge HInv2 s' \wedge 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 intro: HPhase1or2Write-HInv4d, 
auto simp add: Phase1or2Read-def 
  intro: HPhase1or2ReadThen-HInv4d 
  HPhase1or2ReadElse-HInv4d, 
auto simp add: EndPhase1or2-def 
  intro: HEndPhase1-HInv4d 
  HEndPhase2-HInv4d, 
auto intro: HFail-HInv4d, 
auto intro: HEndPhase0-HInv4d simp add: HInv1-def)
qed
end

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

C.5 Invariant 5

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

definition maxBalInp :: state ⇒ nat ⇒ InputsOrNi ⇒ bool 
  where maxBalInp s b v = (∀ bk∈allBlocks s. b ≤ bal bk → inp bk = v)
definition HInv5-inner-R :: state ⇒ Proc ⇒ bool 
  where HInv5-inner-R s p = 
    (maxBalInp s (bal(dblock s p)) (inp(dblock s p)) 
     ∨ (∃ D∈MajoritySet. ∃ q. (∀ d∈D. bal(dblock s p) < mbal(disk s d q) 
       ∧ ¬hasRead s p d q)))
definition HInv5-inner :: state ⇒ Proc ⇒ bool
where $HInv5_{-inner\ s\ p} = (phase\ s\ p = 2 \rightarrow HInv5_{-inner-R\ s\ p})$

definition $HInv5 :\ state \Rightarrow bool$
  where $HInv5\ s = (\forall p.\ HInv5_{-inner\ s\ p})$

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

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

We will use the notation used in the proofs of invariant 4, and prove the lemma $action-HInv5_{-p}$ and $action-HInv5_{-q}$ for each action, for the cases $p = q$ and $p \neq q$ respectively.

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

lemma $HStartBallot-HInv5_{-p}$:
  assumes act : $HStartBallot\ s\ s'\ p$
  and inv : $HInv5_{-inner\ s\ p}$
  shows $HInv5_{-inner\ s'\ p}$ using assms
  by (auto simp add : StartBallot-def HInv5-inner-def)

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

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

proof (auto simp del : HStartBallot-def simp add : allBlocks-def
  dest : HStartBallot-blocksOf-q[OF act])
  fix x pa
  assume x-pa : $x \in blocksOf\ s'\ pa$ and
    x-nblks : $\forall xa.\ x \notin blocksOf\ s\ xa$
  show $x = dblock\ s'\ p$
  proof (cases p=pa)
    case True
    from x-nblks
    have $x \notin blocksOf\ s\ p$
    by auto
    with True subsetD[OF HStartBallot-blocksOf[OF act] x-pa]
show ?thesis
by auto

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

qed

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

qed

lemma HStartBallot-HInv5-q2:
assumes act: HStartBallot s s' p
and pnq: p ≠ q
and inv5-2: ∃D ∈ MajoritySet. ∃qq. (∀d ∈ D. bal(dblock s q) < mbal(disk s d qq)
∧ ¬hasRead s q d qq)
shows ∃D ∈ MajoritySet. ∃qq. (∀d ∈ D. bal(dblock s' q) < mbal(disk s' d qq)
∧ ¬hasRead s' q d qq)
proof –
from act pnq
have disk: disk s' = disk s
93
\[ \forall d. \text{blocksRead}(s) = \text{blocksRead}(s') \]
\[ \text{dblock}(s) = \text{dblock}(s') \]

with inv5-2

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

qed

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

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

lemma \text{HPhase1or2Write-HInv5-1}:
assumes act: \text{HPhase1or2Write}(s, s')
and \( \text{maxBalInp}(s, \text{bal}(\text{dblock}(s, q)), \text{inp}(\text{dblock}(s, q))) \)
and \( \text{inv5-1} : \text{maxBalInp}(s', \text{bal}(\text{dblock}(s', q)), \text{inp}(\text{dblock}(s', q))) \)
shows \( \text{maxBalInp}(s', \text{bal}(\text{dblock}(s, q)), \text{inp}(\text{dblock}(s', q))) \)
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 act: \text{HPhase1or2Write}(s, s')
and \( \text{inv4c} : \text{HInv4c}(s, p) \)
and \( \text{phase}(s, p) = 2 \)
and \( \text{inv5-2} : \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock}(s, p)) < \text{mbal}(\text{disk}(s, d, q)) \land \neg \text{hasRead}(s, p, q)) \)
shows \( \exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal}(\text{dblock}(s', p)) < \text{mbal}(\text{disk}(s', d, q)) \land \neg \text{hasRead}(s', p, q)) \)
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 \( p \neq q \)
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(auto simp add: HInv4c-def MajoritySet-def)
with i1 majorities-intersect
have \( D \cap D \neq \{ \} \) by auto
then obtain \( dd \) where \( dd \in D \cap D \)
by auto
with i1 i2 r1
have bal(dblock s \( p \)) < mbal(disk s \( dd \ q \)) \( \land \) mbal(disk s \( dd \ p \)) = bal(dblock s \( p \))
by auto
thus ?thesis by auto
qed
from act pnq
— dblock and hasRead do not change
have dblock \( s' = dblock s \)
and \( \forall \ d. \ hasRead s' \ p \ d \ q = hasRead s \ p \ d \ q \)
— 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 \( \forall \ d \in D. \ bal(dblock s' \ p) < mbal(disk s' \ d \ q) \ \land \neg hasRead s' \ p \ d \ q \)
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' \( p = 2 \)
and i2: \( \forall \ D \in \text{MajoritySet}. \ \forall \ q. \ \exists \ d \in D. \ bal(dblock s' \ p) < mbal(disk s' \ d \ q) \)
\( \rightarrow \) 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 HPhase1or2Write-HInv5-q2:
assumes act: HPhase1or2Write s s' p d
and pnq: p ≠ q
and inv4a: HInv4a s p
and inv5-2: ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
\land \neg hasRead s q d qq)
shows ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal(dblock s' q) < mbal(disk s' d qq)
\land \neg hasRead s' q d qq)

proof –
from inv5-2
obtain D qq
  where i1: IsMajority D
  and i2: ∀ d ∈ D. bal(dblock s q) < mbal(disk s d qq)
  and i3: ∀ d ∈ D. \neg hasRead s q d qq
by (auto simp add: MajoritySet-def)
from act pnq — dblock and hasRead do not change
have dblock': dblock s' = dblock s
and hasread: ∀ d. hasRead s' q d qq = hasRead s q d qq
by (auto simp add: Phase1or2Write-def hasRead-def)
have ∀ d ∈ D. bal(dblock s' q) < mbal(disk s' d qq) \land \neg hasRead s' q d qq
proof (cases qq = p)
case True
  have bal(dblock s q) < mbal(dblock s p)
  proof –
  from inv4a act i1
  have ∃ d ∈ D. mbal(disk s d p) ≤ mbal(dblock s p)
  by (auto simp add: MajoritySet-def HInv4a-def HInv4a2-def Phase1or2Write-def)
  with True i2
  show bal(dblock s q) < mbal(dblock s p)
  by auto
qed
with hasread dblock' True i1 i2 i3 act
show ?thesis
  by (auto simp add: Phase1or2Write-def)
next
case False
with act i2 i3
show ?thesis
  by (auto simp add: Phase1or2Write-def hasRead-def)
qed
with i1
show ?thesis
  by (auto simp add: MajoritySet-def)
qed
lemma \textit{HPhase1or2Write-HInv5-q}:  
\textbf{assumes} act: \textit{HPhase1or2Write s s' p d}  
and \text{inv}: \textit{HInv5-inner s q}  
and \text{inv4a}: \textit{HInv4a s p}  
and \text{pnq: p\neq q}  
\textbf{shows} \textit{HInv5-inner s' q}  
\textbf{proof} (auto simp add: \textit{HInv5-inner-def HInv5-inner-R-def})  
assume \text{phase' \textbf{act}: phase s q = 2}  
and \text{i2: \forall D\in\text{MajoritySet}. \forall qa. \exists d\in D. \text{bal(dblock s' q)} < \text{mbal(disk s' d qa)}}  
\rightarrow \text{hasRead s' q d qa}  
\textbf{from phase' \textbf{act} have phase: phase s q = 2}  
by (auto simp add: \textit{Phase1or2Write-def})  
show \text{maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))}}  
\textbf{proof} (rule \textit{HPhase1or2Write-HInv5-1}[OF act, of q])  
from \text{HPhase1or2Write-HInv5-q2}[OF act \text{ pnq inv4a} inv i2 phase}  
\textbf{show} \text{maxBalInp s (bal(dblock s q)) (inp(dblock s q))}  
by (auto simp add: \textit{HInv5-inner-def HInv5-inner-R-def}, blast)  
\textbf{qed}

\textbf{qed}

\textbf{theorem} \textit{HPhase1or2Write-HInv5}:  
\textit{[ \textit{HPhase1or2Write s s' p d; HInv5-inner s q; HInv4c s p; HInv4a s p \] \rightarrow HInv5-inner s' q}}  
by (auto simp dest: \textit{HPhase1or2Write-HInv5-q HPhase1or2Write-HInv5-p})

\textbf{lemma} \textit{HPhase1or2ReadThen-HInv5-1}:  
\textbf{assumes} act: \textit{HPhase1or2ReadThen s s' p d r}  
and \text{inv5-1: maxBalInp s' (bal(dblock s q)) (inp(dblock s' q))}}  
\textbf{shows} \text{maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))}}  
\textbf{using} \textit{assms and HPhase1or2ReadThen-blocksOf[OF act]}  
\textbf{by} (auto simp add: \textit{Phase1or2ReadThen-def maxBalInp-def allBlocks-def})

\textbf{lemma} \textit{HPhase1or2ReadThen-HInv5-p2}:  
\textbf{assumes} act: \textit{HPhase1or2ReadThen s s' p d r}  
and \text{inv4c: HInv4c s p}  
and \text{inv2c: Inv2c-inner s p}  
and \text{phase: phase s p = 2}  
and \text{inv5-2: \exists D\in\text{MajoritySet}. \exists q. (\forall d\in D. \text{bal(dblock s p)} < \text{mbal(disk s d q)}}  
\wedge \neg \text{hasRead s p d q)}}  
\textbf{shows} \text{\exists D\in\text{MajoritySet}. \exists q. (\forall d\in D. \text{bal(dblock s' p)} < \text{mbal(disk s' d q)}}  
\wedge \neg \text{hasRead s' p d q)}}  
\textbf{proof}  
\textbf{from inv5-2}  
\textbf{obtain} D q  
\textbf{where i1: IsMajority D}  
and \text{i2: \forall d\in D. \text{bal(dblock s p)} < \text{mbal(disk s d q)}}  
and \text{i3: \forall d\in D. \neg \text{hasRead s p d q)}}  
\textbf{by} (auto simp add: \textit{MajoritySet-def})  
\textbf{from inv2c phase}
have \( \text{bal}(\text{dblock } s \ p) = \text{mbal}(\text{dblock } s \ p) \) by (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 (auto simp add: Phase1or2ReadThen-def)

moreover

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

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

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

proof -

from 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 (auto simp add: HInv4c-def MajoritySet-def)

with \text{i1 majorities-intersect}

have \( D \cap D1 \neq {} \) by auto

then obtain \( dd \) where \( dd \in D \cap D1 \)

by auto

with \text{i1 i2 r1}

have \( \text{bal}(\text{dblock } s \ p) < \text{mbal}(\text{disk } s \ dd \ q) \land \text{mbal}(\text{disk } s \ dd \ p) = \text{bal}(\text{dblock } s \ p) \)

by auto

thus ?thesis by auto

qed

from \text{pnr act}

have \( \text{hasRead}' : \forall d \in D. \text{hasRead } s' p d q = \text{hasRead } s p d q \)

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

from \text{act pnq}

— \text{dblock and disk do not change}

have \( \text{dblock } s' = \text{dblock } s \)

and \( \forall d. \text{disk } s' = \text{disk } s \)

by (auto simp add: Phase1or2ReadThen-def)

with \text{i2 hasRead'} i3

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

by auto

with \text{i1}

show ?thesis

by (auto simp add: MajoritySet-def)

qed

lemma HPhase1or2ReadThen-HInv5-p:

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

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

and \text{inv4}: HInv4c \( s \ p \)

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

shows HInv5-inner \( s' \ p \)

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

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

and \( \forall d \in \text{MajoritySet}. \forall q. \exists d \in D. \text{bal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s' d q) \imp \text{hasRead } s' p d q \)
with act have phase: phase s p = 2
by (auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal (dblock s' p)) (inp (dblock s' p))
proof (rule HPhase1or2ReadThen-HInv5-1[of act, of p])
  from inv2c
  have Inv2c-inner s p by (auto simp add: Inv2c-def)
  from HPhase1or2ReadThen-HInv5-p2[of act inv4 this phase] inv i2 phase
  show maxBalInp s (bal (dblock s p)) (inp (dblock s p))
    by (auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

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

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

lemma HPhase1or2ReadThen-HInv5-q:
assumes act: HPhase1or2ReadThen s s' p d r
and inv: HInv5-inner s q
and inv4a: HInv4a s p
and pnq: p ≠ q
shows HInv5-inner s' q
proof(auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase s' q = 2
and i2: ∀ D∈MajoritySet. ∀ qa. ∃ d∈D. bal (dblock s' q) < mbal (disk s' d qa)
→→ hasRead s' q d qa
from phase' act have phase: phase s q = 2
by(auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal (dblock s' q)) (inp (dblock s' q))
proof(rule HPhase1or2ReadThen-HInv5-1[OF act, of q])
from HPhase1or2ReadThen-HInv5-q2[OF act pnq inv4a] i2 phase
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed

theorem HPhase1or2ReadThen-HInv5:
[ HPhase1or2ReadThen s s' p d r; HInv5-inner s q;
  HInv5-e 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-HInv5-p:
HEndPhase2 s s' p → HInv5-inner s' p
by(auto simp add: EndPhase2-def HInv5-inner-def)

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

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

lemma HEndPhase2-HInv5-q2:
  assumes act: HEndPhase2 s s' p
  and pnq: p\# q
  and inv5-2: \exists D \in MajoritySet. \exists qq. (\forall d \in D. bal(dblock s q) < mbal(disk s d qq)
  \wedge \neg hasRead s q d qq)
  shows \exists D \in MajoritySet. \exists qq. (\forall d \in D. bal(dblock s' q) < mbal(disk s' d qq)
  \wedge \neg hasRead s' q d qq)
proof -
  from act pnq
  have disk: disk s' = disk s
    and blocksRead: \forall d. blocksRead s' q d = blocksRead s q d
    and dblock: dblock s' q = dblock s q
    by(auto simp add: 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\# 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-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 q. (\forall d \in D. bal(dblock s' q) < mbal(disk s' d q)
  \wedge \neg hasRead s' p d q))
proof -
have \( \exists bk \in \text{allBlocks } s. \, \text{bal}(\text{dblock } s') \leq \text{bal } bk \land bk \neq \text{dblock } s' \) p

proof –

from asm4 obtain bk
where p31: \( bk \in \text{allBlocks } s' \land \text{bal}(\text{dblock } s') \leq \text{bal } bk \land bk \neq \text{dblock } s' \) p
by (auto simp add: maxBalImp-def)
then obtain q where p32: \( bk \in \text{blocksOf } s' \) q
by (auto simp add: allBlocks-def)
from act
have dblock: \( p \neq q \Longrightarrow \text{dblock } s' q = \text{dblock } s q \)
by (auto simp add: EndPhase1-def)
have bk \( \in \text{blocksOf } s q \)
proof (cases \( p = q \))
  case True
  with p32 p31 HEndPhase1-blocksOf[OF act]
  show \(?thesis\)
  by auto
next
  case False
  from dblock[OF False subsetD[OF HEndPhase1-blocksOf[OF act, of q] p32]
  show \(?thesis\)
  by (auto simp add: blocksOf-def)
qed
with p31
show \(?thesis\)
by (auto simp add: allBlocks-def)
qed
then obtain bk where p22: \( bk \in \text{allBlocks } s \land \text{bal}(\text{dblock } s') \leq \text{bal } bk \land bk \neq \text{dblock } s' \) p by auto
have \( \exists q \in \text{UNIV} - \{p\}. \, bk \in \text{blocksOf } s q \)
proof –

from p22 obtain q where bk: \( bk \in \text{blocksOf } s q \)
by (auto simp add: allBlocks-def)
from 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 \( H\text{inv4}b s p \) by (auto simp add: Hinv4-def)
ultimately
have \( p \neq q \)
using bk
by (auto simp add: Hinv4-def Hinv4b-def)
with bk

102
show \( ? \)thesis
by auto
qed
then obtain \( q \) where \( p23: q \in \text{UNIV} - \{ p \} \land bk \in \text{blocksOf} \ s\ q \)
by auto
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \ bal(d\ block\ s'\ p) \leq \ mbal(disk\ s\ d\ q) \)
proof
from \( p23 \) inv4
have \( \forall d'. \exists D \in \text{MajoritySet}. \forall d \in D. \ bal\ bk \leq \ mbal(disk\ s\ d\ q) \)
by(auto simp add: HInv4-def HInv4d-def)
from \( i4d\ p22 \)
show \( ? \)thesis
by force
qed
then obtain \( D \) where \( Dmaj: D \in \text{MajoritySet} \) and \( p24: (\forall d \in D. \ bal(d\ block\ s'\ p) \leq \ mbal(disk\ s\ d\ q)) \)
by(auto simp add: HInv4-def HInv4d-def)
proof
from \( inv2c \)
have \( \text{Inv2c-inner} \ s\ p \)
by(auto simp add: Inv2c-def)
with \( \text{act} \)
have \( \text{bal-pos}: 0 < \ bal(d\ block\ s'\ p) \)
by(auto simp add: Inv2c-inner-def EndPhase1-def)
with \( \text{inv2a} \)
have \( \bal(d\ block\ s'\ p) \in \text{Ballot}\ p \cup \{ 0 \} \)
by(auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with \( \text{bal-pos} \) have \( \text{bal-in-p}: \bal(d\ block\ s'\ p) \in \text{Ballot}\ p \)
by auto
from \( \text{inv2a} \) have \( \text{Inv2a-inner} \ s\ q \)
by(auto simp add: Inv2a-def)
then obtain \( D \) where \( Dmaj: D \in \text{MajoritySet} \)
by(auto simp add: EndPhase1-def InitializePhase-def hasRead-def)
lemma union-inclusion:
\[ A \subseteq A'; B \subseteq B' \] \implies A \cup B \subseteq A' \cup B'
by blast

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

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

case True
from x-nblks
have x \notin blocksOf s p
by auto
with True subsetD[OF HEndPhase1-blocksOf[OF act] x-pa]
show ?thesis
  by auto
next
case False
  from x-nblks subsetD[OF HEndPhase1-blocksOf-q[OF act 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≠q
  and asm4: ¬maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
  shows (∃D∈MajoritySet. ∃qq. (∀d∈D. bal(dblock s' q) < mbal(disk s d qq)
    ∧ ¬hasRead s q d qq))
proof –
  from act pnq
  have phase s' q = phase s q
    and phase-p: phase s p = 1
    and disk: disk s' = disk s
    and dblock: dblock s' q = dblock s q
    and bal: bal(dblock s' p) = mbal(dblock s p)
    by(auto simp add: HEndPhase1-def InitializePhase-def)
  with phase'
  have phase: phase s q = 2 by auto
  from phase inv2c
  have bal-dblq: bal(dblock s q) ∈ Ballot q
    by(auto simp add: Inv2c-def Inv2c-inner-def)
  have ∃D∈MajoritySet. ∃qq. (∀d∈D. bal(dblock s q) < mbal(disk s d qq)
    ∧ ¬hasRead s q d qq)
proof(cases maxBalInp s (bal(dblock s q)) (inp(dblock s q))]
case True
  have p21: bal(dblock s q) < bal(dblock s' p) ∧ inp(dblock s q) ≠ inp(dblock s'
    p)
  proof –
    from True asm4 dblock HEndPhase1-allBlocks[OF act]
    have p32: bal(dblock s q) ≤ bal(dblock s' p)
      ∧ inp(dblock s q) = inp(dblock s' p)
    by(auto simp add: maxBalInp-def)
    from inv2a
have \( \text{bal}(\text{dblock } s' p) \in \text{Ballot } p \cup \{ \emptyset \} \)
by (auto simp add: Inv2a-def Inv2a-inner-def
\hspace{1cm} Inv2a-innermost-def blocks0f-def)

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

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

with \( p32 \)
show \(?\)thesis
by auto

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

qed

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

have \( \text{p23} : \forall d \in D. \text{\{block=\text{dblock } s q, proc=q\} } \notin \text{blocksRead } s p d \)
proof –
have \( \text{dblock } s q \in \text{allBlocksRead } s p \longrightarrow \text{inp}(\text{dblock } s' p) = \text{inp}(\text{dblock } s q) \)
proof auto
assume \( \text{dblock-q } \text{dblock } s q \in \text{allBlocksRead } s p \)
from \( \text{inv2a-q} \)
have \((\text{bal}(\text{dblock}\ s\ q)=0) = (\text{inp}(\text{dblock}\ s\ q) = \text{NotAnInput})\)
  by(auto simp add: \text{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: \text{nonInitBlks-def blocksSeen-def blocksOf-def Inv2a-innermost-def})
with \text{act}
have \text{dblock-max: inp}(\text{dblock}\ s\ p)=\text{inp}(\text{maxBlk}\ s\ p)
  by(auto simp add: \text{EndPhase1-def})
from \text{maxBlk-in-nonInitBlks[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: \text{allBlocks-def nonInitBlks-def blocksSeen-def blocksOf-def rdBy-def allBlocksRead-def allRdBlks-def})
with True \text{subsetD[OF this max-in-nib]}
have \text{bal\ (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: \text{maxBalInp-def})
with \text{maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]}
dblock-q-nib dblock-max
show \text{inp}(\text{dblock}\ s'\ p)=\text{inp}(\text{dblock}\ s\ q)
  by auto
qed
with \text{p21}
have \text{dblock\ s\ q} \notin \text{block' allRdBlks\ s\ p}
  by(auto simp add: \text{allBlocksRead-def})
hence \forall d. \text{dblock\ s\ q} \notin \text{block' blocksRead\ s\ p\ d}
  by(auto simp add: \text{allRdBlks-def})
thus \text{?thesis}
  by force
qed
have \text{p24: \forall d\in D. \neg(\exists br\in blocksRead\ s\ q\ d. \text{bal}(\text{dblock}\ s\ q) \leq \text{mbal}\ (\text{block}\ br))}
proof
  from \text{inv2c phase}
  have \forall d. \forall br\in blocksRead\ s\ q\ d. \text{mbal}(\text{block}\ br) < \text{mbal}(\text{dblock}\ s\ q)
    and \text{bal}(\text{dblock}\ s\ q) = \text{mbal}(\text{dblock}\ s\ q)
    by(auto simp add: \text{Inv2c-def Inv2c-inner-def})
  thus \text{?thesis}
    by force
qed
have \text{p25: \forall d\in D. \neg\text{hasRead}\ s\ q\ d\ p}
proof auto
fix d
assume \text{d-in-D: d} \in D
and \text{hasRead-qdp: hasRead\ s\ q\ d\ p}
have \text{p31: (\text{block'=dblock}\ s\ p, proc'=p)\in blocksRead\ s\ q\ d}
proof
  from \text{d-in-D p22}
have hasRead-pdq; hasRead s p d q by auto
with hasRead-qdp phase phase-p inv3
have HInv3-R s q p d
  by(auto simp add: HInv3-def HInv3-inner-def HInv3-L-def)
with p23 d-in-D
show ?thesis
  by(auto simp add: HInv3-R-def)
qed

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

with p22
show ?thesis
  by auto

next
case False
with inv phase
show ?thesis
  by(auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def)
qed

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

theorem HEndPhase1-HInv5:
  assumes act: HEndPhase1 s s' p
  and inv: HInv5 s
  and inv1: Inv1 s
  and inv2a: Inv2a s
  and inv2a': Inv2a s'
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  and inv3: HInv3 s
  and inv4: HInv4 s
  shows HInv5-inner s' q
  using HEndPhase1-HInv5-p[OF act inv4 inv2a inv2a' inv2c]
lemma \textit{HFail-HInv5-p}:
\begin{align*}
H\text{Fail} \ s \ s' \ p & \implies H\text{Inv5-inner} \ s' \ p \\
\text{by}(\text{auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def})
\end{align*}

lemma \textit{HFail-blocksOf-q}:
\begin{align*}
\text{assumes act: }& H\text{Fail} \ s \ s' \ p \\
\text{and }& pnq: p \neq q \\
\text{shows }& \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \\
\text{using }& \text{assms} \\
\text{by}(\text{auto simp add: Fail-def InitializePhase-def blocksOf-def rdBy-def})
\end{align*}

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

lemma \textit{HFail-HInv5-q1}:
\begin{align*}
\text{assumes act: }& H\text{Fail} \ s \ s' \ p \\
\text{and }& pnq: p \neq q \\
\text{and inv2a: }& \text{Inv2a-inner} \ s' \ q \\
\text{and inv5-1: }& \text{maxBalInp} \ s \ (\text{bal}(\text{dblock} \ s \ q)) \ (\text{inp}(\text{dblock} \ s \ q)) \\
\text{shows }& \text{maxBalInp} \ s' \ (\text{bal}(\text{dblock} \ s' \ q)) \ (\text{inp}(\text{dblock} \ s' \ q)) \\
\text{proof}(\text{auto simp add: maxBalInp-def}) \\
\text{fix }& bk \\
\text{assume }& bk: bk \in \text{allBlocks} \ s' \\
& \text{and bal: } \text{bal} \ (\text{dblock} \ s' \ q) \leq \text{bal} \ bk
\end{align*}
from act pnq
have dblock': dblock s' q = dblock s q by (auto simp add: Fail-def)
from subsetD[OF HFail-allBlocks[OF act] bk]
show inp bk = inp (dblock s' q)
proof
  assume bk: bk ∈ allBlocks s
  with inv5-1 dblock' bal
  show ?thesis
    by (auto simp add: maxBalInp-def)
next
  assume bk: bk ∈ {dblock s' p}
  with act have bk-init: bk = InitDB
  with bal
  have bal (dblock s' q)=0
    by (auto simp add: InitDB-def)
  with inv2a
  have inp (dblock s' q)= NotAnInput
    by (auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with bk-init
  show ?thesis
    by (auto simp add: InitDB-def)
qed
qed
lemma HFail-HInv5-q2:
  assumes act: HFail s s' p
  and pnq: p≠q
  and inv5-2: ∃D∈MajoritySet. ∃qq.
    ( ∀ d∈D. bal(dblock s q) < mbal(disk s d qq)
     ∧ ¬hasRead s q d qq)
  shows ∃D∈MajoritySet. ∃qq.
    ( ∀ d∈D. bal(dblock s' q) < mbal(disk s' d qq)
     ∧ ¬hasRead s' q d qq)
proof
  from act pnq
  have disk: disk s' = disk s
    and blocksRead: ∀ d. blocksRead s' q d = blocksRead s q d
    and dblock: dblock s' q = dblock s q
    by (auto simp add: Fail-def InitializePhase-def)
  with inv5-2
  show ?thesis
    by (auto simp add: hasRead-def)
qed
lemma HFail-HInv5-q:
  assumes act: HFail s s' p
  and inv: HInv5-inner s q
  and pnq: p≠q
  and inv2a: Inv2a s'

shows $H_{\text{inv5-inner}} s' q$

proof (auto simp add: $H_{\text{inv5-inner-def}}$ $H_{\text{inv5-inner-R-def}}$)

assume phase': phase $s' q = 2$
and $nR2$: $\forall D \in \text{MajoritySet}$. $\forall qa. \exists d \in D. \text{bal}(\text{dblock} s' q) < \text{mbal}(\text{disk} s' d qa)$ $\rightarrow$ $\text{hasRead} s' q d qa$ (is $?P s'$)

from $H_{\text{fail-\text{inv5-q2}}} [\text{OF} \ act \ pnq]$
have $\neg (\neg P s) \implies \neg (\neg P s')$
y by auto

with $nR2$
have $P$: $?P s$
y by blast

from $\text{inv2a}$
have $\text{inv2a'}$: $\text{Inv2a-inner} s' q$ by (auto simp add: $\text{Inv2a-def}$)

from $\text{act \ pnq \ phase'}$
have phase $s q = 2$
y by (auto simp add: $\text{Fail-def}$ $\text{split: if-split-asm}$)

with $\text{inv} \ H_{\text{fail-\text{inv5-q1}}} [\text{OF} \ act \ pnq \ inv2a'] \ P$
show $\text{maxBalInp} s' (\text{bal}(\text{dblock} s' q)) (\text{inp}(\text{dblock} s' q))$
y by (auto simp add: $H_{\text{inv5-inner-def}}$ $H_{\text{inv5-inner-R-def}}$)

qed

theorem $H_{\text{fail-\text{inv5}}}$:

$[ \ H_{\text{fail}} s s' p; \ H_{\text{inv5-inner}} s q; \ H_{\text{inv2a}} s' q \ ] \implies H_{\text{inv5-inner}} s' q$
b y (blast dest: $H_{\text{fail-\text{inv5-q}}} \ H_{\text{fail-\text{inv5-p}}}$)

lemma $H_{\text{Phase0Read-\text{inv5-p}}}$:

$H_{\text{Phase0Read}} s s' p d \implies H_{\text{inv5-inner}} s' p$
b y (auto simp add: $H_{\text{Phase0Read-def}}$ $H_{\text{inv5-inner-def}}$)

lemma $H_{\text{Phase0Read-allBlocks}}$:

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

lemma $H_{\text{Phase0Read-\text{inv5-1}}}$:

assumes $\text{act}: H_{\text{Phase0Read}} s s' p d$
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 $\text{assms}$ and $H_{\text{Phase0Read-blocksOf}[\text{OF} \ act]}$
b y (auto simp add: $H_{\text{Phase0Read-def}}$ $H_{\text{maxBalInp-def}}$ $H_{\text{allBlocks-def}}$)

lemma $H_{\text{Phase0Read-\text{inv5-q2}}}$:

assumes $\text{act}: H_{\text{Phase0Read}} s s' p d$
and $\text{pnq}: p \neq q$
and $\text{inv5-2}: \exists D \in \text{MajoritySet}. \exists qq. (\forall d \in D. \text{bal}(\text{dblock} s q) < \text{mbal}(\text{disk} d s qq))$

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

proof –
from act pnq
have disk: $\text{disk} s' = \text{disk} s$
and blocksRead: $\forall d. \text{blocksRead} s' q d = \text{blocksRead} s q d$
and dblock: $\text{dblock} s' q = \text{dblock} s q$
by(auto simp add: 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: $\forall D \in \text{MajoritySet}. \forall qa. \exists d \in D. \text{bal}(\text{dblock}s' q d qa) < \text{mbal}(\text{disk} s' d qa)$
$\rightarrow \text{hasRead} s' q d qa$
from phase' act have phase: phase s q = 2
by(auto simp add: Phase0Read-def)
show maxBalInp s' ($\text{bal}(\text{dblock} s' q)) \ (\text{inp}(\text{dblock} s' q))$)
proof(rule HPhase0Read-HInv5-1[OF act, of q])
from HPhase0Read-HInv5-q[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:
$\left[ \begin{array}{c} \text{HPhase0Read} s s' p d; \text{HInv5-inner} s q \end{array} \right] \implies \text{HInv5-inner} s' q$
by(blast dest: HPhase0Read-HInv5-q HPhase0Read-HInv5-p)

lemma HEndPhase0-HInv5-p:
HEndPhase0 s s' p $\implies$ HInv5-inner s' p
by(auto simp add: EndPhase0-def HInv5-inner-def)

lemma HEndPhase0-blocksOf-q:
assumes act: HEndPhase0 s s' p
and pnq: p$\neq$q
shows blocksOf s' q $\subseteq$ blocksOf s q
proof –
from act pnq
have dblock: $\{\text{dblock} s' q\} \subseteq \{\text{dblock} s q\}$
and disk: $\text{disk} s' = \text{disk} s$
and blks: blocksRead s′ q = blocksRead s q
by (auto simp add: EndPhase0-def InitializePhase-def)
from disk
have disk': \{disk s d q | d ∈ UNIV\} ⊆ \{disk s d q | d ∈ UNIV\} (is ?D' ⊆ ?D)
  by auto
from pnq act
have \{\text{UN} qq d. rdBy s' q qq d\} ⊆ \{\text{UN} qq d. rdBy s q qq d\}
  by (auto simp add: EndPhase0-def InitializePhase-def rdBy-def split: if-split_asm, blast)
hence \{\text{block} br | \text{br} ∈ \{\text{UN} qq d. rdBy s' q qq d\}\} ⊆ \{\text{block} br | \text{br} ∈ \{\text{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
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 maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
proof(auto simp add: maxBalInp-def)
fix bk
assume bk: bk ∈ allBlocks s'
and bal: bal (dblock s' q) ≤ bal bk
from act pnq
have dblock': dblock s' q = dblock s q by(auto simp add: EndPhase0-def)
from subsetD[OF HEndPhase0-allBlocks[OF act] bk]
show inp bk = inp (dblock s' q)
proof
assume bk: bk ∈ allBlocks s
with inv5-1 dblock'
show thesis
by(auto simp add: maxBalInp-def)
next
assume bk: bk ∈ {dblock s' p}
with HEndPhase0-some[OF act inv1]
have ∃ ba∈allBlocksRead s p. bal ba = bal (dblock s' p) ∧ inp ba = inp (dblock s' p)
by(auto simp add: EndPhase0-def)
then obtain ba
where ba-blksread: ba∈allBlocksRead s p
and ba-balinp: bal ba = bal (dblock s' p) ∧ inp ba = inp (dblock s' p)
by auto
have allBlocksRead s p ⊆ allBlocks s
by(auto simp add: allBlocksRead-def allRdBlks-def
allBlocks-def blocksOf-def rdBy-def)
from subsetD[OF this ba-blksread] ba-balinp bal bk dblock' inv5-1
show thesis
by(auto simp add: maxBalInp-def)
qed
qed

lemma HEndPhase0-Hinv5-q2:
assumes act: HEndPhase0 s s' p
and pnq: p ≠ q
and inv5-2: ∃ D∈MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s d q) < mbal(disk s d)
q)
shows ∃ D∈MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s' d q) < mbal(disk s' d q)
∧ ¬hasRead s q d qq)
proof –
from act pnq
have disk: disk s' = disk s
and blocksRead: ∀ d. blocksRead s' q d = blocksRead s q d
and dblock: dblock s' q = dblock s q
by(auto simp add: EndPhase0-def InitializePhase-def)
with inv5-2
show thesis
by (auto simp add: hasRead-def)

qed

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

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

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

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

theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

C.6 Lemma I2f

To prove the final conjunct we will use the predicate valueChosen(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. (∀ d ∈ D. 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 bk-noninit act
show ?thesis
  by (auto simp add: EndPhase1-def)
qed

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

\textbf{lemma} \texttt{HEndPhase1-valueChosen2:}
\begin{align*}
\text{assumes act: \texttt{HEndPhase1 s s’ q} } \\
\text{and \texttt{asm4: \forall d \in D. } b \leq bal(disk s d p) } \\
\quad \land (\forall q. (\text{phase s q = 1} \\
\quad \land b \leq mbal(dblock s q) \\
\quad \land hasRead s q d p)
\rightarrow (\exists br \in blocksRead s q d. b \leq bal(block br))) (\text{is ?P s})
\end{align*}
shows \texttt {?P s’}
proof (auto)
fix \texttt{d}
assume \texttt{d: d \in D}
with \texttt{act asm4}
show \texttt{b \leq bal (disk s’ d p)}
by (auto simp add: EndPhase1-def)
fix \texttt{d q}
assume \texttt{d: d \in D}
and \texttt{phase’: phase s’ q = Suc 0}
and \( \text{dblk-mbal}: b \leq \text{mbal}(\text{dblock } s') \)

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

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

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

qed

theorem HEndPhase1-valueChosen:
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 \( \text{v-input: } v \in \text{Inputs} \)
shows \( \text{valueChosen } s' v \)

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

from \( \text{HEndPhase1-maxBalInp}[OF \, \text{act \, asm1 \, asm2 \, asm3 \, asm4 \, inv1 \, inv2a \, inv2b]} \)
have \( \text{maxBalInp } s' b v . \)
with \( \text{HEndPhase1-valueChosen2}[OF \, \text{act \, asm4]} \, \text{asm1 \, asm2} \)
show \( ?\text{thesis} \)
by (auto simp add: valueChosen-def)

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

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

with dblk-mbal

have b≤mbal(dblock s q) by auto

moreover

from act hasRead

have hasRead s q d p

by (auto simp add: StartBallot-def InitializePhase-def

hasRead-def split: if-split-asm)

ultimately

have ∃ br∈blocksRead s q d. b≤ bal(block br)

using p$j1 asm4 d

by blast

with act hasRead

show ∃ br∈blocksRead s′ q d. b≤ bal(block br)

by (auto simp add: StartBallot-def InitializePhase-def

hasRead-def)

qed

theorem HStartBallot-valueChosen:

assumes act: HStartBallot s s′ q

and vc: valueChosen s v

and v-input: v ∈ Inputs

shows valueChosen s′ v

proof -

from vc

obtain b p D where

asm1: b ∈ (UN p. Ballot p)

and asm2: D∈MajoritySet

and asm3: maxBalInp s b v

and asm4: ∀ d∈D. b ≤ bal(disk s d p)

∧ (∀ q. (phase s q = 1 

∧ b ≤ mbal(dblock s q)

∧ hasRead s q d p)

→ (∃ br∈blocksRead s q d. b ≤ bal(block br))))

by (auto simp add: valueChosen-def)

from HStartBallot-maxBalInp[OF act asm3]

have maxBalInp s′ b v .

with HStartBallot-valueChosen2[OF act asm4] asm1 asm2

show ?thesis

by (auto simp add: valueChosen-def)

qed

lemma HPhase1or2Write-maxBalInp:

assumes act: HPhase1or2Write s s′ q d

and asm3: maxBalInp s b v

shows maxBalInp s′ b v

proof (auto simp add: maxBalInp-def)

fix bk

assume bk: bk ∈ allBlocks s′
and \( b \)-bal: \( b \leq \text{bal} \) bk
from subsetD \( \text{OF Phase1or2Write-allBlocks[OF act]} \) bk
show \( \text{inp bk} = v \)
  by (auto simp add: maxBalInp-def)
qed

lemma \( \text{Phase1or2Write-valueChosen2:} \)
assumes \( \text{act: Phase1or2Write s s' pp d} \)
and \( \text{asm2: D\in MajoritySet} \)
and \( \text{asm4: } \forall \ d \in D. \ b \leq \text{bal(disk s d p)} \)
\[
\AND (\forall q . \text{phase s q = 1} \)
\AND (b \leq \text{mbal(dblock s q)})
\AND (\forall q . \text{hasRead s q d p})
\rightarrow (\exists \ br \in \text{blocksRead s q d. } b \leq \text{bal(block br)}) \) (is \( ?P s \))
and \( \text{inv4: HInv4a s pp} \)
shows \( ?P s' \)
proof (auto)
  fix d1
  assume \( d: d1 \in D \)
  show \( b \leq \text{bal(disk s' d1 p)} \)
  proof (cases d1 = d \AND pp = p)
    case True
    with \( \text{inv4 act} \)
    have \( \text{HInv4a2 s p} \)
    by (auto simp add: Phase1or2Write-def HInv4a-def)
    with \( \text{asm2 majority-intersect} \)
    have \( \exists \ dd \in D. \ \text{bal(disk s dd p) \leq bal(dblock s p)} \)
    by (auto simp add: HInv4a2-def MajoritySet-def)
    then obtain \( \text{dd where p41: } \dd \in D \ \AND \ \text{bal(disk s dd p) \leq bal(dblock s p)} \)
    by auto
    from \( \text{asm4 p41} \)
    have \( b \leq \text{bal(disk s dd p)} \)
    by auto
    with \( \text{p41} \)
    have \( \text{p42: } b \leq \text{bal(dblock s p)} \)
    by auto
    from \( \text{act True} \)
    have \( \text{dblock s p = disk s' d p} \)
    by (auto simp add: Phase1or2Write-def)
    with \( \text{p42 True} \)
    show \( ?\text{thesis} \)
    by auto
  next
  case False
  with \( \text{act asm4 d} \)
  show \( ?\text{thesis} \)
  by (auto simp add: Phase1or2Write-def)
qed
next
fix \( d \) \( q \)

**assume** \( d : d \in D \)

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

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

**and** hasRead: hasRead \( s' \) \( q \) \( d \) \( p \)

**from** \( \text{phase'} \) \( \text{act} \) hasRead

**have** \( p31: \text{phase} \, s \, q = 1 \)

**and** \( p32: \text{dblock} \, s' \, q = \text{dblock} \, s \, q \)

by (auto simp add: Phase1or2Write-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** \( \text{act} \) hasRead

**have** hasRead \( s \) \( q \) \( d \) \( p \)

by (auto simp add: Phase1or2Write-def InitializePhase-def

hasRead-def split : if-split-asm)

**ultimately**

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

**using** \( p31 \) \( \text{asm}4 \) \( d \)

**by** blast

**with** \( \text{act} \) hasRead

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

**by** (auto simp add: Phase1or2Write-def InitializePhase-def

hasRead-def)

**qed**

**theorem** HPhase1or2Write-valueChosen:

**assumes** act: HPhase1or2Write \( s \, s' \, q \, d \)

**and** valueChosen: \( v \) \( \in \text{Inputs} \)

**and** \( v' = \text{valueChosen} \, s \, v \)

**shows** \( v' = \text{valueChosen} \, s' \, v \)

**proof** —

**from** \( v \)

**obtain** \( b \) \( d \) \( D \) **where**

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

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

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

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

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

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

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

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

by (auto simp add: valueChosen-def)

**from** HPhase1or2Write-maxBalInp[OF act asm3]

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

**with** HPhase1or2Write-valueChosen2[OF act asm2 asm4 inv4] \( \text{asm}1 \) \( \text{asm}2 \)

**show** \( \text{?thesis} \sto 124 \)
by (auto simp add: valueChosen-def)

qed

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

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

theorem HPhase1or2ReadThen-valueChosen:
assumes act: HPhase1or2ReadThen s s' q d p
and vc: valueChosen s v
and v-input: v ∈ 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 HPhase1or2ReadThen-maxBalInp[OF act asm3]
have maxBalInp s' b v .
with HPhase1or2ReadThen-valueChosen2[OF act asm4] asm1 asm2
show ?thesis
by(auto simp add: valueChosen-def)
qed

theorem HPhase1or2ReadElse-valueChosen:
[ HPhase1or2ReadElse s s' p d r; valueChosen s v; v∈ Inputs ]
⇒ 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 \( \exists \, br \in \text{blocksRead} \, s' \, q \, d \cdot b \leq \text{bal(block br)} \)

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

qed

theorem \text{HEndPhase2-valueChosen}:  
assumes \text{act}: \text{HEndPhase2} \, s \, s' \, q  
and \text{vc}: \text{valueChosen} \, s \, v  
and \text{v-input}: \, v \in \text{Inputs}  
shows \text{valueChosen} \, s' \, v  

proof – 
  from \text{vc} 
  obtain \, b \, p \, D \ where  
  \text{asm1}: \, b \in (\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(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 \, br \in \text{blocksRead} \, s \, q \, d \cdot b \leq \text{bal(block br)}))  
  by (auto simp add: \text{valueChosen-def})
  from \text{HEndPhase2-maxBalInp}[OF \text{act} \, \text{asm3}]  
  have \text{maxBalInp} \, s' \, b \, v .  
  with \text{HEndPhase2-valueChosen2}[OF \text{act} \, \text{asm4}] \, \text{asm1} \, \text{asm2}  
  show ?thesis  
  by (auto simp add: \text{valueChosen-def})

qed

lemma \text{HFail-maxBalInp}:  
assumes \text{act}: \text{HFail} \, s \, s' \, q  
and \text{asm1}: \, b \in (\text{UN} \, p. \, \text{Ballot} \, p)  
and \text{asm3}: \, \text{maxBalInp} \, s \, b \, v  
shows \text{maxBalInp} \, s' \, b \, v  

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

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

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

qed

theorem HFail-valueChosen:
  assumes act: HFail s s' q
  and vc: valueChosen s v
  and v-input: v ∈ Inputs
  shows valueChosen s' v

proof -
  from vc
  obtain b p D where
    asm1: b ∈ (UN p. Ballot p)
    and asm2: D ∈ MajoritySet
    and asm3: maxBalInp s b v
  and asm4: ∀ d ∈ D. b ≤ bal (disk s d p)
    ∧ (∀ q. (phase s q = 1
    ∧ b ≤ mbal (dblock s q)
    ∧ hasRead s q d p
    ) −→ (∃ br ∈ blocksRead s q d. b ≤ bal (block br)))

  by (auto simp add: valueChosen-def)

  from HFail-maxBalInp[OF act asm1 asm3]
  have maxBalInp s' b v.
  with HFail-valueChosen2[OF act asm4] asm1 asm2
  show ?thesis
    by (auto simp add: valueChosen-def)

qed

lemma HPhase0Read-maxBalInp:
  assumes act: HPhase0Read s s' q d
  and asm3: maxBalInp s b v
  shows maxBalInp s' b v

proof (auto simp add: maxBalInp-def)

fix bk

assume bk: bk ∈ allBlocks s'
  and b-bal: b ≤ bal bk

from subsetD[OF HPhase0Read-allBlocks[OF act] bk] asm3 b-bal

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

qed

lemma HPhase0Read-valueChosen2:
  assumes act: HPhase0Read s s' qq dd
  and asm4: ∀ d ∈ D. b ≤ bal (disk s d p)
    ∧ (∀ q. (phase s q = 1
    ∧ b ≤ mbal (dblock s q)
    ∧ hasRead s q d p
    ) −→ (∃ br ∈ blocksRead s q d. b ≤ bal (block br))) (is ?P s)
  shows ?P s'

proof (auto)
fix 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
qed

theorem HPhase0Read-valueChosen:
  assumes act: HPhase0Read s s' q d
  and vc: valueChosen s v
  and v-input: v ∈ Inputs
  shows valueChosen s' v
proof –
from vc
obtain b p D where
  asm1: b ∈ (UN p. Ballot p)
and asm2: D∈MajoritySet
and asm3: maxBalInp s b v
and \text{asm4}: \forall d \in D. \ b \leq \text{bal}(\text{disk} \ s \ d \ p) \\
\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))) \\
\text{by (auto simp add: valueChosen-def)} \\
\text{from HPhase0Read-maxBalInp[OF act asm3]} \\
\text{have maxBalInp s' b v}.

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

\text{lemma HEndPhase0-maxBalInp:} \\
\text{assumes act: HEndPhase0 s s' q} \\
\text{and \text{asm3}: maxBalInp s b v} \\
\text{and inv1: Inv1 s} \\
\text{shows maxBalInp s' b v} \\
\text{proof (auto simp add: maxBalInp-def)} \\
\text{fix bk} \\
\text{assume bk: bk \in allBlocks s'} \\
\text{and b-bal: b \leq bal bk} \\
\text{from subsetD[OF HEndPhase0-allBlocks[OF act] bk]} \\
\text{show inp bk = v} \\
\text{proof} \\
\text{assume bk: bk \in allBlocks s} \\
\text{with asm3 b-bal} \\
\text{show ?thesis} \\
\text{by (auto simp add: maxBalInp-def)} \\
\text{next} \\
\text{assume bk: bk \in \{dblock s' q\}} \\
\text{with HEndPhase0-some[OF act inv1] act} \\
\text{have \exists ba \in allBlocksRead s q. bal ba = bal (dblock s' q) \land inp ba = inp (dblock s' q)} \\
\text{by (auto simp add: EndPhase0-def)} \\
\text{then obtain ba} \\
\text{where ba-blksread: ba \in allBlocksRead s q} \\
\text{and ba-balinp: bal ba = bal (dblock s' q) \land inp ba = inp (dblock s' q)} \\
\text{by auto} \\
\text{have allBlocksRead s q \subseteq allBlocks s} \\
\text{by (auto simp add: allBlocksRead-def allRdBlks-def} \\
\text{ allBlocks-def blocksOf-def rdBy-def)} \\
\text{from subsetD[OF this ba-blksread] ba-balinp bk b-bal asm3} \\
\text{show ?thesis} \\
\text{by (auto simp add: maxBalInp-def)} \\
\text{qed} \\
\text{qed}
Lemma \( \text{HEndPhase0-valueChosen2} \):

Assumes \( \text{act: HEndPhase0 } s \ s' \ q \)

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

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

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

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

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

Shows \( ?P s' \)

Proof (auto)

Fix \( d \)

Assume \( d \in D \)

With \( \text{act asm4} \)

Show \( b \leq \text{bal} \text{(disk } s' \ d \ p) \)

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

Fix \( q \)

Assume \( d \in D \)

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

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

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

From \( \text{phase'} \text{ act hasRead} \)

Have \( p31: \text{phase } s \ q = 1 \)

And \( p32: \text{dblock } s' \ q = \text{dblock } s \ q \)

By (auto simp add: \text{EndPhase0-def InitializePhase-def hasRead-def split : if-split-asm})

With \( \text{dblk-mbal} \)

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

Moreover

From \( \text{act hasRead} \)

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

By (auto simp add: \text{EndPhase0-def InitializePhase-def hasRead-def split: if-split-asm})

Ultimately

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

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

By blast

With \( \text{act hasRead} \)

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

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

qed

Theorem \( \text{HEndPhase0-valueChosen} \):

Assumes \( \text{act: HEndPhase0 } s \ s' \ q \)

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

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

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

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

Proof –

From \( \text{vc} \)
obtain \( b \ p \ D \) where
- \( \text{asm1}: b \in (\text{UN p. Ballot p}) \)
- \( \text{asm2}: D \in \text{MajoritySet} \)
- \( \text{asm3}: \text{maxBalInp} \ s \ b \ v \)
- \( \text{asm4}: \forall \ d \in D. \ b \leq \text{bal}(\text{disk s d p}) \)

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

by (auto simp add: valueChosen-def)

from HEndPhase0-maxBalInp[OF act asm3 inv1]

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

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, \( \text{valueChosen}(\text{chosen}) \) holds, and each processor’s output equals either \( \text{chosen} \) or \( \text{NotAnInput} \).

definition \( HInv6 :: \text{state} \Rightarrow \text{bool} \) where
\( HInv6 \ s = ((\text{chosen s} \neq \text{NotAnInput} \rightarrow \text{valueChosen s (chosen s))} \land (\forall \ p. \text{outpt s p} \in \{\text{chosen s}, \text{NotAnInput}\})) \)

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

lemma HEndPhase2-Inv6-1:
assumes 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': \( \text{chosen s'} \neq \text{NotAnInput} \)
shows \( \text{valueChosen} \ s' (\text{chosen s'}) \)
proof (cases chosen \( s = \text{NotAnInput} \))
from inv5 act
have inv5R: \( \text{HInv5-inner-R s} \ p \)
and phase: \( \text{phase s p} = 2 \)
and ep2-maj: \( \text{IsMajority} \ \{d. \ d \in \text{disksWritten s p} \} \)

end

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

C.7 Invariant 6

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

definition \( HInv6 :: \text{state} \Rightarrow \text{bool} \) where
\( HInv6 \ s = ((\text{chosen s} \neq \text{NotAnInput} \rightarrow \text{valueChosen s (chosen s))} \land (\forall \ p. \text{outpt s p} \in \{\text{chosen s}, \text{NotAnInput}\})) \)

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

lemma HEndPhase2-Inv6-1:
assumes 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': \( \text{chosen s'} \neq \text{NotAnInput} \)
shows \( \text{valueChosen} \ s' (\text{chosen s'}) \)
proof (cases chosen \( s = \text{NotAnInput} \))
from inv5 act
have inv5R: \( \text{HInv5-inner-R s} \ p \)
and phase: \( \text{phase s p} = 2 \)
and ep2-maj: \( \text{IsMajority} \ \{d. \ d \in \text{disksWritten s p} \} \)

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

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

\textbf{case True}

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

\textbf{proof}−

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

\textbf{proof} auto

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

\textbf{from \(ep2-maj\ Dmaj\ majorities-intersect\)

\textbf{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 \(\text{dinD}\)

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

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

by auto

\textbf{from \(inv2b\)

\textbf{have} \(\text{Inv2b-inner} s p d\)

by(auto simp add: Inv2b-def)

with \(ddisk\)

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

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

with \(inv2c\) phase

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

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

with \(dhasR\ \text{dinD}\)

\textbf{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

\textbf{qed}

\textbf{with \(inv5R\)

\textbf{show} ?thesis

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

\textbf{qed}

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

\textbf{proof}−

\textbf{from \(act\)

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

by(auto simp add: EndPhase2-def)

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

\textbf{proof} auto

\textbf{fix} \(q\)

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

\textbf{from \(outpt'\ pnq\)

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

by(auto simp add: EndPhase2-def)

\textbf{with \(True\ inv2c\)

\textbf{135}
show outpt s' q = NotAnInput
  by(auto simp add: Inv2c-def Inv2c-inner-def)
q)

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: ∃ pa. outpt s' pa ≠ NotAnInput =⇒ pa = p
    by(auto)
  with outpt'-pa
  have outpt s' p ≠ NotAnInput
    by(auto)
  from some-equality[of λp. 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)
q)
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)
q)

from ep2-maj inv2b majorities-intersect
have ∃ D ∈ MajoritySet. (∀ d ∈ D. disk s d p = dblock s p
  ∧ (∀ q ∈ UNIV - {p}. hasRead s p d q))
  by(auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def)
then obtain D
  where Dmaj: D ∈ MajoritySet
  and p34: ∀ d ∈ D. disk s d p = dblock s p
  ∧ (∀ q ∈ UNIV - {p}. hasRead s p d q)
  by(auto)
have p35: ∀ q. ∀ d ∈ D. (phase s q = 1 ∧ bal(dblock s p) ≤ mbal(dblock s q) ∧ hasRead s q d p)
  =⇒ (∃ block = dblock s p, proc = p) ∈ blocksRead s q d

proof auto
  fix q d
  assume dD: d ∈ 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 \( \text{bal}(\text{dblock } s \; p) = \text{mbal}(\text{dblock } s \; p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)

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

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

from phase phase-q
have \( p \neq q \) by auto

with \( p34 \; d\overline{D} \)
have \( \text{hasRead } s \; p \; d \)
by auto

with phase phase-q hasRead \( \text{inv3} \) \( p41 \)
show \( (\exists \text{br} \in \text{blocksRead } s \; q \; d. \; \text{bal}(\text{block } br) = \text{bal}(\text{dblock } s \; p)) \)

proof (auto)
fix \( q \; d \)
assume \( d\overline{D}: d \in D \) 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 \;
from phase-q act
have phase \( s' \; q = \text{phase } s \; q \) and \( \text{dblock } s' \; q = \text{dblock } s \; q \) and \( \text{hasRead } s' \; q \; d \)
by (auto simp add: EndPhase2-def hasRead-def InitializePhase-def)

with \( p35 \; \text{phase}-q \) bal hasRead \( d\overline{D} \)
have \( (\text{block}=\text{dblock } s \; p, \; \text{proc}=p) \in \text{blocksRead } s' \; q \; d \)
by auto
thus \( \exists \text{br} \in \text{blocksRead } s' \; q \; d. \; \text{bal}(\text{block } br) = \text{bal}(\text{dblock } s \; p) \)
by force

qed

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

hasRead \( s' \; q \; d \)

by force

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

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

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

lemma valueChosen-equal:
assumes v: valueChosen s v
and w: valueChosen s w
shows $v = w$ using assms
proof (auto simp add: valueChosen-def)
fix a b aa ba p D pa Da
assume max-v: maxBalInp s b v
  and Dmaj: $D \in MajoritySet$
  and asm-v: $\forall d \in D. \ b \leq bal (disk s d p)$ \land
    ($\forall q. \ phase s q = Suc 0 \land$
      $b \leq mbal (dblock s q) \land hasRead s q d p \longrightarrow$
      $(\exists br \in blocksRead s q d. \ b \leq bal (block br)))$
and max-w: maxBalInp s ba w
  and Damaj: $Da \in MajoritySet$
  and asm-w: $\forall d \in Da. \ ba \leq bal (disk s d pa)$ \land
    ($\forall q. \ phase s q = Suc 0 \land$
     $ba \leq mbal (dblock s q) \land hasRead s q d pa \longrightarrow$
     $(\exists br \in blocksRead s q d. \ ba \leq bal (block br)))$
from asm-v
have asm-v: $\forall d \in D. \ b \leq bal (disk s d p)$ by auto
from asm-w
have asm-w: $\forall d \in Da. \ ba \leq bal (disk s d pa)$ by auto
show $v = w$
proof (cases $b \leq ba$)
case True
  from valueChosen-equal-case[OF max-v Dmaj asm-v max-w Damaj asm-w True]
  show ?thesis .
next
case False
  from valueChosen-equal-case[OF max-w Damaj asm-w max-v Dmaj asm-v]
False
show \(?thesis\)
  by auto
qed

lemma \(H\text{EndPhase2-Inv6-2}\):
  assumes \(act: H\text{EndPhase2} s s' p\)
  and \(inv: H\text{Inv6} s\)
  and \(inv2b: Inv2b s\)
  and \(inv2c: Inv2c s\)
  and \(inv3: H\text{Inv3} s\)
  and \(inv5: H\text{Inv5-inner} s p\)
  and \(asm: \text{outpt} s' r \neq \text{NotAnInput}\)
  shows \(\text{outpt} s' r = \text{chosen} s'\)
proof(cases \(\text{chosen} s = \text{NotAnInput}\))
case \(True\)
  with \(inv2c\)
  have \(\forall q. \text{outpt} s q = \text{NotAnInput}\)
    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 \(p31: \text{valueChosen} s (\text{chosen} s)\)
    by(auto simp add: \(H\text{Inv6}\)-def)
  with \(False\) \(act\)
  have \(\text{chosen} s' \neq \text{NotAnInput}\)
    by(auto simp add: \(HNextPart\)-def)
  from \(H\text{EndPhase2-Inv6-1}\[OF act inv inv2b inv2c inv3 inv5 this]\]
  have \(p32: \text{valueChosen} s'(\text{chosen} s')\).
  from \(False\) \(\text{InputsOrNi}\)
  have \(\text{chosen} s \in \text{Inputs}\) by auto
  from \(\text{valueChosen-equal}[OF H\text{EndPhase2-valueChosen}[OF act p31 this] p32]\)
  have \(p33: \text{chosen} s = \text{chosen} s'\).
  from \(act\)
  have \(maj: \text{IsMajority} \{d. d \in \text{disksWritten} s p\}
    \land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead} s p d q)\) (is \(\text{IsMajority} ?D)\)
    and \(phase: \text{phase} s p = 2\)
    by(auto simp add: \(EndPhase2\)-def)
  show \(?thesis\)
proof(cases \(\text{outpt} s r = \text{NotAnInput}\))
case \(True\)
  with \(asm\) \(act\)
  have \(p41: r=p\)
    by(auto simp add: \(EndPhase2\)-def split: if-split-asm)
  from \(maj\)
have \( p_{42} \) : \( \exists D \in \text{MajoritySet}. \forall d \in D. \forall q \in \text{UNIV} - \{ p \}. \text{hasRead } s \ p \ d \ q \)
by(auto simp add: MajoritySet-def)

have \( p_{43} \) : \( \neg \left( \exists D \in \text{MajoritySet}. \forall d \in D. \forall q \in \text{UNIV} - \{ p \}. \right) \left( \text{bal(dblock } s \ p \ d \ q \rangle < \text{mbal(disk } s \ d \ q \rangle \right) \wedge \neg\text{hasRead } s \ p \ d \ q \}

proof auto
fix \( D \) \( q \)
assume \( Dmaj \) : \( D \in \text{MajoritySet} \)
show \( \exists d \in D. \text{bal(dblock } s \ p \rangle < \text{mbal(disk } s \ d \ q \rangle \rightarrow \text{hasRead } s \ p \ d \ q \)
proof(cases \( p = q \))
assume \( pq \) : \( p = q \)
thus \(?thesis \)
proof auto
from \( \text{maj majorities-intersect } Dmaj \)
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 \( d \in \text{disksWritten } s \ p \) \text{ and } \( d \in ?D \cap D \)
by auto
hence \( dD \) : \( d \in D \) by auto
from \( d \) inv2b
have disk \( s \ d \ p = \text{dblock } s \ p \) 
by(auto simp add: Inv2b-def Inv2b-inner-def)
with \( \text{inv2c phase} \)
have \( \text{bal(dblock } s \ p \rangle = \text{mbal(disk } s \ d \ p \rangle \) 
by(auto simp add: Inv2c-def Inv2c-inner-def)
with \( dD \) \( pq \)
show \( \exists d \in D. \text{bal(dblock } s \ q \rangle < \text{mbal(disk } s \ d \ q \rangle \rightarrow \text{hasRead } s \ q \ d \ q \)
by auto
qed
next
case False
with \( p_{42} \)
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \text{hasRead } s \ p \ d \ q \)
by auto
with \( \text{majorities-intersect } Dmaj \)
show ?thesis 
by(auto simp add: MajoritySet-def, blast)
qed
qed
with \( \text{inv5 act} \)
have \( p_{44} \) : \( \text{maxBalInp } s \ (\text{bal(dblock } s \ p \rangle) (\text{inp(dblock } s \ p \rangle) \) 
by(auto simp add: EndPhase2-def HInv5-inner-def HInv5-inner-R-def)

have \( \exists b \in \text{allBlocks } s. \exists b \in \text{UN } p. \text{Ballot } p. (\text{maxBalInp } s \ b \ (\text{chosen } s)) \wedge b \leq \text{bal } bk \)
proof
have disk-allblks : \( \forall d \ p. \text{disk } s \ d \ p \in \text{allBlocks } s \)
by(auto simp add: allBlocks-def blocksOf-def)
from p31
have \( \exists b \in (\text{UN } p. \text{ Ballot } p), \maxBalInp s b \text{ (chosen s)} \land \)
(\( \exists p. \exists D \in \text{MajoritySet} \land (\forall d \in D. \ b \leq \text{bal}(\text{disk } s d p)) \))
  by (auto simp add: valueChosen-def, force)
with majority-nonempty obtain \( b, p, D, d \)
  where IsMajority \( D \land b \in (\text{UN } p. \text{ Ballot } p) \land \)
  \( \maxBalInp s b \text{ (chosen s)} \land d \in D \land b \leq \text{bal}(\text{disk } s d p) \)
  by (auto simp add: MajoritySet-def, blast)
with disk-allblks
show ?thesis
  by (auto)
qed
then obtain \( bk, b \)
  where p45-bk: \( bk \in \text{allBlocks } s \land b \leq \text{bal } bk \)
  and p45-b: \( b \in (\text{UN } p. \text{ Ballot } p) \land (\maxBalInp s b \text{ (chosen s)}) \)
  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 ?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 ?thesis
  by (auto simp add: maxBalInp-def)
qed
with p41 p33 act
show ?thesis
  by (auto simp add: EndPhase2-def)
next
case False
from inv2c
have Inv2c-inner \( s r \)
  by (auto simp add: Inv2c-def)
with False asm inv2c act
have outpt \( s' r = \text{outpt } s r \)
  by (auto simp add: Inv2c-inner-def, EndPhase2-def
  split: if-split-asm)
with inv p33 False
show ?thesis
  by (auto simp add: HInv6-def)
qed
**theorem HEndPhase2-Inv6:**

- **assumes** act: HEndPhase2 s s' p
- and inv: HInv6 s
- and inv2b: Inv2b s
- and inv2c: Inv2c s
- and inv3: HInv3 s
- and inv5: HInv5-inner s p

**shows** HInv6 s'

**proof** (auto simp add: HInv6-def)

**assume** chosen s' ≠ NotAnInput

from HEndPhase2-Inv6-1[OF act inv inv2b inv2c inv3 inv5 this]

**show** valueChosen s' (chosen s')

**next**

**fix** p

**assume** outpt s' p ≠ NotAnInput

from HEndPhase2-Inv6-2[OF act inv inv2b inv2c inv3 inv5 this]

**show** outpt s' p = chosen s'

qed

**lemma outpt-chosen:**

- **assumes** outpt: outpt s = outpt s'
- and inv2c: Inv2c s
- and nextp: HNextPart s s'

**shows** chosen s' = chosen s

**proof** –

**from** inv2c

**have** chosen s = NotAnInput → (∀ p. outpt s p = NotAnInput)

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

with outpt nextp

**show** ?thesis

by (auto simp add: HNextPart-def)

qed

**lemma outpt-Inv6:**

\[
\text{outpt} s = \text{outpt} s' \land \forall p, \text{outpt} s p \in \{\text{chosen} s, \text{NotAnInput}\}.
\]

**using** outpt-chosen.

**by** auto

**theorem HStartBallot-Inv6:**

- **assumes** act: HStartBallot s s' p
- and inv: HInv6 s
- and inv2c: Inv2c s

**shows** HInv6 s'

**proof** –

**from** outpt-chosen act inv2c inv

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

theorem HPhase1or2Write-Inv6:
  assumes act: HPhase1or2Write s s' p d
  and inv: HInv6 s
  and inv4: HInv4a s p
  and inv2c: Inv2c s
  shows HInv6 s'
proof –
  from outpt-chosen act inv2c inv
  have chosen s' ≠ NotAnInput → valueChosen s (chosen s')
    by(auto simp add: Phase1or2Write-def HInv6-def)
  from HPhase1or2Write-valueChosen[OF act] inv4 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 \( t_1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \,(\text{chosen } s') \)
by auto
from act
have \( \text{outpt } s = \text{outpt } s' \)
by(auto simp add: \text{Phase1or2ReadThen-def})
from \text{outpt-Inv6[OF outpt]} \, act \text{inv2c inv}
have \( \forall p. \, \text{outpt } s' \, p = \text{chosen } s' \lor \text{outpt } s' \, p = \text{NotAnInput} \)
by(auto simp add: \text{HInv6-def})
with \( t_1 \)
show \( \text{thesis} \)
by(simp add: \text{HInv6-def})
qed

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

theorem \text{HEndPhase1-Inv6}:  
assumes act: \text{HEndPhase1 } s \, s' \, p  
and inv: \text{HInv6 } s  
and inv1: \text{Inv1 } s  
and inv2a: \text{Inv2a } s  
and inv2b: \text{Inv2b } s  
and inv2c: \text{Inv2c } s  
shows \text{HInv6 } s'  
proof --  
from \text{outpt-chosen act inv2c inv}
have \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \,(\text{chosen } s') \)
by(auto simp add: \text{EndPhase1-def} \text{HInv6-def})
from \text{HEndPhase1-valueChosen[OF act]} \, \text{inv1 inv2a inv2b this InputsOrNi}  
have \( t_1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \,(\text{chosen } s') \)
by auto
from act
have \( \text{outpt } s = \text{outpt } s' \)
by(auto simp add: \text{EndPhase1-def})
from \text{outpt-Inv6[OF outpt]} \, act \text{inv2c inv}
have \( \forall p. \, \text{outpt } s' \, p = \text{chosen } s' \lor \text{outpt } s' \, p = \text{NotAnInput} \)
by(auto simp add: \text{HInv6-def})
with \( t_1 \)
show \( \text{thesis} \)
by(simp add: \text{HInv6-def})
qed

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

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

theorem \( \text{HFail-Inv6}: \)
assumes \( \text{act: HFail } s s' p \)
and \( \text{inv: Hinv6 } s \)
and \( \text{inv2c: Inv2c } s \)
shows \( \text{Hinv6 } s' \)
proof –
from \( \text{outpt-chosen-2 act inv2c inv} \)
have \( \text{chosen } s' \neq \text{NotAnInput } \rightarrow \text{valueChosen } s (\text{chosen } s') \)
by (auto simp add: Fail-def Hinv6-def)
from \( \text{HFail-valueChosen}[\text{OF act}] \) this \( \text{InputsOrNi} \)
have \( \text{t1: chosen } s' \neq \text{NotAnInput } \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
by auto
from \( \text{act} \)
have \( \text{outpt: outpt } s' = (\text{outpt } s) (p:=\text{NotAnInput}) \)
by (auto simp add: Fail-def)
from \( \text{outpt-Hinv6-2}[\text{OF outpt}] \) act \( \text{inv2c inv} \)
have \( \forall p. \text{outpt } s' \ p = \text{chosen } s' \lor \text{outpt } s' \ p = \text{NotAnInput} \)
by (auto simp add: Hinv6-def)
with \( \text{t1} \)
show \( \text{?thesis} \)
by (simp add: Hinv6-def)
qed
theorem HPhase0Read-Inv6:
assumes act: HPhase0Read s s' p d
and inv: HInv6 s
and inv2c: Inv2c s
shows HInv6 s'
proof
  from outpt-chosen act inv2c inv
  have chosen s' \neq NotAnInput \rightarrow valueChosen s (chosen s')
    by (auto simp add: Phase0Read-def HInv6-def)
  from HPhase0Read-valueChosen[OF act] this InputsOrNi
  have t1: chosen s' \neq NotAnInput \rightarrow valueChosen s' (chosen s')
    by auto
  from act
  have outpt: outpt s = outpt s'
    by (auto simp add: Phase0Read-def)
  from outpt-Inv6[OF outpt] act inv2c inv
  have \forall p. outpt s' p = chosen s' \lor outpt s' p = NotAnInput
    by (auto simp add: HInv6-def)
  with t1
  show ?thesis
    by (simp add: HInv6-def)
qed

HInv1 \land HInv2 \land HInv2' \land HInv3 \land HInv4 \land HInv5 \land HInv6 is an invariant of HNext.
lemma \textbf{I2f}: \\
\textbf{assumes} \texttt{nxt:} \quad \texttt{HNext \; s \; s'} \\
\textbf{and} \quad \texttt{inv:} \quad \texttt{HInv1 \; s} \; \land \; \texttt{HInv2 \; s} \; \land \; \texttt{HInv3 \; s} \; \land \; \texttt{HInv4 \; s} \; \land \; \texttt{HInv5 \; s} \; \land \\
\texttt{HInv6 \; s} \\
\textbf{shows} \quad \texttt{HInv6 \; s'} \; \textbf{using} \; \texttt{assms} \\
\textit{by} (\texttt{auto simp add: HNext-def Next-def,} \\
\texttt{auto simp add: HInv2-def intro: HStartBallot-Inv6,} \\
\texttt{auto intro: HPhase0Read-Inv6,} \\
\texttt{auto simp add: HInv4-def intro: HPhase1or2Write-Inv6,} \\
\texttt{auto simp add: Phase1or2Read-def} \\
\texttt{intro: HPhase1or2ReadThen-Inv6} \\
\texttt{HPhase1or2ReadElse-Inv6,} \\
\texttt{auto simp add: EndPhase1or2-def HInv1-def HInv5-def} \\
\texttt{intro: HEndPhase1-Inv6} \\
\texttt{HEndPhase2-Inv6,} \\
\texttt{auto intro: HFail-Inv6,} \\
\texttt{auto intro: HEndPhase0-Inv6}) \\
\textit{end} \\

\textbf{theory} \; \textit{DiskPaxos-Invariant} \; \textbf{imports} \; \textit{DiskPaxos-Inv6} \; \textbf{begin} \\

\textbf{C.8 \; The Complete Invariant} \\

definition \texttt{HInv :: \textit{state} ⇒ bool} \\
\textbf{where} \\
\texttt{HInv \; s \; = \; (HInv1 \; s} \\
\texttt{\land \; HInv2 \; s} \\
\texttt{\land \; HInv3 \; s} \\
\texttt{\land \; HInv4 \; s} \\
\texttt{\land \; HInv5 \; s} \\
\texttt{\land \; HInv6 \; s)} \\

\textbf{theorem} \textbf{I1:} \\
\texttt{HInit \; s} \; \Longrightarrow \; \texttt{HInv \; s} \\
\textbf{using} \; \texttt{HInit-HInv1 \; HInit-HInv2 \; HInit-HInv3} \\
\texttt{HInit-HInv4 \; HInit-HInv5 \; HInit-HInv6} \\
\textit{by} (\texttt{auto simp add: HInv-def}) \\

\textbf{theorem} \textbf{I2:} \\
\textbf{assumes} \texttt{inv:} \; \texttt{HInv \; s} \\
\textbf{and} \; \texttt{nxt:} \; \texttt{HNext \; s \; s'} \\
\textbf{shows} \; \texttt{HInv \; s'} \\
\textbf{using} \; \texttt{inv \; I2a[OF \; nxt]} \; \texttt{ I2b[OF \; nxt]} \; \texttt{ I2c[OF \; nxt]} \\
\texttt{ I2d[OF \; nxt]} \; \texttt{ I2e[OF \; nxt]} \; \texttt{ I2f[OF \; nxt]} \\
\textit{by} (\texttt{simp add: HInv-def}) \\

148
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 = ⟨|input = inpt s, ioutput = outpt s,  ichosen=chosen s,  iallInput = allInput s⟩

theorem R1:
\[ \text{HInit } s; \text{ is } = s2is \ s \implies \text{HInit is} \]

by (auto simp add: HInit-def HInit-def s2is-def Init-def)

**Theorem R2b:**

**Assumes** inv: HInv s

and inv': HInv s'

and nxt: HNext s s'

and srel: is = s2is s \& is' = s2is s'

**Shows** (\exists p. IFail is is' \lor IChoose is is' \lor is = is')

**Proof** (auto)

**Assume** chg-vars: is \neq is'

with srel

have s-change: inpt s \neq inpt s' \lor outpt s \neq outpt s'

\lor chosen s \neq chosen s' \lor allInput s \neq allInput s'

by (auto simp add: s2is-def)

from inv have inv2c5: \forall p. inpt s p \in allInput s

\&(chosen s = NotAnInput \implies 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' \neq inpt s \lor outpt s' \neq outpt s

by (auto simp add: HNext-def Next-def HNextPart-def)

with nxt

have \exists p. Fail s s' p \lor EndPhase2 s s' p

by (auto simp add: HNext-def Next-def)

StartBallot-def Phase0Read-def Phase1or2Write-def

Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def

EndPhase1or2-def EndPhase1-def EndPhase0-def)

then obtain p where fail-or-endphase2: Fail s s' p \lor 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 \lor 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 \cup (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 ip′ = (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-uni: 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: (∀x. outpt s' x ≠ NotAnInput → x = p)
by auto

from p32 True nxt some-equality[of λp. outpt s' p ≠ NotAnInput, OF outpt-nni
some-eq]
have p33: chosen s' = outpt s' p
by (auto simp add: HNext-def HNextPart-def)
with endphase2
have chosen s' = inp (dblock s p) ∧ outpt s' = (outpt s) (p := inp (dblock s p))
by (auto simp add: EndPhase2-def)
with True p22
have if (chosen s = NotAnInput)
then (∃ip ∈ allInput s. chosen s' = ip ∧ outpt s' = (outpt s) (p := ip))
else (outpt s' = (outpt s) (p := chosen s) ∧ chosen s' = chosen s)
by auto
moreover
from endphase2 inv2c5 nxt
have inpt s' = inpt s ∧ allInput s' = allInput s
by (auto simp add: EndPhase2-def HNext-def HNextPart-def)
ultimately
show ?thesis
  using srel p31
by (auto simp add: IChoose-def s2is-def)

next
case False
with nxt
have p31: chosen s' = chosen s
by (auto simp add: HNext-def HNextPart-def)
from inv'
have inv6: HInv6 s'
by (auto simp add: HInv-def)
have p32: outpt s' p = chosen s
proof
from endphase2
have outpt s' p = inp (dblock s p)
by (auto simp add: EndPhase2-def)
moreover
from inv6 p31
have outpt s' p ∈ {chosen s, NotAnInput}
by (auto simp add: HInv6-def)
ultimately
show ?thesis
  using outpt-nni
  by auto
qed
from srel False
have IChoose is is' p
proof(clarsimp simp add: IChoose-def s2is-def)
  from endphase2 inv2c
  have outpt s p = NotAnInput
    by(auto simp add: EndPhase2-def Inv2c-inner-def)
moreover
from endphase2 p31 p32 False
have outpt s' = (outpt s) (p:= chosen s) ∧ chosen s' = chosen s
  by(auto simp add: EndPhase2-def)
moreover
from endphase2 nxt inv2c5
have inpt s' = inpt s ∧ allInput s' = allInput s
  by(auto simp add: EndPhase2-def HNext-def HNextPart-def)
ultimately
show outpt s p = NotAnInput
  ∧ outpt s' = (outpt s) (p:= chosen s) ∧ chosen s' = chosen s
  ∧ inpt s' = inpt s ∧ allInput s' = allInput s
  by auto
qed
thus ?thesis
  by auto
qed
qed
thus ∃p. IFail is is' p ∨ IChoose is is' p
  by auto
qed

end