Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff Stephan Merz

September 19, 2015

Abstract

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

Contents

1 Introduction 2

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

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

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

5 Conclusion 11

A TLA\(^+\) correctness specification 12

B Disk Paxos Algorithm Specification 13
1 Introduction

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

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

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

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

2 The Disk Paxos Algorithm

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

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

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

The consensus algorithm of Disk Paxos is called Disk Synod. In it, each processor has an assigned block on each disk. Also it has a local memory that contains its current block (called the \( d_{block} \)), and other state variables (see figure 1). When a process \( p \) starts it contains an input value \( input[p] \) 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 \( input[p] \) 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 \( outpt \)).

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

In each block, processors maintain three values:

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

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: \( input \) and \( output \). To formalize the property stating that all processors should choose the same value and that this value should have been an input of a processor, we need variables that represent all past inputs and the value chosen as result. Consequently, an Inner submodule is introduced, which adds two variables: \( allInput \) and \( chosen \). Our Synod module will be obtained by existentially quantifying these variables of the Inner module.

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

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

\[
\text{HDiskSynodSpec} \triangleq \text{HInit} \wedge \Box \text{HNext}_{\{\text{vars, chosen, allInput}\}}
\]

where \text{HInit} describes the initial state of the algorithm and \text{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:

\[
\text{ISpec} \triangleq \text{Init} \wedge \Box \text{INext}_{\{\text{input, output, chosen, allInput}\}}
\]

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

\text{THEOREM R1} \quad \text{HInit} \Rightarrow \text{IIInit}

\text{THEOREM R2} \quad \text{HInit} \wedge \Box \text{HNext}_{\{\text{vars, chosen, allInput}\}} \Rightarrow \Box \text{INext}_{\text{iVars}}

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

\text{THEOREM R2a} \quad \text{HInit} \wedge \Box \text{HNext}_{\{\text{vars, chosen, allInput}\}} \Rightarrow \Box \text{HInv}

\text{THEOREM R2b} \quad \text{HInv} \wedge \text{HInv'} \wedge \text{HNext} \Rightarrow \text{INext} \lor (\text{UNCHANGED iVars})

A predicate satisfying \text{HInv} is said to be an invariant of \text{HDiskSynodSpec}. To prove \text{R2a}, we make \text{HInv} strong enough to satisfy:
Again, we have TLA proof rules that say that \( I_1 \) and \( I_2 \) imply \( R2a \). In summary, \( R2b, I_1, \) and \( I_2 \) together imply \( HDiskSynodSpec \Rightarrow ISpec \).

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

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

## 3 Translating from TLA\(^+\) to Isabelle/HOL

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

### 3.1 Typed vs. Untyped

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

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

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

Isabelle/HOL:

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

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

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

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

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

For example, see figure 2. The type 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⁺:

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

Isabelle/HOL:

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

Figure 3: Translation of an action

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

3.2 Primed Variables

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

3.3 Restructuring the specification

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

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

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

\[ \text{Phase1or2Read} \overset{\text{def}}{=} \text{Phase1or2ReadThen} \lor \text{Phase1or2ReadElse} \]

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:

\[ \text{HInv2} \overset{\text{def}}{=} \text{Inv2a} \land \text{Inv2b} \land \text{Inv2c} \]

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

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

we write:

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

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

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

Now we can express that we want to obtain the fact

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

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

4 Structure of the Correctness Proof

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

There are informal proofs for invariants $HInv3$-$HInv6$ 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

\[\text{MODULE Synod}\]
\[\text{EXTENDS Naturals}\]
\[\text{CONSTANT } N, \text{ Inputs}\]
\[\text{ASSUME } (N \in \text{Nat}) \land (N > 0)\]
\[\text{Proc } \triangleq 1..N\]
\[\text{NotAnInput } \triangleq \text{choose } c : c \notin \text{Inputs}\]
\[\text{VARIABLES inputs, output}\]
\[\text{MODULE Inner}\]
\[\text{VARIABLES allInput, chosen}\]

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

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

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

\[\text{INext} \triangleq \exists p \in \text{Proc } : \text{IChoose}(p) \lor \text{IFail}(p)\]
\[\text{ISpec} \triangleq \text{Init } \land \square[\text{INext}]\langle \text{input, output, chosen, allInput}\rangle\]

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

theory DiskPaxos-Model imports Main begin

This is the specification of the Disk Synod algorithm.

typedec InputsOrNi
typedec Disk
typedec Proc

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

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

majorities-intersect:
  ∀ S T. IsMajority(S) ∧ IsMajority(T) −→ S ∩ T ≠ {}

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

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

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

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

definition AllBallots :: nat set
  where AllBallots = (UN p. Ballot p)

record
  DiskBlock =
\[\begin{align*} mba &::= \text{nat} \\
bal &::= \text{nat} \\
inp &::= \text{InputsOrNi} \\
\textbf{definition} \text{InitDB} &::= \text{DiskBlock} \\
\textbf{where} \text{InitDB} &\equiv (\| \text{mba} = 0, \text{bal} = 0, \text{inp} = \text{NotAnInput} \|) \\
\textbf{record} \\
\text{BlockProc} &= \\
\text{block} &::= \text{DiskBlock} \\
\text{proc} &::= \text{Proc} \\
\textbf{record} \\
\text{state} &= \\
\text{inpt} &::= \text{Proc} \Rightarrow \text{InputsOrNi} \\
\text{outpt} &::= \text{Proc} \Rightarrow \text{InputsOrNi} \\
\text{disk} &::= \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \\
\text{dblock} &::= \text{Proc} \Rightarrow \text{DiskBlock} \\
\text{phase} &::= \text{Proc} \Rightarrow \text{nat} \\
\text{disksWritten} &::= \text{Proc} \Rightarrow \text{Disk} \\
\text{blocksRead} &::= \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{BlockProc} \Rightarrow \text{set} \\
\text{allInput} &::= \text{InputsOrNi} \Rightarrow \text{set} \\
\text{chosen} &::= \text{InputsOrNi} \\
\textbf{definition} \text{hasRead} &::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\textbf{where} \text{hasRead} &\equiv (\exists \text{br} \in \text{blocksRead} s p d. \text{proc br} = q) \\
\textbf{definition} \text{allRdBlks} &::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{BlockProc} \Rightarrow \text{set} \\
\textbf{where} \text{allRdBlks} &\equiv (\text{UN} d. \text{blocksRead} s p d) \\
\textbf{definition} \text{allBlocksRead} &::= \text{state} \Rightarrow \text{Proc} \Rightarrow \text{DiskBlock} \Rightarrow \text{set} \\
\textbf{where} \text{allBlocksRead} &\equiv \text{block }^+ (\text{allRdBlks} s p) \\
\textbf{definition} \text{Init} &::= \text{state} \Rightarrow \text{bool} \\
\textbf{where} \\
\text{Init} &\equiv (\text{range } \text{inpt} s \subseteq \text{Inputs} \\
&\& \text{outpt} s = (\lambda p. \text{NotAnInput}) \\
&\& \text{disk} s = (\lambda d p. \text{InitDB}) \\
&\& \text{phase} s = (\lambda p. 0) \\
&\& \text{dblock} s = (\lambda p. \text{InitDB}) \\
&\& \text{disksWritten} s = (\lambda p. \{\}) \\
&\& \text{blocksRead} s = (\lambda p d. \{\})) \\
\textbf{definition} \text{InitializePhase} &::= \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \\
\textbf{where} \\
\text{InitializePhase} &\equiv (\text{Init} s \Rightarrow \text{blocksRead} s \Rightarrow \text{Init} s') p = \text{false} \\
\end{align*}\]
(disksWritten \ s' = (\ p := \{\}))
& blocksRead \ s' = (\ p := (\ \lambda \ d. *) )\)

**definition** \texttt{StartBallot :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool}

**where**
\texttt{StartBallot \ s \ s' \ p =}
\( (phase \ s \ p \in \{1, 2\}) \)
& phase \ s' = (phase \ s)(p := 1) 
& (\exists b \in Ballop \ p. \)
\( mbal(dblock \ s \ p) < b \)
& dblock \ s' = (dblock \ s)(p := (dblock \ s \ p)(mbal := b \}))
& \texttt{InitializePhase \ s \ s' \ p}
& inpt \ s' = inpt \ s \ & outpt \ s' = outpt \ s \ & disk \ s' = disk \ s)

**definition** \texttt{Phase1or2Write :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow bool}

**where**
\texttt{Phase1or2Write \ s \ s' \ p \ d =}
\( (phase \ s \ p \in \{1, 2\}) \)
& disk \ s' = (disk \ s)(d := (disk \ s \ d)(p := (dblock \ s \ p)))
& disksWritten \ s' = (\ p := (disksWritten \ s \ p) \cup \{d\})
& inpt \ s' = inpt \ s \ & outpt \ s' = outpt \ s
& phase \ s' = phase \ s \ & dblock \ s' = dblock \ s
& blocksRead \ s' = blocksRead \ s)

**definition** \texttt{Phase1or2ReadThen :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool}

**where**
\texttt{Phase1or2ReadThen \ s \ s' \ p \ d \ q =}
\( (d \in disksWritten \ s \ p) \)
& mbal(disk \ s \ d \ q) < mbal(dblock \ s \ p)
& blocksRead \ s' = (\ p := (blocksRead \ s \ p)(d := (blocksRead \ s \ p \cup \{d\}))
& inpt \ s' = inpt \ s \ & outpt \ s' = outpt \ s
& disk \ s' = disk \ s \ & phase \ s' = phase \ s
& dblock \ s' = dblock \ s \ & disksWritten \ s' = disksWritten \ s)

**definition** \texttt{Phase1or2ReadElse :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool}

**where**
\texttt{Phase1or2ReadElse \ s \ s' \ p \ d \ q =}
\( (d \in disksWritten \ s \ p) \)
& \texttt{StartBallot \ s \ s' \ p}

**definition** \texttt{Phase1or2Read :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool}

**where**
\texttt{Phase1or2Read \ s \ s' \ p \ d \ q =}
\( (\ \lambda \ p \ q. \) \texttt{Phase1or2ReadThen \ s \ s' \ p \ d \ q)
\lor \texttt{Phase1or2ReadElse \ s \ s' \ p \ d \ q) \)

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

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

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

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

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

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

definition \( \text{Fail} :: \text{state} \Rightarrow \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool} \)
\where \( \text{Fail} \ s \ s' \ p = \) (\( \exists ip \in \text{Inputs}. \ \text{inpt} \ s' = (\text{inpt} \ s) \ (p := ip) \land \text{phase} \ s' = (\text{phase} \ s) \ (p := 0) \land \text{dblock} \ s' = (\text{dblock} \ s) \ (p := \text{InitDB}) \)

16
\[\text{outpt} \ s' = (\text{outpt} \ s) \ (p := \text{NotAnInput})\]
\[\text{InitializePhase} \ s \ s' \ p\]
\[\text{disk} \ s' = \text{disk} \ s\]

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

**where**

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

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

**where**

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

**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**

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

**Definition**

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

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

**Definition**

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

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

**Definition**

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

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

18
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** \( \text{HNextPart-Inv1} : [ \text{HInv1 } s ; \text{HNextPart } s s' ; \text{Inv1 } s' ] \implies \text{HInv1 } s' \)

by (auto simp add: HNextPart-def Inv1-def)

**theorem** \( \text{HInit-HInv1} : \text{HInit } s \implies \text{HInv1 } s \)

by (auto simp add: HInit-def Inv1-def Init-def allRdBlks-def)

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

assumes \( \text{inv} : \text{HInv1 } s \)

and \( \text{asm} : \forall \ p. \ \text{allRdBlks } s' p \subseteq \text{insert } bk \ (\text{allRdBlks } s p) \)

shows \( \forall \ p. \ \text{finite } (\text{allRdBlks } s' p) \)

**proof**

fix \( pp \)

from \( \text{inv} \)

have \( \forall \ p. \ \text{finite } (\text{allRdBlks } s p) \)

by (simp add: Inv1-def)

hence \( \text{finite } (\text{allRdBlks } s pp) \)

by blast

with \( \text{asm} \)

show \( \text{finite } (\text{allRdBlks } s' pp) \)

by (auto intro: finite-subset)

qed

**theorem** \( \text{HPhase1or2ReadThen-HInv1} : \)

assumes \( \text{inv1} : \text{HInv1 } s \)

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

shows \( \text{HInv1 } s' \)

**proof**

— we focus on the last conjunct of Inv1

from \( \text{act} \)

have \( \forall \ p. \ \text{allRdBlks } s' p \subseteq \text{allRdBlks } s p \cup \{(\text{block } = \text{disk } s d q, \ \text{proc } = q)\} \)

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

with \( \text{inv1} \)

have \( \forall \ p. \ \text{finite } (\text{allRdBlks } s' p) \)

by (blast dest: allRdBlks-finite)

— the others conjuncts are trivial

with \( \text{inv1 } \text{act} \)

show \( \text{thesis} \)

by (auto simp add: Inv1-def Phase1or2ReadThen-def HNextPart-def)

qed

**theorem** \( \text{HEndPhase1-HInv1} : \)

assumes \( \text{inv1} : \text{HInv1 } s \)

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

shows \( \text{HInv1 } s' \)

**proof**

from \( \text{inv1 } \text{act} \)


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

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

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

theorem \( \text{HPhase1or2ReadElse-Inv1} \):
assumes \( \text{act} \): \( \text{HPhase1or2ReadElse} \ s \ s' \) \( \text{p} \) \( \text{d} \) \( \text{q} \)
and \( \text{inv1} \): \( \text{HInv1} \ s \)
shows \( \text{HInv1} \ s' \)
using \( \text{HStartBallot-Inv1[OF inv1]} \) act
by (auto simp add: \( \text{Phase1or2ReadElse-def} \))

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

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

theorem HPhase0Read-HInv1:
  assumes inv1: HInv1 s
  and act: HPhase0Read s s' p d
  shows HInv1 s'
proof –
  — we focus on the last conjunct of Inv1
  from act
  have \( \forall pp. \text{allRdBlks } s' pp \subseteq \text{allRdBlks } s pp \cup \{ (|block = \text{disk } s d p, \text{proc} = p) \} \)
    by (auto simp add: Phase0Read-def allRdBlks-def split: split_if_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 HInv1-def [simp del]

HInv1 is an invariant of HNext

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

end

theory DiskPaxos-Inv2 imports DiskPaxos-Inv1 begin

C.2 Invariant 2

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

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

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

definition allBlocks :: state ⇒ DiskBlock set
where allBlocks s = (UN p. blocksOf s p)

**definition Inv2a-innermost :: state ⇒ Proc ⇒ DiskBlock ⇒ bool**

**where**

Inv2a-innermost s p bk =
  (mbal bk ∈ (Ballot p) ∪ {0})
  ∧ bal bk ∈ (Ballot p) ∪ {0}
  ∧ (bal bk = 0) = (inp bk = NotAnInput)
  ∧ bal bk ≤ mbal bk
  ∧ inp bk ∈ (allInput s) ∪ {NotAnInput})

**definition Inv2a-inner :: state ⇒ Proc ⇒ bool**

**where** Inv2a-inner s p = (∀ bk ∈ blocksOf s p. Inv2a-innermost s p bk)

**definition Inv2a :: state ⇒ bool**

**where** Inv2a s = (∀ p. Inv2a-inner s p)

**definition Inv2b-inner :: state ⇒ Proc ⇒ Disk ⇒ bool**

**where**

Inv2b-inner s p d =

((d ∈ disksWritten s p −→
  (phase s p ∈ {1, 2} ∧ disk s d p = dblock s p))
  ∧ (phase s p ∈ {1, 2} −→
    ( (blocksRead s p d ≠ {} −→ d ∈ disksWritten s p)
    ∧ ¬ hasRead s p d p)))

**definition Inv2b :: state ⇒ bool**

**where** Inv2b s = (∀ p d. Inv2b-inner s p d)

**definition Inv2c-inner :: state ⇒ Proc ⇒ bool**

**where**

Inv2c-inner s p =

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

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

definition $\text{HInv2} :: \ state \ \Rightarrow \ bool$

where $\text{HInv2} \ s = (\text{Inv2a} \ s \ \land \ \text{Inv2b} \ s \ \land \ \text{Inv2c} \ s)$

C.2.1 Proofs of Invariant 2 a

theorem $\text{HInit-Inv2a}$:

assumes $\text{inv}$: $\text{Inv2a} \ s$

and $\text{act}$: $\text{HPhase1or2ReadThen} \ s \ s' \ p \ d \ q$

shows $\text{Inv2a} \ s'$

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

fix $pp \ bk$

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

with $\text{inv}$ $\text{HPhase1or2ReadThen-blocksOf} [OF \ \text{act}]$

have $\text{Inv2a-innermost} \ s \ pp \ bk$

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

with $\text{act}$

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

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

qed

lemma $\text{InitializePhase-rdBy}$:

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

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

lemma $\text{HStartBallot-blocksOf}$:

$\text{HStartBallot} \ s \ s' \ p \ \Rightarrow \ \text{blocksOf} \ s' \ q \ \subseteq \ \text{blocksOf} \ s \ q \ \cup \ \{ \text{dblock} \ s' \ q \}$

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

dest: $\text{subsetD} [OF \ \text{InitializePhase-rdBy}]$)

lemma $\text{HStartBallot-Inv2a-dblock}$:

assumes $\text{act}$: $\text{HStartBallot} \ s \ s' \ p$

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

proof –
from act
have mbal': mbal (dblock s' p) ∈ Ballot p
  by (auto simp add: StartBallot-def)

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

with act
have inp': inp (dblock s' p) = inp (dblock s p)
  by (auto simp add: StartBallot-def)

from act
have mbal (dblock s p) ≤ mbal (dblock s' p)
  by (auto simp add: StartBallot-def)

with bal' inv2a
have bal-mbal: bal (dblock s' p) ≤ mbal (dblock s' p)
  by (auto simp add: Inv2a-innermost-def)

from act
have allInput s ⊆ allInput s'
  by (auto simp add: HNextPart-def InitializePhase-def Inv2a-innermost-def)

with mbal' bal' inp' bal-mbal act inv2a
show ?thesis
by (auto simp add: Inv2a-innermost-def)
qed

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

theorem HStartBallot-Inv2a:
  assumes inv: Inv2a s
  and act: HStartBallot s s' p
  shows Inv2a s'
proof (clarsimp simp add: Inv2a-def Inv2a-inner-def)
  fix q bk
  assume bk: bk ∈ blocksOf s' q
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:
  \[ HEndPhase2 s s' p \implies \text{blocksOf } s' q \subseteq \text{blocksOf } s q \]
by (auto simp add: EndPhase2-def blocksOf-def
dest: subsetD[OF InitializePhase-rdBy])

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

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

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

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

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

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

**Lemma** `HEndPhase0-blocksOf`:

`HEndPhase0 s s' p ⇒ blocksOf s' q ⊆ blocksOf s q ∪ {dblock s' q}`

by (auto simp add: `EndPhase0-def blocksOf-def`)

**Lemma** `HEndPhase0-blocksRead`:

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

`EndPhase0` has the additional difficulty of having a `choose` expression. We prove that there exists an `x` such that the predicate of the `choose` expression holds, and then apply `someI`: `?P ?x ⇒ ?P (Eps ?P)`.

**Lemma** `HEndPhase0-some`:

assumes `act`: `HEndPhase0 s s' p`
and `inv1`: `Inv1 s`
sows `∃ b ∈ allBlocksRead s p. (∀ t ∈ allBlocksRead s p. bal t ≤ bal b)
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal (SOME b. b ∈ allBlocksRead s p
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal b)))`
proof
  from `inv1`
  have `finite (bal `allBlocksRead s p)` by (simp add: `Inv1-def allBlocksRead-def`)
  moreover
  from `HEndPhase0-blocksRead[OF act]`
  have `?S ≠ {}`
  by (auto simp add: `allBlocksRead-def allRdBlks-def`)
  ultimately
  have `Max `?S ∈ `?S and `∀ t ∈ `?S. t ≤ Max `?S` by `auto`
  hence `∃ r ∈ `?S. `∀ t ∈ `?S. t ≤ r` ...
  then obtain `mblk`
  where `mblk ∈ allBlocksRead s p`
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk)` by (simp add: `allBlocksRead-def allRdBlks-def`)
  ultimately
  have `Max `?S ∈ `?S and `∀ t ∈ `?S. t ≤ Max `?S` by `auto`
  hence `∃ r ∈ `?S. `∀ t ∈ `?S. t ≤ r` ...
  then obtain `mblk`
  where `mblk ∈ allBlocksRead s p`
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk)` by (simp add: `allBlocksRead-def allRdBlks-def`)
  ultimately
  have `Max `?S ∈ `?S and `∀ t ∈ `?S. t ≤ Max `?S` by `auto`
  hence `∃ r ∈ `?S. `∀ t ∈ `?S. t ≤ r` ...
  then obtain `mblk`
  where `mblk ∈ allBlocksRead s p`
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk)` by (simp add: `allBlocksRead-def allRdBlks-def`)
  ultimately
  have `Max `?S ∈ `?S and `∀ t ∈ `?S. t ≤ Max `?S` by `auto`
  hence `∃ r ∈ `?S. `∀ t ∈ `?S. t ≤ r` ...
  then obtain `mblk`
  where `mblk ∈ allBlocksRead s p`
  ∧ (∀ t ∈ allBlocksRead s p. bal t ≤ bal mblk)` by (simp add: `allBlocksRead-def allRdBlks-def`)

by auto
thus ?thesis
  by (rule someI)
qed

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

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

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

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

\text{lemma HEndPhase0-Inv2a-dblock-q:} \\
\text{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)
**lemma** HEndPhase1-blocksOf:
  HEndPhase1\ s\ \ s'\ \ p\ \ \Rightarrow\ \ blocksOf\ s'\ q\ \subseteq\ \ blocksOf\ s\ q\ \cup\ \{\text{dblock}\ s'\ q\}
*by* (auto simp add: EndPhase1-def blocksOf-def dest: subsetD[OF InitializePhase-rdBy])

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

**lemma** blocksOf-nonInitBlks:
  (∀p bk. bk ∈ blocksOf s p → P bk)
  \ \Rightarrow\ \ bk ∈ nonInitBlks s p → P bk
*by*(auto simp add: allRdBlks-def blocksOf-def nonInitBlks-def blocksSeen-def allBlocksRead-def rdBy-def, blast)
lemma maxBlk-allInput:
  assumes inv: Inv2a s
  and mblk: maxBlk s p ∈ nonInitBlks s p
  shows inp (maxBlk s p) ∈ allInput s
proof –
  from inv
  have blocks: ∀ p bk. bk ∈ blocksOf s p
    → inp bk ∈ (allInput s) ∪ {NotAnInput}
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def)
  from mblk NotAnInput
  have inp (maxBlk s p) ≠ NotAnInput
    by (auto simp add: nonInitBlks-def)
  with mblk blocksOf-nonInitBlks[OF blocks]
  show ?thesis
    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 inv1
have mbal': mbal' (dblock s' p) ∈ Ballot p
  by (auto simp add: EndPhase1-def Inv2c-def Inv2c-inner-def)

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

moreover
from act inv2
have inp': inp' (dblock s' p) ∈ allInput s'
  by (blast dest: HEndPhase1-dblock-allInput)

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

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

lemma HEndPhase1-Inv2a-dblock-q:
  assumes act: HEndPhase1 s s' p
  and inv1: HInv1 s
  and 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

and \( inv: \text{Inv2a } s \)
and \( inv2c: \text{Inv2c-inner } s \ p \)
shows \( \text{Inv2a } s' \)

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

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

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

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

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

theorem HPhase1or2Write-Inv2b:
\[ [ \text{Inv2b } s ; \text{HPhase1or2Write } s \ s' p \ d ] \]
\( \rightarrow \) Inv2b s'

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

theorem HPhase1or2ReadElse-Inv2b:
  \[ \begin{array}{l}
  \text{Inv2b } s \\
  \text{HPhase1or2ReadElse } s \ s' \ p \ d \ q
  \end{array} \implies \text{Inv2b } s' \]
by (auto simp add: Phase1or2ReadElse-def StartBallot-def hasRead-def
     InitializePhase-def Inv2b-def Inv2b-inner-def)

theorem HEndPhase1-Inv2b:
  \[ \begin{array}{l}
  \text{Inv2b } s \\
  \text{HEndPhase1 } s \ s' \ p
  \end{array} \implies \text{Inv2b } s' \]
by (auto simp add: EndPhase1-def InitializePhase-def
     Inv2b-def Inv2b-inner-def hasRead-def)

theorem HFail-Inv2b:
  \[ \begin{array}{l}
  \text{Inv2b } s \\
  \text{HFail } s \ s' \ p
  \end{array} \implies \text{Inv2b } s' \]
by (auto simp add: Fail-def InitializePhase-def
     Inv2b-def Inv2b-inner-def hasRead-def)

theorem HEndPhase2-Inv2b:
  \[ \begin{array}{l}
  \text{Inv2b } s \\
  \text{HEndPhase2 } s \ s' \ p
  \end{array} \implies \text{Inv2b } s' \]
by (auto simp add: EndPhase2-def InitializePhase-def
     Inv2b-def Inv2b-inner-def hasRead-def)

theorem HPhase0Read-Inv2b:
  \[ \begin{array}{l}
  \text{Inv2b } s \\
  \text{HPhase0Read } s \ s' \ p \ d
  \end{array} \implies \text{Inv2b } s' \]
by (auto simp add: Phase0Read-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: HNextPart s s'
  and inv2c: Inv2c s
  and outpt': \forall p. outpt s' p = (if phase s' p = 3
                     then inp(dblock s' p)
                     else NotAnInput)
  and inp-dblk: \forall p. inp(dblock s' p) \in allInput s' \cup\{NotAnInput\}
shows chosen s' \in allInput s' \cup\{NotAnInput\}
using hnp outpt' inp-dblk inv2c

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

lemma HNextPart-chosen:
    assumes hnp: HNextPart s s'
    shows chosen s' = NotAnInput \rightarrow (\forall p. outpt s' p = NotAnInput)
using hnp

proof(auto simp add: HNextPart-def split: split-if-asm)
fix p pa
assume o1: outpt s' p \neq NotAnInput
and o2: outpt s' (SOME p. outpt s' p \neq NotAnInput) = NotAnInput
from o1
have \exists p. outpt s' p \neq NotAnInput
  by auto
hence outpt s' (SOME p. outpt s' p \neq NotAnInput) \neq NotAnInput
  by(rule someI-ex)
with o2
show outpt s' pa = NotAnInput
  by simp
qed

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

theorem HPhase1or2ReadThen-Inv2c:
    assumes inv: Inv2c s
    and act: HPhase1or2ReadThen s s' p d q
    and inv2a: Inv2a s
    shows Inv2c s'
proof --
from inv2a act
have inv2a': Inv2a s'
  by(blast dest: HPhase1or2ReadThen-Inv2a)
from act inv
have outpt': \forall p. outpt s' p = (if phase s' p = 3
    then inp(dblock s' p)
    else NotAnInput)
  by(auto simp add: Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: \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 \(\forall p. \text{ inpt } s' p \in \text{allInput } s'
\land (\text{chosen } s' = \text{NotAnInput} \rightarrow \text{outpt } s' p = \text{NotAnInput})\)
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 \(\text{inv}: \text{Inv2c } s\)
and \(\text{act}: \text{HStartBallot } s s' p\)
and \(\text{inv2a}: \text{Inv2a } s\)
shows \(\text{Inv2c } s'\)
proof
  from act
  have phase': phase \( s' p = 1\)
    by(simp add: StartBallot-def)
  from act
  have phase: phase \( s p \in \{1, 2\}\)
    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': \(\forall p. \text{outpt } s' p = \text{(if phase } s' p = 3
    then inp(dblock } s' p)\)
    else NotAnInput)
    by(auto simp add: StartBallot-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: \(\forall p. \text{inp (dblock } s' p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}\)
    by(auto simp add: Inv2a-def Inv2a-inner-def
    Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen \( s' \in \text{allInput } s' \cup \{\text{NotAnInput}\}\)
    by(auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: \( \forall p. \text{inpt } s' p \in \text{allInput } s' \)
\( \land (\text{chosen } s' = \text{NotAnInput} \rightarrow \text{outpt } s' p = \text{NotAnInput}) \)
by(auto dest: HNextPart-chosen HNextPart-allInput)
with phase' mbal' bal' outpt' chosen' act inv blks'
show ?thesis
by(auto simp add: StartBallot-def InitializePhase-def
Inv2c-def Inv2c-inner-def)
qed

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

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

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

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

42
have inv2a': Inv2a s' by (blast dest: HEndPhase2-Inv2a)
from act inv
have outpt': ∀ p. outpt s' p = (if phase s' p = 3 then inp(dblock s' p) else NotAnInput)
  by (auto simp add: EndPhase2-def Inv2c-def Inv2c-inner-def)
from inv2a'
have dblk: ∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
  by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
with act inv outpt'
have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
  by (auto dest: HNextPart-Inv2c-chosen)
from act inv
have allinp: ∀ p. inpt s' p ∈ allInput s' ∧ (chosen s' = NotAnInput
  → outpt s' p = NotAnInput)
  by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
  by (auto simp add: EndPhase2-def InitializePhase-def Inv2c-def Inv2c-inner-def)
qed

theorem HFail-Inv2c:
  assumes inv: Inv2c s
  and act: HFail s s' p
  and inv2a: Inv2a s
  shows Inv2c s'
proof –
  from inv2a act
  have inv2a': Inv2a s'
    by (blast dest: HFail-Inv2a)
  from act inv
  have outpt': ∀ p. outpt s' p = (if phase s' p = 3 then inp(dblock s' p)
    else NotAnInput)
    by (auto simp add: Fail-def Inv2c-def Inv2c-inner-def)
  from inv2a'
  have dblk: ∀ p. inp (dblock s' p) ∈ allInput s' ∪ {NotAnInput}
    by (auto simp add: Inv2a-def Inv2a-inner-def Inv2a-innermost-def blocksOf-def)
  with act inv outpt'
  have chosen': chosen s' ∈ allInput s' ∪ {NotAnInput}
    by (auto dest: HNextPart-Inv2c-chosen)
  from act inv
  have allinp: ∀ p. inpt s' p ∈ allInput s' ∧ (chosen s' = NotAnInput
    → outpt s' p = NotAnInput)
by (auto dest: HNextPart-chosen HNextPart-allInput)
with outpt' chosen' act inv
show ?thesis
  by (auto simp add: Fail-def InitializePhase-def
       Inv2c-def Inv2c-inner-def)
qed

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

theorem HEndPhase0-Inv2c:
  assumes inv: "Inv2c s"
  and act: "HEndPhase0 s s' p"
  and inv2a: "Inv2a s"
  and inv1: "Inv1 s"
  shows "Inv2c s'"
proof -
  from inv
  have inv2c-inner s p by (auto simp add: Inv2c-def)
  with inv2a act inv1

44
have $inv2a'$. 

\begin{enumerate}
\item by (blast dest: $HEndPhase0-inv2a$)
\end{enumerate}

**hence bal'**: \(bal(dblock s' \ p) \in \text{Ballot } p \cup \{0\}\)

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

**from act inv**

**have** $mbal'$. \(mbal(dblock s' \ p) \in \text{Ballot } p\)

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

from act inv

**have** $dblk'$. \((\forall d. \forall br \in \text{blocksRead } s' \ p d. \mbal(block \ br) < \mbal(dblock s' \ p))\)

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

from act inv

**have** $outpt'$. \(\forall p. \outpt s' \ p = (\text{if } \text{phase } s' \ p = 3 \ \text{then } \inp(dblock s' \ p)) \ \text{else } \text{NotAnInput}\)

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

from act inv

**have** $dblk'. \forall p. \inp(dblock s' \ p) \in \text{allInput } s' \cup \{\text{NotAnInput}\}$

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

with act inv $outpt'$

**have** $chosen'$. \(\forall p. \ chosen' \ p \in \text{allInput } s' \cup \{\text{NotAnInput}\}\)

by (auto dest: $HNextPart-inv2a'$)

from act inv

**have** $allinp$. \(\forall p. \inp s' \ p \in \text{allInput } s' \land (\text{chosen } s' = \text{NotAnInput} \ \rightarrow \ \text{outpt } s' \ p = \text{NotAnInput})\)

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

with $mbal' \ bal' \ blks' \ outpt' \ chosen' \ act inv$

show $?\text{thesis}$

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

qed

**theorem** $HInit-HInv2$:

$$HInit \ s \Rightarrow HInv2 \ s$$

**using** $HInit-inv2a \ HInit-inv2b \ HInit-inv2c$

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

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

**lemma I2b**:

assumes $nxt$: $HNext \ s \ s'$

and $inv$: $HInv1 \ s \land HInv2 \ s$

shows $HInv2 \ s'$

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

**show** $Inv2a \ s' \ using \ assms$

by (auto simp add: $HInv2\text{-def HNext-def Next-def}$, auto intro: $HStartBallot-inv2a$,
auto intro: HPhase1or2Write-Inv2a,
auto simp add: Phase1or2Read-def
  intro: HPhase1or2ReadThen-Inv2a
  HPhase1or2ReadElse-Inv2a,
auto intro: HPhase0Read-Inv2a,
auto simp add: EndPhase1or2-2-def Inv2c-def
  intro: HEndPhase1-Inv2a
  HEndPhase2-Inv2a,
auto intro: HFail-Inv2a,
auto simp add: HInv1-def
  intro: HEndPhase0-Inv2a)
show Inv2b s' using assms
  by(auto simp add: HInv2-def HNext-def Next-def,
      auto intro: HStartBallot-Inv2b,
      auto intro: HPhase0Read-Inv2b,
      auto intro: HPhase1or2Write-Inv2b,
      auto simp add: Phase1or2Read-def
      intro: HPhase1or2ReadThen-Inv2b
      HPhase1or2ReadElse-Inv2b,
      auto simp add: EndPhase1or2-2-def
      intro: HEndPhase1-Inv2b HEndPhase2-Inv2b,
      auto intro: HFail-Inv2b HEndPhase0-Inv2b)
show Inv2c s' using assms
  by(auto simp add: HInv2-def HNext-def Next-def,
      auto intro: HStartBallot-Inv2c,
      auto intro: HPhase0Read-Inv2c,
      auto intro: HPhase1or2Write-Inv2c,
      auto simp add: Phase1or2Read-def
      intro: HPhase1or2ReadThen-Inv2c
      HPhase1or2ReadElse-Inv2c,
      auto simp add: EndPhase1or2-2-def
      intro: HEndPhase1-Inv2c
      HEndPhase2-Inv2c,
      auto intro: HFail-Inv2c,
      auto simp add: HInv1-def intro: HEndPhase0-Inv2c)
qed
end

theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

C.3 Invariant 3

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

definition HInv3-L :: state \( \Rightarrow \) Proc \( \Rightarrow \) Proc \( \Rightarrow \) Disk \( \Rightarrow \) bool
where
\[ H_{inv3-L} \ s \ p \ q \ d = \ (\text{phase} \ s \ p \in \{1,2\} \land \text{phase} \ s \ q \in \{1,2\} \land \text{hasRead} \ s \ p \ d \ q \land \text{hasRead} \ s \ q \ d \ p) \]

definition \( H_{inv3-R} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \)

where
\[ H_{inv3-R} \ s \ p \ q \ d = (\{(\text{block} = \text{dblock} \ s \ q, \text{proc} = q)\} \in \text{blocksRead} \ s \ p \ d \lor \{(\text{block} = \text{dblock} \ s \ p, \text{proc} = p)\} \in \text{blocksRead} \ s \ q \ d) \]

definition \( H_{inv3-inner} :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{Proc} \Rightarrow \text{Disk} \Rightarrow \text{bool} \)

where
\[ H_{inv3-inner} \ s \ p \ q \ d = (H_{inv3-L} \ s \ p \ q \ d \rightarrow H_{inv3-R} \ s \ p \ q \ d) \]

definition \( H_{inv3} :: \text{state} \Rightarrow \text{bool} \)

where
\[ H_{inv3} \ s = (\forall \ p \ q \ d. \ H_{inv3-inner} \ s \ p \ q \ d) \]

C.3.1 Proofs of Invariant 3

theorem \( H_{init-Hinv3} : H_{init} \ s \Rightarrow H_{inv3} \ s \)

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

lemma \( InitPhase-Hinv3-p \):

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

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

lemma \( InitPhase-Hinv3-q \):

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

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

lemma \( H_{inv3-L-sym} : H_{inv3-L} \ s \ p \ q \ d \Rightarrow H_{inv3-L} \ s \ q \ p \ d \)

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

lemma \( H_{inv3-R-sym} : H_{inv3-R} \ s \ p \ q \ d \Rightarrow H_{inv3-R} \ s \ q \ p \ d \)

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

lemma \( Phase1or2ReadThen-Hinv3-pq \):

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

and \( \text{inv-L'} : H_{inv3-L} \ s' \ p \ q \ d \)

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

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

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

proof –

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

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

by (auto simp add: \( \text{Phase1or2ReadThen-def} \ H_{inv3-L-def} \ H_{inv3-L-def} \text{hasRead-def split: split-if-asm} \))
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 HInv3-L-sym[of h3l']
      show ?thesis
        by (auto dest: Phase1or2ReadThen-HInv3-pq HInv3-R-sym)
    next
      case False
      from nh3l h3l' act
      have (¬hasRead s pp dd qq ∨ ¬hasRead s qq dd pp)
\begin{itemize}
\item \(\land \ \text{hasRead} \ s' \ pp \ dd \ qq \ \land \ \text{hasRead} \ s' \ qq \ dd \ pp\)
\item by (auto simp add: HInv3-L-def Phase1or2ReadThen-def)
\item with \(\text{act False}\)
\item show \(\?\text{thesis}\)
\item by (auto dest: Phase1or2ReadThen-HInv3-hasRead)
\item qed
\item qed
\item qed
\end{itemize}

lemma \(\text{StartBallot-HInv3-p}\):
\[
\begin{array}{c}
\text{[ StartBallot} \ s \ s' \ p; \ HInv3-L \ s' \ p \ q \ d \ ] \\
\Rightarrow \ HInv3-R \ s' \ p \ q \ d
\end{array}
\]
by (auto simp add: StartBallot-def dest: InitPhase-HInv3-p)

lemma \(\text{StartBallot-HInv3-q}\):
\[
\begin{array}{c}
\text{[ StartBallot} \ s \ s' \ q; \ HInv3-L \ s' \ p \ q \ d \ ] \\
\Rightarrow \ HInv3-R \ s' \ p \ q \ d
\end{array}
\]
by (auto simp add: StartBallot-def dest: InitPhase-HInv3-q)

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

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

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

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

49
with inv act
show ?thesis
  by (auto simp add: HInv3-inner-def dest: StartBallot-HInv3-t)
qed

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

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

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

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

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

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

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

lemma EndPhase1-HInv3:
assumes act: EndPhase1 s s' t
and inv: HInv3-inner s p q d
shows HInv3-inner s' p q d
proof(cases t=p \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:
[ HEndPhase1 s s' p; HInv3 s ] \implies HInv3 s'
by(auto simp add: HInv3-def dest: EndPhase1-HInv3)

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

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

lemma EndPhase2-HInv3-nL:
[ EndPhase2 s s' t; \neg HInv3-L s p q d; t\neq p; t\neq q ]
\implies \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≠p; t≠ 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≠p; t≠ 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 ∨ 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; ¬HInv3-L s p q d; t≠p; t≠ q ]
  \implies ¬HInv3-L s' p q d
by(auto simp add: Fail-def InitializePhase-def HInv3-L-def hasRead-def)
lemma Fail-HInv3-R:
\[
\begin{align*}
\text{Fail} & \ s \ s' \ t \\
\text{HInv3} & \ s \ p \ q \ d \\
\text{t} & \neq p \\
\text{t} & \neq q \\
\implies & \ 
\text{HInv3} \ s' \ p \ q \ d \\
\end{align*}
\]
by (auto simp add: Fail-def InitializePhase-def HInv3-R-def hasRead-def)

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

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

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

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

lemma EndPhase0-HInv3-p:
[ [ \text{EndPhase0 } s \quad s' \quad p; \quad \text{HInv3-L } s' \quad p \quad q \quad d ] ]
\Rightarrow \text{HInv3-R } s' \quad p \quad q \quad d
by (auto simp add: EndPhase0-def dest: InitPhase-HInv3-p)

lemma EndPhase0-HInv3-q:
[ [ \text{EndPhase0 } s \quad s' \quad q; \quad \text{HInv3-L } s' \quad p \quad q \quad d ] ]
\Rightarrow \text{HInv3-R } s' \quad p \quad q \quad d
by (auto simp add: EndPhase0-def dest: InitPhase-HInv3-q)

lemma EndPhase0-HInv3-nL:
[ [ \text{EndPhase0 } s \quad s' \quad t; \quad \neg \text{HInv3-L } s \quad p \quad q \quad d; \quad t \neq p; \quad t \neq q ] ]
\Rightarrow \neg \text{HInv3-L } s' \quad p \quad q \quad d
by (auto simp add: EndPhase0-def InitializePhase-def
      HInv3-L-def hasRead-def)

lemma EndPhase0-HInv3-R:
[ [ \text{EndPhase0 } s \quad s' \quad t; \quad \text{HInv3-R } s \quad p \quad q \quad d; \quad t \neq p; \quad t \neq q ] ]
\Rightarrow \text{HInv3-R } s' \quad p \quad q \quad d
by (auto simp add: EndPhase0-def InitializePhase-def
      HInv3-R-def hasRead-def)

lemma EndPhase0-HInv3-t:
[ [ \text{EndPhase0 } s \quad s' \quad t; \quad \text{HInv3-inner } s \quad p \quad q \quad d; \quad t \neq p; \quad t \neq q ] ]
\Rightarrow \text{HInv3-inner } s' \quad p \quad q \quad d
by (auto simp add: HInv3-inner-def
      dest: EndPhase0-HInv3-nL EndPhase0-HInv3-R)

lemma EndPhase0-HInv3:
  assumes act: \text{EndPhase0 } s \quad s' \quad t
  and inv: \text{HInv3-inner } s \quad p \quad q \quad d
  shows \text{HInv3-inner } s' \quad p \quad q \quad d
proof (cases \( t = p \) \lor \( t = q \))
  case True
  with act inv
  show \( \neg \text{thesis} \)
    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:
  \[ \text{HEndPhase0 } s \ s' \ p; \ HInv3 \ s \ \implies \ HInv3 \ s' \]
  by (auto simp add: HInv3-def dest: EndPhase0-HInv3)

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

lemma I2c:
  assumes \( \text{next: } HNext \ s \ s' \)
  and \( \text{inv: } HInv1 \ s \land HInv2 \ s \land HInv3 \ s \)
  shows \( HInv3 \ s' \) using \( \text{assms} \)
  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 $HInv4a1 \ s \ p = (\forall bk \in \text{blocksOf} \ s \ p. \ \text{bal} \ bk \leq \text{mbal} (\text{dblock} \ s \ p))$

**Definition** $HInv4a2 :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where

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

**Definition** $HInv4a :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where

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

**Definition** $HInv4b :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where

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

**Definition** $HInv4c :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where

$HInv4c \ s \ p = (\forall \text{phase} \ s \ p \in \{2,3\} \rightarrow (\exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{mbal(disk} \ s \ d \ p) = \text{bal(dblock} \ s \ p)))$

**Definition** $HInv4d :: \text{state} \Rightarrow \text{Proc} \Rightarrow \text{bool}$

where

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

**Definition** $HInv4 :: \text{state} \Rightarrow \text{bool}$

where

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

The initial state implies Invariant 4.

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

**Using** $\text{Disk-isMajority}$

**By** (auto simp add: $HInit-def$ $Init-def$ $HInv4-def$ $HInv4a-def$ $HInv4a1-def$

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

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

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

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

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

lemma \( \text{HStartBallot-HInv4a-p:} \)
assumes \( \text{act: HStartBallot } s s' p \)
and \( \text{inv: HInv4a } s p \)
and \( \text{inv2a: Inv2a-inner } s' p \)
shows $H_{Inv4a} s' p$
using act inv inv2a
proof –
  from act
  have phase: $0 < \text{phase} s p$
    by (auto simp add: StartBallot-def)
  from act inv inv2a
  show thesis
    by (auto simp del: HStartBallot-def simp add: HInv4a-def phase
         elim: HStartBallot-HInv4a1 HStartBallot-HInv4a2)
qed

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

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

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

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

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

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

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

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

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

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

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

60
shows $HInv4a1 \ s' p$
proof (auto simp add: $HInv4a1$-def)
 fix $bk$
 assume $bk$: $bk \in \text{blocksOf} \ s' p$
 from $bk \ \text{HEndPhase1-blocksOf}[OF \ act]$ have $bk \in \{ \text{dblock} \ s' p \} \cup \text{blocksOf} \ s p$
 by blast
 with $act \ inv$
 show $bal \ bk \leq mbal \ (\text{dblock} \ s' p)$
 by (auto simp add: $HInv4a$-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 $\text{disk} \ s \ d \ p \in \text{blocksOf} \ s \ p$
 by (auto simp add: blocksOf-def)
 with $inv2a$
 have $bal(\text{disk} \ s \ d \ p) \leq mbal \ (\text{disk} \ s \ d \ p)$
 by (auto simp add: 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 $HInv4a \ s' \ p$

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

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

theorem HEndPhase1-HInv4a:
  \[ HEndPhase1 \ s \ s' \ p; HInv4a \ s \ q; Inv2a \ s \ \rightarrow \ HInv4a \ s' \ q \]
by (blast dest: HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q)

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

theorem HPhase0Read-HInv4a:
  \[ HPhase0Read \ s \ s' \ p \ d; HInv4a \ s \ q \ \rightarrow \ HInv4a \ s' \ q \]
by (auto simp add: Phase0Read-def HInv4a-def HInv4a1-def HInv4a2-def InitializePhase-def
blocksOf-def rdBy-def)
theorem HEndPhase2-HInv4a:
[ HEndPhase2 s s' p; HInv4a s q ] \implies HInv4a s' q
by(auto simp add: HEndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def
     InitializePhase-def blocksOf-def rdBy-def)

lemma allSet:
  assumes aPQ: \( \forall a. \forall r \in P a. Q r \) and \( rb: rb \in P d \)
  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: HInv4d s p
and inv2c: Inv2c-inner s p
  shows \( \exists \, d. \exists \, rb \in \text{blocksRead} \, s \, p \, d. \, \text{bal} \, bk \leq \text{mbal} (\text{block} \, rb) \)
proof -
  from bk inv4d have \( \exists \, D1 \in \text{MajoritySet}. \forall d \in D1. \, \text{bal} \, bk \leq \text{mbal} (\text{disk} \, s \, d \, p) \) — 4.2
    by(auto simp add: HInv4d-def)
  with \( \text{majorities-intersect} \)
  have p43: \( \forall D \in \text{MajoritySet}. \exists \, d \in D. \, \text{bal} \, bk \leq \text{mbal} (\text{disk} \, s \, d \, p) \)
    by(simp add: MajoritySet-def, blast)
  from act
  have \( \forall d. \forall \, rb \in \text{blocksRead} \, s \, p \, d. \, \text{block} \, rb = \text{disk} \, s \, d \, p \) — 5.1
    by(auto simp add: Inv2c-inner-def)
  hence \( \forall d. \, \text{hasRead} \, s \, p \, d \)
    \( \implies (\exists \, rb \in \text{blocksRead} \, s \, p \, d. \, \text{block} \, rb = \text{disk} \, s \, d \, p) \) — 5.2
    (is \( \forall d. \, ?H \, d \implies ?P \, d \))
    by(auto simp add: hasRead-def)
  with act
  have p53: \( \exists \, D \in \text{MajoritySet}. \forall d \in D. \, ?P \, d \)
    by(auto simp add: MajoritySet-def EndPhase0-def)
  show \(?thesis\)
proof -
  from p43 p53
  have \( \exists \, D \in \text{MajoritySet}. \, (\exists \, d \in D. \, \text{bal} \, bk \leq \text{mbal} (\text{disk} \, s \, d \, p)) \)
    \( \land (\forall d \in D. \, ?P \, d) \)
    by(auto
  thus \(?thesis\)
    by force
qed
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: 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

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≠{}
    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 \( \text{disk} \ s \ d \ p \in \text{block} ^ {'} \ \text{blocksRead} \ s \ p \ d \)
by force
thus \( \text{disk} \ s \ d \ p \in \text{allBlocksRead} \ s \ p \)
by(auto simp add: allBlocksRead-def allRdBlks-def)

qed

lemma \( \text{HEndPhase0-41} \):
assumes act: \( \text{HEndPhase0} \ s \ s ^ {'} \ p \)
and inv1: \( \text{Inv1} \ s \)
and inv2c: \( \text{Inv2c-inner} \ s \ p \)
shows \( \exists D \in \text{MajoritySet}. \ \forall d \in D. \ \text{mbal}(\text{disk} \ s \ d \ p) \leq \text{mbal}(\text{dblock} \ s ^ {'} \ p) \)
\& \( \text{bal}(\text{disk} \ s \ d \ p) \leq \text{bal}(\text{dblock} \ s ^ {'} \ p) \)

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

qed

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

qed

lemma \( \text{HEndPhase0-Hinv4a2-p} \):
assumes act: \( \text{HEndPhase0} \ s \ s ^ {'} \ p \)
and inv1: \( \text{Inv1} \ s \)
and $inv2c$: \texttt{Inv2c-inner s p}

shows $Hinv4a2 s' p$

proof\((\text{simp add: } Hinv4a2-def)\)

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

lemma \texttt{HEndPhase0-Hinv4a-p}:
assumes \texttt{act: HEndPhase0 s s' p}
and $inv2a$: \texttt{Inv2a s}
and $inv2$: \texttt{Inv2c s}
and $inv4d$: $Hinv4d s p$
and $inv1$: \texttt{Inv1 s}
and $inv$: $Hinv4a s p$
shows $Hinv4a s' p$
proof
  from \texttt{inv2}
  have $inv2c$: $Inv2c-inner s p$
    by\((\text{auto simp add: Inv2c-def})\)
  with \texttt{inv1 inv2a act}
  have $inv2a'$: $Inv2a s'$
    by\((\text{blast dest: HEndPhase0-Inv2a})\)
  from \texttt{act}
  have $phase s' p = 1$
    by\((\text{auto simp add: EndPhase0-def})\)
  with \texttt{act inv inv2c inv4d inv2a' inv1}
  show $?thesis$
    by\((\text{auto simp add: Hinv4a-def simp del: HEndPhase0-def elim: HEndPhase0-Hinv4a1-p HEndPhase0-Hinv4a2-p})\)
qed

lemma \texttt{HEndPhase0-Hinv4a-q}:
assumes \texttt{act: HEndPhase0 s s' p}
and $inv$: $Hinv4a s q$
and $p \neq q$
shows $Hinv4a s' q$
proof

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

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

C.4.2 Proofs of Invariant 4b

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

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

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

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

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

69
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)
    \and bal (dblock s p) = bal (dblock s' p)
    by (auto simp add: StartBallot-def)
  from HStartBallot-blocksOf[OF act] bk
  have bk \in \{dblock s' p\} \cup blocksOf s p
    by blast
  thus bal bk < mbal (dblock s' p)
proof
  assume bk: bk \in \{dblock s' p\}
  from inv2a
  have bal (dblock s p) \leq mbal (dblock s p)
    by (auto simp add: Inv2a-innermost-def)
  with p42 bk
show ?thesis by auto

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

lemma HStartBallot-HInv4b-q:
assumes act: HStartBallot 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: 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 dstk' dblock'
have blocksOf s' q ⊆ blocksOf s q
  by (auto simp add: blocksOf-def rdBy-def, blast)
with inv phase' dblock'
show ?thesis
  by (auto simp add: HInv4b-def)
qed

theorem HStartBallot-HInv4b:
assumes act: HStartBallot s s' p
and inv2a: Inv2a s
and inv4b: HInv4b s q
and inv4a: HInv4a s p
shows 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 ?thesis
  by (blast dest: HStartBallot-HInv4b-p)
next
case False
with act inv4b
show \(?thesis
  by(blast dest: HStartBallot-HInv4b-q)
qed

theorem HPhase1or2Write-HInv4b:
[ HPhase1or2Write s s' p d; HInv4b s q ] \implies HInv4b s' q
by(auto simp add: Phase1or2Write-def HInv4b-def
  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 pq: 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 pq: 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
have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
  by(auto simp add: EndPhase2-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 HEndPhase2-HInv4b:
  assumes act: HEndPhase2 s s' p
  and inv: HInv4b s q
  shows HInv4b s' q
proof(cases p=q)
  case True
  with HEndPhase2-HInv4b-p[OF act]
  show ?thesis by simp
next
  case False
  from act pnq
  have disk': disk s' = disk s
    and dblock': dblock s' q = dblock s q
    and phase': phase s' q = phase s q
    by(auto simp add: Fail-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by(auto simp add: Fail-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q ⊆ blocksOf s q
    by(auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  with inv phase' dblock'
  show ?thesis
    by(auto simp add: HInv4b-def)
qed

lemma HFail-HInv4b-p:
  HFail s s' p ⇒ HInv4b s' p
by(auto simp add: Fail-def HInv4b-def)

lemma HFail-HInv4b-q:
  assumes act: HFail 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: Fail-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by(auto simp add: Fail-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have blocksOf s' q ⊆ blocksOf s q
    by(auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
  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: Phase0Read-def HInv4b-def)

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

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

75
C.4.3 Proofs of Invariant 4c

**Lemma HStartBallot-HInv4c-p:**

\[ [ \text{HStartBallot } s \ s' \ p; \text{HInv4c } s \ p] \implies \text{HInv4c } s' \ p \]

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

**Lemma HStartBallot-HInv4c-q:**

assumes act: HStartBallot \( s \ s' \ p \)

and inv: HInv4c \( s \ q \)

and pnq: \( p \neq q \)

shows HInv4c \( s' \ q \)

proof –

from act pnq

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

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

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

by (auto simp add: StartBallot-def)

with inv

show \( ?\text{thesis} \)

by (auto simp add: HInv4c-def)

qed

**Theorem HStartBallot-HInv4c:**

\[ [ \text{HStartBallot } s \ s' \ p; \text{HInv4c } s \ q] \implies \text{HInv4c } s' \ q \]

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

**Lemma HPhase1or2Write-HInv4c-p:**

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

and inv: HInv4c \( s \ p \)

and inv2c: Inv2c \( s \)

shows HInv4c \( s' \ p \)

proof (cases phase \( s' \ p = 2 \))

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

show \( ?\text{thesis} \)

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

from act phase'

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

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

by (auto simp add: Phase1or2Write-def)

from phase' inv2c act

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

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

with bal

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

by auto

with inv phase act

show \( \exists D.\text{ IsMajority } D \)

\( \wedge (\forall d \in D. \text{mbal}(\text{disk } s' \ d \ p) = \text{bal}(\text{dblock } s' \ p)) \)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma HPhase1or2Write-HInv4d-q:
assumes act: HPhase1or2Write s s' p d
and inv: HInv4d s q
and pq: p \neq q
shows HInv4d s' q
proof
from act pq have disk': \forall d. disk s' d q = disk s d q
by (auto simp add: Phase1or2Write-def)
from act pq have blocksRead': \forall q. allRdBlks s' q \subseteq allRdBlks s q
by (auto simp add: Phase1or2Write-def InitializePhase-def allRdBlks-def)
with act pq have blocksOf s' q \subseteq blocksOf s q

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

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

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

qed

theorem HPhase1or2Write-HInv4d:
  \([ H\text{Phase1or2Write } s s' p d; H\text{Inv4d } s q; H\text{Inv4a } s p ] \implies H\text{Inv4d } 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 \( \exists D \in \text{MajoritySet}. \forall d \in D. \ bal\ bk \leq \text{mbal}(\text{disk } s d p) \)
    
    by (auto simp add: HInv4d-def)

  with \( \text{act} \)
  show \( \exists D \in \text{MajoritySet}. \forall d \in D. \ bal\ bk \leq \text{mbal}(\text{disk } s' d p) \)
    
    by (auto simp add: Phase1or2ReadThen-def)

qed

lemma HPhase1or2ReadThen-HInv4d-q:
  assumes act: HPhase1or2ReadThen s s' p d r
  and inv: HInv4d s q
  and pnq: p \( \neq \) q
  shows HInv4d s' q
proof
  from act pnq
  have disk': \( \text{disk } s' = \text{disk } s \)
    
    by (auto simp add: Phase1or2ReadThen-def)

  from pnq
  have blocksOf s' q \( \subseteq \) \text{blocksOf } s q
    
    by (auto simp add: Phase1or2ReadThen-def allRdBlks-def blocksOf-def rdBy-def)

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

theorem HPhase1or2ReadThen-HInv4d:
[ HPhase1or2ReadThen s s' p d r; HInv4d s q ] \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 subsetD[OF HEndPhase1-blocksOf[OF act bk]]
show \exists D \in MajoritySet. \forall d \in D. bal bk \leq mbal (disk s' d p)
proof
assume bk: bk \in blocksOf s p
with inv disk'
show ?thesis
by (auto simp add: HInv4d-def)
next
assume bk: bk \in \{dblock s' p\}
with p31
show ?thesis
by force
qed

lemma HEndPhase1-HInv4d-q:
assumes act: HEndPhase1 s s' p
and inv: HInv4d s q
and pnq: p ≠ q
shows HInv4d s' q

proof
  from act pnq
  have disk': disk s' = disk s
    and dblock': dblock s' q = dblock s q
    by (auto simp add: EndPhase1-def)
  from act pnq
  have blocksRead': ∀ q. allRdBlks s' q ⊆ allRdBlks s q
    by (auto simp add: EndPhase1-def InitializePhase-def allRdBlks-def)
  with disk' dblock'
  have 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 (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: 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 inv: HInv4\(d\) s q
and pmq: p\(\neq\)q
shows HInv4\(d\) s’ q

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

theorem HEndPhase2-HInv4d:
\[ HEndPhase2 s s' p; HInv4d s q \implies HInv4d s' q \]
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 \(\in\) blocksOf s’ p
  from act
  have disk\': disk s’ = disk s
    by(auto simp add: Fail-def)
  from subsetD[OF HFail-blocksOf[OF act] bk]
  show \(\exists \) 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 act
  have bal bk = 0
    by(auto simp add: Fail-def InitDB-def
            Disk-isMajority)
  with Disk-isMajority
  show \(?\)thesis
    by(auto simp add: MajoritySet-def)

86
lemma HFail-HInv4d-q:
assumes act: HFail s s' p
and inv: HInv4d s q
and pnq: p ≠ q
shows HInv4d s' q
proof
from act pnq
have \(\text{disk}'\): \(\text{disk} \ s' = \text{disk} \ s\)
  and \(\text{dblock}'\): \(\text{dblock} \ s' q = \text{dblock} \ s \ q\)
  by (auto simp add: Fail-def)
from act pnq
have \(\text{blocksRead}'\): \(\forall \ q. \ \text{allRdBlks} \ s' q \subseteq \text{allRdBlks} \ s \ q\)
  by (auto simp add: Fail-def InitializePhase-def allRdBlks-def)
with \(\text{disk}'\) \(\text{dblock}'\)
have \(\text{blocksOf} \ s' q \subseteq \text{blocksOf} \ s q\)
  by (auto simp add: allRdBlks-def blocksOf-def rdBy-def, blast)
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 \(\text{disk}'\)
show \(?\text{thesis}\)
by (auto simp add: HInv4d-def)
qed

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

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

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

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

lemma HEndPhase0-HInv4d-p:
  assumes act: HEndPhase0 s s' p
  and inv: HInv4d s p
  and inv2c: Inv2c s
  and inv1: Inv1 s
  shows HInv4d s' p
proof(clarsimp simp add: HInv4d-def)
  fix bk
assume bk: bk ∈ blocksOf s' p
from subsetD(OF HEndPhase0-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}
from inv2c
have inv2c-inner: Inv2c-inner s p
by (auto simp add: Inv2c-def)
from bk HEndPhase0-some[OF act inv1]
  HEndPhase0-blocksOf2[OF act inv2c-inner] act
have bal bk ∈ bal ' (blocksOf s p)
by (auto simp add: EndPhase0-def)
with inv
show ?thesis
by (auto simp add: HInv4d-def)
qed

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

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

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

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

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

qed

end

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

C.5 Invariant 5

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

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

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

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

**C.5.1 Proof of Invariant 5**

The initial state implies Invariant 5.

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

We will use the notation used in the proofs of invariant 4, and prove the lemma \( \text{action-HInv5-p} \) and \( \text{action-HInv5-q} \) for each action, for the cases \( p = q \) and \( p \neq q \) respectively.

Also, for each action we will define an \( \text{action-allBlocks} \) lemma in the same way that we defined \( \text{-blocksOf} \) lemmas in the proofs of \( HInv2 \). 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: \( HInv5-\text{inner} \ s \ p \)
shows \( HInv5-\text{inner} \ s' \ p \)
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 \( \text{blocksOf} \ s' \ q \subseteq \text{blocksOf} \ s \ q \)
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: \( x \in \text{blocksOf} \ s' \ pa \) and \( x-na: \forall \ xa. \ x \notin \text{blocksOf} \ s \ xa \)
show \( x=\text{dblock} \ s' \ p \)
proof (cases \( p=pa \))
  case True
  from \( x-na \) \( x \notin \text{blocksOf} \ s \ p \)
  by auto
with True subsetD[OF HStartBallot-blocksOf[OF act] x-pa]
show \( \exists \text{thesis} \) 
by auto

next 
case False 
from \( x\text{-nblks} \subset \text{D[OF HStartBallot-blocksOf-q[OF act False] x-pa]} \) 
show \( \exists \text{thesis} \) 
by auto
qed
qed

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

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

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

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

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

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

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

proof
from inv5-2
obtain \( D \) \( q \)
where i1: IsMajority \( D \)
and i2: \( \forall d \in D \). bal (dblock \( s \) \( p \)) < mbal (disk \( s \) \( d \) \( q \))
and i3: \( \forall d \in D \). \( \neg \text{hasRead} \ s \ p \ d \ q \)
by (auto simp add: MajoritySet-def)
have pnq: \( p \neq q \)
proof
from inv4c phase
obtain \( D1 \) where r1: IsMajority \( D1 \) \( \land \) \( \forall d \in D1 \). mbal (disk \( s \) \( d \) \( p \)) = bal (dblock \( s \) \( p \))

94
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 \(s' \, p \land d \, q = hasRead s \, p \land d \, q\)
— In all disks \(q\) blocks don’t change
and \(\forall d.\) disk \(s' \, d \land q = disk s \land d \, q\)
by (auto simp add: Phase1or2Write-def hasRead-def)
with i2 i1 i3 majority-nonempty
have \(\forall d \in D.\) bal \(dblock s' \, p < mbal(disk s' \, d \, q) \land \neg hasRead s' \, p \land d \, q\)
  by auto
with i1
show \(?\)thesis
  by (auto simp add: MajoritySet-def)
qed

lemma \(HPhase1or2Write-HInv5-p:\)
assumes act: \(HPhase1or2Write\, s \, s' \, p \land d\)
and inv: \(HInv5-inner\, s \, p\)
and inv4: \(HInv4c\, s \, p\)
shows \(HInv5-inner\, s \, s' \, p\)
proof (auto simp add: HInv5-inner-def HInv5-inner-R-def)
assume phase': phase \(s' \, p = 2\)
and i2: \(\forall D \in MajoritySet.\, \forall q.\, \exists d \in D.\) bal \(dblock s' \, p < mbal (disk s' \, d \, q)\)
\(\rightarrow hasRead s' \, p \land d \, q\)
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

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

qed

theorem HPhase1or2Write-HInv5:
[ HPhase1or2Write s s' p d; HInv5-inner s q;
  HInv4c s p; HInv4a s p ] → HInv5-inner s'
by (blast dest: HPhase1or2Write-HInv5-q HPhase1or2Write-HInv5-p)

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

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

97
have \( \text{bal}(\text{dblock } s \ p) = \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: \text{Inv2c-inner-def})

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

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

proof
from \text{inv4c phase}
obtain \( D_1 \) where
r1: \( \text{IsMajority } D_1 \land (\forall d \in D_1. \text{mbal}(\text{disk } s \ d \ p) = \text{bal}(\text{dblock } s \ p)) \)
by (auto simp add: \text{HInv4c-def MajoritySet-def})

with \( \varepsilon \) \text{majorities-intersect}
have \( D \cap D_1 \neq \{} \) by \text{auto}
then obtain \( dd \) where
dd \in \( D \cap D_1 \)
by \text{auto}

with \( \varepsilon \ \varepsilon \ r1 \)
have \( \text{bal}(\text{dblock } s \ p) < \text{mbal}(\text{disk } s \ dd \ q) \land \text{mbal}(\text{disk } s \ dd \ p) = \text{bal}(\text{dblock } s \ p) \)
by \text{auto}

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

qed

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

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

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

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

qed

lemma \text{HPhase1or2ReadThen-HInv5-p:}
assumes \( \text{act}: \text{HPhase1or2ReadThen } s \ s' \ p \ d \ r \)
and \( \text{inv}: \text{HInv5-inner } s \ p \)
and \( \text{inv4}: \text{HInv4c } s \ p \)
and \( \text{inv2c}: \text{Inv2c } s \)
sows \( \text{HInv5-inner } s' \ p \)
proof (auto simp add: \text{HInv5-inner-def HInv5-inner-R-def})
assume \( \text{phase}': \text{phase } s' \ p = 2 \)
    and \( \varepsilon \) : \( \forall D' \in \text{MajoritySet}. \forall q, \exists d \in D. \text{bal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s' \ d \ q) \)
    \( \rightarrow \text{hasRead } s' \ p \ d \ q \)

98
with act have phase: phase s p = 2
  by(auto simp add: Phase1or2ReadThen-def)
show maxBalInp s' (bal (dblock s' p)) (inp (dblock s' p))
proof(rule HPhase1or2ReadThen-HInv5-1[OF act, of p])
  from inv2c
  have Inv2c-inner s p by(auto simp add: Inv2c-def)
  from HPhase1or2ReadThen-HInv5-p2[OF act inv4 this phase]
  have 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 d qq) < mbal (disk s d qq)
                                             ∧ ¬ hasRead s q d qq)
  shows ∃ D ∈ MajoritySet. ∃ qq. (∀ d ∈ D. bal (dblock s' q d qq) < mbal (disk s' d qq)
                                             ∧ ¬ hasRead s' q d qq)
proof -
  from inv5-2
  obtain D qq
    where i1: Is Majority D
    and i2: ∀ d ∈ D. bal (dblock s q d qq) < mbal (disk s d qq)
    and i3: ∀ d ∈ D. ¬ hasRead s q d qq
      by(auto simp add: MajoritySet-def)
  from act pnq
    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 d qq) < 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-HInv5-2[OF act pnq inv4a] inv i2 phase
show maxBalInp s (bal (dblock s q)) (inp (dblock s q))
by(auto simp add: HInv5-inner-def HInv5-inner-R-def, blast)
qed
qed

theorem HPhase1or2ReadThen-HInv5:
[ HPhase1or2ReadThen s s' p d r; HInv5-inner s q; 
  HInv5-inner s q ] ⇒ HInv5-inner s' q
by(blast dest: HPhase1or2ReadThen-HInv5-HInv5-1)

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: ∃ 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: 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 ] ⇒ HInv5-inner s' q
by(blast dest: HEndPhase2-HInv5-q HEndPhase2-HInv5-p)

lemma HEndPhase1-HInv5-p:
  assumes act: HEndPhase1 s s' p
  and inv4: HInv4 s
  and inv2a: Inv2a s
  and inv2a': Inv2a' s'
  and inv2c: Inv2c s
  and asm4: ¬maxBalInp s' (bal(dblock s' p)) (inp(dblock s' p))
  shows (∃ D∈MajoritySet. ∃ q. (∀ d∈D. bal(dblock s' p) < mbal(disk s' d q)
    ∧ ¬hasRead s' p d q))
proof –

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

proof
  - from \( \text{asm}_4 \)
    obtain \( bk \)
      where \( p31: bk \in \text{allBlocks } s' \land \text{bal}(\text{dblock } s' p) \leq \text{bal } bk \land bk \neq \text{dblock } s' p \)
      by (auto simp add: \text{maxBalImp-def})
    then obtain \( q \) where \( p32: bk \in \text{blocksOf } s' q \)
      by (auto simp add: \text{blocksOf-def})
  from \( \text{act} \)
  have \( \text{dblock}: p \neq q \rightarrow \text{dblock } s' q = \text{dblock } s q \)
    by (auto simp add: \text{EndPhase1-def})
  have \( bk \in \text{blocksOf } s q \)
  proof (cases \( p = q \))
    case True
      with \( p32 \) \( p31 \) \text{HEndPhase1-blocksOf[OF act]}
      show \( \theta \text{thesis} \)
        by auto
  next
    case False
    from \( \text{dblock[OF False]} \) \text{subsetD[OF HEndPhase1-blocksOf[OF act, of q] p32]}
    show \( \theta \text{thesis} \)
      by (auto simp add: \text{blocksOf-def})
  qed
  with \( p31 \)
  show \( \theta \text{thesis} \)
    by (auto simp add: \text{allBlocks-def})
  qed

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

have \( \exists q \in \text{UNIV} - \{ p \}. \ bk \in \text{blocksOf } s q \)
proof
  - from \( p22 \)
    obtain \( q \) where \( bk: bk \in \text{blocksOf } s q \)
      by (auto simp add: \text{blocksOf-def})
  from \( \text{act} \) \( p22 \)
  have \( \text{mbal}(\text{dblock } s p) \leq \text{bal } bk \)
    by (auto simp add: \text{EndPhase1-def})
  moreover
  from \( \text{act} \)
  have \( \text{phase } s p = 1 \)
    by (auto simp add: \text{EndPhase1-def})
  moreover
  from \( \text{inv}_4 \)
  have \( \text{HInv4b } s p \) by (auto simp add: \text{HInv4-def})
  ultimately
  have \( p \neq q \)
    using \( bk \)
    by (auto simp add: \text{HInv4-def HInv4b-def})
  with \( bk \)
show \( \text{thesis} \)
by auto
qed

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

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

from \( p23 \ \text{inv4} \)
have \( \text{i4d: } \exists D \in \text{MajoritySet}. \forall d \in D. \, \text{bal} bk \leq \text{mbal}(\text{disk } s \ d \ q) \)
by (auto simp add: \( \text{HInv4-def } \text{HInv4d-def} \))

from \( \text{i4d } p22 \)
show \( \text{thesis} \)
by force
qed

then obtain \( D \) where \( D_{\text{maj}}: D \in \text{MajoritySet} \) and \( p24: (\forall d \in D. \, \text{bal}(\text{dblock } s' \ p) \leq \text{mbal}(\text{disk } s \ d \ q)) \)
by auto

have \( p25: (\forall d \in D. \, \text{bal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s \ d \ q)) \)
proof –

from \( \text{inv2c} \)
have \( \text{Inv2c-inner } s \ p \)
by (auto simp add: \( \text{Inv2c-def} \))
with \( \text{act} \)

have \( \text{bal-pos: } 0 < \text{bal}(\text{dblock } s' \ p) \)
by (auto simp add: \( \text{Inv2c-inner-def } \text{EndPhase1-def} \))

with \( \text{inv2a'} \)
have \( \text{bal}(\text{dblock } s' \ p) \in \text{Ballot } p \cup \{0\} \)
by (auto simp add: \( \text{Inv2a-def } \text{Inv2a-inner-def} \text{Inv2a-innermost-def } \text{blocksOf-def} \))

with \( \text{bal-pos} \) have \( \text{bal-in-p: } \text{bal}(\text{dblock } s' \ p) \in \text{Ballot } p \)
by auto

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

hence \( (\forall d \in D. \, \text{mbal}(\text{disk } s \ d \ q) \in \text{Ballot } q \cup \{0\}) \)
by (auto simp add: \( \text{Inv2a-inner-def } \text{Inv2a-innermost-def } \text{blocksOf-def} \))

with \( p24 \) \( \text{bal-pos} \)

have \( (\forall d \in D. \, \text{mbal}(\text{disk } s \ d \ q) \in \text{Ballot } q \)
by force
with \( \text{Ballot-disj } p23 \) \( \text{bal-in-p} \)

have \( (\forall d \in D. \, \text{mbal}(\text{disk } s \ d \ q) \neq \text{bal}(\text{dblock } s' \ p) \)
by force
with \( p23 \) \( p24 \)
show \( \text{thesis} \)
by force
qed

with \( p23 \) \( \text{act} \)

have \( (\forall d \in D. \, \text{mbal}(\text{dblock } s' \ p) < \text{mbal}(\text{disk } s' \ d \ q) \land \neg \text{hasRead } s' \ p \ d \ q) \)
by (auto simp add: \( \text{EndPhase1-def } \text{InitializePhase-def } \text{hasRead-def} \))

with \( D_{\text{maj}} \)

103
show ?thesis
by blast
qed

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

lemma HEndPhase1-blocksOf-q:
assumes act: HEndPhase1 s s' p
and pmq: p\neq q
shows blocksOf s' q \subseteq blocksOf s q
proof –
from act pmq
have dblock: \{dblock s' q\} \subseteq \{dblock s q\}
and disk: disk s' = disk s
and blk: blocksRead s' q = blocksRead s q
by(auto simp add: HEndPhase1-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 pmq act
have (UN qq d . rdBy s' q qq d) \subseteq (UN qq d . rdBy s q qq d)
by(auto simp add: HEndPhase1-def InitializePhase-def rdBy-def split: split-if-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 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]
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:\n  \( ¬\maxBalInp s' (\text{bal}(\text{dblock} s' q)) (\text{inp}(\text{dblock} s' q)) \)

  shows (∃ D∈MajoritySet. ∃ qq. (∀ d∈D. bal(\text{dblock} s' q) < mbal(\text{disk} s d qq)
  ∧ ¬hasRead s' q d qq))

proof –
  from act pnq
  have phase s' q = phase s q
  and phase-p: phase s p = 1
  and disk: disk s' = disk s
  and dblock: dblock s' q = dblock s q
  and bal: bal(dblock s' p) = mbal(dblock s p)
  by(auto simp add: EndPhase1-def InitializePhase-def)
  with phase'
  have phase: phase s q = 2 by auto
  from phase inv2c
  have bal-dblk-q: bal(dblock s q) ∈ Ballot q
  by(auto simp add: Inv2c-def Inv2c-inner-def)
  have ∃ D∈MajoritySet. ∃ qq. (∀ d∈D. bal(dblock s q) < mbal(dblock 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
\quad Inv2a-innermost-def blocksOf-def)

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

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

then obtain D
where p22: \( D \in \text{MajoritySet} \land (\forall d \in D. \text{bal}(\text{dblock } s \ q) < \text{mbal}(\text{disk } s \ d \ p) \land \text{hasRead } s \ p \ d \ q) \)
by auto
have p23: \( \forall d \in D. (\exists \text{block}=\text{dblock } s \ q, \text{proc}=q) \notin \text{blocksRead } s \ p \ d \)
proof –
have dblock \( s \ q \in \text{allBlocksRead } s \ p \longrightarrow \text{inp}(\text{dblock } s' \ p) = \text{inp}(\text{dblock } s \ q) \)
proof auto
assume dblock-q: \( \text{dblock } s \ q \in \text{allBlocksRead } s \ p \)
from inv2a-q
have \((\text{bal}(\text{dblock}\ s\ q)=0) = (\text{inp}(\text{dblock}\ s\ q) = \text{NotAnInput})\)
  by\((\text{auto simp add: Inv2a-def Inv2a-inner-def blocksOf-def Inv2a-innermost-def})\)
with \(\text{bal-dblk-q Ballot-nzero dblock-q InputsOrNi}\)
have \(\text{dblock-q-nib}: \text{dblock}\ s\ q \in \text{nonInitBlks}\ s\ p\)
  by\((\text{auto simp add: nonInitBlks-def blocksSeen-def blocksOf-def Inv2a-innermost-def})\)
with \(\text{act}\)
have \(\text{dblock-max}: \text{inp}(\text{dblock}\ s\' p) = \text{inp}(\text{maxBlk}\ s\ p)\)
  by\((\text{auto simp add: EndPhase1-def})\)
from \(\text{maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]}\)
have \(\text{max-in-nib}: \text{maxBlk}\ s\ p \in \text{nonInitBlks}\ s\ p\ ..\)
hence \(\text{nonInitBlks}\ s\ p \subseteq \text{allBlocks}\ s\)
  by\((\text{auto simp add: allBlocks-def nonInitBlks-def blocksSeen-def blocksOf-def rdBy-def allBlocksRead-def allRdBlks-def})\)
with \(\text{True subsetD[OF this max-in-nib]}\)
have \(\text{bal}(\text{dblock}\ s\ q) \leq \text{bal}(\text{maxBlk}\ s\ p) \rightarrow \text{inp}(\text{maxBlk}\ s\ p) = \text{inp}(\text{dblock}\ s\ q)\)
  by\((\text{auto simp add: maxBalInp-def})\)
with \(\text{maxBlk-in-nonInitBlks[OF dblock-q-nib inv1]}\)
dblock-q-nib dblock-max
show \(\text{inp}(\text{dblock}\ s\' p) = \text{inp}(\text{dblock}\ s\ q)\)
  by auto
qed
with \(p21\)
have \(\text{dblock}\ s\ q \notin \text{block}\ \cdot\ \text{allRdBlks}\ s\ p\)
  by\((\text{auto simp add: allBlocksRead-def})\)
hence \(\forall d. \text{dblock}\ s\ q \notin \text{block}\ \cdot\ \text{blocksRead}\ s\ p\ d\)
  by\((\text{auto simp add: allRdBlks-def})\)
thus \(?\text{thesis}\)
  by force
qed
have \(p24: \forall d \in D. \neg(\exists\ br\in\ \text{blocksRead}\ s\ q\ d. \text{bal}(\text{dblock}\ s\ q) \leq \text{mbal}(\text{block}\ br))\)
proof –
  from \(\text{inv2c phase}\)
  have \(\forall d. \forall br\in\text{blocksRead}\ s\ q\ d. \text{mbal}(\text{block}\ br) < \text{mbal}(\text{dblock}\ s\ q)\)
    and \(\text{bal}(\text{dblock}\ s\ q) = \text{mbal}(\text{dblock}\ s\ q)\)
    by\((\text{auto simp add: Inv2c-def Inv2c-inner-def})\)
  thus \(?\text{thesis}\)
    by force
qed
have \(p25: \forall d \in D. \neg\text{hasRead}\ s\ q\ d\ p\)
proof \text{auto}\nfix \(d\)
assume \(d\in D: d \in D\)
and \(\text{hasRead-qdp}: \text{hasRead}\ s\ q\ d\ p\)
have \(p31: (\exists\ \text{block}=\text{dblock}\ s\ p, \text{proc}=p)\in\text{blocksRead}\ s\ q\ d\)
proof –
  from \(d\in D\ p22\)
107
have \textit{hasRead-pdq} \textit{hasRead s p d q} by auto
with \textit{hasRead-qdp phase phase-p inv3}
have \textit{HInv3-R s q p d}
by(auto simp add: \textit{HInv3-def} \textit{HInv3-inner-def} \textit{HInv3-L-def})
with p23 d-in-D
show ?thesis
by(auto simp add: \textit{HInv3-R-def})
qed
from p21 act
have p32: \textit{bal(dblock s q)} < \textit{mbal(dblock s p)}
by(auto simp add: \textit{EndPhase1-def})
with p31 d-in-D \textit{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: \textit{HInv5-def} \textit{HInv5-inner-def} \textit{HInv5-inner-R-def})
qed
then obtain \textit{D} \textit{qq}
where \textit{D} \in \textit{MajoritySet} \land \forall d \in \textit{D}. \textit{bal(dblock s q)} < \textit{mbal(disk s d \textit{qq})} \land \neg \textit{hasRead s q d \textit{qq}}
by(auto)
moreover
from act pnq
have \forall d. \textit{hasRead s' q d \textit{qq}} = \textit{hasRead s q d \textit{qq}}
by(auto simp add: \textit{EndPhase1-def InitializePhase-def hasRead-def})
ultimately show ?thesis
using disk dblock
by auto
qed

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

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

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

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

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

lemma HFail-HInv5-q:
  assumes act: HFail s s' p
  and inv: HInv5-inner s q
  and pnq: p≠q
  and inv2a: Inv2a s'
  ....

110
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 $nR2$: $\forall D \in \text{MajoritySet}.$

$\forall qa. \exists d \in D. \ \text{bal (dblock } s' \ q) < \text{mbal (disk } d \ qa) \Longrightarrow \text{hasRead } s' \ q \ d \ qa \ (\text{is } ?P \ s')$

from $H\text{Fail-HInv5-q2}[OF act pnq]$ have $\neg (\neg ?P \ s) \Longrightarrow \neg(\neg ?P \ s')$

by auto

with $nR2$

have $P$: $\neg ?P \ s$

by blast

from $\text{inv2a}$

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

from act pnq phase'

have phase $ s \ q = 2$

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

with $\text{inv } H\text{Fail-HInv5-q1}[OF act pnq inv2a'] \ P$

show $\text{maxBalInp } s' (\text{bal (dblock } s' \ q)) (\text{inp (dblock } s' \ q))$

by(auto simp add: $H\text{Inv5-inner-def } H\text{Inv5-inner-R-def}$)

qed

theorem $H\text{Fail-HInv5}$:

[ $H\text{Fail } s s' p; \ H\text{Inv5-inner } s \ q; \ H\text{Inv5-inner-R } s' \ q$ ] $\Longrightarrow H\text{Inv5-inner } s' \ q$

by(blast dest: $H\text{Fail-HInv5-q } H\text{Fail-HInv5-p}$)

lemma $H\text{Phase0Read-HInv5-p}$:

$H\text{Phase0Read } s s' p d \Longrightarrow H\text{Inv5-inner } s' \ p$

by(auto simp add: $H\text{Phase0Read-def } H\text{Inv5-inner-def}$)

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

assumes act: $H\text{Phase0Read } s s' p d$

shows allBlocks $ s' \subseteq \text{allBlocks } s$

using $H\text{Phase0Read-blocksOf}[OF act]$

by(auto simp add: allBlocks-def)

lemma $H\text{Phase0Read-HInv5-1}$:

assumes act: $H\text{Phase0Read } s s' p d$

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

shows $\text{maxBalInp } s' (\text{bal (dblock } s' \ q)) (\text{inp (dblock } s' \ q))$

using assms and $H\text{Phase0Read-blocksOf}[OF act]$

by(auto simp add: $H\text{Phase0Read-def } maxBalInp-def allBlocks-def$)

lemma $H\text{Phase0Read-HInv5-q2}$:

assumes act: $H\text{Phase0Read } s s' p d$

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

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

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

111
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 $s' = disk s$
  and blocksRead: $\forall d. \text{blocksRead} s' q d = \text{blocksRead} s q d$
  and dblock: $\text{dblock} s' q = \text{dblock} s q$
by(auto simp add: Phase0Read-def InitializePhase-def)
with inv5-2
show $?thesis$
  by(auto simp add: hasRead-def)
qed

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

point

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

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

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

112
and blks: blocksRead s' q = blocksRead s q

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

from disk

have disk': {disk s' d q | d . d ∈ UNIV} ⊆ {disk s d q | d . d ∈ UNIV} (is ?D' ⊆ ?D)

  by auto

from pnq act

have (UN qq d. rdBy s' q qq d) ⊆ (UN qq d. rdBy s q qq d)

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

  rdBy-def split: split-if-asm, blast)

hence {block br | br. br ∈ (UN qq d. rdBy s' q qq d)} ⊆

  {block br | br. br ∈ (UN qq d. rdBy s q qq d)}

  (is ?R' ⊆ ?R)

  by 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-q[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 \textit{maxBalInp} \ s' \ (\textit{bal}(\textit{dblock} \ s' \ q)) \ (\textit{inp}(\textit{dblock} \ s' \ q))

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

fix \ bk

assume \ bk: \ bk \in\ allBlocks \ s'

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

from \ act \ pnq

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

from \ \textit{subsetD}[OF \ \textit{HEndPhase0-allBlocks}[OF \ \textit{act} \ bk]

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

proof

assume \ bk: \ bk \in\ allBlocks \ s

with \ inv5-1 \ \textit{dblock}' \ \textit{bal}

show \ ?thesis

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

next

assume \ bk: \ bk \in\ \{\textit{dblock} \ s' \ p\}

with \ \textit{HEndPhase0-some}[OF \ \textit{act} \ \textit{inv1} \ \textit{act}]

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

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

then obtain \ \textit{ba}

where \ \textit{ba-blksread}: \ \textit{ba} \in\ \textit{allBlocksRead} \ s \ p

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

by \ auto

have \ \textit{allBlocksRead} \ s \ p \subseteq \ allBlocks \ s

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

from \ \textit{subsetD}[OF \ \textit{this} \ \textit{ba-blksread}] \ \textit{ba-balinp} \ \textit{bal} \ \textit{bk} \ \textit{dblock}' \ inv5-1

show \ ?thesis

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

ged

ged

lemma \ \textit{HEndPhase0-Hinv5-q2}:

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

and \ pnq: \ p \neq q

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

\land \ \neg\textit{hasRead} \ s \ q \ d \ \textit{qq})

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

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

proof –

from \ act \ pnq

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

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

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

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

with \ inv5-2

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

qed

lemma HEndPhase0-HInv5-q:
  assumes act: HEndPhase0 s s' p
  and inv1: 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-HEndPhase0-HInv5-p)

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

lemma I2e:
  assumes nxt: HNext s s'
  and inv: HInv1 s ∧ HInv2 s ∧ HInv2 s' ∧ HInv3 s ∧ HInv3 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.

115
**definition** valueChosen :: state ⇒ InputsOrNi ⇒ bool

**where**

\[ \text{valueChosen } s \ v = \]
\[ (\exists b \in (\text{UN } p. \text{Ballot } p). \]
\[ \text{maxBalInp } s \ b \ v \]
\[ \land (\exists p. \exists D \in \text{MajoritySet} . (\forall d \in D. \ b \leq \text{bal}(\text{disk } s \ d \ p) \]
\[ \land (\forall q. (\text{phase } s \ q = 1 \]
\[ \land b \leq \text{mbal}(\text{dblock } s \ q) \]
\[ \land \text{hasRead } s \ q \ d \ p) \]
\[ ) \rightarrow (\exists b \in \text{blocksRead } s \ q \ d. \ b \leq \text{bal}(\text{block } br)) \]

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

**proof**

- from bk-blocksOf inv2a
  have inv2a-bk: Inv2a-innermost s r bk
    by (auto simp add: Inv2a-def Inv2a-inner-def)

- from Ballot-nzero asm1
  have 0 < b by auto

- with b-bal
  have 0 < 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
         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 jasm3 act True
    have p42: ∃ br ∈ blocksRead s q d. b ≤ bal(block br)
      by (auto simp add: EndPhase1-def)
    from True act
    have thesis-L: b ≤ bal(dblock s' q)
      by (auto simp add: EndPhase1-def)
    from p42
    have inp(dblock s' q) = v

117
proof auto
fix br
assume br: br ∈ blocksRead s q d
and b-bal: b ≤ bal (block br)
hence br-rdBy: br ∈ (UN q d. rdBy s (proc br) q d)
  by(auto simp add: rdBy-def)
hence br-blksof: block br ∈ blocksOf s (proc br)
  by(auto simp add: blocksOf-def)
from br have br-bseen: block br ∈ blocksSeen s q
  by(auto simp add: blocksSeen-def allBlocksRead-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 \( \text{asm3 HEndPhase1-allBlocks[OF act]} \)
show \(?\text{thesis}\)
by (auto simp add: maxBalInp-def)
qed

next
case \(False\)
have \(\text{dblock } s' q \in \text{allBlocks } s'\)
  by (auto simp add: allBlocks-def blocksOf-def)
show \(?\text{thesis}\)
proof (auto simp add: maxBalInp-def)
  fix \(bk\)
  assume \(bk: bk \in \text{allBlocks } s\)
  and \(b-bal: b \leq \text{bal } bk\)
  from \(\text{subsetD[OF HEndPhase1-allBlocks[OF act] bk]}\)
  show \(\text{inp } bk = v\)
proof
  assume \(bk: bk \in \text{allBlocks } s\)
  with \(\text{asm3 b-bal}\)
  show \(?\text{thesis}\)
  by (auto simp add: maxBalInp-def)
next
  assume \(bk: bk \in \{ \text{dblock } s' q \}\)
  from \(\text{act False}\)
  have \(\sim b \leq \text{bal } (\text{dblock } s' q)\)
  by (auto simp add: EndPhase1-def)
  with \(bk b-bal\)
  show \(?\text{thesis}\)
  by (auto)
qed
qed

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

with \( \text{act} \)

have \( p_{31}: \text{phase} s q = 1 \)
  and \( p_{32}: \text{dblock} s' q = \text{dblock} s q \)
  by (auto simp add: \text{EndPhase1-def} split: split-if-asm)

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

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

moreover

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

with \( \text{act} \)

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

ultimately

have \( \exists br \in \text{blocksRead} s q d. b \leq \text{bal} (\text{block} br) \)
  using \( p_{31} \) \( \text{asm4} d \)
  by blast

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

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

qed

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

assumes \( \text{act}: \text{HEndPhase1} s s' q \)
and \( \text{vc}: \text{valueChosen} s v \)

and \( \text{inv1}: \text{Inv1} s \)
and \( \text{inv2a}: \text{Inv2a} s \)
and \( \text{inv2b}: \text{Inv2b} s \)
and \( \text{v-input}: v \in \text{Inputs} \)

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

proof

from \( \text{vc} \)

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

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

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

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

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

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

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

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

ultimately

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

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

by blast

with \( \text{act hasRead} \)

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

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

qed

theorem \( \text{HStartBallot-valueChosen} \):

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

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

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

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

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

fix \( bk \)

assume \( bk: bk \in \text{allBlocks } s' \)
and $b$-bal: $b \leq \text{bal}_{bk}$
from $\text{subsetD}[\text{OF HPhase1or2Write-allBlocks}[\text{OF act} \ bk]]$ asm3 $b$-bal
show $\text{inp}_{bk} = v$
  by(auto simp add: maxBalInp-def)
qed

lemma $\text{HPhase1or2Write-valueChosen2}$:
  assumes act: $\text{HPhase1or2Write} \ s \ s' \ pp \ d$
  and asm2: $D \in \text{MajoritySet}$
  and asm4: $\forall d \in D. \ b \leq \text{bal}(\text{disk} \ s \ d \ p)$
      $\land$ $(\forall q. (\text{phase} \ s \ q = 1$
      $\land \ b \leq \text{mbal}(\text{dblock} \ s \ q)$
      $\land \ \text{hasRead} \ s \ q \ d \ p)$
      $\rightarrow$ $(\exists \text{br} \in \text{blocksRead} \ s \ q \ d. \ b \leq \text{bal}(\text{block} \ br))$ (is ?P $s$)
  and inv4: $\text{HInv4a} \ s \ pp$
shows $\exists P s'$
proof(auto)
  fix $d1$
  assume $d: d1 \in D$
  show $b \leq \text{bal}(\text{disk} \ s' \ d1 \ p)$
  proof(cases $d1 = d \land pp = p$
  case True
    with inv4 act
    have $H\text{Inv4a}2 \ s \ p$
      by(auto simp add: $\text{Phase1or2Write-def}$ $\text{HInv4a-def}$)
    with asm2 majorities-intersect
    have $\exists d d1 \in D. \ \text{bal}(\text{disk} \ s \ dd \ p) \leq \text{bal}(\text{dblock} \ s \ p)$
      by(auto simp add: $\text{HInv4a2-def}$ $\text{MajoritySet-def}$)
    then obtain $dd$ where p41: $dd \in D \land \text{bal}(\text{disk} \ s \ dd \ p) \leq \text{bal}(\text{dblock} \ s \ p)$
      by auto
    from asm4 p41
    have $b \leq \text{bal}(\text{disk} \ s \ dd \ p)$
      by auto
    with p41
    have p42: $b \leq \text{bal}(\text{dblock} \ s \ p)$
      by auto
    from act True
    have $\text{dblock} \ s \ p = \text{disk} \ s' \ d \ p$
      by(auto simp add: $\text{Phase1or2Write-def}$)
    with p42 True
    show $\exists \text{thesis}$
      by(auto)
  next
  case False
    with act asm4 d
    show $\exists \text{thesis}$
      by(auto simp add: $\text{Phase1or2Write-def}$)
qed
next
fix $d\ q$
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 $\text{phase'}\ \text{act}\ \text{hasRead}$
have $p31$: $\text{phase}\ s\ q = 1$
and $p32$: $\text{dblock}\ s' q = \text{dblock}\ s q$
by (auto simp add: $\text{Phase1or2Write-def}\ \text{InitializePhase-def}$
\hspace{1cm} \text{hasRead-def}\ \text{split : split-if-asm})
with $\text{dblk-mbal}$
have $b \leq \text{mbal} (\text{dblock}\ s q)$ by auto
moreover
from $\text{act}\ \text{hasRead}$
have $\text{hasRead}\ s q d p$
by (auto simp add: $\text{Phase1or2Write-def}\ \text{InitializePhase-def}$
\hspace{1cm} \text{hasRead-def}\ \text{split : split-if-asm})
ultimately
have $\exists br \in \text{blocksRead}\ s\ q\ d.\ b \leq \text{bal}(\text{block}\ br)$
using $p31\ \text{asm4}\ d$
by blast
with $\text{act}\ \text{hasRead}$
show $\exists br \in \text{blocksRead}\ s' q\ d.\ b \leq \text{bal}(\text{block}\ br)$
by (auto simp add: $\text{Phase1or2Write-def}\ \text{InitializePhase-def}$
\hspace{1cm} \text{hasRead-def})
qed

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

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 s)
  shows ?P s'
proof (auto)
  fix dd
  assume d: dd ∈ D
  with act asm4
  show b ≤ bal (disk s' dd p)
    by (auto simp add: Phase1or2ReadThen-def)
  fix dd qq
  assume d: dd ∈ D
  and phase': phase s' qq = Suc 0
  and dblk-mbal: b ≤ mbal (dblock s' qq)
  and hasRead: hasRead s' qq dd p
  show ∃ br ∈ blocksRead s' qq dd. b ≤ bal (block br)
proof (cases d = dd ∧ qq = q ∧ pp = p)
  case True
  from d asm4
  have b ≤ bal (disk s dd p)
    by auto
  with act True
  show ?thesis
    by (auto simp add: Phase1or2ReadThen-def)
next
  case False
  with phase' act
  have p31: phase s qq = 1
and p32: 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: split-if-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)
  \( \land (\forall q. \begin{array}{l}
    \text{phase } s q = 1
    \\
    \land b ≤ mbal (dblock s q)
    \\
    \land hasRead s q d p
  \end{array} ) \rightarrow (\exists br ∈ blocksRead s q d. b ≤ bal (block br)))\)
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: \( \text{maxBalInp } s b v \)
  shows \( \text{maxBalInp } s' b v \)
proof (auto simp add: maxBalInp-def)
  fix bk
  assume bk: \( bk \in \text{allBlocks } s' \)
  and b-bal: \( b \leq \text{bal } bk \)
  from \( \text{subsetD OF HEndPhase2-allBlocks[OF } \text{act} ] \) bk asm3 b-bal
  show \( \text{inp } bk = v \)
    by (auto simp add: maxBalInp-def)
qed

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

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

qed

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

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

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

qed

theorem HFail-valueChosen:
  assumes act: HFail s s' q
  and vc: valueChosen s v
  and v-input: \( v \in \text{Inputs} \)
  shows valueChosen s' v
proof
  from vc obtain b p D where
    \( \text{asm1: } b \in (\text{UN } p \cdot \text{Ballot } p) \)
  and asm2: D\( \in \text{MajoritySet} \)
  and asm3: maxBalInp s b v
  and asm4: \( \forall d \in D. \ b \leq \text{bal}(\text{disk } s d p) \)
    \( \land (\forall q. (\ \text{phase } s q = 1 \)
    \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 } 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 \in \text{allBlocks } s'
  and b-bal: b\leq \text{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: \( \forall d \in D. \ b \leq \text{bal}(\text{disk } s d p) \)
    \( \land (\forall q. (\ \text{phase } s q = 1 \)
    \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 } br)) \)
(is \?P s)
  shows \?P s'
proof (auto)
fix \(d\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

next
fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})

fix \(d \ q\)
assume \(d: d \in D\) with \(\text{act asm4}\)
show \(b \leq \text{bal} (\text{disk } s' \ d \ p)\)
by (\text{auto simp add: Phase0Read-def})
and asm4: \( \forall d \in D. \ b \leq \text{bal}(\text{disk} s d p) \)
\[ \land (\forall q. (\text{phase} s q = 1 \land b \leq \text{mbal}(\text{dblock} s q) \land \text{hasRead} s q d p) \rightarrow (\exists br \in \text{blocksRead} s q d. \ b \leq \text{bal}(\text{block} br))) \]

by (auto simp add: valueChosen-def)

from HPhase0Read-maxBalInp[OF act asm3]
have maxBalInp s' b v.
with HPhase0Read-valueChosen2[OF act asm4] asm1 asm2
show ?thesis
by (auto simp add: valueChosen-def)

qed

lemma HEndPhase0-maxBalInp:
assumes act: HEndPhase0 s s' q
and asm3: maxBalInp s b v
and inv1: Inv1 s
shows maxBalInp s' b v
proof (auto simp add: maxBalInp-def)
fix bk
assume bk: bk \in allBlocks s'
and b-bal: \( b \leq \text{bal} \ bk \)
from subsetD[OF HEndPhase0-allBlocks[OF act] bk]
show inp bk = v
proof
assume bk: bk \in allBlocks s
with asm3 b-bal
show ?thesis
by (auto simp add: maxBalInp-def)

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

lemma \(\text{HEndPhase0-valueChosen2}\):
\[
\text{assumes } \text{act}: \text{HEndPhase0 } s \; s' \; q \\
\text{and } \text{asm4}: \forall d \in D. \; \; b \leq \text{bal(disk } s \; d \; p) \\
\land (\forall q. ( \text{phase } s \; q = 1 \\
\land b \leq m\text{bal(dblock } s \; q) \\
\land \text{hasRead } s \; q \; d \; p) \\
\rightarrow (\exists br \in \text{blocksRead } s \; q \; d. \; \; b \leq \text{bal(block } br))) \; \; (\text{is } ?P \; s)
\]
shows ?P \; s'
proof (auto)
fix \; d
assume \; d: \; d \in D
with \; \text{act} \; \text{asm4}
show \; b \leq \text{bal(disk } s' \; d \; p) \\
by (auto simp add: \text{EndPhase0-def})
fix \; d \; q
assume \; d: \; d \in D \\
and \; \text{phase'}: \; \text{phase } s' \; q = \text{Suc 0} \\
and \; \text{dblk-mbal}: \; b \leq m\text{bal(dblock } s' \; q) \\
and \; \text{hasRead}: \; \text{hasRead } s' \; q \; d \; p
from \; \text{phase'} \; \text{act} \; \text{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} \text{InitializePhase-def} \text{hasRead-def split: split-if-asm})
with \; \text{dblk-mbal}
have \; b \leq m\text{bal(dblock } s \; q) \; \text{by auto}
moreover
from \; \text{act} \; \text{hasRead}
have \; \text{hasRead } s \; q \; d \; p \\
by (auto simp add: \text{EndPhase0-def} \text{InitializePhase-def} \text{hasRead-def split: split-if-asm})
ultimately
have \; \exists br \in \text{blocksRead } s \; q \; d. \; b \leq \text{bal(block } br) \\
using \; p31 \; \text{asm4} \; d \\
by blast
with \; \text{act} \; \text{hasRead}
show \; \exists br \in \text{blocksRead } s' \; q \; d. \; b \leq \text{bal(block } br) \\
by (auto simp add: \text{EndPhase0-def} \text{InitializePhase-def} \text{hasRead-def})
qed

theorem \(\text{HEndPhase0-valueChosen}\):
\[
\text{assumes } \text{act}: \text{HEndPhase0 } s \; s' \; q \\
\text{and } \text{vc}: \text{valueChosen } s \; v \\
\text{and } \text{v-input}: \; v \in \text{Inputs} \\
\text{and } \text{inv1}: \; \text{Inv1 } s \\
\text{shows } \text{valueChosen } s' \; v
\]
proof –
from \; \text{vc}
obtain \( b \) \( p \) \( D \) where

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

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

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

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

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

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

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

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

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

show \( \text{thesis} \)

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

qed

end

theory \( \text{DiskPaxos-Inv6 imports DiskPaxos-Chosen begin} \)

C.7 Invariant 6

The final conjunct of \( \text{HInv} \) 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 \( \text{HInv6 :: state } \Rightarrow \text{ bool} \)

where

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

theorem \( \text{HInit-HInv6: } \text{HInit } s \Rightarrow \text{HInv6 } s \)

by (auto simp add: \( \text{HInit-def Init-def InitDB-def HInv6-def} \))

lemma \( \text{HEndPhase2-Inv6-1:} \)

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

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

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

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

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

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

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

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

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

from \( \text{inv5 act} \)

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

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

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

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

\textbf{case True}

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

proof
  have \(\neg(\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal (dblock s p) < mbal (disk s d q)} \land \neg \text{hasRead s p d q})\))
  proof
    fix D q
    assume Dmaj: D \in MajoritySet
    from ep2-maj Dmaj majorities-intersect
    have \(\exists d \in D. d \in \text{disksWritten s p}\)
    \(\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q})\)
    by (auto simp add: MajoritySet-def, blast)
    then obtain d
      where dinD: d \in D
      and ddisk: d \in \text{disksWritten s p}
      and dhasR: \(\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q}\)
      by auto
    from inv2b
    have Inv2b-inner s p d
    by (auto simp add: Inv2b-def)
    with ddisk
    have disk s d p = dblock s p
    by (auto simp add: Inv2b-inner-def)
    with inv2c phase
    have bal (dblock s p) = mbal (disk s d p)
    by (auto simp add: Inv2c-def Inv2c-inner-def)
    with dhasR dinD
    show \(\exists d \in D. \text{bal (dblock s p) < mbal (disk s d q)} \longrightarrow \text{hasRead s p d q}\)
    by auto
  qed
  with inv5R
  show \(?\text{thesis}\)
  by (auto simp add: HInv5-inner-R-def)
qed

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

proof
  from act
  have outpt': outpt s' = (outpt s) (p := inp (dblock s p))
  by (auto simp add: EndPhase2-def)
  have outpt'\'-q: \(\forall q. p \neq q \longrightarrow \text{outpt s'} q = \text{NotAnInput}\)
  proof
    fix q
    assume pnq: p \neq q
    from outpt' pnq
    have outpt s' q = outpt s q
    by (auto simp add: EndPhase2-def)
    with True inv2c


  135
show output $s' \neq \text{NotAnInput}$
  by (auto simp add: Inv2c-def Inv2c-inner-def)
qed

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

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

from ep2-maj inv2b majorities-intersect
have $\exists D \in \text{MajoritySet}. \forall d \in D. \text{disk s d p = dblock s p}$
  $\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q})$
  by (auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def)
then obtain $D$
  where Dmaj: $D \in \text{MajoritySet}$
  and p34: $\forall d \in D. \text{disk s d p = dblock s p}$
  $\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead s p d q})$
  by auto
have p35: $\forall q. \forall d \in D. (\text{phase s q=1} \land \text{bal(dblock s p)\leq mbal(dblock s q)} \land \text{hasRead s q d p})$
  $\implies (|\text{block=dblock s p, proc=p}| \in \text{blocksRead s q d})$
proof auto
  fix $q d$
  assume dD: $d \in D \land \text{phase-q: phase s q= Suc 0}$
  and bal-mbal: $\text{bal(dblock s p)\leq mbal(dblock s q)} \land \text{hasRead: hasRead s q d p}$
  from phase inv2c
have \( \text{bal}(\text{dblock } s \ p) = \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)
moreover
from inv2c phase
have \( \forall \ br \in \text{blocksRead } s \ p \ d. \ \text{mbal}(\text{block } br) < \text{mbal}(\text{dblock } s \ p) \)
by (auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
have p41: \( \{\text{block= } \text{dblock } s \ q, \ \text{proc= } q\}\in \text{blocksRead } s \ p \ d \)
using bal-mbal
by auto
from phase phase-q
have \( p \neq q \) by auto
with p34 dD
have \( \text{hasRead } s \ p \ d \ q \)
by auto
with phase phase-q hasRead inv3 p41
show \( (\exists\ \text{br} \in \text{blocksRead } s \ p \ d. \ \text{bal}(\text{block } br) = \text{bal}(\text{dblock } s \ p)) \)
proof (auto)
fix \( q \ d \)
assume dD: \( d \in D \) and phase-q: \( \text{phase } s' q = \text{Suc } 0 \)
and bal: \( \text{bal } (\text{dblock } s \ p) \leq \text{mbal } (\text{dblock } s' q) \)
and hasRead: \( \text{hasRead } s' q d \ p \)
from phase-q act
have phase s' q=phase s q \( \land \text{dblock } s' q=\text{dblock } s q \) \( \land \text{hasRead } s' q d \ p=\text{hasRead } s q d \ p \)
by (auto simp add: EndPhase2-def HasRead-def InitializePhase-def)
with p35 phase-q bal hasRead dD
have \( \{\text{block= } \text{dblock } s \ p, \ \text{proc= } p\}\in \text{blocksRead } s' q d \)
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: \( \forall \ q. \ \forall d \in D. \ \text{phase } s' q = 1 \ \land \ \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \) \( \land \)
hasRead s' q d p
\( \rightarrow (\exists\ \text{br} \in \text{blocksRead } s' q d. \ \text{bal}(\text{block } br) \leq \text{bal}(\text{block } br)) \)
by force
from act
have bal-dblock: \( \text{bal}(\text{dblock } s' p) = \text{bal}(\text{dblock } s \ p) \)
and disk: disk s' = disk s
by (auto simp add: EndPhase2-def)
from bal-dblock p33
have maxBalInp s' (bal (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 maxBalInp s' (\( \text{bal}(\text{dblock } s \ p) \)) (chosen s') \land
  (\( \exists D \in \text{MajoritySet}. \\forall d \in D. \text{bal}(\text{dblock } s \ p) \leq \text{bal}(\text{disk } s' \ d \ p) \land
    (\forall q. \text{phase } s' q = \text{Suc } 0 \land
    \text{bal}(\text{dblock } s \ p) \leq \text{mbal}(\text{dblock } s' q) \land \text{hasRead } s' q d p \rightarrow
    (\exists \text{br } \in \text{blocksRead } s' q d. \text{bal}(\text{dblock } s \ p) \leq \text{bal}(\text{block } \text{br})))))
  using p36-2 Dmaj
  by auto
moreover
from phase inv2c
have \( \text{bal}(\text{dblock } s \ p) \in \text{Ballot } p \)
  by (auto simp add: Inv2c-def Inv2c-inner-def)
ultimately
show \(?thesis\)
  by (auto simp add: valueChosen-def)
next
case False
with act
have p31: chosen s' = chosen s
  by (auto simp add: HNextPart-def)
from False inv
have valueChosen s (chosen s)
  by (auto simp add: HInv6-def)
from HEndPhase2-valueChosen[OF act this] p31 False InputsOrNi
show \(?thesis\)
  by auto
qed

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

139
show \( ?\text{thesis} \)
by auto
qed

lemma \( H\text{EndPhase2-Inv6-2} \):
assumes \( \text{act}: H\text{EndPhase2} s s' p \)
and \( \text{inv}: H\text{Inv6} s \)
and \( \text{inv2b}: \text{Inv2b} s \)
and \( \text{inv2c}: \text{Inv2c} s \)
and \( \text{inv3}: H\text{Inv3} s \)
and \( \text{inv5}: H\text{Inv5-inner} s p \)
and \( \text{asm}: \text{outpt} s' r \neq \text{NotAnInput} \)
shows \( \text{outpt} s' r = \text{chosen} s' \)
proof (cases chosen \( s = \text{NotAnInput} \))
case True
with \( \text{inv2c} \)
have \( \forall q. \text{outpt} s q = \text{NotAnInput} \)
by (auto simp add: Inv2c-def Inv2c-inner-def)
with True \( \text{act} \) \( \text{asm} \)
show \( ?\text{thesis} \)
by (auto simp add: HNextPart-def split: split-if-asm)
next
case False
with \( \text{inv} \)
have \( p31: \text{valueChosen} s (\text{chosen} s) \)
by (auto simp add: HInv6-def)
with False \( \text{act} \)
have \( \text{chosen} s' \neq \text{NotAnInput} \)
by (auto simp add: HNextPart-def)
from \( H\text{EndPhase2-Inv6-1}[O F \text{act inv inv2b inv2c inv3 inv5 this}] \)
have \( p32: \text{valueChosen} s' (\text{chosen} s') \).
from False \( \text{InputsOrNi} \)
have \( \text{chosen} s \in \text{Inputs} \) by auto
from \( \text{valueChosen-equal}[O F \text{HEndPhase2-valueChosen}[O F \text{act p31 this}] p32] \)
have \( p33: \text{chosen} s = \text{chosen} s' \).
from act
have \( \text{maj}: \text{IsMajority} \{d. \ d \in \text{disksWritten} s p \}
\land (\forall q \in \text{UNIV} - \{p\}. \text{hasRead} s p d q)\} \) (is \( \text{IsMajority} \ ?D \)
and \( \text{phase}: \text{phase} s p = 2 \)
by (auto simp add: HNextPart-def)
show \( ?\text{thesis} \)
proof (cases \( \text{outpt} s r = \text{NotAnInput} \))
case True
with \( \text{asm} \) \( \text{act} \)
have \( p41: r = p \)
by (auto simp add: HNextPart-def split: split-if-asm)
from \( \text{maj} \)

140
have p42: \( \exists D \in \text{MajoritySet}. \forall d \in D. \forall q \in \text{UNIV} - \{ p \}. \text{hasRead s p d q} \)
by (auto simp add: MajoritySet-def)

have p43: (\( \neg (\exists D \in \text{MajoritySet}. \exists q. (\forall d \in D. \text{bal(dblock s p) < mbal(disk s d q)}) \))

proof auto
fix D q
assume Dmaj: D \in \text{MajoritySet}
show \( \exists d \in D. \text{bal(dblock s p) < mbal(disk s d q)} \text{ hasRead s p d q} \)
proof (cases p=q)
assume pq: p=q
thus ?thesis
proof auto
from maj majorities-intersect Dmaj
have ?D\cap\{\}\{\}
by (auto simp add: MajoritySet-def)
hence \( \exists d \in ?D\cap D. d \in \text{disksWritten s p by auto} \)
then obtain d where d: d \in \text{disksWritten s p and d} \in ?D\cap D
by auto
hence d\ D: d \in D by auto
from d inv2b
have disk s d p = dblock s p
by (auto simp add: Inv2b-def Inv2b-inner-def)
with inv2c phase
have bal(dblock s p) = mbal(disk s d p)
by (auto simp add: Inv2c-def Inv2c-inner-def)
with d\ D pq
show \( \exists d \in D. \text{bal(dblock s q) < mbal(disk s d q)} \text{ hasRead s q d q} \)
by auto
qed
next
case False
with p42
have \( \exists D \in \text{MajoritySet}. \forall d \in D. \text{hasRead s p d q} \)
by auto
with majorities-intersect Dmaj
show ?thesis
by (auto simp add: MajoritySet-def, blast)
qed

qed

with inv5 act
have p44: \text{maxBalInp s (bal(dblock s p)) (inp(dblock s p))}
by (auto simp add: EndPhase2-def HInv5-inner-def HInv5-inner-R-def)

have \( \exists b \in \text{allBlocks s}. \exists b \in (\text{UN p. Ballot p}). (\text{maxBalInp s b (chosen s)}) \land b \leq \text{bal bk} \)

proof –
have disk-allblks: \( \forall d p. \text{disk s d p} \in \text{allBlocks s} \)
by (auto simp add: allBlocks-def blocksOf-def)
from p31
have \( \exists b \in (UN p. \text{Ballot } p), \text{maxBalInp } s \ b \ (\text{chosen } s) \land \\
(\exists p. \exists D \in \text{MajoritySet}.(\forall d \in D. \ b \leq \text{bal}(disk s d p))) \)
  by (auto simp add: valueChosen-def, force)
with majority-nonempty obtain b p D d
  where IsMajority D \land b \in (UN p. \text{Ballot } p) \land \\
  \text{maxBalInp } s \ b \ (\text{chosen } s) \land d \in D \land b \leq \text{bal}(disk s d p)
  by (auto simp add: MajoritySet-def, blast)
with disk-allblks
  show ?thesis
  by (auto)
qed
then obtain bk b
  where p45-bk: bk \in allBlocks s \land b \leq bal bk
  and p45-b: b \in (UN p. \text{Ballot } p) \land (\text{maxBalInp } s \ b \ (\text{chosen } s))
  by auto
have p46: \text{inp}(dblock s p) = \text{chosen } s
proof (cases b \leq \text{bal}(dblock s p))
  case True
  have dblock s p \in allBlocks s
    by (auto simp add: allBlocks-def blocksOf-def)
  with p45-b True
    show ?thesis
    by (auto simp add: maxBalInp-def)
next
  case False
  from p44 p45-bk False
  have \text{inp} bk = \text{inp}(dblock s p)
    by (auto simp add: maxBalInp-def)
  with p45-b p45-bk
    show ?thesis
    by (auto simp add: maxBalInp-def)
qed
with p41 p33 act
  show ?thesis
  by (auto simp add: EndPhase2-def)
next
  case False
  from inv2c
  have Inv2c-inner s r
    by (auto simp add: Inv2c-def)
  with False asm inv2c act
  have \text{outpt } s' = \text{outpt } s r
    by (auto simp add: Inv2c-inner-def EndPhase2-def
      split: split-if-asm)
  with inv p33 False
    show ?thesis
    by (auto simp 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:
[ outpt s = outpt s′; ∀ p. outpt s p ∈ {chosen s, NotAnInput};
  Inv2c s; HNextPart s s′ ] → ∀ p. outpt s′ p ∈ {chosen s′, NotAnInput}
using assms and 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: HStartBallot-def HInv6-def) 
from HStartBallot-valueChosen[OF act] this InputsOrNi 
have \( t1 \colon \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
  by auto 
from act 
have \( \text{outpt} \colon \text{outpt } s = \text{outpt } s' \)
  by (auto simp add: StartBallot-def) 
from outpt-Inv6[OF outpt] act inv2c inv 
have \( \forall p. \text{outpt } s' p = \text{chosen } s' \vee \text{outpt } s' p = \text{NotAnInput} \)
  by (auto simp add: HInv6-def) 
with \( t1 \)
show \( ?\text{thesis} \)
  by (simp add: HInv6-def)
qed

theorem HPhase1or2Write-Inv6:
  assumes \( \text{act} \colon \text{HPhase1or2Write } s s' p d \)
  and \( \text{inv} \colon \text{HInv6 } s \)
  and \( \text{inv4} \colon \text{HInv4a } s p \)
  and \( \text{inv2c} \colon \text{Inv2c } s \)
  shows \( \text{HInv6 } s' \)
proof 
  from outpt-chosen act inv2c inv 
  have \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
    by (auto simp add: Phase1or2Write-def HInv6-def) 
  from HPhase1or2Write-valueChosen[OF act] inv4 this InputsOrNi 
  have \( t1 \colon \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
    by auto 
  from act 
  have \( \text{outpt} \colon \text{outpt } s = \text{outpt } s' \)
    by (auto simp add: Phase1or2Write-def) 
  from outpt-Inv6[OF outpt] act inv2c inv 
  have \( \forall p. \text{outpt } s' p = \text{chosen } s' \vee \text{outpt } s' p = \text{NotAnInput} \)
    by (auto simp add: HInv6-def) 
  with \( t1 \)
  show \( ?\text{thesis} \)
    by (simp add: HInv6-def)
qed

theorem HPhase1or2ReadThen-Inv6:
  assumes \( \text{act} \colon \text{HPhase1or2ReadThen } s s' p d q \)
  and \( \text{inv} \colon \text{HInv6 } s \)
  and \( \text{inv2c} \colon \text{Inv2c } s \)
  shows \( \text{HInv6 } s' \)
proof 
  from outpt-chosen act inv2c inv 
  have \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' (\text{chosen } s') \)
    by (auto simp add: Phase1or2ReadThen-def HInv6-def) 
  from HPhase1or2ReadThen-valueChosen[OF act] this InputsOrNi
have \( t_1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \) (chosen \( s' \))

by auto

from act

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

by (auto simp add: Phase1or2ReadThen-def)

from outpt-Inv6[OF outpt] act inv2c inv

have \( \forall p. \text{outpt } s' p = \text{chosen } s' \vee \text{outpt } s' p = \text{NotAnInput} \)

by (auto simp add: HInv6-def)

with \( t_1 \)

show \(?thesis\)

by (simp add: HInv6-def)

qed

theorem HPhase1or2ReadElse-Inv6:

assumes \( \text{act: HPhase1or2ReadElse } s \\ s' p d q \)

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

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

shows \( \text{HInv6 } s' \)

using assms and HStartBallot-Inv6

by (auto simp add: Phase1or2ReadElse-def)

theorem HEndPhase1-Inv6:

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

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

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

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

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

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

shows \( \text{HInv6 } s' \)

proof 

from outpt-chosen act inv2c inv

have \( \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s \) (chosen \( s' \))

by (auto simp add: EndPhase1-def HInv6-def)

from HEndPhase1-valueChosen[OF act] inv1 inv2a inv2b this InputsOrNi

have \( t_1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen } s' \) (chosen \( s' \))

by auto

from act

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

by (auto simp add: EndPhase1-def)

from outpt-Inv6[OF outpt] act inv2c inv

have \( \forall p. \text{outpt } s' p = \text{chosen } s' \vee \text{outpt } s' p = \text{NotAnInput} \)

by (auto simp add: HInv6-def)

with \( t_1 \)

show \(?thesis\)

by (simp add: HInv6-def)

qed

lemma outpt-chosen-2:

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

\textbf{qed}

\textbf{theorem} HEndPhase0-Inv6:
\begin{itemize}
  \item assumes act: HEndPhase0 \(s \ s' \ p\)
  \item and inv: HInv6 \(s\)
  \item and inv1: Inv1 \(s\)
  \item and inv2c: Inv2c \(s\)
  \item shows HInv6 \(s'\)
\end{itemize}
\textbf{proof} –
\begin{itemize}
  \item from outpt-chosen act inv2c inv
  \item have chosen \(s' \neq \text{NotAnInput} \rightarrow \text{valueChosen} \ s \ (\text{chosen } s')\)
    \begin{itemize}
      \item by (auto simp add: HInv6-def)
    \end{itemize}
  \item from HEndPhase0-valueChosen[OF act] inv1 this InputsOrNi
  \item have \(t1: \text{chosen } s' \neq \text{NotAnInput} \rightarrow \text{valueChosen} \ s' \ (\text{chosen } s')\)
    \begin{itemize}
      \item by auto
    \end{itemize}
\end{itemize}
\textbf{qed}

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

definition HInv :: state ⇒ bool
where
HInv s = (HInv1 s ∧ HInv2 s ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s ∧ HInv6 s)

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

theorem I2:
assumes inv: HInv s
and nxt: HNext s s'
shows HInv s'
using inv I2a[OF nxt] I2b[OF nxt] I2c[OF nxt] I2d[OF nxt]
I2e[OF nxt] I2f[OF nxt]
by (simp add: HInv-def)
theory DiskPaxos imports DiskPaxos-Invariant begin

C.9 Inner Module

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

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

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

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

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

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

theorem R1:

end
\begin{align*}
\text{theorem } R2b: \\
\text{assumes } \text{inv}: \text{HInv } s \\
\text{and } \text{inv’}: \text{HInv } s' \\
\text{and } \text{nxt}: \text{HNext } s s' \\
\text{and } \text{srel}: is = s2is s \land is' = s2is s' \\
\text{shows } (\exists \ p. \text{IFail } is is' p \lor \text{IChoose } is is' p) \lor is = is'
\end{align*}

\text{proof (auto)}
\begin{align*}
\text{assume } \text{chg-vars}: is \neq is' \\
\text{with } \text{srel} \\
\text{have } \text{s-change}: & \ \text{inpt } s \neq \text{inpt } s' \lor \text{outpt } s \neq \text{outpt } s' \\
& \lor \text{chosen } s \neq \text{chosen } s' \lor \text{allInput } s \neq \text{allInput } s' \\
\text{by (auto simp add: s2is-def)} \\
\text{from } \text{inv} \\
\text{have } \text{inv2c5}: \forall \ p. \ \text{inpt } s p \in \text{allInput } s \\
\land (\text{chosen } s = \text{NotAnInput} \implies \text{outpt } s p = \text{NotAnInput}) \\
\text{by (auto simp add: HInv-def HInv2-def Inv2c-def Inv2c-inner-def)} \\
\text{from } \text{nxt } \text{s-change } \text{inv2c5} \\
\text{have } \text{inpt } s' \neq \text{inpt } s \lor \text{outpt } s' \neq \text{outpt } s \\
\text{by (auto simp add: HNext-def Next-def HNextPart-def)} \\
\text{with } \text{nxt} \\
\text{have } \exists \ p. \text{Fail } s s' p \lor \text{EndPhase2 } s s' p \\
\text{by (auto simp add: HNext-def Next-def StartBallot-def Phase0Read-def Phase1or2Write-def} \\
\text{Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def} \\
\text{EndPhase1or2-def EndPhase1-def EndPhase0-def)} \\
\text{then obtain } p \text{ where fail-or-endphase2: Fail } s s' p \lor \text{EndPhase2 } s s' p \\
\text{by auto} \\
\text{from } \text{inv} \\
\text{have } \text{inv2c}: \text{Inv2c-inner } s p \\
\text{by (auto simp add: HInv-def HInv2-def Inv2c-def)} \\
\text{from } \text{fail-or-endphase2 have } \text{IFail } is is' p \lor \text{IChoose } is is' p \\
\text{proof} \\
\text{assume } \text{fail}: \text{Fail } s s' p \\
\text{hence } \text{phase'}: \text{phase } s' p = 0 \\
\text{and } \text{outpt}: \text{outpt } s' = (\text{outpt } s) (p := \text{NotAnInput}) \\
\text{by (auto simp add: Fail-def)} \\
\text{have } \text{IFail } is is' p \\
\text{proof} \\
\text{from } \text{fail srel} \\
\text{have ioutput } is' = (\text{ioutput } is) (p := \text{NotAnInput}) \\
\text{by (auto simp add: Fail-def s2is-def)} \\
\text{moreover} \\
\text{from } \text{nxt} \\
\text{have all-nxt: allInput } s' = \text{allInput } s \cup (\text{range } (\text{inpt } s')) \\
\text{by (auto simp add: HNext-def HNextPart-def)} \\
\text{from } \text{fail srel}
\end{align*}
have \( \exists ip \in \text{Inputs}. \ (i\text{input is}')(i\text{input is})(p:= ip) \)
by(auto simp add: Fail-def s2is-def)
then obtain ip where ip-Input: ip\in\text{Inputs} and i\text{input is}' = (i\text{input is})(p:= ip)
by(auto)
moreover from outpt srel nxt inv2c
have ichosen is' = ichosen is
by(auto simp add: HNext-def HNextPart-def s2is-def Inv2c-inner-def)
ultimately show \?thesis using ip-Input
by(auto simp add: IFail-def)
qed
thus \?thesis by(auto)
next assume endphase2: EndPhase2 s s' p
from endphase2
have phase s p =2
by(auto simp add: EndPhase2-def)
with inv2c: Ballot-nzero
have bal-dblk-nzero: bal(dblock s p)\neq 0
by(auto simp add: Inv2c-inner-def)
moreover from inv
have inv2a-dblock: Inv2a-innermost s p (dblock s p)
by(auto simp add: Hinv-def Hinv2-def Inv2a-def Inv2a-inner-def blocksOf-def)
ultimately have p22: inp (dblock s p) \subseteq allInput s
by(auto simp add: Inv2a-innermost-def)
from inv
have allInput s \subseteq Inputs
by(auto simp add: Hinv-def Hinv1-def)
with p22 NotAnInput endphase2
have outpt-uni: outpt s' p \neq NotAnInput
by(auto simp add: EndPhase2-def)
show \?thesis proof(cases chosen s = NotAnInput)
case True
with inv2c5
have p31: \( \forall q. \ outpt s q = \text{NotAnInput} \)
by(auto)
with endphase2
have p32: \( \forall q \in UNIV - \{p\}. \ outpt s' q = \text{NotAnInput} \)
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 inp t s' = inp t ∧ 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
    ∧ inp t s' = inp t ∧ 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