Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff   Stephan Merz

April 17, 2016

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

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 nth command. Hence, each processor p starts with an input value (a command), and it may output a value (the command to be executed). The problem is solved if all processors eventually output the same value and this value was a value of $input[p]$ for some p (under certain assumptions, in our case that there exists at least one non-faulty processor that can write and read a majority of disks).

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

2.1   Informal description of the algorithm.

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

Processors

Processor 1	dblock
• inpt, outpt,	mbal
• phase	bal
• disksWritten	inp
• blocksRead	

Processor 2	dblock
• inpt, outpt,	mbal
• phase	bal
• disksWritten	inp
• blocksRead	

. . . .

Processor N	dblock
• inpt, outpt,	mbal
• phase	bal
• disksWritten	inp
• blocksRead	

Disks

disk 1			
p1	p2	-------	pn
mbal	mbal		mbal
bal	bal		bal
inp	inp		inp

disk 2			
p1	p2	-------	pn
mbal	mbal		mbal
bal	bal		bal
inp	inp		inp

. . . .

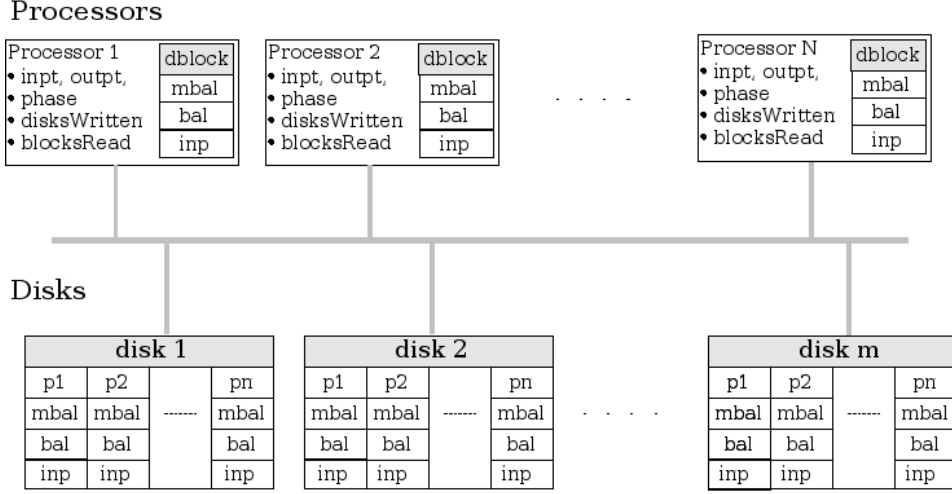| disk m			
p1	p2	-------	pn
mbal	mbal		mbal
bal	bal		bal
inp	inp		inp

Figure 1: A network of processors and disks.

of the) *Inner* module is implied by the (translation to Isabelle/HOL of the) algorithm module *HDiskSynod*.

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

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

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

Analogously, we have the specification of the *Inner* module:

$$ISpec \triangleq IInit \wedge \Box[INext]_{\langle input,output,chosen,allInput \rangle}$$

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

THEOREM $R1$ $HInit \Rightarrow IInit$
THEOREM $R2$ $HInit \wedge \Box[HNext]_{\langle vars,chosen,allInput \rangle} \Rightarrow \Box[INext]_{ivars}$

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

THEOREM $R2a$ $HInit \wedge \Box[HNext]_{\langle vars,chosen,allInput \rangle} \Rightarrow \Box HInv$
THEOREM $R2b$ $HInv \wedge HInv' \wedge HNext \Rightarrow INext \vee (\text{UNCHANGED } ivars)$

A predicate satisfying *HInv* is said to be an invariant of *HDiskSynodSpec*. To prove $R2a$, we make *HInv* strong enough to satisfy:

5

TLA$^+$	Isabelle/HOL
$\exists\, d \in D \;:\; disk[d][q].bal = bk$	$\exists\, d \in D.\; bal(disk\; s\; d\; q)\;=\;bk$
CHOOSE $x.P\,x$	$\varepsilon\; x.\; P\; x$
$phase' = [phase \text{ EXCEPT } ![p] = 1]$	$phase\; s' = (phase\; s)(p := 1)$
UNION $\{blocksOf(p)\;:\;p\in Proc\}$	$UN\; p.\; blocksOf\; s\; p$
UNCHANGED v	$v\; s' = v\; s$

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

> THEOREM $I1$ $HInit \Rightarrow HInv$
>
> THEOREM $I2$ $HInv \wedge HNext \Rightarrow HInv'$

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

Finding a predicate $HInv$ that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present $HInv$ as a conjunction of 6 predicates $HInv1, \ldots, HInv6$, where $HInv1$ is a simple "type invariant" and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of $HInv\,i$ by the algorithm's next-state relation relies on all $HInv\,j$ (for $j \leq i$) being true in the state before the transition. In our proofs we are going to use exactly the same tactic.

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

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

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

3.1   Typed vs. Untyped

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

---

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

TLA$^+$:

CONSTANT *Inputs*

$NotAnInput \triangleq$ CHOOSE $c : c \notin Inputs$

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

Isabelle/HOL:

typedecl *InputsOrNi*

consts
  $Inputs :: InputsOrNi \text{ set}$
  $NotAnInput :: InputsOrNi$

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

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

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

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

For example, see figure 2. The type *InputsOrNi* models the members of the set *Inputs*, and the element *NotAnInput*. We record the fact that *NotAnInput* is not in *Inputs*, with axiom *NotAnInput*. Now, looking at the type of the *inp* field of the *DiskBlock* record in the TLA$^+$ specification, we see that its type should be *InputsOrNi*. However, this is not the same type as $Inputs \cup \{NotAnInput\}$, 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$^+$:

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

Isabelle/HOL:

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

Figure 3: Translation of an action

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

3.2 Primed Variables

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

3.3 Restructuring the specification

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

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

- Formulate lemmas that use these definitions.

- Selectively unfold these definitions in proofs, instead of adding $Let\text{-}def$ 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:

$$Phase1or2Read \quad \triangleq \quad Phase1or2ReadThen \lor Phase1or2ReadElse$$

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:

$$HInv2 \quad \triangleq \quad Inv2a \land Inv2b \land Inv2c$$

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:

$Inv2a\ s \equiv \forall\, p.\ \forall\, bk\, \in\, blocksOf\ s\ p.\ \ldots$

we write:

$Inv2a\text{-}innermost\ ::\ state \Rightarrow Proc \Rightarrow DiskBlock \Rightarrow bool$
$Inv2a\text{-}innermost\ s\ p\ bk \equiv \ldots$

$Inv2a\text{-}inner\ ::\ state \Rightarrow Proc \Rightarrow bool$
$Inv2a\text{-}inner\ s\ p \equiv \forall\, bk\, \in\, blocksOf\ s\ p.\ Inv2a\text{-}innermost\ \ s\ p\ bk$

$Inv2a\ ::\ state \Rightarrow bool$
$Inv2a\ s \equiv \forall\, p.\ Inv2a\text{-}inner\ s\ p$

Now we can express that we want to obtain the fact

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

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

4 Structure of the Correctness Proof

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

4.1 Going from Informal Proofs to Formal Proofs

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

As an example of these omissions, the invariant should state that the set $allRdBlks$ is finite. This is needed to choose a block with a maximum ballot number in action $EndPhase1$. Interestingly, this omission cannot be detected with finite-state model checking. As another example, it was omitted that it is necessary to assume that $HInv4$ and $HInv5$ hold in the previous state to prove lemma $I2f$.

Although our proofs were based on the informal ones, the high-level structure was often different. When proving that a predicate I was an invariant of $Next$, we preferred proving the invariance of I for each action, rather than a big theorem proving the invariance of I for the $Next$ action. As a consequence, a proof for some actions was often similar to the proof of some other action. For example, the proof of the invariance of $HInv3$ for the $EndPhase0$ and $Fail$ actions is almost the same. This means we could have made only one proof for the two actions, shortening the length of the complete proof. Nevertheless, proving each action separately could be tackled more easily by Isabelle, as it had fewer facts to deal with, and proving a new lemma was often a simple matter of copy, paste and renaming of definitions. Once the invariance of a given predicate has been proved for each simple action, proving it for the $Next$ action is easy since the $Next$ action is a disjunction of all actions.

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

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

5 Conclusion

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

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

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

References

[GL00]   Eli Gafni and Leslie Lamport. Disk Paxos. In *International Symposium on Distributed Computing*, pages 330–344, 2000.

[KMM00] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[Lam95]   Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, /1995.

[Lam98]   Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[Lam02]   Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Nip03]   Tobias Nipkow. Structured Proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, volume 2646, pages 259–278, 2003.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Pac01]   Carlos Pacheco. Reasoning about TLA actions, 2001.

[Sch90]   Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[Wen02]   Markus M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents.* PhD thesis, Institut fur Informatik, TU München, 2002.

A   TLA$^+$ correctness specification

─────────────── MODULE *Synod* ───────────────

EXTENDS *Naturals*
CONSTANT *N, Inputs*
ASSUME $(N \in Nat) \wedge (N > 0)$
$Proc \triangleq 1..N$
$NotAnInput \triangleq$ CHOOSE $c : c \notin Inputs$
VARIABLES *inputs, output*

───────────── MODULE *Inner* ─────────────

VARIABLES *allInput, chosen*

─────────────────────────────────────

$IInit \triangleq \wedge input \in [Proc \rightarrow Inputs]$
$\qquad\qquad \wedge output = [p \in Proc \mapsto NotAnInput]$
$\qquad\qquad \wedge chosen = NotAnInput$
$\qquad\qquad \wedge allInput = input[p] : p \in Proc$

$IChoose(p) \triangleq$
$\quad \wedge output[p] = NotAnInput$
$\quad \wedge$ IF $chosen = NotAnInput$
$\qquad$ THEN $ip \in allInput : \wedge chosen' = ip$
$\qquad\qquad\qquad\qquad\qquad\quad \wedge output' = [output$ EXCEPT $![p] = ip]$
$\qquad$ ELSE $\wedge output' = [output$ EXCEPT $![p] = chosen]$
$\qquad\qquad\quad \wedge$ UNCHANGED *chosen*
$\quad \wedge$ UNCHANGED $\langle input, allInput \rangle$

$IFail(p) \triangleq \wedge output' = [output$ EXCEPT $![p] = NotAnInput]$
$\qquad\qquad\quad \wedge \exists ip \in Inputs : \wedge input' = [input$ EXCEPT $![p] = ip]$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge allInput' = allInput \cup \{ip\}$

$INext \triangleq \exists p \in Proc : IChoose(p) \vee IFail(p)$
$ISpec \triangleq IInit \wedge \Box[INext]_{\langle input,output,chosen,allInput \rangle}$

─────────────────────────────────────

$IS(chosen, allInput) \triangleq$ INSTANCE *Inner*
$SynodSpec \triangleq \exists chosen, allInput : IS(chosen, allInput)!ISpec$

─────────────────────────────────────

B Disk Paxos Algorithm Specification

theory *DiskPaxos-Model* **imports** *Main* **begin**

This is the specification of the Disk Synod algorithm.

typedecl *InputsOrNi*
typedecl *Disk*
typedecl *Proc*

axiomatization
  Inputs :: *InputsOrNi set* **and**
  NotAnInput :: *InputsOrNi* **and**
  Ballot :: *Proc* $\Rightarrow$ *nat set* **and**
  IsMajority :: *Disk set* $\Rightarrow$ *bool*
where
  NotAnInput: *NotAnInput* $\notin$ *Inputs* **and**
  InputsOrNi: (*UNIV* :: *InputsOrNi set*) = *Inputs* $\cup$ {*NotAnInput*} **and**
  Ballot-nzero: $\forall\, p.\ 0 \notin Ballot\ p$ **and**
  Ballot-disj: $\forall\, p\ q.\ p \neq q \longrightarrow (Ballot\ p) \cap (Ballot\ q) = \{\}$ **and**
  Disk-isMajority: *IsMajority*(*UNIV*) **and**
  majorities-intersect:
    $\forall\, S\ T.\ IsMajority(S) \wedge IsMajority(T) \longrightarrow S \cap T \neq \{\}$

lemma *ballots-not-zero* [*simp*]:
  $b \in Ballot\ p \Longrightarrow 0 < b$
proof (*rule ccontr*)
  assume *b*: $b \in Ballot\ p$
  and *contr*: $\neg\ (0 < b)$
  from *Ballot-nzero*
  have $0 \notin Ballot\ p$ **..**
  with *b contr*
  show *False*
    by *auto*
qed

lemma *majority-nonempty* [*simp*]: *IsMajority*(S) $\Longrightarrow$ S $\neq$ {}
proof(*auto*)
  from *majorities-intersect*
  have *IsMajority*({}) $\wedge$ *IsMajority*({}) $\longrightarrow$ {} $\cap$ {} $\neq$ {}
    by *auto*
  thus *IsMajority* {} $\Longrightarrow$ *False*
    by *auto*
qed

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

record
  DiskBlock =

mbal:: *nat*
        bal :: *nat*
        inp :: *InputsOrNi*

definition *InitDB* :: *DiskBlock*
    where *InitDB* = (| *mbal* = 0, *bal* = 0, *inp* = *NotAnInput* |)

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

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

        allInput :: *InputsOrNi set*
        chosen :: *InputsOrNi*

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

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

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

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

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

14

$(disksWritten\ s' = (disksWritten\ s)(p := \{\})$
$\&\ blocksRead\ s' = (blocksRead\ s)(p := (\lambda d.\ \{\})))$

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

definition *Phase1or2Write* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *bool*
where
  Phase1or2Write s s' p d =
   (phase s p $\in$ *{1, 2}*
  $\wedge\ disk\ s' = (disk\ s)\ (d := (disk\ s\ d)\ (p := dblock\ s\ p))$
  $\wedge\ disksWritten\ s' = (disksWritten\ s)\ (p := (disksWritten\ s\ p) \cup \{d\})$
  $\wedge\ inpt\ s' = inpt\ s\ \wedge\ outpt\ s' = outpt\ s$
  $\wedge\ phase\ s' = phase\ s\ \wedge\ dblock\ s' = dblock\ s$
  $\wedge\ blocksRead\ s' = blocksRead\ s)$

definition *Phase1or2ReadThen* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  Phase1or2ReadThen s s' p d q =
   (d $\in$ *disksWritten s p*
  $\&\ mbal(disk\ s\ d\ q) < mbal(dblock\ s\ p)$
  $\&\ blocksRead\ s' = (blocksRead\ s)(p := (blocksRead\ s\ p)(d :=$
          $(blocksRead\ s\ p\ d) \cup \{(\!|\ block = disk\ s\ d\ q,$
                     $proc = q\ |\!)\}))$
  $\&\ inpt\ s' = inpt\ s\ \&\ outpt\ s' = outpt\ s$
  $\&\ disk\ s' = disk\ s\ \&\ phase\ s' = phase\ s$
  $\&\ dblock\ s' = dblock\ s\ \&\ disksWritten\ s' = disksWritten\ s)$

definition *Phase1or2ReadElse* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  Phase1or2ReadElse s s' p d q =
   (d $\in$ *disksWritten s p*
  $\wedge\ StartBallot\ s\ s'\ p)$

definition *Phase1or2Read* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  Phase1or2Read s s' p d q =
   (Phase1or2ReadThen s s' p d q
   $\vee\ Phase1or2ReadElse\ s\ s'\ p\ d\ q)$

definition *blocksSeen* :: *state* $\Rightarrow$ *Proc* $\Rightarrow$ *DiskBlock set*

where *blocksSeen s p = allBlocksRead s p ∪ {dblock s p}*

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

definition *maxBlk :: state ⇒ Proc ⇒ DiskBlock*
where
  maxBlk s p =
    (SOME b. b ∈ nonInitBlks s p ∧ (∀ c ∈ nonInitBlks s p. bal c ≤ bal b))

definition *EndPhase1 :: state ⇒ state ⇒ Proc ⇒ bool*
where
  EndPhase1 s s′ p =
    (IsMajority {d . d ∈ disksWritten s p
                    ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)}
    ∧ phase s p = 1
    ∧ dblock s′ = (dblock s) (p := dblock s p
          (bal := mbal(dblock s p),
            inp :=
            (if nonInitBlks s p = {}
              then inpt s p
              else inp (maxBlk s p))
))
    ∧ outpt s′ = outpt s
    ∧ phase s′ = (phase s) (p := phase s p + 1)
    ∧ InitializePhase s s′ p
    ∧ inpt s′ = inpt s ∧ disk s′ = disk s)

definition *EndPhase2 :: state ⇒ state ⇒ Proc ⇒ bool*
where
  EndPhase2 s s′ p =
    (IsMajority {d . d ∈ disksWritten s p
                    ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)}
    ∧ phase s p = 2
    ∧ outpt s′ = (outpt s) (p:= inp (dblock s p))
    ∧ dblock s′ = dblock s
    ∧ phase s′ = (phase s) (p := phase s p + 1)
    ∧ InitializePhase s s′ p
    ∧ inpt s′ = inpt s ∧ disk s′ = disk s)

definition *EndPhase1or2 :: state ⇒ state ⇒ Proc ⇒ bool*
  where *EndPhase1or2 s s′ p = (EndPhase1 s s′ p ∨ EndPhase2 s s′ p)*

definition *Fail :: state ⇒ state ⇒ Proc ⇒ bool*
where
  Fail s s′ p =
    (∃ ip ∈ Inputs. inpt s′ = (inpt s) (p := ip)
    ∧ phase s′ = (phase s) (p := 0)
    ∧ dblock s′= (dblock s) (p := InitDB)

$\land$ *outpt s′* = (*outpt s*) (*p := NotAnInput*)
$\land$ *InitializePhase s s′ p*
$\land$ *disk s′= disk s*)

definition *Phase0Read* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *bool*
where
  Phase0Read s s′ p d =
   (*phase s p* = *0*
  $\land$ *blocksRead s′* = (*blocksRead s*) (*p* := (*blocksRead s p*) (*d* := *blocksRead s p d*
$\cup$ {(| *block* = *disk s d p*, *proc* = *p* |)}))
   $\land$ *inpt s′* = *inpt s* & *outpt s′* = *outpt s*
   $\land$ *disk s′* = *disk s* & *phase s′* = *phase s*
   $\land$ *dblock s′* = *dblock s* & *disksWritten s′* = *disksWritten s*)

definition *EndPhase0* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  EndPhase0 s s′ p =
   (*phase s p* = *0*
  $\land$ *IsMajority* ({*d. hasRead s p d p*})
  $\land$ ($\exists$ *b* $\in$ *Ballot p*.
    ($\forall$ *r* $\in$ *allBlocksRead s p. mbal r* < *b*)
   $\land$ *dblock s′* = (*dblock s*) (*p*:=
         (*SOME r*.   *r* $\in$ *allBlocksRead s p*
            $\land$ ($\forall$ *s* $\in$ *allBlocksRead s p. bal s* $\leq$ *bal r*)) (| *mbal* := *b* |)))
  $\land$ *InitializePhase s s′ p*
  $\land$ *phase s′* = (*phase s*) (*p*:= *1*)
  $\land$ *inpt s′* = *inpt s* $\land$ *outpt s′* = *outpt s* $\land$ *disk s′* = *disk s*)

definition *Next* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *bool*
where
  Next s s′ = ($\exists$ *p*.
        StartBallot s s′ p
      $\lor$ ($\exists$ *d*.   *Phase0Read s s′ p d*
         $\lor$ *Phase1or2Write s s′ p d*
         $\lor$ ($\exists$ *q. q* $\neq$ *p* $\land$ *Phase1or2Read s s′ p d q*))
      $\lor$ *EndPhase1or2 s s′ p*
      $\lor$ *Fail s s′ p*
      $\lor$ *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
  HInit s =
   (*Init s*
  & *chosen s* = *NotAnInput*
  & *allInput s* = *range* (*inpt s*))

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

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

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

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

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

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

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

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

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

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

definition
  HFail :: state ⇒ state ⇒ Proc ⇒ bool **where**
  HFail s s′ p = (Fail s s′ p ∧ HNextPart s s′)

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

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

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

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

end

C   Proof of Disk Paxos' Invariant

theory *DiskPaxos-Inv1* **imports** *DiskPaxos-Model* **begin**

C.1   Invariant 1

This is just a type Invariant.

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

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

declare *HInv1-def* [*simp*]

We added the assertion that the set *allRdBlksp* is finite for every process *p*; one may therefore choose a block with a maximum ballot number in action *EndPhase*1.

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

lemma *HNextPart-Inv1*: ⟦ *HInv1 s*; *HNextPart s s'*; *Inv1 s'* ⟧ ⟹ *HInv1 s'*
  by(*auto simp add*: *HNextPart-def Inv1-def*)

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

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

theorem *HPhase1or2ReadThen-HInv1*:
  assumes *inv1*: *HInv1 s*
  and *act*: *HPhase1or2ReadThen s s' p d q*
  shows *HInv1 s'*
proof −
  — we focus on the last conjunct of Inv1
  from *act*
  have ∀ *p*. *allRdBlks s' p* ⊆ *allRdBlks s p* ∪ {(⎜*block = disk s d q, proc = q*⎟)}
    by(*auto simp add*: *Phase1or2ReadThen-def allRdBlks-def*
           split: *if-split-asm*)
  with *inv1*
  have ∀ *p*. *finite* (*allRdBlks s' p*)
    by(*blast dest*: *allRdBlks-finite*)
  — the others conjuncts are trivial
  with *inv1 act*
  show *?thesis*
    by(*auto simp add*: *Inv1-def Phase1or2ReadThen-def HNextPart-def*)
qed

theorem *HEndPhase1-HInv1*:
  assumes *inv1*: *HInv1 s*
  and *act*: *HEndPhase1 s s' p*
  shows *HInv1 s'*
proof −
    from *inv1 act*

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

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

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

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

theorem *HEndPhase2-HInv1*:
　assumes *inv1*: *HInv1 s*
　and *act*: *HEndPhase2 s s′ p*
　shows *HInv1 s′*
proof −
　from *inv1 act*
　have *Inv1 s′*
　　by(*auto simp add*: *Inv1-def EndPhase2-def InitializePhase-def allRdBlks-def*)
　with *inv1 act*

show *?thesis*
  by(*auto simp del*: *HInv1-def elim*: *HNextPart-Inv1*)
qed

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

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

theorem *HEndPhase0-HInv1*:
  assumes *inv1*: *HInv1 s*
  and *act*: *HEndPhase0 s s′ p*
  shows *HInv1 s′*
proof −
  from *inv1 act*
  have *Inv1 s′*
    by(*auto simp add*: *Inv1-def 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*]

*HInv*1 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 *Inv*2*a*, *Inv*2*b*, and *Inv*2*c*. The main difficulty is in proving the preservation of the first conjunct.

definition *rdBy* :: *state* $\Rightarrow$ *Proc* $\Rightarrow$ *Proc* $\Rightarrow$ *Disk* $\Rightarrow$ *BlockProc set*
where
  rdBy s p q d =
    {*br . br* $\in$ *blocksRead s q d* $\wedge$ *proc br* = *p*}

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

definition *allBlocks* :: *state* $\Rightarrow$ *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. inpt s p ∈ allInput s
      ∧ (chosen s = NotAnInput ⟶ outpt s p = NotAnInput)))

definition *Inv2c :: state ⇒ bool*

where *Inv2c s = (∀ p. Inv2c-inner s p)*

definition *HInv2* :: *state ⇒ bool*
  where *HInv2 s = (Inv2a s ∧ Inv2b s ∧ Inv2c s)*

C.2.1 Proofs of Invariant 2 a

theorem *HInit-Inv2a*: *HInit s ⟶ Inv2a s*
by (*auto simp add*: *HInit-def Init-def Inv2a-def Inv2a-inner-def*
            Inv2a-innermost-def rdBy-def blocksOf-def
            InitDB-def)

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

lemma *HPhase1or2ReadThen-blocksOf*:
  ⟦ *HPhase1or2ReadThen s s′ p d q* ⟧ ⟹ *blocksOf s′ r ⊆ blocksOf s r*
  by(*auto simp add*: *Phase1or2ReadThen-def blocksOf-def rdBy-def*)

theorem *HPhase1or2ReadThen-Inv2a*:
  assumes *inv*: *Inv2a s*
  and *act*: *HPhase1or2ReadThen s s′ p d q*
  shows *Inv2a s′*
proof (*clarsimp simp add*: *Inv2a-def Inv2a-inner-def*)
  fix *pp bk*
  assume *bk*: *bk ∈ blocksOf s′ pp*
  with *inv HPhase1or2ReadThen-blocksOf*[*OF act*]
  have *Inv2a-innermost s pp bk*
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def*)
  with *act*
  show *Inv2a-innermost s′ pp bk*
    by(*auto simp add*: *Inv2a-innermost-def HNextPart-def*)
qed

lemma *InitializePhase-rdBy*:
  InitializePhase s s′ p ⟹ rdBy s′ pp qq dd ⊆ rdBy s pp qq dd
by(*auto simp add*: *InitializePhase-def rdBy-def*)

lemma *HStartBallot-blocksOf*:
  HStartBallot s s′ p ⟹ blocksOf s′ q ⊆ blocksOf s q ∪ {dblock s′ q}
by(*auto simp add*: *StartBallot-def blocksOf-def*
      dest: *subsetD*[*OF InitializePhase-rdBy*])

lemma *HStartBallot-Inv2a-dblock*:
  assumes *act*: *HStartBallot s s′ p*
  and *inv2a*: *Inv2a-innermost s p (dblock s p)*

25

shows *Inv2a-innermost s′ p (dblock s′ p)*
proof −
　from *act*
　have *mbal′: mbal (dblock s′ p) ∈ Ballot p*
　　by(*auto simp add: StartBallot-def*)
　from *act*
　have *bal′: bal (dblock s′ p) = bal (dblock s p)*
　　by(*auto simp add: StartBallot-def*)
　with *act*
　have *inp′: inp (dblock s′ p) = inp (dblock s p)*
　　by(*auto simp add: StartBallot-def*)
　from *act*
　have *mbal (dblock s p) ≤ mbal (dblock s′ p)*
　　by(*auto simp add: StartBallot-def*)
　with *bal′ inv2a*
　have *bal-mbal: bal (dblock s′ p) ≤ mbal (dblock s′ p)*
　　by(*auto simp add: Inv2a-innermost-def*)
　from *act*
　have *allInput s ⊆ allInput s′*
　　by(*auto simp add: HNextPart-def*)
　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* $\in$ *blocksOf s q* $\longrightarrow$ *Inv2a-innermost s q bk*
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def*)
from *bk HStartBallot-blocksOf*[*OF act*]
have *bk* $\in$ {*dblock s' q*} $\cup$ *blocksOf s q*
  by *blast*
thus *Inv2a-innermost s' q bk*
proof
  assume *bk-dblock*: *bk* $\in$ {*dblock s' q*}
  from *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* $\in$ *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
qed

lemma *HPhase1or2Write-blocksOf*:
  ⟦ *HPhase1or2Write s s' p d* ⟧ $\Longrightarrow$ *blocksOf s' r* $\subseteq$ *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* $\in$ *blocksOf s' q*
  from *inv bk HPhase1or2Write-blocksOf*[*OF act*]
  have *inp-q-bk*: *Inv2a-innermost s q bk*
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def*)
  with *act*
  show *Inv2a-innermost s' q bk*
    by(*auto simp add*: *Inv2a-innermost-def HNextPart-def*)
qed

theorem *HPhase1or2ReadElse-Inv2a*:
  assumes *inv*: *Inv2a s*
  and *act*:   *HPhase1or2ReadElse s s' p d q*
  shows *Inv2a s'*

proof −
  from *act*
  have *HStartBallot s s′ p*
    by(*simp add*: *Phase1or2ReadElse-def*)
  with *inv*
  show *?thesis*
    by(*auto elim*: *HStartBallot-Inv2a*)
qed

lemma *HEndPhase2-blocksOf*:
  ⟦ *HEndPhase2 s s′ p* ⟧ ⟹ *blocksOf s′ q ⊆ blocksOf s q*
  by(*auto simp add*: *EndPhase2-def blocksOf-def*
            dest: *subsetD*[*OF InitializePhase-rdBy*])

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

lemma *HFail-blocksOf*:
  HFail s s′ p ⟹ *blocksOf s′ q ⊆ blocksOf s q ∪ {dblock s′ q}*
by(*auto simp add*: *Fail-def blocksOf-def*
        dest: *subsetD*[*OF InitializePhase-rdBy*])

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

show *?thesis*
  by(*auto simp add*: *Fail-def HNextPart-def*
    InitializePhase-def Inv2a-innermost-def)
qed

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

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

theorem *HPhase0Read-Inv2a*:
  assumes *inv*: *Inv2a s*
  and   *act*: *HPhase0Read s s' p d*
  shows   *Inv2a s'*
proof(*clarsimp simp add*: *Inv2a-def Inv2a-inner-def*)
  fix *q bk*
  assume *bk*: *bk* ∈ *blocksOf s' q*
  from *inv bk HPhase0Read-blocksOf*[*OF act*]

have *inp-q-bk*: *Inv2a-innermost s q bk*
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def*)
with *act*
show *Inv2a-innermost s′ q bk*
  by(*auto simp add*: *Inv2a-innermost-def HNextPart-def*)
qed

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

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

*EndPhase*0 has the additional difficulty of having a choose expression. We prove that there exists an x such that the predicate of the choose expression holds, and then apply *someI*: *?P ?x ⟹ ?P (Eps ?P)*.

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

30

```
    by auto
  thus ?thesis
    by (rule someI)
qed


lemma HEndPhase0-dblock-allBlocksRead:
  assumes act: HEndPhase0 s s′ p
  and     inv1: Inv1 s
  shows   dblock s′ p ∈ (λx. x (|mbal:= mbal(dblock s′ p)|)) ' 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′
  and inv2a: Inv2a-innermost s p (dblock s′ p)
  shows inp (dblock s′ p) ∈ allInput s′ ∪ {NotAnInput}
 proof −
  from act
  have allInput s′ = allInput s ∪ (range (inpt s′))
    by(simp add: HNextPart-def)
  moreover
  from inv2a
  have inp (dblock s′ p) ∈ allInput s ∪ {NotAnInput}
    by(simp add: Inv2a-innermost-def)
  ultimately show ?thesis
    by blast
qed


lemma HEndPhase0-Inv2a-allBlocksRead:
  assumes act: HEndPhase0 s s′ p
  and inv2a: Inv2a-inner s p
  and inv2c: Inv2c-inner s p
  shows ∀ t ∈ (λx. x (|mbal:= mbal (dblock s′ p)|)) ' allBlocksRead s p.
        Inv2a-innermost s p t
proof −
  from act
  have mbal′: mbal (dblock s′ p) ∈ Ballot p
    by(auto simp add: EndPhase0-def)
  from inv2c act
  have allproc-p: ∀ d. ∀ br ∈ blocksRead s p d. proc br = p
    by(simp add: Inv2c-inner-def EndPhase0-def)
  with inv2a
  have allBlocks-inv2a: ∀ t ∈ allBlocksRead s p. Inv2a-innermost s p t
  proof(auto simp add: Inv2a-inner-def allBlocksRead-def
                  allRdBlks-def blocksOf-def rdBy-def)
    fix d bk
    assume bk-in-blocksRead: bk ∈ blocksRead s p d
      and inv2a-bk: ∀ x∈    {u. ∃ d. u = disk s d p}
                      ∪ {block br |br. (∃ q d. br ∈ blocksRead s q d)
```

$\land proc\ br = p\}.\ Inv2a\text{-}innermost\ s\ p\ x$

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

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

lemma *HEndPhase0-Inv2a-dblock-q*:
 assumes *act*: *HEndPhase0 s s′ p*

and *inv1*: *Inv1 s*
  and *inv2a*: *Inv2a-inner s q*
  and *inv2c*: *Inv2c-inner s p*
  shows *Inv2a-innermost s′ q (dblock s′ q)*
proof(*cases p=q*)
  case *True*
  with *act inv2a inv2c inv1*
  show *?thesis*
    by(*blast dest*: *HEndPhase0-Inv2a-dblock*)
next
  case *False*
  from *inv2a*
  have *inv-q-dblock*: *Inv2a-innermost s q (dblock s q)*
    by(*auto simp add*: *Inv2a-inner-def blocksOf-def*)
  with *False act*
  show *?thesis*
    by(*clarsimp simp add*: *EndPhase0-def HNextPart-def*
      InitializePhase-def Inv2a-innermost-def)
qed

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

qed
qed

lemma *HEndPhase1-blocksOf*:
  HEndPhase1 s s′ p $\Longrightarrow$ *blocksOf s′ q* $\subseteq$ *blocksOf s q* $\cup$ {*dblock s′ q*}
by (*auto simp add*: *EndPhase1-def blocksOf-def*
        dest: *subsetD*[*OF InitializePhase-rdBy*])

lemma *maxBlk-in-nonInitBlks*:
  assumes *b*: *b* $\in$ *nonInitBlks s p*
  and *inv1*: *Inv1 s*
  shows   *maxBlk s p* $\in$ *nonInitBlks s p*
       $\wedge$ ($\forall c \in$ *nonInitBlks s p. bal c* $\leq$ *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* $\neq$ {}
    by *auto*
  with *nibals-finite*
  have *Max ?S* $\in$ *?S* **and** $\forall bb \in$ *?S. bb* $\leq$ *Max ?S* **by** *auto*
  hence $\exists mb \in$ *?S.* $\forall bb \in$ *?S. bb* $\leq$ *mb* **..**
  then obtain *mblk*
    where   *mblk* $\in$ *nonInitBlks s p*
        $\wedge$ ($\forall c \in$ *nonInitBlks s p. bal c* $\leq$ *bal mblk*)
        (**is** *?P mblk*)
    by *auto*
  hence *?P* (*SOME b. ?P b*)
    by (*rule someI*)
  thus *?thesis*
    by (*simp add*: *maxBlk-def*)
qed

lemma *blocksOf-nonInitBlks*:
  ($\forall p\ bk.\ bk \in$ *blocksOf s p* $\longrightarrow$ *P bk*)
      $\Longrightarrow$ *bk* $\in$ *nonInitBlks s p* $\longrightarrow$ *P bk*
  by(*auto simp add*: *allRdBlks-def blocksOf-def nonInitBlks-def*
              blocksSeen-def allBlocksRead-def rdBy-def,
    blast)

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

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

lemma *HEndPhase1-Inv2a-dblock*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv1*: *HInv1 s*
  and *inv2*: *Inv2a s*
  and *inv2c*: *Inv2c-inner s p*
  shows *Inv2a-innermost s′ p* (*dblock s′ p*)
proof −
  from *inv1 act* **have** *inv1′*: *HInv1 s′*

35

by(*blast dest*: *HEndPhase1-HInv1*)
from *inv2*
have *inv2a*: *Inv2a-innermost s p* (*dblock s p*)
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def blocksOf-def*)
from *act inv2c*
have $mbal'$: *mbal* (*dblock s' p*) $\in$ *Ballot p*
  by (*auto simp add*: *EndPhase1-def Inv2c-def Inv2c-inner-def*)
moreover
from *act*
have bal': *bal* (*dblock s' p*) $=$ *mbal* (*dblock s p*)
  by (*auto simp add*: *EndPhase1-def*)
moreover
from *act inv1 inv2*
have inp': *inp* (*dblock s' p*) $\in$ *allInput s'*
  by(*blast dest*: *HEndPhase1-dblock-allInput*)
moreover
with $inv1'$ *NotAnInput*
have *inp* (*dblock s' p*) $\neq$ *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*: *Inv2a s*
and *inv2c*: *Inv2c-inner s p*
shows *Inv2a s′*
proof (*clarsimp simp add*: *Inv2a-def Inv2a-inner-def*)
  fix *q bk*
  assume *bk-in-bks*: *bk* ∈ *blocksOf s′ q*
  with *HEndPhase1-blocksOf*[*OF act*]
  have *dblock-blocks*: *bk* ∈ {*dblock s′ q*} ∪ *blocksOf s q*
    by *blast*
  thus *Inv2a-innermost s′ q bk*
  proof
    assume *bk* ∈ {*dblock s′ q*}
    with *act inv1 inv2c inv*
    show *?thesis*
      by(*blast dest*: *HEndPhase1-Inv2a-dblock-q*)
  next
    assume *bk-in-blocks*: *bk* ∈ *blocksOf s q*
    with *inv*
    have *Inv2a-innermost s q bk*
      by(*auto simp add*: *Inv2a-def Inv2a-inner-def*)
    with *act* **show** *?thesis*
      by(*auto simp add*: *EndPhase1-def HNextPart-def*
        InitializePhase-def Inv2a-innermost-def)
  qed
qed

C.2.2   Proofs of Invariant 2 b

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

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

theorem *HPhase1or2ReadThen-Inv2b*:
  ⟦ *Inv2b s*; *HPhase1or2ReadThen s s′ p d q* ⟧
  ⟹ *Inv2b s′*
by (*auto simp add*: *Phase1or2ReadThen-def Inv2b-def*
        Inv2b-inner-def hasRead-def)

theorem *HStartBallot-Inv2b*:
  ⟦ *Inv2b s*; *HStartBallot s s′ p* ⟧
  ⟹ *Inv2b s′*
  by(*auto simp add*:*StartBallot-def InitializePhase-def*
        Inv2b-def Inv2b-inner-def hasRead-def)

theorem *HPhase1or2Write-Inv2b*:
  ⟦ *Inv2b s*; *HPhase1or2Write s s′ p d* ⟧
  ⟹ *Inv2b s′*

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

theorem *HPhase1or2ReadElse-Inv2b*:
  ⟦ *Inv2b s*; *HPhase1or2ReadElse s s' p d q* ⟧
   ⟹ *Inv2b s'*
by (*auto simp add*: *Phase1or2ReadElse-def StartBallot-def hasRead-def*
                   InitializePhase-def Inv2b-def Inv2b-inner-def)

theorem *HEndPhase1-Inv2b*:
  ⟦ *Inv2b s*; *HEndPhase1 s s' p* ⟧ ⟹ *Inv2b s'*
by (*auto simp add*: *EndPhase1-def InitializePhase-def*
                   Inv2b-def Inv2b-inner-def hasRead-def)

theorem *HFail-Inv2b*:
  ⟦ *Inv2b s*; *HFail s s' p* ⟧
   ⟹ *Inv2b s'*
by (*auto simp add*: *Fail-def InitializePhase-def*
                   Inv2b-def Inv2b-inner-def hasRead-def)

theorem *HEndPhase2-Inv2b*:
  ⟦ *Inv2b s*; *HEndPhase2 s s' p* ⟧ ⟹ *Inv2b s'*
by (*auto simp add*: *EndPhase2-def InitializePhase-def*
                   Inv2b-def Inv2b-inner-def hasRead-def)

theorem *HPhase0Read-Inv2b*:
  ⟦ *Inv2b s*; *HPhase0Read s s' p d* ⟧ ⟹ *Inv2b s'*
by (*auto simp add*: *Phase0Read-def Inv2b-def*
                   Inv2b-inner-def hasRead-def)

theorem *HEndPhase0-Inv2b*:
  ⟦ *Inv2b s*; *HEndPhase0 s s' p* ⟧ ⟹ *Inv2b s'*
by (*auto simp add*: *EndPhase0-def InitializePhase-def*
                   Inv2b-def Inv2b-inner-def hasRead-def)

C.2.3   Proofs of Invariant 2 c

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

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

using *hnp outpt′ inp-dblk inv2c*
proof(*auto simp add*: *HNextPart-def Inv2c-def Inv2c-inner-def*
       split: *if-split-asm*)
qed

lemma *HNextPart-chosen*:
  assumes *hnp*: *HNextPart s s′*
  shows *chosen s′ = NotAnInput* $\longrightarrow$ ($\forall$ *p*. *outpt s′ p = NotAnInput*)
using *hnp*
proof(*auto simp add*: *HNextPart-def split*: *if-split-asm*)
  fix *p pa*
  assume *o1*: *outpt s′ p* $\neq$ *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* $]\!]$ $\Longrightarrow$ $\forall$ *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*

39

have $\forall\, p. \quad inpt\ s'\ p \in allInput\ s'$
$$\wedge\ (chosen\ s' = NotAnInput \longrightarrow outpt\ s'\ p = NotAnInput)$$
  by(*auto dest*: *HNextPart-chosen HNextPart-allInput*)
with *outpt′ chosen′ act inv*
show *?thesis*
  by(*auto simp add*: *Phase1or2ReadThen-def Inv2c-def Inv2c-inner-def*)
qed

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

from *act inv*
have *allinp*: $\forall\, p.\quad inpt\ s'\ p\ \in allInput\ s'$
$\qquad\qquad\qquad \wedge\ (chosen\ s' = NotAnInput$
$\qquad\qquad\qquad\qquad\qquad \longrightarrow outpt\ s'\ p = NotAnInput)$
  by(*auto dest*: *HNextPart-chosen HNextPart-allInput*)
with *phase' mbal' bal' outpt' chosen' act inv blks'*
show *?thesis*
by(*auto simp add*: *StartBallot-def InitializePhase-def*
$\qquad\qquad\quad$ *Inv2c-def Inv2c-inner-def*)
qed

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

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

theorem *HEndPhase1-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HEndPhase1 s s' p*
  and *inv2a*: *Inv2a s*

and *inv1*: *HInv1 s*
shows *Inv2c s′*
proof −
  from *inv*
  have *Inv2c-inner s p* **by** (*auto simp add*: *Inv2c-def*)
  with *inv2a act inv1*
  have *inv2a′*: *Inv2a s′*
    by(*blast dest*: *HEndPhase1-Inv2a*)
  from *act inv*
  have *mbal′*: *mbal(dblock s′ p)* ∈ *Ballot p*
    by(*auto simp add*: *EndPhase1-def Inv2c-def Inv2c-inner-def*)
  from *act*
  have *bal′*: *bal(dblock s′ p)* = *mbal (dblock s′ p)*
    by(*auto simp add*: *EndPhase1-def*)
  from *act inv*
  have *blks′*: (∀ *d*. ∀ *br* ∈ *blocksRead s′ p d*.
                    mbal(block br) < *mbal(dblock s′ p)*)
    by(*auto simp add*: *EndPhase1-def InitializePhase-def*
                  Inv2c-def Inv2c-inner-def)
  from *act inv*
  have *outpt′*: ∀ *p*. *outpt s′ p* = (*if phase s′ p* = *3*
                                then inp(dblock s′ p)
                                else NotAnInput)
    by(*auto simp add*: *EndPhase1-def Inv2c-def Inv2c-inner-def*)
  from *inv2a′*
  have *dblk*: ∀ *p*. *inp (dblock s′ p)* ∈ *allInput s′* ∪ {*NotAnInput*}
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
                  Inv2a-innermost-def blocksOf-def)
  with *act inv outpt′*
  have *chosen′*: *chosen s′* ∈ *allInput s′* ∪ {*NotAnInput*}
    by(*auto dest*: *HNextPart-Inv2c-chosen*)
  from *act inv*
  have *allinp*: ∀ *p*.   *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′*: $\forall p.$ *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*: $\forall p.$ *inp* (*dblock s′ p*) $\in$ *allInput s′* $\cup$ {*NotAnInput*}
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
                   Inv2a-innermost-def blocksOf-def)
with *act inv outpt′*
have *chosen′*: *chosen s′* $\in$ *allInput s′* $\cup$ {*NotAnInput*}
  by(*auto dest*: *HNextPart-Inv2c-chosen*)
from *act inv*
have *allinp*: $\forall p.$ *inpt s′ p* $\in$ *allInput s′*
             $\wedge$ (*chosen s′* = *NotAnInput*
                     $\longrightarrow$ *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′*: $\forall p.$ *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*: $\forall p.$ *inp* (*dblock s′ p*) $\in$ *allInput s′* $\cup$ {*NotAnInput*}
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
                     Inv2a-innermost-def blocksOf-def)
  with *act inv outpt′*
  have *chosen′*: *chosen s′* $\in$ *allInput s′* $\cup$ {*NotAnInput*}
    by(*auto dest*: *HNextPart-Inv2c-chosen*)
  from *act inv*
  have *allinp*: $\forall p.$ *inpt s′ p* $\in$ *allInput s′* $\wedge$ (*chosen s′* = *NotAnInput*
               $\longrightarrow$ *outpt s′ p* = *NotAnInput*)

by(*auto dest*: *HNextPart-chosen HNextPart-allInput*)
        with *outpt′ chosen′ act inv*
        show *?thesis*
          by(*auto simp add*: *Fail-def InitializePhase-def*
                        Inv2c-def Inv2c-inner-def)
qed

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

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

44

have *inv2a′*: *Inv2a s′*
  by(*blast dest*: *HEndPhase0-Inv2a*)
hence *bal′*: *bal*(*dblock s′ p*) ∈ *Ballot p* ∪ {*0*}
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
             Inv2a-innermost-def blocksOf-def)
from *act inv*
have *mbal′*: *mbal*(*dblock s′ p*) ∈ *Ballot p*
  by(*auto simp add*: *EndPhase0-def Inv2c-def Inv2c-inner-def*)
from *act inv*
have *blks′*: (∀ *d*. ∀ *br* ∈ *blocksRead s′ p d*.
              mbal(*block br*) < *mbal*(*dblock s′ p*))
  by(*auto simp add*: *EndPhase0-def InitializePhase-def*
             Inv2c-def Inv2c-inner-def)
from *act inv*
have *outpt′*: ∀ *p*. *outpt s′ p* = (*if phase s′ p = 3*
                     then inp(*dblock s′ p*)
                     else NotAnInput)
  by(*auto simp add*: *EndPhase0-def Inv2c-def Inv2c-inner-def*)
from *inv2a′*
have *dblk*: ∀ *p*. *inp* (*dblock s′ p*) ∈ *allInput s′* ∪ {*NotAnInput*}
  by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
             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*: *EndPhase0-def InitializePhase-def*
             Inv2c-def Inv2c-inner-def)
qed

theorem *HInit-HInv2*:
  HInit s ⟹ *HInv2 s*
using *HInit-Inv2a HInit-Inv2b HInit-Inv2c*
by(*auto simp add*: *HInv2-def*)

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

lemma *I2b*:
  assumes *nxt*: *HNext s s′*
  and *inv*: *HInv1 s* ∧ *HInv2 s*
  shows *HInv2 s′*
proof(*auto simp add*: *HInv2-def*)
  show *Inv2a s′* **using** *assms*
    by (*auto simp add*: *HInv2-def HNext-def Next-def*,
        auto intro: *HStartBallot-Inv2a*,

45

auto intro: *HPhase1or2Write-Inv2a*,
  auto simp add: *Phase1or2Read-def*
   intro: *HPhase1or2ReadThen-Inv2a*
    HPhase1or2ReadElse-Inv2a,
  auto intro: *HPhase0Read-Inv2a*,
  auto simp add: *EndPhase1or2-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-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-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

HInv3-L s p q d = (*phase s p* ∈ {*1,2*}
         ∧ *phase s q* ∈ {*1,2*}
         ∧ *hasRead s p d q*
         ∧ *hasRead s q d p*)

definition *HInv3-R* :: *state* ⇒ *Proc* ⇒ *Proc* ⇒ *Disk* ⇒ *bool*
where
   HInv3-R s p q d = ((⦇*block*= *dblock s q*, *proc*= *q*⦈) ∈ *blocksRead s p d*
             ∨ (⦇*block*= *dblock s p*, *proc*= *p*⦈) ∈ *blocksRead s q d*)

definition *HInv3-inner* :: *state* ⇒ *Proc* ⇒ *Proc* ⇒ *Disk* ⇒ *bool*
   where *HInv3-inner s p q d* = (*HInv3-L s p q d* ⟶ *HInv3-R s p q d*)

definition *HInv3* :: *state* ⇒ *bool*
   where *HInv3 s* = (∀ *p q d*. *HInv3-inner s p q d*)

C.3.1 Proofs of Invariant 3

theorem *HInit-HInv3*: *HInit s* ⟹ *HInv3 s*
  by(*simp add*: *HInit-def Init-def HInv3-def*
          HInv3-inner-def HInv3-L-def HInv3-R-def)

lemma *InitPhase-HInv3-p*:
  ⟦ *InitializePhase s s′ p*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *InitializePhase-def HInv3-inner-def*
             hasRead-def HInv3-L-def HInv3-R-def)

lemma *InitPhase-HInv3-q*:
  ⟦ *InitializePhase s s′ q* ; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *InitializePhase-def HInv3-inner-def*
             hasRead-def HInv3-L-def HInv3-R-def)

lemma *HInv3-L-sym*: *HInv3-L s p q d* ⟹ *HInv3-L s q p d*
  by(*auto simp add*: *HInv3-L-def*)

lemma *HInv3-R-sym*: *HInv3-R s p q d* ⟹ *HInv3-R s q p d*
  by(*auto simp add*: *HInv3-R-def*)

lemma *Phase1or2ReadThen-HInv3-pq*:
  assumes *act*: *Phase1or2ReadThen s s′ p d q*
  and   *inv-L′*: *HInv3-L s′ p q d*
  and   *pq*: *p≠q*
  and   *inv2b*: *Inv2b s*
  shows   *HInv3-R s′ p q d*
proof −
  from *inv-L′ act pq*
  have *phase s q* ∈ {*1,2*} ∧ *hasRead s q d p*
    by(*auto simp add*: *Phase1or2ReadThen-def HInv3-L-def*
          hasRead-def split: *if-split-asm*)

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

lemma *Phase1or2ReadThen-HInv3-hasRead*:
  ⟦ *¬hasRead s pp dd qq*;
    Phase1or2ReadThen s s′ p d q;
    pp≠p ∨ qq≠q ∨ dd≠d⟧
  ⟹ *¬hasRead s′ pp dd qq*
  by(*auto simp add: hasRead-def Phase1or2ReadThen-def*)

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

$\land$ *hasRead s′ pp dd qq* $\land$ *hasRead s′ qq dd pp*
        by(*auto simp add*: *HInv3-L-def Phase1or2ReadThen-def*)
      with *act False*
      show *?thesis*
        by(*auto dest*: *Phase1or2ReadThen-HInv3-hasRead*)
    qed
  qed
qed

lemma *StartBallot-HInv3-p*:
  ⟦ *StartBallot s s′ p*; *HInv3-L s′ p q d* ⟧
        $\implies$ *HInv3-R s′ p q d*
by(*auto simp add*: *StartBallot-def dest*: *InitPhase-HInv3-p*)

lemma *StartBallot-HInv3-q*:
  ⟦ *StartBallot s s′ q*; *HInv3-L s′ p q d* ⟧
        $\implies$ *HInv3-R s′ p q d*
  by(*auto simp add*: *StartBallot-def dest*: *InitPhase-HInv3-q*)

lemma *StartBallot-HInv3-nL*:
  ⟦ *StartBallot s s′ t*; ¬*HInv3-L s p q d*; *t≠p*; *t≠ q* ⟧
        $\implies$ ¬*HInv3-L s′ p q d*
  by(*auto simp add*: *StartBallot-def InitializePhase-def*
                HInv3-L-def hasRead-def)

lemma *StartBallot-HInv3-R*:
  ⟦ *StartBallot s s′ t*; *HInv3-R s p q d*; *t≠p*; *t≠ q* ⟧
          $\implies$ *HInv3-R s′ p q d*
  by(*auto simp add*: *StartBallot-def InitializePhase-def*
                HInv3-R-def hasRead-def)

lemma *StartBallot-HInv3-t*:
  ⟦ *StartBallot s s′ t*; *HInv3-inner s p q d*; *t≠p*; *t≠ q* ⟧
            $\implies$ *HInv3-inner s′ p q d*
  by(*auto simp add*: *HInv3-inner-def*
      dest: *StartBallot-HInv3-nL StartBallot-HInv3-R*)

lemma *StartBallot-HInv3*:
  assumes *act*: *StartBallot s s′ t*
  and   *inv*: *HInv3-inner s p q d*
  shows   *HInv3-inner s′ p q d*
proof(*cases t=p* $\lor$ *t=q*)
  case *True*
  with *act inv*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def*
        dest: *StartBallot-HInv3-p StartBallot-HInv3-q*)
next
  case *False*

49

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

theorem *HStartBallot-HInv3*:
  ⟦ *HStartBallot s s′ p*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  by(*auto simp add*: *HInv3-def dest*: *StartBallot-HInv3*)

theorem *HPhase1or2ReadElse-HInv3*:
  ⟦ *HPhase1or2ReadElse s s′ p d q*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  by(*auto simp add*: *Phase1or2ReadElse-def HInv3-def*
        dest: *StartBallot-HInv3*)

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

lemma *EndPhase1-HInv3-p*:
  ⟦ *EndPhase1 s s′ p*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
by(*auto simp add*: *EndPhase1-def dest*: *InitPhase-HInv3-p*)

lemma *EndPhase1-HInv3-q*:
  ⟦ *EndPhase1 s s′ q*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *EndPhase1-def dest*: *InitPhase-HInv3-q*)

lemma *EndPhase1-HInv3-nL*:

$\llbracket$ *EndPhase1 s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\implies$ ¬*HInv3-L s′ p q d*
by(*auto simp add*: *EndPhase1-def InitializePhase-def*
HInv3-L-def hasRead-def)

lemma *EndPhase1-HInv3-R*:
$\llbracket$ *EndPhase1 s s′ t*; *HInv3-R s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\implies$ *HInv3-R s′ p q d*
by(*auto simp add*: *EndPhase1-def InitializePhase-def*
HInv3-R-def hasRead-def)

lemma *EndPhase1-HInv3-t*:
$\llbracket$ *EndPhase1 s s′ t*; *HInv3-inner s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\implies$ *HInv3-inner s′ p q d*
by(*auto simp add*: *HInv3-inner-def dest*: *EndPhase1-HInv3-nL*
EndPhase1-HInv3-R)

lemma *EndPhase1-HInv3*:
assumes *act*: *EndPhase1 s s′ t*
and *inv*: *HInv3-inner s p q d*
shows *HInv3-inner s′ p q d*
proof(*cases t=p* ∨ *t=q*)
case *True*
with *act inv*
show *?thesis*
by(*auto simp add*: *HInv3-inner-def*
dest: *EndPhase1-HInv3-p EndPhase1-HInv3-q*)
next
case *False*
with *inv act*
show *?thesis*
by(*auto simp add*: *HInv3-inner-def dest*: *EndPhase1-HInv3-t*)
qed

theorem *HEndPhase1-HInv3*:
$\llbracket$ *HEndPhase1 s s′ p*; *HInv3 s* $\rrbracket$ $\implies$ *HInv3 s′*
by(*auto simp add*: *HInv3-def dest*: *EndPhase1-HInv3*)

lemma *EndPhase2-HInv3-p*:
$\llbracket$ *EndPhase2 s s′ p*; *HInv3-L s′ p q d* $\rrbracket$ $\implies$ *HInv3-R s′ p q d*
by(*auto simp add*: *EndPhase2-def dest*: *InitPhase-HInv3-p*)

lemma *EndPhase2-HInv3-q*:
$\llbracket$ *EndPhase2 s s′ q*; *HInv3-L s′ p q d* $\rrbracket$ $\implies$ *HInv3-R s′ p q d*
by(*auto simp add*: *EndPhase2-def dest*: *InitPhase-HInv3-q*)

lemma *EndPhase2-HInv3-nL*:
$\llbracket$ *EndPhase2 s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\implies$ ¬*HInv3-L s′ p q d*

by(*auto simp add*: *EndPhase2-def InitializePhase-def*
              HInv3-L-def hasRead-def)

lemma *EndPhase2-HInv3-R*:
  ⟦ *EndPhase2 s s′ t*; *HInv3-R s p q d*; *t≠p*; *t≠ q* ⟧
                  ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *EndPhase2-def InitializePhase-def*
              HInv3-R-def hasRead-def)

lemma *EndPhase2-HInv3-t*:
  ⟦ *EndPhase2 s s′ t*; *HInv3-inner s p q d*; *t≠p*; *t≠ q* ⟧
                  ⟹ *HInv3-inner s′ p q d*
  by(*auto simp add*: *HInv3-inner-def*
          dest: *EndPhase2-HInv3-nL EndPhase2-HInv3-R*)

lemma *EndPhase2-HInv3*:
  assumes *act*: *EndPhase2 s s′ t*
  and   *inv*: *HInv3-inner s p q d*
  shows   *HInv3-inner s′ p q d*
proof(*cases t=p ∨ t=q*)
  case *True*
  with *act inv*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def*
            dest: *EndPhase2-HInv3-p EndPhase2-HInv3-q*)
next
  case *False*
  with *inv act*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def dest*: *EndPhase2-HInv3-t*)
qed

theorem *HEndPhase2-HInv3*:
  ⟦ *HEndPhase2 s s′ p*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  by(*auto simp add*: *HInv3-def dest*: *EndPhase2-HInv3*)

lemma *Fail-HInv3-p*:
  ⟦ *Fail s s′ p*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
by(*auto simp add*: *Fail-def dest*: *InitPhase-HInv3-p*)

lemma *Fail-HInv3-q*:
  ⟦ *Fail s s′ q*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *Fail-def dest*: *InitPhase-HInv3-q*)

lemma *Fail-HInv3-nL*:
  ⟦ *Fail s s′ t*; ¬*HInv3-L s p q d*; *t≠p*; *t≠ q* ⟧
          ⟹ ¬*HInv3-L s′ p q d*
  by(*auto simp add*: *Fail-def InitializePhase-def*
              HInv3-L-def hasRead-def)

lemma *Fail-HInv3-R*:
  ⟦ *Fail s s′ t*; *HInv3-R s p q d*; *t≠p*; *t≠ q* ⟧
            ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *Fail-def InitializePhase-def*
               HInv3-R-def hasRead-def)

lemma *Fail-HInv3-t*:
  ⟦ *Fail s s′ t*; *HInv3-inner s p q d*; *t≠p*; *t≠ q* ⟧
            ⟹ *HInv3-inner s′ p q d*
  by(*auto simp add*: *HInv3-inner-def*
            dest: *Fail-HInv3-nL Fail-HInv3-R*)

lemma *Fail-HInv3*:
  assumes *act*: *Fail s s′ t*
  and   *inv*: *HInv3-inner s p q d*
  shows   *HInv3-inner s′ p q d*
proof(*cases t=p ∨ t=q*)
  case *True*
  with *act inv*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def*
             dest: *Fail-HInv3-p Fail-HInv3-q*)
next
  case *False*
  with *inv act*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def dest*: *Fail-HInv3-t*)
qed

theorem *HFail-HInv3*:
  ⟦ *HFail s s′ p*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  by(*auto simp add*: *HInv3-def dest*: *Fail-HInv3*)

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

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

lemma *EndPhase0-HInv3-p*:
  ⟦ *EndPhase0 s s′ p*; *HInv3-L s′ p q d* ⟧
        ⟹ *HInv3-R s′ p q d*
by(*auto simp add*: *EndPhase0-def dest*: *InitPhase-HInv3-p*)

lemma *EndPhase0-HInv3-q*:
  ⟦ *EndPhase0 s s′ q*; *HInv3-L s′ p q d* ⟧
        ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *EndPhase0-def dest*: *InitPhase-HInv3-q*)

lemma *EndPhase0-HInv3-nL*:
  ⟦ *EndPhase0 s s′ t*; *¬HInv3-L s p q d*; *t≠p*; *t≠ q* ⟧
        ⟹ *¬HInv3-L s′ p q d*
  by(*auto simp add*: *EndPhase0-def InitializePhase-def*
            HInv3-L-def hasRead-def)

lemma *EndPhase0-HInv3-R*:
  ⟦ *EndPhase0 s s′ t*; *HInv3-R s p q d*; *t≠p*; *t≠ q* ⟧
        ⟹ *HInv3-R s′ p q d*
  by(*auto simp add*: *EndPhase0-def InitializePhase-def*
            HInv3-R-def hasRead-def)

lemma *EndPhase0-HInv3-t*:
  ⟦ *EndPhase0 s s′ t*; *HInv3-inner s p q d*; *t≠p*; *t≠ q* ⟧
        ⟹ *HInv3-inner s′ p q d*
  by(*auto simp add*: *HInv3-inner-def*
       dest: *EndPhase0-HInv3-nL EndPhase0-HInv3-R*)

lemma *EndPhase0-HInv3*:
  assumes *act*: *EndPhase0 s s′ t*
  and   *inv*: *HInv3-inner s p q d*
  shows   *HInv3-inner s′ p q d*
proof(*cases t=p ∨ t=q*)
  case *True*
  with *act inv*
  show *?thesis*
    by(*auto simp add*: *HInv3-inner-def*
          dest: *EndPhase0-HInv3-p EndPhase0-HInv3-q*)
next

case *False*
with *inv act*
show *?thesis*
  by(*auto simp add*: *HInv3-inner-def dest*: *EndPhase0-HInv3-t*)
qed

theorem *HEndPhase0-HInv3*:
  ⟦ *HEndPhase0 s s′ p*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  by(*auto simp add*: *HInv3-def dest*: *EndPhase0-HInv3*)

*HInv*1 ∧ *HInv*2 ∧ *HInv*3 is an invariant of *HNext*.

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

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* ⇒ *Proc* ⇒ *bool*

55

where *HInv4a1 s p = (∀ bk ∈ blocksOf s p. bal bk ≤ mbal (dblock s p))*

definition *HInv4a2 :: state ⇒ Proc ⇒ bool*
where
  HInv4a2 s p = (∀ D ∈ MajoritySet.(∃ d ∈ D. mbal(disk s d p) ≤ mbal(dblock s p)

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

definition *HInv4a :: state ⇒ Proc ⇒ bool*
  where *HInv4a s p = (phase s p ≠ 0 ⟶ HInv4a1 s p ∧ HInv4a2 s p)*

definition *HInv4b :: state ⇒ Proc ⇒ bool*
  where *HInv4b s p = (phase s p = 1 ⟶ (∀ bk ∈ blocksOf s p. bal bk < mbal(dblock s p)))*

definition *HInv4c :: state ⇒ Proc ⇒ bool*
  where *HInv4c s p = (phase s p ∈ {2,3} ⟶ (∃ D∈MajoritySet. ∀ d∈D. mbal(disk s d p) = bal (dblock s p)))*

definition *HInv4d :: state ⇒ Proc ⇒ bool*
  where *HInv4d s p = (∀ bk ∈ blocksOf s p. ∃ D∈MajoritySet. ∀ d∈D. bal bk ≤ mbal (disk s d p))*

definition *HInv4 :: state ⇒ bool*
  where *HInv4 s = (∀ p. HInv4a s p ∧ HInv4b s p ∧ HInv4c s p ∧ HInv4d s p)*

The initial state implies Invariant 4.

theorem *HInit-HInv4*: *HInit s ⟹ HInv4 s*
  using *Disk-isMajority*
  by(*auto simp add*: *HInit-def Init-def HInv4-def HInv4a-def HInv4a1-def*
              HInv4a2-def HInv4b-def HInv4c-def HInv4d-def
              MajoritySet-def blocksOf-def InitDB-def rdBy-def)

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

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

C.4.1   Proofs of Invariant 4a

lemma *HStartBallot-HInv4a1*:
  assumes *act*: *HStartBallot s s′ p*
  and *inv*: *HInv4a1 s p*
  and *inv2a*: *Inv2a-inner s′ p*
  shows *HInv4a1 s′ p*
proof(*auto simp add*: *HInv4a1-def*)
  fix *bk*

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

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

lemma $HStartBallot\text{-}HInv4a\text{-}p$:
  assumes $act\colon HStartBallot\ s\ s'\ p$
  and $inv\colon HInv4a\ s\ p$
  and $inv2a\colon Inv2a\text{-}inner\ s'\ p$

shows *HInv4a s′ p*
using *act inv inv2a*
proof −
  from *act*
  have *phase*: *0 < phase s p*
    by(*auto simp add*: *StartBallot-def*)
  from *act inv inv2a*
  show *?thesis*
    by(*auto simp del*: *HStartBallot-def simp add*: *HInv4a-def phase*
          elim: *HStartBallot-HInv4a1 HStartBallot-HInv4a2*)
qed

lemma *HStartBallot-HInv4a-q*:
  assumes *act*: *HStartBallot s s′ p*
  and *inv*: *HInv4a s q*
  and *pnq*: *p≠q*
  shows *HInv4a s′ q*
proof −
  from *act pnq*
  have *blocksOf s′ q ⊆ blocksOf s q*
    by(*auto simp add*: *StartBallot-def InitializePhase-def*
          blocksOf-def rdBy-def)
  with *act inv pnq*
  show *?thesis*
    by(*auto simp add*: *StartBallot-def HInv4a-def*
          HInv4a1-def HInv4a2-def)
qed

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

lemma *Phase1or2Write-HInv4a1*:

$\llbracket$ *Phase1or2Write s s′ p d*; *HInv4a1 s q* $\rrbracket \Longrightarrow$ *HInv4a1 s′ q*
by(*auto simp add*: *Phase1or2Write-def HInv4a1-def*
              blocksOf-def rdBy-def)

lemma *Phase1or2Write-HInv4a2*:
  $\llbracket$ *Phase1or2Write s s′ p d*; *HInv4a2 s q* $\rrbracket \Longrightarrow$ *HInv4a2 s′ q*
  by(*auto simp add*: *Phase1or2Write-def HInv4a2-def*)

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

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

lemma *HPhase1or2ReadThen-HInv4a2*:
  $\llbracket$ *HPhase1or2ReadThen s s′ p d r*; *HInv4a2 s q* $\rrbracket \Longrightarrow$ *HInv4a2 s′ q*
  by(*auto simp add*: *Phase1or2ReadThen-def HInv4a2-def*)

lemma *HPhase1or2ReadThen-HInv4a-p*:
  assumes *act*: *HPhase1or2ReadThen s s′ p d r*
  and *inv*: *HInv4a s p*
  and *inv2b*: *Inv2b s*
  shows *HInv4a s′ p*
proof −
  from *act inv2b*
  have *phase s p* ∈ {*1,2*}
    by(*auto simp add: Phase1or2ReadThen-def Inv2b-def Inv2b-inner-def*)
  with *act inv*
  show *?thesis*
    by(*auto simp del: HPhase1or2ReadThen-def simp add: HInv4a-def*
        elim: HPhase1or2ReadThen-HInv4a1-p HPhase1or2ReadThen-HInv4a2)
qed

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

theorem *HPhase1or2ReadThen-HInv4a*:
  ⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4a s q*; *Inv2b s* ⟧ ⟹ *HInv4a s′ q*
  by(*blast dest: HPhase1or2ReadThen-HInv4a-p HPhase1or2ReadThen-HInv4a-q*)

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

lemma *HEndPhase1-HInv4a1*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a1 s p*

shows *HInv4a1 s′ p*
proof(*auto simp add*: *HInv4a1-def*)
  fix *bk*
  assume *bk*: *bk* ∈ *blocksOf s′ p*
  from *bk HEndPhase1-blocksOf*[*OF act*]
  have *bk* ∈ {*dblock s′ p*} ∪ *blocksOf s p*
    by *blast*
  with *act inv*
  show *bal bk* ≤ *mbal* (*dblock s′ p*)
    by(*auto simp add*: *HInv4a-def HInv4a1-def EndPhase1-def*)
qed

lemma *HEndPhase1-HInv4a2*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a2 s p*
  and *inv2a*: *Inv2a s*
  shows *HInv4a2 s′ p*
proof(*auto simp add*: *HInv4a2-def*)
  fix *D*
  assume *Dmaj*: *D* ∈ *MajoritySet*
  from *inv Dmaj*
  have ∃ *d*∈*D*. *mbal* (*disk s d p*) ≤ *mbal* (*dblock s p*)
        ∧ *bal* (*disk s d p*) ≤ *bal* (*dblock s p*)
    by(*auto simp add*: *HInv4a2-def*)
  then obtain *d*
    where *d-cond*: *d*∈*D*
             ∧ *mbal* (*disk s d p*) ≤ *mbal* (*dblock s p*)
             ∧ *bal* (*disk s d p*) ≤ *bal* (*dblock s p*)
    by *auto*
  have *disk s d p* ∈ *blocksOf s p*
    by(*auto simp add*: *blocksOf-def*)
  with *inv2a*
  have *bal*(*disk s d p*) ≤ *mbal* (*disk s d p*)
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def Inv2a-innermost-def*)
  with *act d-cond*
  have *d*∈*D*
      ∧ *mbal* (*disk s′ d p*) ≤ *mbal* (*dblock s′ p*)
      ∧ *bal* (*disk s′ d p*) ≤ *bal* (*dblock s′ p*)
    by(*auto simp add*: *EndPhase1-def*)
  with *Dmaj*
  show ∃ *d*∈*D*. *mbal* (*disk s′ d p*) ≤ *mbal* (*dblock s′ p*)
        ∧ *bal* (*disk s′ d p*) ≤ *bal* (*dblock s′ p*)
    by *auto*
qed

lemma *HEndPhase1-HInv4a-p*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a s p*
  and *inv2a*: *Inv2a s*

61

shows *HInv4a s′ p*
proof −
  from *act*
  have *phase*: *0 < phase s p*
    by(*auto simp add*: *EndPhase1-def*)
  with *act inv inv2a*
  show *?thesis*
    by(*auto simp del*: *HEndPhase1-def simp add*: *HInv4a-def*
          elim: *HEndPhase1-HInv4a1 HEndPhase1-HInv4a2*)
qed

lemma *HEndPhase1-HInv4a-q*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a s q*
  and *pnq*: *p≠q*
  shows *HInv4a s′ q*
proof −
  from *act pnq*
  have *dblock s′ q = dblock s q ∧ disk s′ = disk s*
    by(*auto simp add*: *EndPhase1-def*)
  moreover
  from *act pnq*
  have *∀ p d. rdBy s′ q p d ⊆ rdBy s q p d*
    by(*auto simp add*: *EndPhase1-def InitializePhase-def rdBy-def*)
  hence (*UN p d. rdBy s′ q p d*) ⊆ (*UN p d. rdBy s q p d*)
    by(*auto, blast*)
  ultimately
  have *blocksOf s′ q ⊆ blocksOf s q*
    by(*auto simp add*: *blocksOf-def, 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* ⟧ ⟹ *HInv4a s′ q*
  by(*blast dest*: *HEndPhase1-HInv4a-p HEndPhase1-HInv4a-q*)

theorem *HFail-HInv4a*:
  ⟦ *HFail s s′ p*; *HInv4a s q* ⟧ ⟹ *HInv4a s′ q*
  by(*auto simp add*: *Fail-def HInv4a-def HInv4a1-def*
               HInv4a2-def InitializePhase-def
               blocksOf-def rdBy-def)

theorem *HPhase0Read-HInv4a*:
  ⟦ *HPhase0Read s s′ p d*; *HInv4a s q* ⟧ ⟹ *HInv4a s′ q*
  by(*auto simp add*: *Phase0Read-def HInv4a-def HInv4a1-def*
               HInv4a2-def InitializePhase-def
               blocksOf-def rdBy-def)

theorem *HEndPhase2-HInv4a*:
  ⟦ *HEndPhase2 s s′ p*; *HInv4a s q* ⟧ ⟹ *HInv4a s′ q*
  by(*auto simp add: EndPhase2-def HInv4a-def HInv4a1-def HInv4a2-def*
                InitializePhase-def blocksOf-def rdBy-def)

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

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

qed

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

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

have *disk s d p* ∈ *block ' blocksRead s p d*
  by *force*
thus *disk s d p* ∈ *allBlocksRead s p*
  by(*auto simp add: allBlocksRead-def allRdBlks-def*)
qed

lemma *HEndPhase0-41*:
  assumes *act*: *HEndPhase0 s s' p*
  and   *inv1*: *Inv1 s*
  and   *inv2c*: *Inv2c-inner s p*
  shows ∃ *D*∈*MajoritySet*. ∀ *d*∈*D*.   *mbal*(*disk s d p*) ≤ *mbal*(*dblock s' p*)
                        ∧ *bal*(*disk s d p*) ≤ *bal*(*dblock s' p*)
proof −
  from *act HEndPhase0-some*[*OF act inv1*]
  have *p51*: ∀ *br*∈*allBlocksRead s p*.   *mbal br* < *mbal*(*dblock s' p*)
                        ∧ *bal br* ≤ *bal*(*dblock s' p*)
    and *a*: *IsMajority*({*d*. *hasRead s p d p*})
    and *phase*: *phase s p = 0*
    by(*auto simp add: EndPhase0-def*)+
  from *inv2c phase*
  have (∀ *d*∈{*d*. *hasRead s p d p*}. *disk s d p* ∈ *allBlocksRead s p*)
    by(*auto dest: hasRead-allBlks*)
  with *p51*
  have (∀ *d*∈{*d*. *hasRead s p d p*}.   *mbal*(*disk s d p*) ≤ *mbal*(*dblock s' p*)
                        ∧ *bal*(*disk s d p*) ≤ *bal*(*dblock s' p*))
    by *force*
  with *a* **show** *?thesis*
    by(*auto simp add: MajoritySet-def*)
qed

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

lemma *HEndPhase0-HInv4a2-p*:
  assumes *act*: *HEndPhase0 s s' p*
  and   *inv1*: *Inv1 s*

65

and   *inv2c*: *Inv2c-inner s p*
shows *HInv4a2 s′ p*
proof(*simp add*: *HInv4a2-def*)
  from *act*
  have *disk′*: *disk s′ = disk s*
    by(*simp add*: *EndPhase0-def*)
  from *act inv1 inv2c*
  have $\exists\, D \in MajoritySet.\ \forall\, d \in D.\quad mbal(disk\ s\ d\ p) \leq mbal(dblock\ s′\ p)$
                              $\wedge\ bal(disk\ s\ d\ p) \leq bal(dblock\ s′\ p)$
    by(*blast dest*: *HEndPhase0-41*)
  from *Majority-exQ[OF this]*
  have $\forall\, D \in MajoritySet.\ \exists\, d \in D.\quad mbal(disk\ s\ d\ p) \leq mbal(dblock\ s′\ p)$
                              $\wedge\ bal(disk\ s\ d\ p) \leq bal(dblock\ s′\ p)$
  (**is** *?P (disk s)*) **.**
  from *ssubst[OF disk′, of ?P, OF this]*
  show $\forall\, D \in MajoritySet.\ \exists\, d \in D.\ mbal\ (disk\ s′\ d\ p) \leq mbal\ (dblock\ s′\ p)$
                            $\wedge\ bal\ (disk\ s′\ d\ p) \leq bal\ (dblock\ s′\ p)$ **.**
qed

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

lemma *HEndPhase0-HInv4a-q*:
  assumes *act*: *HEndPhase0 s s′ p*
  and *inv*: *HInv4a s q*
  and *pnq*: *p≠q*
  shows *HInv4a s′ q*
proof −

from *act pnq*
have *dblock s′ q = dblock s q ∧ disk s′ = disk s*
  by(*auto simp add: EndPhase0-def*)
moreover
from *act pnq*
have *∀ p d. rdBy s′ q p d ⊆ rdBy s q p d*
  by(*auto simp add: EndPhase0-def InitializePhase-def rdBy-def*)
hence (*UN p d. rdBy s′ q p d*) ⊆ (*UN p d. rdBy s q p d*)
  by(*auto, blast*)
ultimately
have *blocksOf s′ q ⊆ blocksOf s q*
  by(*auto simp add: blocksOf-def, blast*)
with *act inv pnq*
show *?thesis*
  by(*auto simp add: EndPhase0-def HInv4a-def HInv4a1-def HInv4a2-def*)
qed

theorem *HEndPhase0-HInv4a*:
  ⟦ *HEndPhase0 s s′ p*; *HInv4a s q*; *HInv4d s p*;
    Inv2a s; *Inv1 s*; *Inv2a s*; *Inv2c s*⟧
  ⟹ *HInv4a s′ q*
  by(*blast dest: HEndPhase0-HInv4a-p HEndPhase0-HInv4a-q*)

C.4.2   Proofs of Invariant 4b

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

lemma *HEndPhase0-dblock-mbal*:
  ⟦ *HEndPhase0 s s′ p* ⟧
    ⟹ *∀ br∈allBlocksRead s p. mbal br < mbal(dblock s′ p)*
  by(*auto simp add: EndPhase0-def*)

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

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

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

68

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

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

show *?thesis*
  by(*auto simp add*: *HInv4b-def*)
qed

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

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

show *?thesis* **by** *auto*
  next
    assume *bk*: *bk* ∈ *blocksOf s p*
    from *phase inv4a*
    have *p41*: *HInv4a1 s p*
      by(*auto simp add*: *HInv4a-def*)
    with *p42 bk*
    show *?thesis*
      by(*auto simp add*: *HInv4a1-def*)
  qed
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 *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 *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* ⟧ ⟹ *HInv4b s′ q*
  by(*auto simp add*: *Phase1or2Write-def HInv4b-def*
               blocksOf-def rdBy-def)

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

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

theorem *HPhase1or2ReadThen-HInv4b*:
  ⟦ *HPhase1or2ReadThen s s′ p d q*; *HInv4b s r*⟧ ⟹ *HInv4b s′ r*
  by(*blast dest*: *HPhase1or2ReadThen-HInv4b-p*
            HPhase1or2ReadThen-HInv4b-q)

theorem *HPhase1or2ReadElse-HInv4b*:
  ⟦ *HPhase1or2ReadElse s s′ p d q*; *HInv4b s r*;
    Inv2a s; *HInv4a s p* ⟧
  ⟹ *HInv4b s′ r*
using *HStartBallot-HInv4b*
 by(*auto simp add*: *Phase1or2ReadElse-def*)

lemma *HEndPhase1-HInv4b-p*:
  HEndPhase1 s s′ p ⟹ *HInv4b s′ p*
  by(*auto simp add*: *EndPhase1-def HInv4b-def*)

lemma *HEndPhase1-HInv4b-q*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *pnq*: *p≠q*

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

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

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

lemma *HEndPhase2-HInv4b-q*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *pnq*: *p≠q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
proof −
  from *act pnq*
  have *disk′*: *disk s′=disk s*
   and *dblock′*: *dblock s′ q=dblock s q*
   and *phase′*: *phase s′ q =phase s q*
    by(*auto simp add*: *EndPhase2-def*)
  from *act pnq*

have *blocksRead'*: ∀ *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 *HEndPhase2-HInv4b-q*[*OF act False inv*]
  show *?thesis* .
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 *pnq*: *p≠q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
proof −
  from *act pnq*
  have *disk′*: *disk s′=disk s*
   and *dblock′*: *dblock s′ q=dblock s q*
   and *phase′*: *phase s′ q =phase s q*
    by(*auto simp add*: *Phase0Read-def*)
  from *HPhase0Read-blocksOf*[*OF act*] *inv phase′ dblock′*
  show *?thesis*
    by(*auto simp add*: *HInv4b-def*)
qed

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

C.4.3 Proofs of Invariant 4c

lemma *HStartBallot-HInv4c-p*:
  ⟦ *HStartBallot s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
  by(*auto simp add*: *StartBallot-def HInv4c-def*)

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

theorem *HStartBallot-HInv4c*:
  ⟦ *HStartBallot s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
  by(*blast dest*: *HStartBallot-HInv4c-p HStartBallot-HInv4c-q*)

lemma *HPhase1or2Write-HInv4c-p*:
  assumes *act*: *HPhase1or2Write s s′ p d*
     and *inv*: *HInv4c s p*
    and *inv2c*: *Inv2c s*
   shows *HInv4c s′ p*
proof(*cases phase s′ p = 2*)
  assume *phase′*: *phase s′ p = 2*
  show *?thesis*
  proof(*auto simp add*: *HInv4c-def phase′ MajoritySet-def*)
    from *act phase′*
    have *bal*: *bal(dblock s′ p)=bal(dblock s p)*
      and *phase*: *phase s p = 2*
      by(*auto simp add*: *Phase1or2Write-def*)
    from *phase′ inv2c act*
    have *mbal(disk s′ d p)=bal(dblock s p)*
      by(*auto simp add*: *Phase1or2Write-def Inv2c-def Inv2c-inner-def*)
    with *bal*
    have *bal(dblock s′ p) = mbal(disk s′ d p)*
      by *auto*
    with *inv phase act*
    show ∃ *D*.   *IsMajority D*
            ∧ (∀ *d*∈*D*. *mbal* (*disk s′ d p*) = *bal* (*dblock s′ p*))
      by(*auto simp add*: *HInv4c-def Phase1or2Write-def MajoritySet-def*)
  qed

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

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

theorem *HPhase1or2Write-HInv4c*:
  ⟦ *HPhase1or2Write s s′ p d*; *HInv4c s q*; *Inv2c s* ⟧
        ⟹ *HInv4c s′ q*
  by(*blast dest*: *HPhase1or2Write-HInv4c-p*
         HPhase1or2Write-HInv4c-q)

lemma *HPhase1or2ReadThen-HInv4c-p*:
  ⟦ *HPhase1or2ReadThen s s′ p d q*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
  by(*auto simp add*: *Phase1or2ReadThen-def HInv4c-def*)

lemma *HPhase1or2ReadThen-HInv4c-q*:
  assumes *act*: *HPhase1or2ReadThen s s′ p d r*
  and *inv*: *HInv4c s q*
  and *pnq*: *p≠q*
  shows *HInv4c s′ q*
proof −
  from *act pnq*
  have *phase*: *phase s′ q = phase s q*
   and *dblock*: *dblock s q = dblock s′ q*
   and *disk*: *disk s′ = disk s*
    by(*auto simp add*: *Phase1or2ReadThen-def*)
  with *inv*
  show *?thesis*
    by(*auto simp add*: *HInv4c-def*)
qed

theorem *HPhase1or2ReadThen-HInv4c*:
  ⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4c s q*⟧
        ⟹ *HInv4c s′ q*
  by(*blast dest*: *HPhase1or2ReadThen-HInv4c-p*
              HPhase1or2ReadThen-HInv4c-q)

theorem *HPhase1or2ReadElse-HInv4c*:
  ⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
using *HStartBallot-HInv4c*
  by(*auto simp add*: *Phase1or2ReadElse-def*)

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

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

theorem *HEndPhase1-HInv4c*:
  ⟦ *HEndPhase1 s s′ p*; *HInv4c s q*; *Inv2b s*⟧ ⟹ *HInv4c s′ q*
  by(*blast dest*: *HEndPhase1-HInv4c-p HEndPhase1-HInv4c-q*)

lemma *HEndPhase2-HInv4c-p*:
  ⟦ *HEndPhase2 s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
  by(*auto simp add*: *EndPhase2-def HInv4c-def*)

lemma *HEndPhase2-HInv4c-q*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *inv*: *HInv4c s q*
  and *pnq*: *p≠q*
  shows *HInv4c s′ q*
proof −
  from *act pnq*
  have *phase*: *phase s′ q = phase s q*
   and *dblock*: *dblock s q = dblock s′ q*
   and *disk*: *disk s′ = disk s*
    by(*auto simp add*: *EndPhase2-def*)
  with *inv*
  show *?thesis*
    by(*auto simp add*: *HInv4c-def*)
qed

theorem *HEndPhase2-HInv4c*:
  ⟦ *HEndPhase2 s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
  by(*blast dest*: *HEndPhase2-HInv4c-p HEndPhase2-HInv4c-q*)

lemma *HFail-HInv4c-p*:
  ⟦ *HFail s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
  by(*auto simp add*: *Fail-def HInv4c-def*)

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

theorem *HFail-HInv4c*:
  ⟦ *HFail s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
  by(*blast dest*: *HFail-HInv4c-p HFail-HInv4c-q*)

lemma *HPhase0Read-HInv4c-p*:

$\llbracket$ *HPhase0Read s s′ p d*; *HInv4c s p* $\rrbracket$ $\Longrightarrow$ *HInv4c s′ p*
by(*auto simp add*: *Phase0Read-def HInv4c-def*)

lemma *HPhase0Read-HInv4c-q*:
  assumes *act*: *HPhase0Read s s′ p d*
  and *inv*: *HInv4c s q*
  and *pnq*: *p*≠*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*:
  $\llbracket$ *HPhase0Read s s′ p d*; *HInv4c s q* $\rrbracket$ $\Longrightarrow$ *HInv4c s′ q*
  by(*blast dest*: *HPhase0Read-HInv4c-p HPhase0Read-HInv4c-q*)

lemma *HEndPhase0-HInv4c-p*:
  $\llbracket$ *HEndPhase0 s s′ p*; *HInv4c s p* $\rrbracket$ $\Longrightarrow$ *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*:
  $\llbracket$ *HEndPhase0 s s′ p*; *HInv4c s q* $\rrbracket$ $\Longrightarrow$ *HInv4c s′ q*
  by(*blast dest*: *HEndPhase0-HInv4c-p HEndPhase0-HInv4c-q*)

C.4.4 Proofs of Invariant 4d

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

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

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

theorem *HStartBallot-HInv4d*:
⟦ *HStartBallot s s' p*; *HInv4d s q*⟧ ⟹ *HInv4d s' q*
by(*blast dest*: *HStartBallot-HInv4d-p HStartBallot-HInv4d-q*)

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

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

by(*auto simp add*: *Phase1or2Write-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 *HPhase1or2Write-HInv4d*:
⟦ *HPhase1or2Write s s′ p d*; *HInv4d s q*; *HInv4a s p*⟧ ⟹ *HInv4d s′ q*
by(*blast dest*: *HPhase1or2Write-HInv4d-p HPhase1or2Write-HInv4d-q*)

lemma *HPhase1or2ReadThen-HInv4d-p*:
assumes *act*: *HPhase1or2ReadThen s s′ p d q*
and *inv*: *HInv4d s p*
shows *HInv4d s′ p*
proof(*clarsimp simp add*: *HInv4d-def*)
fix *bk*
assume *bk*: *bk* ∈ *blocksOf s′ p*
from *act*
have *bal′*: *bal* (*dblock s′ p*) = *bal* (*dblock s p*)
by(*auto simp add*: *Phase1or2ReadThen-def*)
from *subsetD*[*OF HPhase1or2ReadThen-blocksOf*[*OF act*] *bk*] *inv*
have ∃ *D*∈*MajoritySet*. ∀ *d*∈*D*. *bal bk* ≤ *mbal* (*disk s d p*)
by(*auto simp add*: *HInv4d-def*)
with *act*
show ∃ *D*∈*MajoritySet*. ∀ *d*∈*D*. *bal bk* ≤ *mbal* (*disk s′ d p*)
by(*auto simp add*: *Phase1or2ReadThen-def*)
qed

lemma *HPhase1or2ReadThen-HInv4d-q*:
assumes *act*: *HPhase1or2ReadThen s s′ p d r*
and *inv*: *HInv4d s q*
and *pnq*: *p*≠*q*
shows *HInv4d s′ q*
proof −
from *act pnq*
have *disk′*: *disk s′*=*disk s*
by(*auto simp add*: *Phase1or2ReadThen-def*)
from *act pnq*
have *blocksOf s′ q* ⊆ *blocksOf s q*
by(*auto simp add*: *Phase1or2ReadThen-def allRdBlks-def*
blocksOf-def rdBy-def)
from *subsetD*[*OF this*] *inv*
have ∀ *bk*∈*blocksOf s′ q*.
∃ *D*∈*MajoritySet*. ∀ *d*∈*D*. *bal bk* ≤ *mbal*(*disk s d q*)

83

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

theorem *HPhase1or2ReadThen-HInv4d*:
　⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4d s q*⟧ ⟹ *HInv4d s′ q*
　by(*blast dest*: *HPhase1or2ReadThen-HInv4d-p*
　　　　　　 HPhase1or2ReadThen-HInv4d-q)

theorem *HPhase1or2ReadElse-HInv4d*:
　⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv4d s q*⟧ ⟹ *HInv4d s′ q*
using *HStartBallot-HInv4d*
　by(*auto simp add*: *Phase1or2ReadElse-def*)

lemma *HEndPhase1-HInv4d-p*:
　assumes *act*: *HEndPhase1 s s′ p*
　and *inv*: *HInv4d s p*
　and *inv2b*: *Inv2b s*
　and *inv4c*: *HInv4c s p*
　shows *HInv4d s′ p*
proof(*clarsimp simp add*: *HInv4d-def*)
　fix *bk*
　assume *bk*: *bk* ∈ *blocksOf s′ p*
　from *HEndPhase1-HInv4c*[*OF act inv4c inv2b*]
　have *HInv4c s′ p* **.**
　with *act*
　have *p31*: ∃ *D*∈*MajoritySet*.
　　　　　　∀ *d*∈*D. mbal* (*disk s′ d p*) = *bal*(*dblock s′ p*)
　　and *disk′*: *disk s′* = *disk s*
　　by(*auto simp add*: *EndPhase1-def HInv4c-def*)
　from *subsetD*[*OF HEndPhase1-blocksOf*[*OF act*] *bk*]
　show ∃ *D*∈*MajoritySet*. ∀ *d*∈*D. bal bk* ≤ *mbal* (*disk s′ d p*)
　proof
　　assume *bk*: *bk* ∈ *blocksOf s p*
　　with *inv disk′*
　　show *?thesis*
　　　by(*auto simp add*: *HInv4d-def*)
　next
　　assume *bk*: *bk* ∈ {*dblock s′ p*}
　　with *p31*
　　show *?thesis*
　　　by *force*
　qed
qed

lemma *HEndPhase1-HInv4d-q*:
　assumes *act*: *HEndPhase1 s s′ p*

84

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*

85

and *inv*: *HInv4d s q*
and *pnq*: *p≠q*
shows *HInv4d s′ q*
proof −
  from *act pnq*
  have *disk′*: *disk s′=disk s*
    by(*auto simp add*: *EndPhase2-def*)
  from *act pnq*
  have *blocksOf s′ q ⊆ blocksOf s q*
    by(*auto simp add*: *EndPhase2-def InitializePhase-def*
                allRdBlks-def blocksOf-def rdBy-def)
  from *subsetD*[*OF this*] *inv*
  have $\forall$ *bk∈blocksOf s′ q*.
       $\exists$ *D∈MajoritySet*. $\forall$ *d∈D. bal bk ≤ mbal*(*disk s d q*)
    by(*auto simp add*: *HInv4d-def*)
  with *disk′*
  show *?thesis*
  by(*auto simp add*: *HInv4d-def*)
qed

theorem *HEndPhase2-HInv4d*:
  $[\![$ *HEndPhase2 s s′ p*; *HInv4d s q* $]\!] \Longrightarrow$ *HInv4d s′ q*
  by(*blast dest*: *HEndPhase2-HInv4d-p HEndPhase2-HInv4d-q*)

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

qed
qed

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

theorem *HFail-HInv4d*:
  ⟦ *HFail s s′ p*; *HInv4d s q* ⟧ ⟹ *HInv4d s′ q*
  by(*blast dest*: *HFail-HInv4d-p HFail-HInv4d-q*)

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

lemma *HPhase0Read-HInv4d-q*:
　assumes *act*: *HPhase0Read s s′ p d*
　and *inv*: *HInv4d s q*
　and *pnq*: *p≠q*
　shows *HInv4d s′ q*
proof −
　from *act pnq*
　have *disk′*: *disk s′=disk s*
　　by(*auto simp add*: *Phase0Read-def*)
　from *act pnq*
　have *blocksOf s′ q ⊆ blocksOf s q*
　　by(*auto simp add*: *Phase0Read-def allRdBlks-def*
　　　　　　　blocksOf-def rdBy-def)
　from *subsetD*[*OF this*] *inv*
　have *∀ 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
  with *act*
  show ∃ *D*∈*MajoritySet*. ∀ *d*∈*D*. *bal bk* ≤ *mbal* (*disk s′ d p*)
    by(*auto simp add*: *EndPhase0-def*)
qed

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

89

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

theorem *HEndPhase0-HInv4d*:
⟦ *HEndPhase0 s s′ p*; *HInv4d s q*;
   Inv2c s; *Inv1 s*⟧ ⟹ *HInv4d s′ q*
by(*blast dest*: *HEndPhase0-HInv4d-p HEndPhase0-HInv4d-q*)

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

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

auto intro: *HPhase1or2Write-HInv4c*,
          auto simp add: *Phase1or2Read-def*
              intro: *HPhase1or2ReadThen-HInv4c*
                      HPhase1or2ReadElse-HInv4c,
          auto simp add: *EndPhase1or2-def*
              intro: *HEndPhase1-HInv4c*
                      HEndPhase2-HInv4c,
          auto intro: *HFail-HInv4c*,
          auto intro: *HEndPhase0-HInv4c simp add*: *HInv1-def*)
      show *HInv4d s′ p* **using** *assms*
        by(*auto simp add*: *HInv4-def HNext-def Next-def*,
          auto simp add: *HInv2-def*
              intro: *HStartBallot-HInv4d*,
          auto intro: *HPhase0Read-HInv4d*,
          auto intro: *HPhase1or2Write-HInv4d*,
          auto simp add: *Phase1or2Read-def*
              intro: *HPhase1or2ReadThen-HInv4d*
                      HPhase1or2ReadElse-HInv4d,
          auto simp add: *EndPhase1or2-def*
              intro: *HEndPhase1-HInv4d*
                      HEndPhase2-HInv4d,
          auto intro: *HFail-HInv4d*,
          auto intro: *HEndPhase0-HInv4d simp add*: *HInv1-def*)
qed

end

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

C.5   Invariant 5

This invariant asserts that, if a processor p is in phase 2, then either its
bal and *inp* values satisfy *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* $\Rightarrow$ *nat* $\Rightarrow$ *InputsOrNi* $\Rightarrow$ *bool*
  where *maxBalInp s b v* = ($\forall$ *bk*$\in$*allBlocks s*. *b* $\leq$ *bal bk* $\longrightarrow$ *inp bk* = *v*)

definition *HInv5-inner-R* :: *state* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  HInv5-inner-R s p =
      (*maxBalInp s* (*bal*(*dblock s p*)) (*inp*(*dblock s p*))
      $\vee$ ($\exists$ *D*$\in$*MajoritySet*. $\exists$ *q*. ($\forall$ *d*$\in$*D*. *bal*(*dblock s p*) < *mbal*(*disk s d q*)
                          $\wedge$ ¬*hasRead s p d q*)))

definition *HInv5-inner* :: *state* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*

where *HInv5-inner s p = (phase s p = 2 $\longrightarrow$ HInv5-inner-R s p)*

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

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

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

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

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

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

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

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

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

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

lemma HStartBallot-HInv5-q2:
  assumes act: HStartBallot s s′ p
  and pnq: p≠q
  and inv5-2: ∃ D∈MajoritySet. ∃ qq. (∀ d∈D.    bal(dblock s q) < mbal(disk s d
qq)
                                ∧ ¬hasRead s q d qq)
  shows ∃ D∈MajoritySet. ∃ qq. (∀ d∈D.    bal(dblock s′ q) < mbal(disk s′ d qq)
                                ∧ ¬hasRead s′ q d qq)
proof −
  from act pnq
  have   disk: disk s′ = disk s
```

and *blocksRead*: ∀ *d*. *blocksRead s′ q d* = *blocksRead s q d*
  and *dblock*: *dblock s′ q* = *dblock s q*
  by(*auto simp add*: *StartBallot-def InitializePhase-def*)
 with *inv5-2*
 show *?thesis*
  by(*auto simp add*: *hasRead-def*)
qed

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

theorem *HStartBallot-HInv5*:
 ⟦ *HStartBallot s s′ p*; *HInv5-inner s q* ⟧ ⟹ *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*: *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*)
 have *pnq*: *p≠q*
 proof −
  from *inv4c phase*
  obtain *D1* **where** *r1*: *IsMajority D1* ∧ (∀ *d*∈*D1*. *mbal*(*disk s d p*) = *bal* (*dblock s p*))

$\quad$ **by**(*auto simp add*: *HInv4c-def MajoritySet-def*)
$\quad$ **with** *i1 majorities-intersect*
$\quad$ **have** *D∩D1≠{}* **by** *auto*
$\quad$ **then obtain** *dd* **where** *dd∈D∩D1*
$\qquad$ **by** *auto*
$\quad$ **with** *i1 i2 r1*
$\quad$ **have** *bal*(*dblock s p*) < *mbal*(*disk s dd q*) ∧ *mbal*(*disk s dd p*) = *bal* (*dblock s p*)
$\qquad$ **by** *auto*
$\quad$ **thus** *?thesis* **by** *auto*
$\quad$ **qed**
$\quad$ **from** *act pnq*
$\qquad$ — *dblock* and *hasRead* do not change
$\quad$ **have** *dblock s′ = dblock s*
$\quad\quad$ **and** ∀ *d. hasRead s′ p d q = hasRead s p d q*
$\qquad$ — In all disks *q* blocks don't change
$\quad\quad$ **and** ∀ *d. disk s′ d q = disk s d q*
$\quad\quad$ **by**(*auto simp add*: *Phase1or2Write-def hasRead-def*)
$\quad$ **with** *i2 i1 i3 majority-nonempty*
$\quad$ **have** ∀ *d∈D. bal* (*dblock s′ p*) < *mbal* (*disk s′ d q*) ∧ ¬*hasRead s′ p d q*
$\qquad$ **by** *auto*
$\quad$ **with** *i1*
$\quad$ **show** *?thesis*
$\qquad$ **by**(*auto simp add*: *MajoritySet-def*)
qed

lemma *HPhase1or2Write-HInv5-p*:
$\quad$ **assumes** *act*: *HPhase1or2Write s s′ p d*
$\quad$ **and** *inv*: *HInv5-inner s p*
$\quad$ **and** *inv4*: *HInv4c s p*
$\quad$ **shows** *HInv5-inner s′ p*
proof(*auto simp add*: *HInv5-inner-def HInv5-inner-R-def*)
$\quad$ **assume** *phase′*: *phase s′ p = 2*
$\quad\quad$ **and** *i2*: ∀ *D∈MajoritySet*. ∀ *q*. ∃ *d∈D. bal* (*dblock s′ p*) < *mbal* (*disk s′ d q*)
$\longrightarrow$ *hasRead s′ p d q*
$\quad$ **with** *act* **have** *phase*: *phase s p = 2*
$\qquad$ **by**(*auto simp add*: *Phase1or2Write-def*)
$\quad$ **show** *maxBalInp s′* (*bal* (*dblock s′ p*)) (*inp* (*dblock s′ p*))
$\quad$ **proof**(*rule HPhase1or2Write-HInv5-1*[*OF act, of p*])
$\quad\quad$ **from** *HPhase1or2Write-HInv5-p2*[*OF act inv4 phase*] *inv i2 phase*
$\quad\quad$ **show** *maxBalInp s* (*bal* (*dblock s p*)) (*inp* (*dblock s p*))
$\qquad$ **by**(*auto simp add*: *HInv5-inner-def HInv5-inner-R-def*, *blast*)
$\quad$ **qed**
qed

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

lemma *HPhase1or2Write-HInv5-q2*:
  assumes *act*: *HPhase1or2Write s s′ p d*
  and *pnq*: *p≠q*
  and *inv4a*: *HInv4a s p*
   and *inv5-2*: *∃ D∈MajoritySet. ∃ qq. (∀ d∈D.   bal(dblock s q) < mbal(disk s d qq)*
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*: $p{\neq}q$
  shows *HInv5-inner s′ q*
proof(*auto simp add*: *HInv5-inner-def HInv5-inner-R-def*)
  assume *phase′*: *phase s′ q = 2*
    and *i2*: $\forall\, D{\in}MajoritySet.\ \forall\, qa.\ \exists\, d{\in}D.\ bal\ (dblock\ s′\ q) < mbal\ (disk\ s′\ d\ qa)$
$\longrightarrow$ *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*:
  $\llbracket$ *HPhase1or2Write s s′ p d*; *HInv5-inner s q*;
    HInv4c s p; *HInv4a s p* $\rrbracket \Longrightarrow$ *HInv5-inner s′ q*
  by(*blast dest*: *HPhase1or2Write-HInv5-q HPhase1or2Write-HInv5-p*)

lemma *HPhase1or2ReadThen-HInv5-1*:
  assumes *act*: *HPhase1or2ReadThen s s′ p d r*
  and *inv5-1*: *maxBalInp s (bal(dblock s q)) (inp(dblock s q))*
  shows *maxBalInp s′ (bal(dblock s′ q)) (inp(dblock s′ q))*
  using *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*: $\exists\, D{\in}MajoritySet.\ \exists\, q.\ (\forall\, d{\in}D.\quad bal(dblock\ s\ p) < mbal(disk\ s\ d\ q)$
                          $\wedge\ \neg hasRead\ s\ p\ d\ q)$
  shows $\exists\, D{\in}MajoritySet.\ \exists\, q.\ (\forall\, d{\in}D.\quad bal(dblock\ s′\ p) < mbal(disk\ s′\ d\ q)$
                          $\wedge\ \neg hasRead\ s′\ p\ d\ q)$
proof −
  from *inv5-2*
  obtain *D q*
    where *i1*: *IsMajority D*
      and *i2*: $\forall\, d{\in}D.\ bal(dblock\ s\ p) < mbal(disk\ s\ d\ q)$
      and *i3*: $\forall\, d{\in}D.\ \neg hasRead\ s\ p\ d\ q$
    by(*auto simp add*: *MajoritySet-def*)
  from *inv2c phase*

97

have *bal*(*dblock s p*)=*mbal*(*dblock s p*)
 by(*auto simp add*: *Inv2c-inner-def*)
moreover
from *act* **have** *mbal* (*disk s d r*) < *mbal* (*dblock s p*)
 by(*auto simp add*: *Phase1or2ReadThen-def*)
moreover
from *i2* **have** $d \in D \longrightarrow bal(dblock\ s\ p) < mbal(disk\ s\ d\ q)$ **by** *auto*
ultimately have *pnr*: $d \in D \longrightarrow q \neq r$ **by** *auto*
have *pnq*: $p \neq q$
proof −
 from *inv4c phase*
 obtain *D1* **where** *r1*: *IsMajority D1* $\wedge$ ($\forall\ d \in D1$. *mbal*(*disk s d p*) = *bal* (*dblock*
s p))
  by(*auto simp add*: *HInv4c-def MajoritySet-def*)
 with *i1 majorities-intersect*
 have $D \cap D1 \neq \{\}$ **by** *auto*
 then obtain *dd* **where** $dd \in D \cap D1$
  by *auto*
 with *i1 i2 r1*
 have *bal*(*dblock s p*) < *mbal*(*disk s dd q*) $\wedge$ *mbal*(*disk s dd p*) = *bal* (*dblock s p*)
  by *auto*
 thus *?thesis* **by** *auto*
qed
from *pnr act*
have *hasRead′*: $\forall\ d \in D$. *hasRead s′ p d q* = *hasRead s p d q*
 by(*auto simp add*: *Phase1or2ReadThen-def hasRead-def*)
from *act pnq*
  — *dblock* and *disk* do not change
have *dblock s′* = *dblock s*
 and $\forall\ d$. *disk s′* = *disk s*
 by(*auto simp add*: *Phase1or2ReadThen-def*)
with *i2 hasRead′ i3*
have $\forall\ d \in D$. *bal* (*dblock s′ p*) < *mbal* (*disk s′ d q*) $\wedge$ ¬*hasRead s′ p d q*
 by *auto*
with *i1*
show *?thesis*
 by(*auto simp add*: *MajoritySet-def*)
qed

lemma *HPhase1or2ReadThen-HInv5-p*:
 assumes *act*: *HPhase1or2ReadThen s s′ p d r*
 and *inv*: *HInv5-inner s p*
 and *inv4*: *HInv4c s p*
 and *inv2c*: *Inv2c s*
 shows *HInv5-inner s′ p*
proof(*auto simp add*: *HInv5-inner-def HInv5-inner-R-def*)
 assume *phase′*: *phase s′ p = 2*
  and *i2*: $\forall\ D \in MajoritySet$. $\forall\ q$. $\exists\ d \in D$. *bal* (*dblock s′ p*) < *mbal* (*disk s′ d q*)
$\longrightarrow$ *hasRead s′ p d q*

with *act* **have** *phase*: *phase s p = 2*
   by(*auto simp add*: *Phase1or2ReadThen-def*)
 show *maxBalInp s′ (bal (dblock s′ p)) (inp (dblock s′ p))*
 proof(*rule HPhase1or2ReadThen-HInv5-1*[*OF act, of p*])
   from *inv2c*
   have *Inv2c-inner s p* **by**(*auto simp add*: *Inv2c-def*)
   from *HPhase1or2ReadThen-HInv5-p2*[*OF act inv4 this phase*] *inv i2 phase*
   show *maxBalInp s (bal (dblock s p)) (inp (dblock s p))*
     by(*auto simp add*: *HInv5-inner-def HInv5-inner-R-def, blast*)
 qed
qed

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

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

lemma *HPhase1or2ReadThen-HInv5-q*:

99

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

theorem *HPhase1or2ReadThen-HInv5*:
  ⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv5-inner s q*;
    Inv2c s; *HInv4c s p*; *HInv4a s p* ⟧ ⟹ *HInv5-inner s′ q*
  by(*blast dest*: *HPhase1or2ReadThen-HInv5-q HPhase1or2ReadThen-HInv5-p*)

theorem *HPhase1or2ReadElse-HInv5*:
  ⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv5-inner s q* ⟧
    ⟹ *HInv5-inner s′ q*
  using *HStartBallot-HInv5*
  by(*auto simp add*: *Phase1or2ReadElse-def*)

lemma *HEndPhase2-HInv5-p*:
  HEndPhase2 s s′ p ⟹ *HInv5-inner s′ p*
  by(*auto simp add*: *EndPhase2-def HInv5-inner-def*)

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

lemma *HEndPhase2-HInv5-q1*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *pnq*: *p≠q*
  and *inv5-1*: *maxBalInp s (bal(dblock s q)) (inp(dblock s q))*
  shows *maxBalInp s′ (bal(dblock s′ q)) (inp(dblock s′ q))*
proof(*auto simp add*: *maxBalInp-def*)
  fix *bk*
  assume *bk*: *bk ∈ allBlocks s′*
    and *bal*: *bal (dblock s′ q) ≤ bal bk*

100

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*: $p \neq q$
  and *inv5-2*: $\exists D \in MajoritySet. \ \exists qq. \ (\forall d \in D. \quad bal(dblock\ s\ q) < mbal(disk\ s\ d\ qq)$
$\wedge \neg hasRead\ s\ q\ d\ qq)$
  shows $\exists D \in MajoritySet. \ \exists qq. \ (\forall d \in D. \quad bal(dblock\ s'\ q) < mbal(disk\ s'\ d\ qq)$
$\wedge \neg hasRead\ s'\ q\ d\ qq)$
proof −
  from *act pnq*
  have *disk*: *disk s' = disk s*
    and *blocksRead*: $\forall d.\ blocksRead\ s'\ q\ d = blocksRead\ s\ q\ d$
    and *dblock*: *dblock s' q = dblock s q*
    by(*auto simp add: EndPhase2-def InitializePhase-def*)
  with *inv5-2*
  show *?thesis*
    by(*auto simp add: hasRead-def*)
qed

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

theorem *HEndPhase2-HInv5*:
  ⟦ *HEndPhase2 s s' p*; *HInv5-inner s q* ⟧ ⟹ *HInv5-inner s' q*
  by(*blast dest: HEndPhase2-HInv5-q HEndPhase2-HInv5-p*)

lemma *HEndPhase1-HInv5-p*:
  assumes *act*: *HEndPhase1 s s' p*
  and *inv4*: *HInv4 s*
  and *inv2a*: *Inv2a s*
  and *inv2a'*: *Inv2a s'*
  and *inv2c*: *Inv2c s*
  and *asm4*: $\neg maxBalInp\ s'\ (bal(dblock\ s'\ p))\ (inp(dblock\ s'\ p))$
  shows $(\exists D \in MajoritySet. \ \exists q. \ (\forall d \in D. \quad bal(dblock\ s'\ p) < mbal(disk\ s'\ d\ q)$
$\wedge \neg hasRead\ s'\ p\ d\ q))$
proof −

have $\exists\, bk \in allBlocks\ s.\ bal(dblock\ s'\ p) \leq bal\ bk \wedge bk \neq dblock\ s'\ p$
proof $-$
  from *asm4*
  obtain *bk*
    where *p31*: $bk \in allBlocks\ s' \wedge\ \ bal(dblock\ s'\ p) \leq bal\ bk \wedge bk \neq dblock\ s'\ p$
    by(*auto simp add*: *maxBalInp-def*)
  then obtain *q* **where** *p32*: $bk \in blocksOf\ s'\ q$
    by(*auto simp add*: *allBlocks-def*)
  from *act*
  have *dblock*: $p \neq q \implies dblock\ s'\ q = dblock\ s\ q$
    by(*auto simp add*: *EndPhase1-def*)
  have $bk \in blocksOf\ s\ q$
  proof(*cases p=q*)
    case *True*
    with *p32 p31 HEndPhase1-blocksOf*[*OF act*]
    show *?thesis*
      by *auto*
    next
      case *False*
      from *dblock*[*OF False*] *subsetD*[*OF HEndPhase1-blocksOf*[*OF act, of q*] *p32*]
      show *?thesis*
        by(*auto simp add*: *blocksOf-def*)
  qed
  with *p31*
  show *?thesis*
  by(*auto simp add*: *allBlocks-def*)
qed
then obtain *bk* **where** *p22*: $bk \in allBlocks\ s \wedge bal\ (dblock\ s'\ p) \leq bal\ bk \wedge bk \neq$
$dblock\ s'\ p$ **by** *auto*
 have $\exists\, q \in UNIV - \{p\}.\ bk \in blocksOf\ s\ q$
 proof $-$
  from *p22*
  obtain *q* **where** *bk*: $bk \in blocksOf\ s\ q$
    by(*auto simp add*: *allBlocks-def*)
  from *act p22*
  have $mbal(dblock\ s\ p) \leq bal\ bk$
    by(*auto simp add*: *EndPhase1-def*)
  moreover
  from *act*
  have *phase s p = 1*
    by(*auto simp add*: *EndPhase1-def*)
  moreover
  from *inv4*
  have *HInv4b s p* **by**(*auto simp add*: *HInv4-def*)
  ultimately
  have $p \neq q$
    using *bk*
    by(*auto simp add*: *HInv4-def HInv4b-def*)
  with *bk*

show *?thesis*
  by *auto*
qed
then obtain *q* **where** *p23*: *q*∈*UNIV*−{*p*} ∧ *bk* ∈ *blocksOf s q*
  by *auto*
have ∃ *D*∈*MajoritySet*.∀ *d*∈*D*. *bal*(*dblock s′ p*) ≤ *mbal*(*disk s d q*)
proof −
  from *p23 inv4*
  have *i4d*: ∃ *D*∈*MajoritySet*.∀ *d*∈*D*. *bal bk* ≤ *mbal*(*disk s d q*)
    by(*auto simp add*: *HInv4-def HInv4d-def*)
  from *i4d p22*
  show *?thesis*
    by *force*
qed
then obtain *D* **where** *Dmaj*: *D*∈*MajoritySet* **and** *p24*:(∀ *d*∈*D*. *bal*(*dblock s′ p*)
≤ *mbal*(*disk s d q*))
  by *auto*
have *p25*: ∀ *d*∈*D*. *bal*(*dblock s′ p*) < *mbal*(*disk s d q*)
proof −
  from *inv2c*
  have *Inv2c-inner s p*
    by(*auto simp add*: *Inv2c-def*)
  with *act*
  have *bal-pos*: *0* < *bal*(*dblock s′ p*)
    by(*auto simp add*: *Inv2c-inner-def EndPhase1-def*)
  with *inv2a′*
  have *bal*(*dblock s′ p*) ∈ *Ballot p* ∪ {*0*}
    by(*auto simp add*: *Inv2a-def Inv2a-inner-def*
                Inv2a-innermost-def blocksOf-def)
  with *bal-pos* **have** *bal-in-p*: *bal*(*dblock s′ p*) ∈ *Ballot p*
    by *auto*
  from *inv2a* **have** *Inv2a-inner s q* **by**(*auto simp add*: *Inv2a-def*)
  hence ∀ *d*∈*D*. *mbal*(*disk s d q*) ∈ *Ballot q* ∪ {*0*}
    by(*auto simp add*: *Inv2a-inner-def Inv2a-innermost-def*
                blocksOf-def)
  with *p24 bal-pos*
  have ∀ *d*∈*D*. *mbal*(*disk s d q*) ∈ *Ballot q*
    by *force*
  with *Ballot-disj p23 bal-in-p*
  have ∀ *d*∈*D*. *mbal*(*disk s d q*)≠ *bal*(*dblock s′ p*)
    by *force*
  with *p23 p24*
  show *?thesis*
    by *force*
qed
with *p23 act*
have ∀ *d*∈*D*. *bal*(*dblock s′ p*) < *mbal*(*disk s′ d q*) ∧ ¬*hasRead s′ p d q*
  by(*auto simp add*: *EndPhase1-def InitializePhase-def hasRead-def*)
with *Dmaj*

show *?thesis*
  by *blast*
qed

lemma *union-inclusion*:
 ⟦ $A \subseteq A'$; $B \subseteq B'$ ⟧ $\implies A \cup B \subseteq A' \cup B'$
by *blast*

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

lemma *HEndPhase1-allBlocks*:
  assumes *act*: *HEndPhase1 s s′ p*
  shows *allBlocks s′ $\subseteq$ allBlocks s $\cup$ {dblock s′ p}*
proof(*auto simp del*: *HEndPhase1-def simp add*: *allBlocks-def*
       dest: *HEndPhase1-blocksOf-q*[*OF act*])
  fix *x pa*
  assume *x-pa*: *x $\in$ blocksOf s′ pa* **and**
     x-nblks: *∀ 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*]

104

```
      show ?thesis
        by auto
    next
      case False
      from x-nblks subsetD[OF HEndPhase1-blocksOf-q[OF act False] x-pa]
      show ?thesis
        by auto
    qed
qed

lemma HEndPhase1-HInv5-q:
  assumes act: HEndPhase1 s s' p
  and inv: HInv5 s
  and inv1: Inv1 s
  and inv2a: Inv2a s'
  and inv2a-q: Inv2a s
  and inv2b: Inv2b s
  and inv2c: Inv2c s
  and inv3: HInv3 s
  and phase': phase s' q = 2
  and pnq: p≠q
  and asm4: ¬maxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
  shows (∃ D∈MajoritySet. ∃ qq. (∀ d∈D.   bal(dblock s' q) < mbal(disk s' d qq)
                                  ∧ ¬hasRead s' q d qq))
proof −
  from act pnq
  have phase s' q = phase s q
    and phase-p: phase s p = 1
    and disk: disk s' = disk s
    and dblock: dblock s' q = dblock s q
    and bal: bal(dblock s' p) = mbal(dblock s p)
    by(auto simp add: EndPhase1-def InitializePhase-def)
  with phase'
  have phase: phase s q = 2 by auto
  from phase inv2c
  have bal-dblk-q: bal(dblock s q) ∈ Ballot q
    by(auto simp add: Inv2c-def Inv2c-inner-def)
  have ∃ D∈MajoritySet. ∃ qq. (∀ d∈D.   bal(dblock s q) < mbal(disk s d qq)
                                  ∧ ¬hasRead s q d qq)
  proof(cases maxBalInp s (bal(dblock s q)) (inp(dblock s q)))
    case True
    have p21: bal(dblock s q) < bal(dblock s' p) ∧ inp(dblock s q) ≠ inp(dblock s'
p)
    proof −
      from True asm4 dblock HEndPhase1-allBlocks[OF act]
      have p32:   bal(dblock s q)≤ bal(dblock s' p)
              ∧ inp(dblock s q) ≠ inp(dblock s' p)
        by(auto simp add: maxBalInp-def)
      from inv2a
```

105

have $bal(dblock\ s'\ p) \in Ballot\ p \cup \{0\}$
 by(*auto simp add: Inv2a-def Inv2a-inner-def*
      Inv2a-innermost-def blocksOf-def)
moreover
from *Ballot-disj Ballot-nzero pnq*
have $Ballot\ q \cap (Ballot\ p \cup \{0\}) = \{\}$
 by *auto*
ultimately
have $bal(dblock\ s'\ p) \neq bal(dblock\ s\ q)$
 using *bal-dblk-q*
 by *auto*
with *p32*
show *?thesis*
 by *auto*
qed
have $\exists D \in MajoritySet. \forall d \in D.\ bal(dblock\ s\ q) < mbal(disk\ s\ d\ p) \wedge hasRead\ s$
$p\ d\ q$
 proof −
 from *act*
 have $\exists D \in MajoritySet. \forall d \in D.\ d \in disksWritten\ s\ p \wedge (\forall q \in UNIV - \{p\}.\ has$-
$Read\ s\ p\ d\ q)$
  by(*auto simp add: EndPhase1-def MajoritySet-def*)
 then obtain D
  where $act1: \forall d \in D.\ d \in disksWritten\ s\ p \wedge (\forall q \in UNIV - \{p\}.\ hasRead\ s\ p\ d$
$q)$
  and $Dmaj: D \in MajoritySet$
  by *auto*
 from *inv2b*
 have $\forall d.\ Inv2b\text{-}inner\ s\ p\ d$ **by**(*auto simp add: Inv2b-def*)
 with *act1 pnq phase-p bal*
 have $\forall d \in D.\ bal(dblock\ s'\ p) = mbal(disk\ s\ d\ p) \wedge hasRead\ s\ p\ d\ q$
  by(*auto simp add: Inv2b-def Inv2b-inner-def*)
 with *p21 Dmaj*
 have $\forall d \in D.\ bal(dblock\ s\ q) < mbal(disk\ s\ d\ p) \wedge hasRead\ s\ p\ d\ q$
  by *auto*
 with *Dmaj*
 show *?thesis*
  by *auto*
 qed
 then obtain D
  where $p22: D \in MajoritySet \wedge (\forall d \in D.\ bal(dblock\ s\ q) < mbal(disk\ s\ d\ p) \wedge$
$hasRead\ s\ p\ d\ q)$
  by *auto*
 have $p23: \forall d \in D.(\!|block = dblock\ s\ q,\ proc = q|\!) \notin blocksRead\ s\ p\ d$
 proof −
  have $dblock\ s\ q \in allBlocksRead\ s\ p \longrightarrow inp(dblock\ s'\ p) = inp(dblock\ s\ q)$
  proof *auto*
   assume $dblock\text{-}q: dblock\ s\ q \in allBlocksRead\ s\ p$
   from *inv2a-q*

106

have $(bal(dblock\ s\ q)=0) = (inp\ (dblock\ s\ q) = NotAnInput)$
  by(*auto simp add: Inv2a-def Inv2a-inner-def*
                  blocksOf-def Inv2a-innermost-def)
with *bal-dblk-q Ballot-nzero dblock-q InputsOrNi*
have *dblock-q-nib*: $dblock\ s\ q \in nonInitBlks\ s\ p$
  by(*auto simp add: nonInitBlks-def blocksSeen-def*)
with *act*
have *dblock-max*: $inp(dblock\ s'\ p)=inp(maxBlk\ s\ p)$
  by(*auto simp add: EndPhase1-def*)
from *maxBlk-in-nonInitBlks*[*OF dblock-q-nib inv1*]
have *max-in-nib*: $maxBlk\ s\ p \in nonInitBlks\ s\ p$ **..**
hence $nonInitBlks\ s\ p \subseteq allBlocks\ s$
  by(*auto simp add: allBlocks-def nonInitBlks-def*
                  blocksSeen-def blocksOf-def rdBy-def
                  allBlocksRead-def allRdBlks-def)
with *True subsetD*[*OF this max-in-nib*]
have $bal\ (dblock\ s\ q) \le bal\ (maxBlk\ s\ p) \longrightarrow inp\ (maxBlk\ s\ p) = inp\ (dblock\ s\ q)$

  by(*auto simp add: maxBalInp-def*)
with *maxBlk-in-nonInitBlks*[*OF dblock-q-nib inv1*]
  dblock-q-nib dblock-max
show $inp(dblock\ s'\ p) = inp(dblock\ s\ q)$
  by *auto*
qed
with *p21*
have $dblock\ s\ q \notin block\ `\ allRdBlks\ s\ p$
  by(*auto simp add: allBlocksRead-def*)
hence $\forall\ d.\ dblock\ s\ q \notin block\ `\ blocksRead\ s\ p\ d$
  by(*auto simp add: allRdBlks-def*)
thus *?thesis*
  by *force*
qed
have *p24*: $\forall\ d{\in}D.\ \neg\ (\exists\ br{\in}\ blocksRead\ s\ q\ d.\ bal(dblock\ s\ q) \le mbal\ (block\ br))$
proof $-$
  from *inv2c phase*
  have $\forall\ d.\ \forall\ br{\in}blocksRead\ s\ q\ d.\ mbal(block\ br){<}mbal(dblock\ s\ q)$
    and $bal(dblock\ s\ q) = mbal(dblock\ s\ q)$
    by(*auto simp add: Inv2c-def Inv2c-inner-def*)
  thus *?thesis*
    by *force*
qed
have *p25*: $\forall\ d{\in}D.\ \neg hasRead\ s\ q\ d\ p$
proof *auto*
  fix *d*
  assume *d-in-D*: $d \in D$
    and *hasRead-qdp*: $hasRead\ s\ q\ d\ p$
  have *p31*: $(\!|block{=}dblock\ s\ p,\ proc{=}p|\!){\in}blocksRead\ s\ q\ d$
  proof $-$
    from *d-in-D p22*

have *hasRead-pdq*: *hasRead s p d q* **by** *auto*
with *hasRead-qdp phase phase-p inv3*
have *HInv3-R s q p d*
  by(*auto simp add*: *HInv3-def HInv3-inner-def HInv3-L-def*)
with *p23 d-in-D*
show *?thesis*
  by(*auto simp add*: *HInv3-R-def*)
qed
from *p21 act*
have *p32*: *bal*(*dblock s q*) < *mbal*(*dblock s p*)
  by(*auto simp add*: *EndPhase1-def*)
with *p31 d-in-D hasRead-qdp p24*
show *False*
  by(*force*)
qed
with *p22*
show *?thesis*
  by *auto*
next
case *False*
with *inv phase*
show *?thesis*
  by(*auto simp add*: *HInv5-def HInv5-inner-def HInv5-inner-R-def*)
qed
then obtain *D qq*
  where *D*∈*MajoritySet* ∧ (∀ *d*∈*D*.   *bal*(*dblock s q*) < *mbal*(*disk s d qq*)
                                   ∧ ¬*hasRead s q d qq*)
  by *auto*
moreover
from *act pnq*
have ∀ *d*. *hasRead s' q d qq* = *hasRead s q d qq*
  by(*auto simp add*: *EndPhase1-def InitializePhase-def hasRead-def*)
ultimately show *?thesis*
  using *disk dblock*
  by *auto*
qed

theorem *HEndPhase1-HInv5*:
  assumes *act*: *HEndPhase1 s s' p*
  and *inv*: *HInv5 s*
  and *inv1*: *Inv1 s*
  and *inv2a*: *Inv2a s*
  and *inv2a'*: *Inv2a s'*
  and *inv2b*: *Inv2b s*
  and *inv2c*: *Inv2c s*
  and *inv3*: *HInv3 s*
  and *inv4*: *HInv4 s*
shows *HInv5-inner s' q*
  using *HEndPhase1-HInv5-p*[*OF act inv4 inv2a inv2a' inv2c*]

HEndPhase1-HInv5-q[*OF act inv inv1 inv2a′ inv2a inv2b inv2c inv3, of q*]
by(*auto simp add: HInv5-def HInv5-inner-def HInv5-inner-R-def*)

lemma *HFail-HInv5-p*:
  HFail s s′ p $\implies$ *HInv5-inner s′ p*
  by(*auto simp add: Fail-def HInv5-inner-def*)

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

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

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

from *act pnq*
have *dblock′*: *dblock s′ q = dblock s q* **by**(*auto simp add: Fail-def*)
from *subsetD[OF HFail-allBlocks[OF act] bk]*
show *inp bk = inp (dblock s′ q)*
proof
  assume *bk*: *bk ∈ 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*
    by(*auto simp add: Fail-def*)
  with *bal*
  have *bal (dblock s′ q)=0*
    by(*auto simp add: InitDB-def*)
  with *inv2a*
  have *inp (dblock s′ q)= NotAnInput*
    by(*auto simp add: Inv2a-inner-def Inv2a-innermost-def blocksOf-def*)
  with *bk-init*
  show *?thesis*
    by(*auto simp add: InitDB-def*)
qed
qed

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

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

shows *HInv5-inner s′ q*
proof(*auto simp add: HInv5-inner-def HInv5-inner-R-def*)
  assume *phase′*: *phase s′ q = 2*
    and *nR2*: ∀ *D*∈*MajoritySet*.
      ∀ *qa*. ∃ *d*∈*D*. *bal* (*dblock s′ q*) < *mbal* (*disk s′ d qa*) ⟶
             hasRead s′ q d qa (**is** *?P s′*)
  from *HFail-HInv5-q2*[*OF act pnq*]
  have ¬ (*?P s*) ⟹ ¬(*?P s′*)
    by *auto*
  with *nR2*
  have *P*: *?P s*
    by *blast*
  from *inv2a*
  have *inv2a′*: *Inv2a-inner s′ q* **by** (*auto simp add: Inv2a-def*)
  from *act pnq phase′*
  have *phase s q = 2*
    by(*auto simp add: Fail-def split: if-split-asm*)
  with *inv HFail-HInv5-q1*[*OF act pnq inv2a′*] *P*
  show *maxBalInp s′ (bal (dblock s′ q)) (inp (dblock s′ q))*
    by(*auto simp add: HInv5-inner-def HInv5-inner-R-def*)
qed

theorem *HFail-HInv5*:
  ⟦ *HFail s s′ p*; *HInv5-inner s q*; *Inv2a s′* ⟧ ⟹ *HInv5-inner s′ q*
by(*blast dest: HFail-HInv5-q HFail-HInv5-p*)

lemma *HPhase0Read-HInv5-p*:
  HPhase0Read s s′ p d ⟹ *HInv5-inner s′ p*
  by(*auto simp add: Phase0Read-def HInv5-inner-def*)

lemma *HPhase0Read-allBlocks*:
  assumes *act*: *HPhase0Read s s′ p d*
  shows *allBlocks s′ ⊆ allBlocks s*
  using *HPhase0Read-blocksOf*[*OF act*]
  by(*auto simp add: allBlocks-def*)

lemma *HPhase0Read-HInv5-1*:
  assumes *act*: *HPhase0Read s s′ p d*
  and *inv5-1*: *maxBalInp s (bal(dblock s q)) (inp(dblock s q))*
  shows *maxBalInp s′ (bal(dblock s′ q)) (inp(dblock s′ q))*
  using *assms* **and** *HPhase0Read-blocksOf*[*OF act*]
  by(*auto simp add: Phase0Read-def maxBalInp-def allBlocks-def*)

lemma *HPhase0Read-HInv5-q2*:
  assumes *act*: *HPhase0Read s s′ p d*
  and *pnq*: *p≠q*
  and *inv5-2*: ∃ *D*∈*MajoritySet*. ∃ *qq*. (∀ *d*∈*D*.   *bal(dblock s q) < mbal(disk s d qq*)

                      ∧ ¬*hasRead s q d qq*)

111

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

proof −
  from *act pnq*
  have *disk*: *disk s' = disk s*
    and *blocksRead*: $\forall\, d.\ blocksRead\ s'\ q\ d = blocksRead\ s\ q\ d$
    and *dblock*: *dblock s' q = dblock s q*
    by(*auto simp add*: *Phase0Read-def InitializePhase-def*)
  with *inv5-2*
  show *?thesis*
    by(*auto simp add*: *hasRead-def*)
qed

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

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

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

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

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

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

113

shows *maxBalInp s′ (bal(dblock s′ q)) (inp(dblock s′ q))*
proof(*auto simp add: maxBalInp-def*)
  fix *bk*
  assume *bk*: *bk ∈ allBlocks s′*
    and *bal*: *bal (dblock s′ q) ≤ bal bk*
  from *act pnq*
  have *dblock′*: *dblock s′ q = dblock s q* **by**(*auto simp add: EndPhase0-def*)
  from *subsetD[OF HEndPhase0-allBlocks[OF act] bk]*
  show *inp bk = inp (dblock s′ q)*
  proof
    assume *bk*: *bk ∈ allBlocks s*
    with *inv5-1 dblock′ bal*
    show *?thesis*
      by(*auto simp add: maxBalInp-def*)
  next
    assume *bk*: *bk ∈ {dblock s′ p}*
    with *HEndPhase0-some[OF act inv1] act*
    have *∃ ba∈allBlocksRead s p. bal ba = bal (dblock s′ p) ∧ inp ba = inp (dblock s′ p)*
      by(*auto simp add: EndPhase0-def*)
    then obtain *ba*
      where *ba-blksread*: *ba∈allBlocksRead s p*
      and *ba-balinp*: *bal ba = bal (dblock s′ p) ∧ inp ba = inp (dblock s′ p)*
      by *auto*
    have *allBlocksRead s p ⊆ allBlocks s*
      by(*auto simp add: allBlocksRead-def allRdBlks-def*
              allBlocks-def blocksOf-def rdBy-def)
    from *subsetD[OF this ba-blksread] ba-balinp bal bk dblock′ inv5-1*
    show *?thesis*
      by(*auto simp add: maxBalInp-def*)
  qed
qed

lemma *HEndPhase0-HInv5-q2*:
  assumes *act*: *HEndPhase0 s s′ p*
  and *pnq*: *p≠q*
  and *inv5-2*: *∃ D∈MajoritySet. ∃ qq. (∀ d∈D.   bal(dblock s 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: EndPhase0-def InitializePhase-def*)
  with *inv5-2*
  show *?thesis*

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

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

theorem *HEndPhase0-HInv5*:
  ⟦ *HEndPhase0 s s′ p*; *HInv5-inner s q*; *Inv1 s* ⟧ ⟹ *HInv5-inner s′ q*
  by(*blast dest*: *HEndPhase0-HInv5-q HEndPhase0-HInv5-p*)

$HInv1 \wedge HInv2 \wedge HInv3 \wedge HInv4 \wedge HInv5$ is an invariant of $HNext$.

lemma *I2e*:
  assumes *nxt*: *HNext s s′*
  and *inv*: *HInv1 s ∧ HInv2 s ∧ HInv2 s′ ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s*
  shows *HInv5 s′*
  using *assms*
  by(*auto simp add*: *HInv5-def HNext-def Next-def*,
    auto simp add: *HInv2-def intro*: *HStartBallot-HInv5*,
    auto intro: *HPhase0Read-HInv5*,
    auto simp add: *HInv4-def intro*: *HPhase1or2Write-HInv5*,
    auto simp add: *Phase1or2Read-def*
      intro: *HPhase1or2ReadThen-HInv5*
        HPhase1or2ReadElse-HInv5,
    auto simp add: *EndPhase1or2-def HInv1-def HInv4-def HInv5-def*
      intro: *HEndPhase1-HInv5*
        HEndPhase2-HInv5,
    auto intro: *HFail-HInv5*,
    auto intro: *HEndPhase0-HInv5 simp add*: *HInv1-def*)

end

theory *DiskPaxos-Chosen* **imports** *DiskPaxos-Inv5* **begin**

C.6 Lemma I2f

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

definition *valueChosen* :: *state* $\Rightarrow$ *InputsOrNi* $\Rightarrow$ *bool*
where
  valueChosen s v =
  ($\exists\, b \in (\mathit{UN}\ p.\ \mathit{Ballot}\ p)$.
      maxBalInp s b v
    $\wedge$ ($\exists\, p.\ \exists\, D \in \mathit{MajoritySet}.(\forall\, d \in D.\ \ b \leq \mathit{bal}(\mathit{disk}\ s\ d\ p)$
                                  $\wedge (\forall\, q.(\quad \mathit{phase}\ s\ q = 1$
                                          $\wedge\ b \leq \mathit{mbal}(\mathit{dblock}\ s\ q)$
                                          $\wedge\ \mathit{hasRead}\ s\ q\ d\ p$
) $\longrightarrow (\exists\, br \in \mathit{blocksRead}\ s\ q\ d.\ b \leq \mathit{bal}(\mathit{block}\ br))$
                          )))))

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

116

    with *bk-noninit act*
    show *?thesis*
      by(*auto simp add*: *EndPhase1-def*)
qed

lemma *HEndPhase1-maxBalInp*:
  assumes *act*: *HEndPhase1 s s' q*
    and *asm1*: $b \in (UN\ p.\ Ballot\ p)$
    and *asm2*: $D \in MajoritySet$
    and *asm3*: *maxBalInp s b v*
    and *asm4*: $\forall d \in D.\ \ b \le bal(disk\ s\ d\ p)$
                  $\wedge (\forall q.(\ \ phase\ s\ q = 1$
                        $\wedge\ b \le mbal(dblock\ s\ q)$
                        $\wedge\ hasRead\ s\ q\ d\ p$
                      $) \longrightarrow (\exists br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$

    and *inv1*: *Inv1 s*
    and *inv2a*: *Inv2a s*
    and *inv2b*: *Inv2b s*
    shows *maxBalInp s' b v*
proof(*cases* $b \le mbal(dblock\ s\ q)$)
  case *True*
  show *?thesis*
  proof(*cases* $p \ne q$)
    assume *pnq*: $p \ne q$
    have $\exists d \in D.\ hasRead\ s\ q\ d\ p$
    proof −
      from *act*
      have *IsMajority*($\{d.\ d \in disksWritten\ s\ q \wedge (\forall r \in UNIV - \{q\}.\ hasRead\ s\ q\ d$
r)$\}$) (**is** *IsMajority*(*?M*))
        by(*auto simp add*: *EndPhase1-def*)
      with *majorities-intersect asm2*
      have $D \cap\ ?M \ne \{\}$
        by(*auto simp add*: *MajoritySet-def*)
      hence $\exists d \in D.\ (\forall r \in UNIV - \{q\}.\ hasRead\ s\ q\ d\ r)$
        by *auto*
      with *pnq*
      show *?thesis*
        by *auto*
    qed
    then obtain *d* **where** *p41*: $d \in D \wedge hasRead\ s\ q\ d\ p$ **by** *auto*
    with *asm4 asm3 act True*
    have *p42*: $\exists br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)$
      by(*auto simp add*: *EndPhase1-def*)
    from *True act*
    have *thesis-L*: $b \le bal\ (dblock\ s'\ q)$
      by(*auto simp add*: *EndPhase1-def*)
    from *p42*
    have $inp(dblock\ s'\ q) = v$

proof *auto*
  fix *br*
  assume *br*: *br* ∈ *blocksRead s q d*
    and *b-bal*: *b* ≤ *bal* (*block br*)
  hence *br-rdBy*: *br* ∈ (*UN q d. rdBy s* (*proc br*) *q d*)
    by(*auto simp add*: *rdBy-def*)
  hence *br-blksof*: *block br* ∈ *blocksOf s* (*proc br*)
    by(*auto simp add*: *blocksOf-def*)
  from *br* **have** *br-bseen*: *block br*∈ *blocksSeen s q*
    by(*auto simp add*: *blocksSeen-def allBlocksRead-def allRdBlks-def*)
  from *HEndPhase1-valueChosen-inp*[*OF act inv2a asm1 br-blksof br-bseen b-bal asm3 inv1*]
  show *?thesis* .
  qed
  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-inp*[*OF act inv2a asm1 db-blksof db-bseen b-bal asm3 inv1*]
    show *?thesis* .
  qed

118

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

lemma *HEndPhase1-valueChosen2*:
  assumes *act*: *HEndPhase1 s s′ q*
   and *asm4*: ∀ *d*∈*D*. *b* ≤ *bal*(*disk s d p*)
           ∧(∀ *q*.(*phase s q = 1*
                ∧ *b* ≤*mbal*(*dblock s q*)
                ∧ *hasRead s q d p*
) ⟶ (∃ *br*∈*blocksRead s q d*. *b* ≤ *bal*(*block br*))) (**is** *?P s*)
  shows *?P s′*
proof(*auto*)
  fix *d*
  assume *d*: *d*∈*D*
  with *act asm4*
  show *b* ≤ *bal* (*disk s′ d p*)
   by(*auto simp add*: *EndPhase1-def*)
  fix *d q*
  assume *d*: *d*∈*D*
   and *phase′*: *phase s′ q = Suc 0*

119

and *dblk-mbal*: $b \leq mbal\ (dblock\ s'\ q)$
with *act*
have *p31*: *phase s q = 1*
  and *p32*: *dblock s' q = dblock s q*
  by(*auto simp add*: *EndPhase1-def split*: *if-split-asm*)
with *dblk-mbal*
have $b \leq mbal(dblock\ s\ q)$ **by** *auto*
moreover
assume *hasRead*: *hasRead s' q d p*
with *act*
have *hasRead s q d p*
  by(*auto simp add*: *EndPhase1-def InitializePhase-def*
   hasRead-def split: *if-split-asm*)
ultimately
have $\exists\,br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)$
  using *p31 asm4 d*
  by *blast*
with *act hasRead*
show $\exists\,br \in blocksRead\ s'\ q\ d.\ b \leq bal(block\ br)$
  by(*auto simp add*: *EndPhase1-def InitializePhase-def hasRead-def*)
qed

theorem *HEndPhase1-valueChosen*:
  assumes *act*: *HEndPhase1 s s' q*
  and *vc*: *valueChosen s v*
  and *inv1*: *Inv1 s*
  and *inv2a*: *Inv2a s*
  and *inv2b*: *Inv2b s*
  and *v-input*: $v \in Inputs$
  shows *valueChosen s' v*
proof −
  from *vc*
  obtain *b p D* **where**
    asm1: $b \in (UN\ p.\ Ballot\ p)$
   and *asm2*: $D \in MajoritySet$
   and *asm3*: *maxBalInp s b v*
   and *asm4*: $\forall\,d \in D.\ \ b \leq bal(disk\ s\ d\ p)$
         $\wedge(\forall\,q.(\quad phase\ s\ q = 1$
              $\wedge\ b \leq mbal(dblock\ s\ q)$
              $\wedge\ hasRead\ s\ q\ d\ p$
             $) \longrightarrow (\exists\,br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$
   by(*auto simp add*: *valueChosen-def*)
  from *HEndPhase1-maxBalInp*[*OF act asm1 asm2 asm3 asm4 inv1 inv2a inv2b*]
  have *maxBalInp s' b v* .
  with *HEndPhase1-valueChosen2*[*OF act asm4*] *asm1 asm2*
  show *?thesis*
   by(*auto simp add*: *valueChosen-def*)
qed

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

lemma *HStartBallot-valueChosen2*:
  assumes *act*: *HStartBallot s s' q*
    and *asm4*: ∀ *d∈D*.   *b ≤ bal(disk s d p)*
                ∧(∀ *q*.(*phase s q = 1*
                  ∧ *b ≤mbal(dblock s q)*
                  ∧ *hasRead s q d p*
) ⟶ (∃ *br∈blocksRead s q d*. *b ≤ bal(block br)*)) (**is** *?P s*)
  shows *?P s'*
proof(*auto*)
  fix *d*
  assume *d*: *d∈D*
  with *act asm4*
  show *b ≤ bal (disk s' d p)*
    by(*auto simp add*: *StartBallot-def*)
  fix *d q*
  assume *d*: *d∈D*
    and *phase'*: *phase s' q = Suc 0*
    and *dblk-mbal*: *b ≤ mbal (dblock s' q)*
    and *hasRead*: *hasRead s' q d p*
  from *phase' act hasRead*
  have *p31*: *phase s q = 1*
    and *p32*: *dblock s' q = dblock s q*
    by(*auto simp add*: *StartBallot-def InitializePhase-def*

121

hasRead-def split : *if-split-asm*)
  with *dblk-mbal*
  have *b≤mbal(dblock s q)* **by** *auto*
  moreover
  from *act hasRead*
  have *hasRead s q d p*
    by(*auto simp add*: *StartBallot-def InitializePhase-def*
      hasRead-def split: *if-split-asm*)
  ultimately
  have *∃ br∈blocksRead s q d. b≤ bal(block br)*
    using *p31 asm4 d*
    by *blast*
  with *act hasRead*
  show *∃ br∈blocksRead s′ q d. b≤ bal(block br)*
    by(*auto simp add*: *StartBallot-def InitializePhase-def*
                   hasRead-def)
qed

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

lemma *HPhase1or2Write-maxBalInp*:
  assumes *act*: *HPhase1or2Write s s′ q d*
    and *asm3*: *maxBalInp s b v*
  shows *maxBalInp s′ b v*
proof(*auto simp add*: *maxBalInp-def*)
  fix *bk*
  assume *bk*: *bk ∈ allBlocks s′*

and *b-bal*: *b≤ bal bk*
    from *subsetD[OF HPhase1or2Write-allBlocks[OF act] bk] asm3 b-bal*
    show *inp bk = v*
      by(*auto simp add*: *maxBalInp-def*)
qed

lemma *HPhase1or2Write-valueChosen2*:
  assumes *act*: *HPhase1or2Write s s' pp d*
    and *asm2*: *D∈MajoritySet*
    and *asm4*: $\forall d∈D$. $b \le bal(disk\ s\ d\ p)$
                  $\land(\forall q.($ *phase s q = 1*
                        $\land\ b \le mbal(dblock\ s\ q)$
                        $\land\ hasRead\ s\ q\ d\ p$
                        $) \longrightarrow (\exists br∈blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$ (**is** *?P s*)
    and *inv4*: *HInv4a s pp*
  shows *?P s'*
proof(*auto*)
  fix *d1*
  assume *d*: *d1∈D*
  show $b \le bal\ (disk\ s'\ d1\ p)$
  proof(*cases d1=d∧pp=p*)
    case *True*
    with *inv4 act*
    have *HInv4a2 s p*
      by(*auto simp add*: *Phase1or2Write-def HInv4a-def*)
    with *asm2 majorities-intersect*
    have $\exists dd∈D.\ bal(disk\ s\ dd\ p) \le bal(dblock\ s\ p)$
      by(*auto simp add*: *HInv4a2-def MajoritySet-def*)
    then obtain *dd* **where** *p41*: $dd∈D \land bal(disk\ s\ dd\ p) \le bal(dblock\ s\ p)$
      by *auto*
    from *asm4 p41*
    have $b \le bal(disk\ s\ dd\ p)$
      by *auto*
    with *p41*
    have *p42*: $b \le bal(dblock\ s\ p)$
      by *auto*
    from *act True*
    have *dblock s p = disk s' d p*
      by(*auto simp add*: *Phase1or2Write-def*)
    with *p42 True*
    show *?thesis*
      by *auto*
  next
    case *False*
    with *act asm4 d*
    show *?thesis*
      by(*auto simp add*: *Phase1or2Write-def*)
  qed
next

fix *d q*
assume *d*: *d∈D*
  and *phase'*: *phase s' q = Suc 0*
  and *dblk-mbal*: $b \leq mbal\ (dblock\ s'\ q)$
  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: Phase1or2Write-def InitializePhase-def*
                hasRead-def split : if-split-asm)
with *dblk-mbal*
have *b≤mbal*(*dblock s q*) **by** *auto*
moreover
from *act hasRead*
have *hasRead s q d p*
  by(*auto simp add: Phase1or2Write-def InitializePhase-def*
   hasRead-def split: if-split-asm)
ultimately
have $\exists\,br{\in}blocksRead\ s\ q\ d.\ b \leq bal(block\ br)$
  using *p31 asm4 d*
  by *blast*
with *act hasRead*
show $\exists\,br{\in}blocksRead\ s'\ q\ d.\ b \leq bal(block\ br)$
  by(*auto simp add: Phase1or2Write-def InitializePhase-def*
                hasRead-def)
qed

theorem *HPhase1or2Write-valueChosen*:
 assumes *act*: *HPhase1or2Write s s' q d*
 and *vc*: *valueChosen s v*
 and *v-input*: *v ∈ Inputs*
 and *inv4*: *HInv4a s q*
 shows *valueChosen s' v*
proof −
 from *vc*
 obtain *b p D* **where**
     asm1: $b \in (UN\ p.\ Ballot\ p)$
  and *asm2*: *D∈MajoritySet*
  and *asm3*: *maxBalInp s b v*
  and *asm4*: $\forall\,d{\in}D.\ \ b \leq bal(disk\ s\ d\ p)$
             $\wedge(\forall\,q.(\ \ phase\ s\ q = 1$
                  $\wedge\ b \leq mbal(dblock\ s\ q)$
                  $\wedge\ hasRead\ s\ q\ d\ p$
                  $) \longrightarrow (\exists\,br{\in}blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$
  by(*auto simp add: valueChosen-def*)
 from *HPhase1or2Write-maxBalInp*[*OF act asm3*]
 have *maxBalInp s' b v* .
 with *HPhase1or2Write-valueChosen2*[*OF act asm2 asm4 inv4*] *asm1 asm2*
 show *?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*: $\forall d \in D.$ $b \leq bal(disk\ s\ d\ p)$
                 $\wedge (\forall q.($ *phase s q = 1*
                         $\wedge$ $b \leq mbal(dblock\ s\ q)$
                         $\wedge$ *hasRead s q d p*
                         $) \longrightarrow (\exists br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$ (**is** *?P s*)
  shows *?P s′*
proof(*auto*)
  fix *dd*
  assume *d*: *dd∈D*
  with *act asm4*
  show $b \leq bal\ (disk\ s′\ dd\ p)$
    by(*auto simp add*: *Phase1or2ReadThen-def*)
  fix *dd qq*
  assume *d*: *dd∈D*
    and *phase′*: *phase s′ qq = Suc 0*
    and *dblk-mbal*: $b \leq mbal\ (dblock\ s′\ qq)$
    and *hasRead*: *hasRead s′ qq dd p*
  show $\exists br \in blocksRead\ s′\ qq\ dd.\ b \leq bal\ (block\ br)$
  proof(*cases d=dd ∧ qq=q ∧ pp=p*)
    case *True*
    from *d asm4*
    have $b \leq bal(disk\ s\ dd\ p)$
      by *auto*
    with *act True*
    show *?thesis*
      by(*auto simp add*: *Phase1or2ReadThen-def*)
  next
    case *False*
    with *phase′ act*
    have *p31*: *phase s qq = 1*

and *p32*: *dblock s′ qq = dblock s qq*
   by(*auto simp add*: *Phase1or2ReadThen-def*)
  with *dblk-mbal*
  have *b≤mbal(dblock s qq)* **by** *auto*
  moreover
  from *act hasRead False*
  have *hasRead s qq dd p*
   by(*auto simp add*: *Phase1or2ReadThen-def*
 hasRead-def split: *if-split-asm*)
  ultimately
  have *∃ br∈blocksRead s qq dd. b≤ bal(block br)*
   using *p31 asm4 d*
   by *blast*
  with *act hasRead*
  show *∃ br∈blocksRead s′ qq dd. b≤ bal(block br)*
   by(*auto simp add*: *Phase1or2ReadThen-def hasRead-def*)
 qed
qed

theorem *HPhase1or2ReadThen-valueChosen*:
 assumes *act*: *HPhase1or2ReadThen s s′ q d p*
 and *vc*: *valueChosen s v*
 and *v-input*: *v ∈ Inputs*
 shows *valueChosen s′ v*
proof −
 from *vc*
 obtain *b p D* **where**
    asm1: *b ∈ (UN p. Ballot p)*
  and *asm2*: *D∈MajoritySet*
  and *asm3*: *maxBalInp s b v*
  and *asm4*: *∀ d∈D. b ≤ bal(disk s d p)*
          ∧(∀ q.(phase s q = 1
             ∧ b ≤mbal(dblock s q)
             ∧ hasRead s q d p
) ⟶ (∃ br∈blocksRead s q d. b ≤ bal(block br)))
  by(*auto simp add*: *valueChosen-def*)
 from *HPhase1or2ReadThen-maxBalInp*[*OF act asm3*]
 have *maxBalInp s′ b v* .
 with *HPhase1or2ReadThen-valueChosen2*[*OF act asm4*] *asm1 asm2*
 show *?thesis*
  by(*auto simp add*: *valueChosen-def*)
qed

theorem *HPhase1or2ReadElse-valueChosen*:
 ⟦ *HPhase1or2ReadElse s s′ p d r*; *valueChosen s v*; *v∈ Inputs* ⟧
   ⟹ *valueChosen s′ v*
 using *HStartBallot-valueChosen*
 by(*auto simp add*: *Phase1or2ReadElse-def*)

lemma *HEndPhase2-maxBalInp*:
  assumes *act*: *HEndPhase2 s s' q*
    and *asm3*: *maxBalInp s b v*
  shows *maxBalInp s' b v*
proof(*auto simp add*: *maxBalInp-def*)
  fix *bk*
  assume *bk*: *bk ∈ allBlocks s'*
    and *b-bal*: *b≤ bal bk*
  from *subsetD*[*OF HEndPhase2-allBlocks*[*OF act*] *bk*] *asm3 b-bal*
  show *inp bk = v*
    by(*auto simp add*: *maxBalInp-def*)
qed

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

127

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

theorem *HEndPhase2-valueChosen*:
  assumes *act*: *HEndPhase2 s s' q*
  and *vc*: *valueChosen s v*
  and *v-input*: $v \in Inputs$
  shows *valueChosen s' v*
proof −
  from *vc*
  obtain *b p D* **where**
      asm1: $b \in (UN\ p.\ Ballot\ p)$
    and *asm2*: $D \in MajoritySet$
    and *asm3*: *maxBalInp s b v*
    and *asm4*: $\forall\, d \in D.\ \ b \leq bal(disk\ s\ d\ p)$
                $\wedge(\forall\, q.(\quad phase\ s\ q = 1$
                        $\wedge\ b \leq mbal(dblock\ s\ q)$
                        $\wedge\ hasRead\ s\ q\ d\ p$
                      $) \longrightarrow (\exists\, br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$
    by(*auto simp add*: *valueChosen-def*)
  from *HEndPhase2-maxBalInp*[*OF act asm3*]
  have *maxBalInp s' b v* **.**
  with *HEndPhase2-valueChosen2*[*OF act asm4*] *asm1 asm2*
  show *?thesis*
    by(*auto simp add*: *valueChosen-def*)
qed

lemma *HFail-maxBalInp*:
  assumes *act*: *HFail s s' q*
    and *asm1*: $b \in (UN\ p.\ Ballot\ p)$
    and *asm3*: *maxBalInp s b v*
  shows *maxBalInp s' b v*
proof(*auto simp add*: *maxBalInp-def*)
  fix *bk*
  assume *bk*: $bk \in allBlocks\ s'$
    and *b-bal*: $b \leq bal\ bk$
  from *subsetD*[*OF HFail-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*: $bk \in \{dblock\ s'\ q\}$
    with *act*
    have *bal bk = 0*

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

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

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*: $v \in Inputs$
  shows *valueChosen s' v*
proof $-$
  from *vc*
  obtain *b p D* **where**
      asm1: $b \in (UN\ p.\ Ballot\ p)$
    and *asm2*: $D \in MajoritySet$
    and *asm3*: *maxBalInp s b v*
    and *asm4*: $\forall d \in D.\ \ b \le bal(disk\ s\ d\ p)$
                $\wedge (\forall q.(\ \ phase\ s\ q = 1$
                      $\wedge\ b \le mbal(dblock\ s\ q)$
                      $\wedge\ hasRead\ s\ q\ d\ p$
                    $) \longrightarrow (\exists\ br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$
    by(*auto simp add*: *valueChosen-def*)
  from *HFail-maxBalInp*[*OF act asm1 asm3*]
  have *maxBalInp s' b v* **.**
  with *HFail-valueChosen2*[*OF act asm4*] *asm1 asm2*
  show *?thesis*
    by(*auto simp add*: *valueChosen-def*)
qed

lemma *HPhase0Read-maxBalInp*:
  assumes *act*: *HPhase0Read s s' q d*
    and *asm3*: *maxBalInp s b v*
  shows *maxBalInp s' b v*
proof(*auto simp add*: *maxBalInp-def*)
  fix *bk*
  assume *bk*: $bk \in allBlocks\ s'$
    and *b-bal*: $b \le bal\ bk$
  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*: $\forall d \in D.\ \ \ b \le bal(disk\ s\ d\ p)$
              $\wedge (\forall q.(\ \ phase\ s\ q = 1$
                 $\wedge\ b \le mbal(dblock\ s\ q)$
                 $\wedge\ hasRead\ s\ q\ d\ p$
               $) \longrightarrow (\exists\ br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$ (**is** *?P s*)
  shows *?P s'*
proof(*auto*)

130

fix *d*
assume *d*: *d∈D*
with *act asm4*
show *b ≤ bal (disk s′ d p)*
  by(*auto simp add*: *Phase0Read-def*)
next
  fix *d q*
  assume *d*: *d∈D*
    and *phase′*: *phase s′ q = Suc 0*
    and *dblk-mbal*: *b ≤ mbal (dblock s′ q)*
    and *hasRead*: *hasRead s′ q d p*
  from *phase′ act*
  have *qqnq*: *qq≠q*
    by(*auto simp add*: *Phase0Read-def*)
  show *∃ br∈blocksRead s′ q d. b ≤ bal (block br)*
  proof −
    from *phase′ act hasRead*
    have *p31*: *phase s q = 1*
      and *p32*: *dblock s′ q = dblock s q*
      by(*auto simp add*: *Phase0Read-def hasRead-def*)
    with *dblk-mbal*
    have *b≤mbal(dblock s q)* **by** *auto*
    moreover
    from *act hasRead qqnq*
    have *hasRead s q d p*
      by(*auto simp add*: *Phase0Read-def hasRead-def*
              split: *if-split-asm*)
    ultimately
    have *∃ br∈blocksRead s q d. b≤ bal(block br)*
      using *p31 asm4 d*
      by *blast*
    with *act hasRead*
    show *∃ br∈blocksRead s′ q d. b≤ bal(block br)*
      by(*auto simp add*: *Phase0Read-def InitializePhase-def*
                  hasRead-def)
  qed
qed

theorem *HPhase0Read-valueChosen*:
  assumes *act*: *HPhase0Read s s′ q d*
  and *vc*: *valueChosen s v*
  and *v-input*: *v ∈ Inputs*
  shows *valueChosen s′ v*
proof −
  from *vc*
  obtain *b p D* **where**
      asm1: *b ∈ (UN p. Ballot p)*
    and *asm2*: *D∈MajoritySet*
    and *asm3*: *maxBalInp s b v*

131

and *asm4*: $\forall\, d{\in}D$. $b \leq bal(disk\ s\ d\ p)$
                 $\wedge(\forall\, q.(\quad phase\ s\ q\ =\ 1$
                         $\wedge\ b \leq mbal(dblock\ s\ q)$
                         $\wedge\ hasRead\ s\ q\ d\ p$
                         $)\ \longrightarrow\ (\exists\, br{\in}blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$
   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*$\leq$ *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*$\in\{dblock\ s′\ q\}$
   with *HEndPhase0-some*[*OF act inv1*] *act*
   have $\exists\, ba{\in}allBlocksRead\ s\ q.\ bal\ ba\ =\ bal\ (dblock\ s′\ q) \wedge inp\ ba\ =\ inp\ (dblock\ s′\ q)$
    by(*auto simp add*: *EndPhase0-def*)
   then obtain *ba*
    where *ba-blksread*: *ba*$\in$*allBlocksRead s q*
    and *ba-balinp*: *bal ba* = *bal (dblock s′ q)* $\wedge$ *inp ba* = *inp (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
qed

132

lemma *HEndPhase0-valueChosen2*:
  assumes *act*: *HEndPhase0 s s' q*
    and *asm4*: $\forall d \in D.$ $b \leq bal(disk\ s\ d\ p)$
                $\wedge (\forall q.(\quad phase\ s\ q = 1$
                      $\wedge\ b \leq mbal(dblock\ s\ q)$
                      $\wedge\ hasRead\ s\ q\ d\ p$
                      $) \longrightarrow (\exists br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$ (**is** *?P s*)
  shows *?P s'*
proof(*auto*)
  fix *d*
  assume *d*: *d* $\in$ *D*
  with *act asm4*
  show $b \leq bal\ (disk\ s'\ d\ p)$
    by(*auto simp add*: *EndPhase0-def*)
  fix *d q*
  assume *d*: *d* $\in$ *D*
    and *phase'*: *phase s' q = Suc 0*
    and *dblk-mbal*: $b \leq mbal\ (dblock\ s'\ q)$
    and *hasRead*: *hasRead s' q d p*
  from *phase' act hasRead*
  have *p31*: *phase s q = 1*
   and *p32*: *dblock s' q = dblock s q*
    by(*auto simp add*: *EndPhase0-def InitializePhase-def*
                hasRead-def split : *if-split-asm*)
  with *dblk-mbal*
  have $b \leq mbal(dblock\ s\ q)$ **by** *auto*
  moreover
  from *act hasRead*
  have *hasRead s q d p*
    by(*auto simp add*: *EndPhase0-def InitializePhase-def*
      hasRead-def split: *if-split-asm*)
  ultimately
  have $\exists br \in blocksRead\ s\ q\ d.\ b \leq bal(block\ br)$
    using *p31 asm4 d*
    by *blast*
  with *act hasRead*
  show $\exists br \in blocksRead\ s'\ q\ d.\ b \leq bal(block\ br)$
    by(*auto simp add*: *EndPhase0-def InitializePhase-def*
                hasRead-def)
qed

theorem *HEndPhase0-valueChosen*:
  assumes *act*: *HEndPhase0 s s' q*
  and *vc*: *valueChosen s v*
  and *v-input*: *v* $\in$ *Inputs*
  and *inv1*: *Inv1 s*
  shows *valueChosen s' v*
proof −
  from *vc*

```
    obtain b p D where
        asm1: b ∈ (UN p. Ballot p)
      and asm2: D∈MajoritySet
      and asm3: maxBalInp s b v
      and asm4: ∀ d∈D.  b ≤ bal(disk s d p)
                  ∧(∀ q.(    phase s q = 1
                        ∧ b ≤mbal(dblock s q)
                        ∧ hasRead s q d p
                        ) ⟶ (∃ br∈blocksRead s q d. b ≤ bal(block br)))
      by(auto simp add: valueChosen-def)
    from HEndPhase0-maxBalInp[OF act asm3 inv1]
    have maxBalInp s′ b v .
    with HEndPhase0-valueChosen2[OF act asm4] asm1 asm2
    show ?thesis
      by(auto simp add: valueChosen-def)
qed

end
```

theory *DiskPaxos-Inv6* **imports** *DiskPaxos-Chosen* **begin**

C.7 Invariant 6

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

definition *HInv6* :: *state* ⇒ *bool*
where
  HInv6 s = ((*chosen s* ≠ *NotAnInput* ⟶ *valueChosen s* (*chosen s*))
        ∧ (∀ p. *outpt s p* ∈ {*chosen s*, *NotAnInput*}))

theorem *HInit-HInv6*: *HInit s* ⟹ *HInv6 s*
  by(*auto simp add*: *HInit-def Init-def InitDB-def HInv6-def*)

lemma *HEndPhase2-Inv6-1*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *inv*: *HInv6 s*
  and *inv2b*: *Inv2b s*
  and *inv2c*: *Inv2c s*
  and *inv3*: *HInv3 s*
  and *inv5*: *HInv5-inner s p*
  and *chosen′*: *chosen s′* ≠ *NotAnInput*
  shows *valueChosen s′* (*chosen s′*)
proof(*cases chosen s=NotAnInput*)
  from *inv5 act*
  have *inv5R*: *HInv5-inner-R s p*
    and *phase*: *phase s p = 2*
    and *ep2-maj*: *IsMajority* {*d* .   *d* ∈ *disksWritten s p*

134

$$\wedge \; (\forall \, q \in UNIV - \{p\}.\; hasRead\; s\; p\; d\; q)\}$$
   by(*auto simp add: EndPhase2-def HInv5-inner-def*)
  case *True*
  have *p32*: *maxBalInp s* (*bal*(*dblock s p*)) (*inp*(*dblock s p*))
  proof−
   have $\neg(\exists \, D \in MajoritySet.\exists \, q.\; (\forall \, d {\in} D.\; bal\; (dblock\; s\; p) < mbal\; (disk\; s\; d\; q)\; \wedge$
$\neg\; hasRead\; s\; p\; d\; q))$
    proof *auto*
     fix *D q*
     assume *Dmaj*: *D∈MajoritySet*
     from *ep2-maj Dmaj majorities-intersect*
     have $\exists \, d {\in} D.\; d \in disksWritten\; s\; p$
      $\wedge \; (\forall \, q \in UNIV - \{p\}.\; hasRead\; s\; p\; d\; q)$
      by(*auto simp add: MajoritySet-def*, *blast*)
     then obtain *d*
      where *dinD*: *d∈D*
      and *ddisk*: *d* ∈ *disksWritten s p*
      and *dhasR*: $\forall \, q \in UNIV - \{p\}.\; hasRead\; s\; p\; d\; q$
      by *auto*
     from *inv2b*
     have *Inv2b-inner s p d*
      by(*auto simp add: Inv2b-def*)
     with *ddisk*
     have *disk s d p = dblock s p*
      by(*auto simp add: Inv2b-inner-def*)
     with *inv2c phase*
     have *bal* (*dblock s p*) = *mbal*(*disk s d p*)
      by(*auto simp add: Inv2c-def Inv2c-inner-def*)
     with *dhasR dinD*
     show $\exists \, d {\in} D.\; bal\; (dblock\; s\; p) < mbal\; (disk\; s\; d\; q) \longrightarrow hasRead\; s\; p\; d\; q$
      by *auto*
    qed
    with *inv5R*
    show *?thesis*
     by(*auto simp add: HInv5-inner-R-def*)
  qed
  have *p33*: *maxBalInp s'* (*bal*(*dblock s' p*)) (*chosen s'*)
  proof −
   from *act*
   have *outpt'*: *outpt s' = (outpt s)* (*p:= inp (dblock s p*))
    by(*auto simp add: EndPhase2-def*)
   have *outpt'-q*: $\forall \, q.\; p{\neq}q \longrightarrow outpt\; s'\; q = NotAnInput$
   proof *auto*
    fix *q*
    assume *pnq*: *p≠q*
    from *outpt' pnq*
    have *outpt s' q= outpt s q*
     by(*auto simp add: EndPhase2-def*)
    with *True inv2c*

show *outpt s' q= NotAnInput*
  by(*auto simp add: Inv2c-def Inv2c-inner-def*)
qed
from *True act chosen'*
have *chosen s' = inp (dblock s p)*
proof(*auto simp add: HNextPart-def split: if-split-asm*)
  fix *pa*
  assume *outpt'-pa*: *outpt s' pa ≠ NotAnInput*
  from *outpt'-q*
  have *someeq2*: $\bigwedge pa.$ *outpt s' pa ≠ NotAnInput $\implies$ pa=p*
    by *auto*
  with *outpt'-pa*
  have *outpt s' p ≠ NotAnInput*
    by *auto*
  from *some-equality*[*of λp. outpt s' p ≠ NotAnInput, OF this someeq2*]
  have *(SOME p. outpt s' p ≠ NotAnInput) = p* .
  with *outpt'*
  show *outpt s' (SOME p. outpt s' p ≠ NotAnInput) = inp (dblock s p)*
    by *auto*
qed
moreover
from *act*
have *bal(dblock s' p) = bal(dblock s p)*
  by(*auto simp add: EndPhase2-def*)
ultimately
have *maxBalInp s (bal(dblock s' p)) (chosen s')*
  using *p32*
  by *auto*
with *HEndPhase2-allBlocks*[*OF act*]
show *?thesis*
  by(*auto simp add: maxBalInp-def*)
qed
from *ep2-maj inv2b majorities-intersect*
have *∃ D∈MajoritySet. (∀ d∈D.   disk s d p = dblock s p*
                        ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q))
  by(*auto simp add: Inv2b-def Inv2b-inner-def MajoritySet-def*)
then obtain *D*
  where *Dmaj*: *D∈MajoritySet*
  and *p34*: *∀ d∈D.   disk s d p = dblock s p*
  ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)
  by *auto*
have *p35*: *∀ q. ∀ d∈D. (phase s q=1 ∧ bal(dblock s p)≤mbal(dblock s q)∧ hasRead s q d p)*
                    ⟶ (|block=dblock s p, proc=p|)∈blocksRead s q d
proof *auto*
  fix *q d*
  assume *dD*: *d∈D* **and** *phase-q*: *phase s q= Suc 0*
    and *bal-mbal*: *bal(dblock s p)≤mbal(dblock s q)* **and** *hasRead*: *hasRead s q d p*
  from *phase inv2c*

136

have *bal(dblock s p)=mbal(dblock s p)*
 by(*auto simp add: Inv2c-def Inv2c-inner-def*)
moreover
from *inv2c phase*
have $\forall\, br{\in}blocksRead\ s\ p\ d.\ mbal(block\ br) < mbal(dblock\ s\ p)$
 by(*auto simp add: Inv2c-def Inv2c-inner-def*)
ultimately
have *p41*: $(\![block{=}dblock\ s\ q,\ proc{=}q]\!)\notin blocksRead\ s\ p\ d$
 using *bal-mbal*
 by *auto*
from *phase phase-q*
have $p{\neq}q$ **by** *auto*
with *p34 dD*
have *hasRead s p d q*
 by *auto*
with *phase phase-q hasRead inv3 p41*
show $(\![block = dblock\ s\ p,\ proc = p]\!) \in blocksRead\ s\ q\ d$
 by(*auto simp add: HInv3-def HInv3-inner-def*
      HInv3-L-def HInv3-R-def)
qed
 have *p36*: $\forall\, q.\ \forall\, d{\in}D.\ phase\ s'\ q{=}1 \,\wedge\, bal(dblock\ s\ p) \leq mbal(dblock\ s'\ q)\,\wedge$
hasRead s' q d p
        $\longrightarrow (\exists\, br{\in}blocksRead\ s'\ q\ d.\ bal(block\ br) = bal(dblock\ s\ p))$
 proof(*auto*)
 fix *q d*
 assume *dD*: $d \in D$ **and** *phase-q*: *phase s' q = Suc 0*
   and *bal*: *bal (dblock s p)* $\leq$ *mbal (dblock s' q)*
   and *hasRead*: *hasRead s' q d p*
 from *phase-q act*
 have *phase s' q=phase s q* $\wedge$ *dblock s' q=dblock s q* $\wedge$ *hasRead s' q d p=hasRead s q d p* $\wedge$ *blocksRead s' q d=blocksRead s q d*
  by(*auto simp add: EndPhase2-def hasRead-def InitializePhase-def*)
 with *p35 phase-q bal hasRead dD*
 have $(\![block{=}dblock\ s\ p,\ proc{=}p]\!){\in}blocksRead\ s'\ q\ d$
  by *auto*
 thus $\exists\, br{\in}blocksRead\ s'\ q\ d.\ bal(block\ br) = bal(dblock\ s\ p)$
  by *force*
 qed
 hence *p36-2*: $\forall\, q.\ \forall\, d{\in}D.\ phase\ s'\ q{=}1 \,\wedge\, bal(dblock\ s\ p) \leq mbal(dblock\ s'\ q)\,\wedge$
hasRead s' q d p
       $\longrightarrow (\exists\, br{\in}blocksRead\ s'\ q\ d.\ bal(dblock\ s\ p) \leq bal(block\ br))$
 by *force*
 from *act*
 have *bal-dblock*: *bal(dblock s' p)=bal(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 $\forall\,d{\in}D.\ bal(dblock\ s\ p) \le bal(disk\ s'\ d\ p)$
  by *auto*
ultimately
have $maxBalInp\ s'\ (bal(dblock\ s\ p))\ (chosen\ s') \wedge$
      $(\exists\,D{\in}MajoritySet.$
            $\forall\,d{\in}D.\ bal(dblock\ s\ p) \le bal\ (disk\ s'\ d\ p) \wedge$
               $(\forall\,q.\ phase\ s'\ q = Suc\ 0 \wedge$
                  $bal(dblock\ s\ p) \le mbal\ (dblock\ s'\ q) \wedge hasRead\ s'\ q\ d\ p \longrightarrow$
                  $(\exists\,br{\in}blocksRead\ s'\ q\ d.\ bal(dblock\ s\ p) \le bal\ (block\ br))))$
  using *p36-2 Dmaj*
  by *auto*
moreover
from *phase inv2c*
have $bal(dblock\ s\ p){\in}\ Ballot\ p$
  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*: $D \in MajoritySet$
  and *asm-v*: $\forall\,d{\in}D.\ b \le bal\ (disk\ s\ d\ p)$
  and *max-w*: $maxBalInp\ s\ ba\ w$
  and *Damaj*: $Da \in MajoritySet$
  and *asm-w*: $\forall\,d{\in}Da.\ ba \le bal\ (disk\ s\ d\ pa)$
  and *b-ba*: $b{\le}ba$
  shows $v{=}w$
proof $-$
  have $\forall\,d.\ disk\ s\ d\ pa \in allBlocks\ s$
    by(*auto simp add*: *allBlocks-def blocksOf-def*)
  with *majorities-intersect Dmaj Damaj*
  have $\exists\,d{\in}D{\cap}Da.\ disk\ s\ d\ pa \in allBlocks\ s$
    by(*auto simp add*: *MajoritySet-def*, *blast*)
  then obtain *d*

138

where *dinmaj*: *d∈D∩Da* **and** *dab*: *disk s d pa ∈ allBlocks s*
  by *auto*
with *asm-w*
have *ba*: *ba ≤ bal (disk s d pa)*
  by *auto*
with *b-ba*
have *b ≤ bal (disk s d pa)*
  by *auto*
with *max-v dab*
have *v-value*: *inp (disk s d pa) = v*
  by(*auto simp add*: *maxBalInp-def*)
from *ba max-w dab*
have *w-value*: *inp (disk s d pa) = w*
  by(*auto simp add*: *maxBalInp-def*)
with *v-value*
show *?thesis* **by** *auto*
qed

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

show *?thesis*
   by *auto*
  qed
qed

lemma *HEndPhase2-Inv6-2*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *inv*: *HInv6 s*
  and *inv2b*: *Inv2b s*
  and *inv2c*: *Inv2c s*
  and *inv3*: *HInv3 s*
  and *inv5*: *HInv5-inner s p*
  and *asm*: *outpt s′ r ≠ NotAnInput*
  shows *outpt s′ r = chosen s′*
proof(*cases chosen s=NotAnInput*)
  case *True*
  with *inv2c*
  have *∀ q. outpt s q = NotAnInput*
    by(*auto simp add*: *Inv2c-def Inv2c-inner-def*)
  with *True act asm*
  show *?thesis*
    by(*auto simp add*: *EndPhase2-def HNextPart-def*
          split: *if-split-asm*)
next
  case *False*
  with *inv*
  have *p31*: *valueChosen s (chosen s)*
    by(*auto simp add*: *HInv6-def*)
  with *False act*
  have *chosen s′≠ NotAnInput*
    by(*auto simp add*: *HNextPart-def*)
  from *HEndPhase2-Inv6-1*[*OF act inv inv2b inv2c inv3 inv5 this*]
  have *p32*: *valueChosen s′(chosen s′)* .
  from *False InputsOrNi*
  have *chosen s ∈ Inputs* **by** *auto*
  from *valueChosen-equal*[*OF HEndPhase2-valueChosen*[*OF act p31 this*] *p32*]
  have *p33*: *chosen s = chosen s′* .
  from *act*
  have *maj*: *IsMajority {d .   d ∈ disksWritten s p*
                     ∧ (∀ q ∈ UNIV − {p}. hasRead s p d q)} (**is** *IsMajority ?D*)
    and *phase*: *phase s p = 2*
    by(*auto simp add*: *EndPhase2-def*)
  show *?thesis*
  proof(*cases outpt s r = NotAnInput*)
    case *True*
    with *asm act*
    have *p41*: *r=p*
      by(*auto simp add*: *EndPhase2-def split*: *if-split-asm*)
    from *maj*

have *p42*: $\exists D \in MajoritySet. \forall d \in D. \forall q \in UNIV - \{p\}. hasRead\ s\ p\ d\ q$
   by(*auto simp add: MajoritySet-def*)
have *p43*: $\neg(\exists D \in MajoritySet. \exists q. (\forall d \in D.\quad bal(dblock\ s\ p) < mbal(disk\ s\ d\ q)$

$\wedge \neg hasRead\ s\ p\ d\ q))$

   proof *auto*
   fix *D q*
   assume *Dmaj*: $D \in MajoritySet$
   show $\exists d \in D. bal\ (dblock\ s\ p) < mbal\ (disk\ s\ d\ q) \longrightarrow hasRead\ s\ p\ d\ q$
   proof(*cases p=q*)
     assume *pq*: $p=q$
     thus *?thesis*
     proof *auto*
       from *maj majorities-intersect Dmaj*
       have *?D∩D≠{}*
         by(*auto simp add: MajoritySet-def*)
       hence $\exists d \in ?D \cap D. d \in disksWritten\ s\ p$ **by** *auto*
       then obtain *d* **where** *d*: $d \in disksWritten\ s\ p$ **and** $d \in ?D \cap D$
         by *auto*
       hence *dD*: $d \in D$ **by** *auto*
       from *d inv2b*
       have $disk\ s\ d\ p = dblock\ s\ p$
         by(*auto simp add: Inv2b-def Inv2b-inner-def*)
       with *inv2c phase*
       have $bal(dblock\ s\ p) = mbal(disk\ s\ d\ p)$
         by(*auto simp add: Inv2c-def Inv2c-inner-def*)
       with *dD pq*
       show $\exists d \in D. bal\ (dblock\ s\ q) < mbal\ (disk\ s\ d\ q) \longrightarrow hasRead\ s\ q\ d\ q$
         by *auto*
     qed
   next
     case *False*
     with *p42*
     have $\exists D \in MajoritySet. \forall d \in D. hasRead\ s\ p\ d\ q$
       by *auto*
     with *majorities-intersect Dmaj*
     show *?thesis*
       by(*auto simp add: MajoritySet-def*, *blast*)
   qed
 qed
 with *inv5 act*
 have *p44*: $maxBalInp\ s\ (bal(dblock\ s\ p))\ (inp(dblock\ s\ p))$
   by(*auto simp add: EndPhase2-def HInv5-inner-def*
               HInv5-inner-R-def)
 have $\exists bk \in allBlocks\ s. \exists b \in (UN\ p.\ Ballot\ p). (maxBalInp\ s\ b\ (chosen\ s)) \wedge b \leq bal\ bk$
 proof −
   have *disk-allblks*: $\forall d\ p. disk\ s\ d\ p \in allBlocks\ s$
     by(*auto simp add: allBlocks-def blocksOf-def*)

141

from *p31*
have $\exists\, b \in (UN\ p.\ Ballot\ p).\ maxBalInp\ s\ b\ (chosen\ s) \land$
$(\exists\, p.\ \exists\, D \in MajoritySet.(\forall\, d \in D.\ \ b \leq bal(disk\ s\ d\ p)))$
  by(*auto simp add*: *valueChosen-def*, *force*)
with *majority-nonempty* **obtain** *b p D d*
  where *IsMajority D* $\land$ $b \in (UN\ p.\ Ballot\ p) \land$
       maxBalInp s b (*chosen s*) $\land$ $d \in D \land b \leq bal(disk\ s\ d\ p)$
  by(*auto simp add*: *MajoritySet-def*, *blast*)
with *disk-allblks*
show *?thesis*
  by(*auto*)
qed
then obtain *bk b*
  where *p45-bk*: $bk \in allBlocks\ s \land b \leq bal\ bk$
    and *p45-b*: $b \in (UN\ p.\ Ballot\ p) \land (maxBalInp\ s\ b\ (chosen\ s))$
  by *auto*
have *p46*: $inp(dblock\ s\ p) = chosen\ s$
proof(*cases* $b \leq bal(dblock\ s\ p)$)
  case *True*
  have $dblock\ s\ p \in allBlocks\ s$
    by(*auto simp add*: *allBlocks-def blocksOf-def*)
  with *p45-b True*
  show *?thesis*
    by(*auto simp add*: *maxBalInp-def*)
next
  case *False*
  from *p44 p45-bk False*
  have $inp\ bk = inp(dblock\ s\ p)$
    by(*auto simp add*: *maxBalInp-def*)
  with *p45-b p45-bk*
  show *?thesis*
    by(*auto simp add*: *maxBalInp-def*)
qed
with *p41 p33 act*
show *?thesis*
  by(*auto simp add*: *EndPhase2-def*)
next
case *False*
from *inv2c*
have *Inv2c-inner s r*
  by(*auto simp add*: *Inv2c-def*)
with *False asm inv2c act*
have $outpt\ s'\ r = outpt\ s\ r$
  by(*auto simp add*: *Inv2c-inner-def EndPhase2-def*
        split: *if-split-asm*)
with *inv p33 False*
show *?thesis*
  by(*auto simp add*: *HInv6-def*)
qed

qed

theorem *HEndPhase2-Inv6*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *inv*: *HInv6 s*
  and *inv2b*: *Inv2b s*
  and *inv2c*: *Inv2c s*
  and *inv3*: *HInv3 s*
  and *inv5*: *HInv5-inner s p*
  shows *HInv6 s′*
proof(*auto simp add*: *HInv6-def*)
  assume *chosen s′ ≠ 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*: *StartBallot-def HInv6-def*)
　from *HStartBallot-valueChosen*[*OF act*] *this InputsOrNi*
　have *t1*: *chosen s'* $\neq$ *NotAnInput* $\longrightarrow$ *valueChosen s'* (*chosen s'*)
　　by *auto*
　from *act*
　have *outpt*: *outpt s* = *outpt s'*
　　by(*auto simp add*: *StartBallot-def*)
　from *outpt-Inv6*[*OF outpt*] *act inv2c inv*
　have $\forall\, p.$ *outpt s' p* = *chosen s'* $\lor$ *outpt s' p* = *NotAnInput*
　　by(*auto simp add*: *HInv6-def*)
　with *t1*
　show *?thesis*
　　by(*simp add*: *HInv6-def*)
qed

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

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

have *t1*: *chosen s′ ≠ NotAnInput ⟶ valueChosen s′ (chosen s′)*
  by *auto*
from *act*
have *outpt*: *outpt s = outpt s′*
  by(*auto simp add*: *Phase1or2ReadThen-def*)
from *outpt-Inv6*[*OF outpt*] *act inv2c inv*
have *∀ p*. *outpt s′ p = chosen s′ ∨ outpt s′ p = NotAnInput*
  by(*auto simp add*: *HInv6-def*)
with *t1*
show *?thesis*
  by(*simp add*: *HInv6-def*)
qed

theorem *HPhase1or2ReadElse-Inv6*:
  assumes *act*: *HPhase1or2ReadElse s s′ p d q*
  and *inv*: *HInv6 s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
  using *assms* **and** *HStartBallot-Inv6*
  by(*auto simp add*: *Phase1or2ReadElse-def*)

theorem *HEndPhase1-Inv6*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv6 s*
  and *inv1*: *Inv1 s*
  and *inv2a*: *Inv2a s*
  and *inv2b*: *Inv2b s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
proof −
  from *outpt-chosen act inv2c inv*
  have *chosen s′ ≠ NotAnInput ⟶ valueChosen s (chosen s′)*
    by(*auto simp add*: *EndPhase1-def HInv6-def*)
  from *HEndPhase1-valueChosen*[*OF act*] *inv1 inv2a inv2b this InputsOrNi*
  have *t1*: *chosen s′ ≠ NotAnInput ⟶ valueChosen s′ (chosen s′)*
    by *auto*
  from *act*
  have *outpt*: *outpt s = outpt s′*
    by(*auto simp add*: *EndPhase1-def*)
  from *outpt-Inv6*[*OF outpt*] *act inv2c inv*
  have *∀ p*. *outpt s′ p = chosen s′ ∨ outpt s′ p = NotAnInput*
    by(*auto simp add*: *HInv6-def*)
  with *t1*
  show *?thesis*
    by(*simp add*: *HInv6-def*)
qed

lemma *outpt-chosen-2*:
  assumes *outpt*: *outpt s′ = (outpt s) (p:= NotAnInput)*

145

   and *inv2c*: *Inv2c s*
   and *nextp*: *HNextPart s s′*
   shows *chosen s = chosen s′*
proof −
  from *inv2c*
  have *chosen s = NotAnInput* ⟶ (∀ *p*. *outpt s p = NotAnInput*)
   by(*auto simp add*: *Inv2c-inner-def Inv2c-def*)
  with *outpt nextp*
  show *?thesis*
   by(*auto simp add*: *HNextPart-def*)
qed

lemma *outpt-HInv6-2*:
  assumes *outpt*: *outpt s′ = (outpt s) (p:= NotAnInput)*
  and *inv*: ∀ *p*. *outpt s p* ∈ {*chosen s*, *NotAnInput*}
  and *inv2c*: *Inv2c s*
  and *nextp*: *HNextPart s s′*
  shows ∀ *p*. *outpt s′ p* ∈ {*chosen s′*, *NotAnInput*}
proof −
  from *outpt-chosen-2*[*OF outpt inv2c nextp*]
  have *chosen s = chosen s′* **.**
  with *inv outpt*
  show *?thesis*
   by *auto*
qed

theorem *HFail-Inv6*:
  assumes *act*: *HFail s s′ p*
  and *inv*: *HInv6 s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
proof −
  from *outpt-chosen-2 act inv2c inv*
  have *chosen s′* ≠ *NotAnInput* ⟶ *valueChosen s (chosen s′)*
   by(*auto simp add*: *Fail-def HInv6-def*)
  from *HFail-valueChosen*[*OF act*] *this InputsOrNi*
  have *t1*: *chosen s′* ≠ *NotAnInput* ⟶ *valueChosen s′ (chosen s′)*
   by *auto*
  from *act*
  have *outpt*: *outpt s′ = (outpt s) (p:=NotAnInput)*
   by(*auto simp add*: *Fail-def*)
  from *outpt-HInv6-2*[*OF outpt*] *act inv2c inv*
  have ∀ *p*. *outpt s′ p = chosen s′* ∨ *outpt s′ p = NotAnInput*
   by(*auto simp add*: *HInv6-def*)
  with *t1*
  show *?thesis*
   by(*simp add*: *HInv6-def*)
qed

theorem *HPhase0Read-Inv6*:
  assumes *act*: *HPhase0Read s s′ p d*
  and *inv*: *HInv6 s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
proof −
  from *outpt-chosen act inv2c inv*
  have *chosen s′ ≠ NotAnInput ⟶ valueChosen s (chosen s′)*
    by(*auto simp add*: *Phase0Read-def HInv6-def*)
  from *HPhase0Read-valueChosen*[*OF act*] *this InputsOrNi*
  have *t1*: *chosen s′ ≠ NotAnInput ⟶ valueChosen s′ (chosen s′)*
    by *auto*
  from *act*
  have *outpt*: *outpt s = outpt s′*
    by(*auto simp add*: *Phase0Read-def*)
  from *outpt-Inv6*[*OF outpt*] *act inv2c inv*
  have *∀ p. outpt s′ p = chosen s′ ∨ outpt s′ p = NotAnInput*
    by(*auto simp add*: *HInv6-def*)
  with *t1*
  show *?thesis*
    by(*simp add*: *HInv6-def*)
qed

theorem *HEndPhase0-Inv6*:
  assumes *act*: *HEndPhase0 s s′ p*
  and *inv*: *HInv6 s*
  and *inv1*: *Inv1 s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
proof −
  from *outpt-chosen act inv2c inv*
  have *chosen s′ ≠ NotAnInput ⟶ valueChosen s (chosen s′)*
    by(*auto simp add*: *EndPhase0-def HInv6-def*)
  from *HEndPhase0-valueChosen*[*OF act*] *inv1 this InputsOrNi*
  have *t1*: *chosen s′ ≠ NotAnInput ⟶ valueChosen s′ (chosen s′)*
    by *auto*
  from *act*
  have *outpt*: *outpt s = outpt s′*
    by(*auto simp add*: *EndPhase0-def*)
  from *outpt-Inv6*[*OF outpt*] *act inv2c inv*
  have *∀ p. outpt s′ p = chosen s′ ∨ outpt s′ p = NotAnInput*
    by(*auto simp add*: *HInv6-def*)
  with *t1*
  show *?thesis*
    by(*simp add*: *HInv6-def*)
qed

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

lemma *I2f*:
  assumes *nxt*: *HNext s s′*
  and *inv*: *HInv1 s ∧ HInv2 s ∧ HInv2 s′ ∧ HInv3 s ∧ HInv4 s ∧ HInv5 s ∧*
HInv6 s
  shows *HInv6 s′* **using** *assms*
  by(*auto simp add*: *HNext-def Next-def*,
    auto simp add: *HInv2-def intro*: *HStartBallot-Inv6*,
    auto intro: *HPhase0Read-Inv6*,
    auto simp add: *HInv4-def intro*: *HPhase1or2Write-Inv6*,
    auto simp add: *Phase1or2Read-def*
       intro: *HPhase1or2ReadThen-Inv6*
            HPhase1or2ReadElse-Inv6,
    auto simp add: *EndPhase1or2-def HInv1-def HInv5-def*
      intro: *HEndPhase1-Inv6*
           HEndPhase2-Inv6,
    auto intro: *HFail-Inv6*,
    auto intro: *HEndPhase0-Inv6*)

end

theory *DiskPaxos-Invariant* **imports** *DiskPaxos-Inv6* **begin**

C.8 The Complete Invariant

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

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

theorem *I2*:
  assumes *inv*: *HInv s*
  and *nxt*: *HNext s s′*
  shows *HInv s′*
  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*)

end

theory *DiskPaxos* **imports** *DiskPaxos-Invariant* **begin**

C.9 Inner Module

record
Istate =
  iinput :: *Proc* $\Rightarrow$ *InputsOrNi*
  ioutput :: *Proc* $\Rightarrow$ *InputsOrNi*
  ichosen :: *InputsOrNi*
  iallInput :: *InputsOrNi set*

definition *IInit* :: *Istate* $\Rightarrow$ *bool*
where
  IInit s = (*range* (*iinput s*) $\subseteq$ *Inputs*
      $\wedge$ *ioutput s* = (λp. *NotAnInput*)
      $\wedge$ *ichosen s* = *NotAnInput*
      $\wedge$ *iallInput s* = *range* (*iinput s*))

definition *IChoose* :: *Istate* $\Rightarrow$ *Istate* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  IChoose s s$'$ p = (*ioutput s p* = *NotAnInput*
      $\wedge$ (*if* (*ichosen s* = *NotAnInput*)
        then ($\exists ip \in$ *iallInput s*. *ichosen s$'$* = *ip*
                     $\wedge$ *ioutput s$'$* = (*ioutput s*) (*p* := *ip*))
        else (*ioutput s$'$* = (*ioutput s*) (*p*:= *ichosen s*)
          $\wedge$ *ichosen s$'$* = *ichosen s*))
      $\wedge$ *iinput s$'$* = *iinput s* $\wedge$ *iallInput s$'$*= *iallInput s*)

definition *IFail* :: *Istate* $\Rightarrow$ *Istate* $\Rightarrow$ *Proc* $\Rightarrow$ *bool*
where
  IFail s s$'$ p = (*ioutput s$'$* = (*ioutput s*) (*p*:= *NotAnInput*)
      $\wedge$ ($\exists ip \in$ *Inputs*. *iinput s$'$* = (*iinput s*)(*p*:= *ip*)
                  $\wedge$ *iallInput s$'$* = *iallInput s* $\cup$ {*ip*})
      $\wedge$ *ichosen s$'$* = *ichosen s*)

definition *INext* :: *Istate* $\Rightarrow$ *Istate* $\Rightarrow$ *bool*
  where *INext s s$'$* = ($\exists p$. *IChoose s s$'$ p* $\vee$ *IFail s s$'$ p*)

definition *s2is* :: *state* $\Rightarrow$ *Istate*
where
  s2is s = (|*iinput* = *inpt s*,
        ioutput = *outpt s*,
        ichosen=*chosen s*,
        iallInput = *allInput s*|)

theorem *R1*:

$[\![$ *HInit s*; *is = s2is s* $]\!] \Longrightarrow$ *IInit is*
by(*auto simp add*: *HInit-def IInit-def s2is-def Init-def*)

theorem *R2b*:
  assumes *inv*: *HInv s*
  and *inv′*: *HInv s′*
  and *nxt*: *HNext s s′*
  and *srel*: *is=s2is s* ∧ *is′=s2is s′*
  shows (∃ *p*. *IFail is is′ p* ∨ *IChoose is is′ p*) ∨ *is = is′*
proof(*auto*)
  assume *chg-vars*: *is≠is′*
  with *srel*
  have *s-change*:   *inpt s* ≠ *inpt s′* ∨ *outpt s* ≠ *outpt s′*
              ∨ *chosen s* ≠ *chosen s′* ∨ *allInput s* ≠ *allInput s′*
    by(*auto simp add*: *s2is-def*)
  from *inv*
  have *inv2c5*: ∀ *p*.   *inpt s p* ∈ *allInput s*
                  ∧ (*chosen s = NotAnInput* ⟶ *outpt s p = NotAnInput*)
    by(*auto simp add*: *HInv-def HInv2-def Inv2c-def Inv2c-inner-def*)
  from *nxt s-change inv2c5*
  have *inpt s′* ≠ *inpt s* ∨ *outpt s′* ≠ *outpt s*
    by(*auto simp add*: *HNext-def Next-def HNextPart-def*)
  with *nxt*
  have ∃ *p*. *Fail s s′ p* ∨ *EndPhase2 s s′ p*
    by(*auto simp add*: *HNext-def Next-def*
      StartBallot-def Phase0Read-def Phase1or2Write-def
      Phase1or2Read-def Phase1or2ReadThen-def Phase1or2ReadElse-def
      EndPhase1or2-def EndPhase1-def EndPhase0-def)
  then obtain *p* **where** *fail-or-endphase2*: *Fail s s′ p* ∨ *EndPhase2 s s′ p*
    by *auto*
  from *inv*
  have *inv2c*: *Inv2c-inner s p*
    by(*auto simp add*: *HInv-def HInv2-def Inv2c-def*)
  from *fail-or-endphase2* **have** *IFail is is′ p* ∨ *IChoose is is′ p*
  proof
    assume *fail*: *Fail s s′ p*
    hence *phase′*: *phase s′ p = 0*
      and *outpt*: *outpt s′ = (outpt s) (p:= NotAnInput)*
      by(*auto simp add*: *Fail-def*)
    have *IFail is is′ p*
    proof −
      from *fail srel*
      have *ioutput is′ = (ioutput is) (p:= NotAnInput)*
        by(*auto simp add*: *Fail-def s2is-def*)
      moreover
      from *nxt*
      have *all-nxt*: *allInput s′= allInput s* ∪ (*range (inpt s′)*)
        by(*auto simp add*: *HNext-def HNextPart-def*)
      from *fail srel*

150

have $\exists\, ip \in Inputs.\;\; iinput\; is' = (iinput\; is)(p := ip)$
  by (*auto simp add*: *Fail-def s2is-def*)
then obtain *ip* **where** *ip-Input*: $ip \in Inputs$ **and** $iinput\; is' = (iinput\; is)(p := ip)$
  by *auto*
with *inv2c5 srel all-nxt*
have $iinput\; is' = (iinput\; is)(p := ip)$
  $\land\; iallInput\; is' = iallInput\; is \cup \{ip\}$
  by (*auto simp add*: *s2is-def*)
moreover
from *outpt srel nxt inv2c*
have $ichosen\; is' = ichosen\; is$
  by (*auto simp add*: *HNext-def HNextPart-def s2is-def Inv2c-inner-def*)
ultimately
show *?thesis*
  using *ip-Input*
  by (*auto simp add*: *IFail-def*)
qed
thus *?thesis*
  by *auto*
next
assume *endphase2*: $EndPhase2\; s\; s'\; p$
from *endphase2*
have $phase\; s\; p = 2$
  by (*auto simp add*: *EndPhase2-def*)
with *inv2c Ballot-nzero*
have *bal-dblk-nzero*: $bal(dblock\; s\; p) \neq 0$
  by (*auto simp add*: *Inv2c-inner-def*)
moreover
from *inv*
have *inv2a-dblock*: $Inv2a\text{-}innermost\; s\; p\; (dblock\; s\; p)$
 by (*auto simp add*: *HInv-def HInv2-def Inv2a-def Inv2a-inner-def blocksOf-def*)
ultimately
have *p22*: $inp\; (dblock\; s\; p) \in allInput\; s$
  by (*auto simp add*: *Inv2a-innermost-def*)
from *inv*
have $allInput\; s \subseteq Inputs$
  by (*auto simp add*: *HInv-def HInv1-def*)
with *p22 NotAnInput endphase2*
have *outpt-nni*: $outpt\; s'\; p \neq NotAnInput$
  by (*auto simp add*: *EndPhase2-def*)
show *?thesis*
proof (*cases chosen s = NotAnInput*)
  case *True*
  with *inv2c5*
  have *p31*: $\forall\, q.\; outpt\; s\; q = NotAnInput$
    by *auto*
  with *endphase2*
  have *p32*: $\forall\, q \in UNIV - \{p\}.\; outpt\; s'\; q = NotAnInput$

151

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

hence *some-eq*: ($\bigwedge x.\ outpt\ s'\ x \neq NotAnInput \implies x = p$)

by *auto*

from *p32 True nxt some-equality*[*of* $\lambda p.\ outpt\ s'\ p \neq NotAnInput,\ OF\ outpt\text{-}nni\ some\text{-}eq$]

  have *p33*: *chosen* $s' = outpt\ s'\ p$

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

  with *endphase2*

  have *chosen* $s' = inp(dblock\ s\ p) \wedge outpt\ s' = (outpt\ s)(p:=inp(dblock\ s\ p))$

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

  with *True p22*

  have *if* (*chosen* $s = NotAnInput$)

                then ($\exists\ ip \in allInput\ s.\ chosen\ s' = ip$

                                $\wedge\ outpt\ s' = (outpt\ s)\ (p := ip)$)

                else (*outpt* $s' = (outpt\ s)\ (p:= chosen\ s)$

                      $\wedge\ chosen\ s' = chosen\ s$)

  by *auto*

  moreover

  from *endphase2 inv2c5 nxt*

  have *inpt* $s' = inpt\ s \wedge allInput\ s' = allInput\ s$

  by(*auto simp add*: *EndPhase2-def HNext-def HNextPart-def*)

  ultimately

  show *?thesis*

    using *srel p31*

    by(*auto simp add*: *IChoose-def s2is-def*)

  next

  case *False*

  with *nxt*

  have *p31*: *chosen* $s' = chosen\ s$

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

  from *inv'*

  have *inv6*: *HInv6* s'

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

  have *p32*: *outpt* $s'\ p = chosen\ s$

  proof−

    from *endphase2*

    have *outpt* $s'\ p\ = inp(dblock\ s\ p)$

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

    moreover

    from *inv6 p31*

    have *outpt* $s'\ p \in \{chosen\ s,\ NotAnInput\}$

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

    ultimately

    show *?thesis*

      using *outpt-nni*

      by *auto*

  qed

  from *srel False*

  have *IChoose is is'* p

proof(*clarsimp simp add*: *IChoose-def s2is-def*)
  from *endphase2 inv2c*
  have *outpt s p = NotAnInput*
    by(*auto simp add*: *EndPhase2-def Inv2c-inner-def*)
  moreover
  from *endphase2 p31 p32 False*
  have *outpt s′ = (outpt s) (p:= chosen s) ∧ chosen s′ = chosen s*
    by(*auto simp add*: *EndPhase2-def*)
  moreover
  from *endphase2 nxt inv2c5*
  have *inpt s′ = inpt s ∧ allInput s′= allInput s*
    by(*auto simp add*: *EndPhase2-def HNext-def HNextPart-def*)
  ultimately
  show   *outpt s p = NotAnInput*
     ∧ outpt s′ = (outpt s)(p := chosen s) ∧ chosen s′ = chosen s
     ∧ inpt s′ = inpt s ∧ allInput s′ = allInput s
    by *auto*
  qed
  thus *?thesis*
    by *auto*
  qed
 qed
 thus *∃ p. IFail is is′ p ∨ IChoose is is′ p*
  by *auto*
qed

end

153