Proving the Correctness of Disk Paxos in Isabelle/HOL

Mauro Jaskelioff Stephan Merz

March 17, 2025

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 $HInv1$ and $HInv3$) was verified using the theorem prover ACL2 [KMM00]. An implicit assumption of this formalization is that all sets are finite, thus overlooking the fact that there is a missing conjunct in the Disk Paxos invariant (see Section 4).

We set the goal of formally verifying Disk Paxos correctness using the theorem prover Isabelle/HOL [NPW02]. In this way, we could gain more confidence in the correctness of Disk Paxos design and, at the same time, learn to what extent can Isabelle be a useful tool for proving the correctness of distributed systems using a real world example.

In Section 2 we give a brief description of the algorithm and its specification. In Section 3 we describe the translation from TLA$^+$ to Isabelle/HOL and the problems that this translation originated. In Section 4 we discuss how our formal proofs relate to the informal ones in [GL00], and in Section 5 we conclude. The entire specification and all formal proofs can be found in the Appendix.

2 The Disk Paxos Algorithm

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

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

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

2.1 Informal description of the algorithm.

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

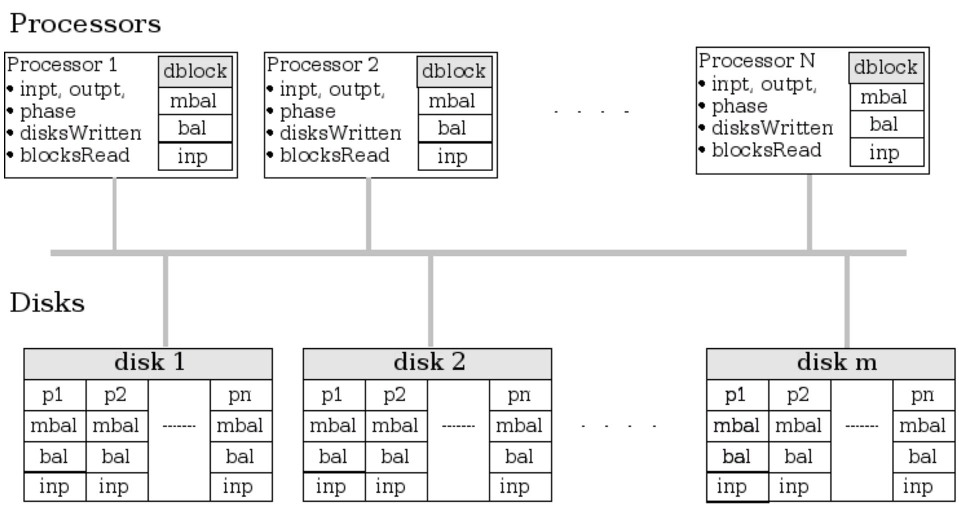

Figure 1: A network of processors and disks.

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:

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

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:

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

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

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

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

TLA$^+$	Isabelle/HOL
$\exists\, d \in D \;:\; disk[d][q].bal = bk$	$\exists\, d \in D.\; bal(disk\ s\ d\ q) \;=\; bk$
CHOOSE $x.P\,x$	$\varepsilon\ x.\ P\ x$
$phase' = [phase$ 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 $I1$ $\quad HInit \Rightarrow HInv$
THEOREM $I2$ $\quad HInv \wedge HNext \Rightarrow HInv'$

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$^+$:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\textsc{constant } \mathit{Inputs}} \\
\mathit{NotAnInput} & \triangleq & \textsc{choose } c : c \notin \mathit{Inputs} \\
\mathit{DiskBlock} & \triangleq & [\mathit{mbal} : (\textsc{union } \mathit{Ballot}(p) : p \in \mathit{Proc}) \cup \{0\}, \\
 & & \;\mathit{bal} \;\; : (\textsc{union } \mathit{Ballot}(p) : p \in \mathit{Proc}) \cup \{0\}, \\
 & & \;\mathit{inp} \;\; : \mathit{Inputs} \cup \{\mathit{NotAnInput}\}]
\end{array}
$$

Isabelle/HOL:

typedecl *InputsOrNi*

consts
Inputs :: *InputsOrNi set*
NotAnInput :: *InputsOrNi*

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

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 $\mathit{Inputs} \cup \{\mathit{NotAnInput}\}$, 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$^+$:

$$
\begin{aligned}
&Phase1or2Write(p, d) \;\triangleq \\
&\quad \land\ phase[p] \in \{1, 2\} \\
&\quad \land\ disk' = [disk \text{ EXCEPT } ![d][p] = dblock[p]] \\
&\quad \land\ disksWritten' = [disksWritten \text{ EXCEPT } ![p] = @ \cup \{d\}] \\
&\quad \land\ \text{UNCHANGED } \langle input, output, phase, dblock, blocksRead \rangle
\end{aligned}
$$

Isabelle/HOL:

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

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:

$$Phase1or2Read \triangleq Phase1or2ReadThen \lor Phase1or2ReadElse$$

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:

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

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 ≡ ∀ *p*. ∀ *bk* ∈ *blocksOf s p*. . . .

we write:

Inv2a-innermost :: *state* ⇒ *Proc* ⇒ *DiskBlock* ⇒ *bool*
Inv2a-innermost s p bk ≡ . . .

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

Inv2a :: *state* ⇒ *bool*
Inv2a s ≡ ∀ *p*. *Inv2a-inner s p*

Now we can express that we want to obtain the fact

Inv2a-innermost s q (dblock s q)

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

4 Structure of the Correctness Proof

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

4.1 Going from Informal Proofs to Formal Proofs

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

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

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

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

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

5 Conclusion

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

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

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

References

[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 $(N \in Nat) \land (N > 0)$
$Proc \triangleq 1..N$
$NotAnInput \triangleq$ CHOOSE $c : c \notin Inputs$
VARIABLES *inputs*, *output*

MODULE *Inner*

VARIABLES *allInput*, *chosen*

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

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

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

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

$IS(chosen, allInput) \triangleq$ 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* ⇒ *nat set* **and**
  IsMajority :: *Disk set* ⇒ *bool*
where
  NotAnInput: *NotAnInput* ∉ *Inputs* **and**
  InputsOrNi: (*UNIV* :: *InputsOrNi set*) = *Inputs* ∪ {*NotAnInput*} **and**
  Ballot-nzero: ∀ *p*. *0* ∉ *Ballot p* **and**
  Ballot-disj: ∀ *p q*. *p* ≠ *q* ⟶ (*Ballot p*) ∩ (*Ballot q*) = {} **and**
  Disk-isMajority: *IsMajority*(*UNIV*) **and**
  majorities-intersect:
    ∀ *S T*. *IsMajority*(*S*) ∧ *IsMajority*(*T*) ⟶ *S* ∩ *T* ≠ {}

lemma *ballots-not-zero* [*simp*]:
  b ∈ *Ballot p* ⟹ *0* < *b*
⟨*proof*⟩

lemma *majority-nonempty* [*simp*]: *IsMajority*(*S*) ⟹ *S* ≠ {}
⟨*proof*⟩

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

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

definition *InitDB* :: *DiskBlock*
  where *InitDB* = ⦇ *mbal* = *0*, *bal* = *0*, *inp* = *NotAnInput* ⦈

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

record
  state =

```
  inpt :: Proc ⇒ InputsOrNi
  outpt :: Proc ⇒ InputsOrNi
  disk :: Disk ⇒ Proc ⇒ DiskBlock
  dblock :: Proc ⇒ DiskBlock
  phase :: Proc ⇒ nat
  disksWritten :: Proc ⇒ Disk set
  blocksRead :: Proc ⇒ Disk ⇒ BlockProc set

  allInput  :: InputsOrNi set
  chosen    :: InputsOrNi


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

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

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

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

definition InitializePhase :: state ⇒ state ⇒ Proc ⇒ bool
 where
 InitializePhase s s' p =
  (disksWritten s' = (disksWritten s)(p := {})
  & blocksRead s' = (blocksRead s)(p := (λd. {})))

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

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 $blocksSeen\ s\ p = allBlocksRead\ s\ p \cup \{dblock\ s\ p\}$

definition $nonInitBlks :: state \Rightarrow Proc \Rightarrow DiskBlock\ set$
where$nonInitBlks\ s\ p = \{bs\ .\ bs \in blocksSeen\ s\ p \wedge inp\ bs \in Inputs\}$

definition $maxBlk :: state \Rightarrow Proc \Rightarrow DiskBlock$
where
$maxBlk\ s\ p =$
$(SOME\ b.\ b \in nonInitBlks\ s\ p \wedge (\forall\, c \in nonInitBlks\ s\ p.\ bal\ c \leq bal\ b))$

definition $EndPhase1 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$
where
$EndPhase1\ s\ s'\ p =$
$(IsMajority\ \{d\ .\ d \in disksWritten\ s\ p$

$\wedge (\forall q \in UNIV - \{p\}.\ hasRead\ s\ p\ d\ q)\}$
$\wedge\ phase\ s\ p = 1$
$\wedge\ dblock\ s' = (dblock\ s)\ (p := dblock\ s\ p$
$(\!|\ bal := mbal(dblock\ s\ p),$
$inp :=$
$(if\ nonInitBlks\ s\ p = \{\}$
$then\ inpt\ s\ p$
$else\ inp\ (maxBlk\ s\ p))$
$|\!)\ )$
$\wedge\ outpt\ s' = outpt\ s$
$\wedge\ phase\ s' = (phase\ s)\ (p := phase\ s\ p + 1)$
$\wedge\ InitializePhase\ s\ s'\ p$
$\wedge\ inpt\ s' = inpt\ s \wedge disk\ s' = disk\ s)$

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

definition $EndPhase1or2 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$
where $EndPhase1or2\ s\ s'\ p = (EndPhase1\ s\ s'\ p \vee EndPhase2\ s\ s'\ p)$

definition $Fail :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$
where
$Fail\ s\ s'\ p =$
$(\exists ip \in Inputs.\ inpt\ s' = (inpt\ s)\ (p := ip)$
$\wedge\ phase\ s' = (phase\ s)\ (p := 0)$
$\wedge\ dblock\ s'= (dblock\ s)\ (p := InitDB)$
$\wedge\ outpt\ s' = (outpt\ s)\ (p := NotAnInput)$
$\wedge\ InitializePhase\ s\ s'\ p$
$\wedge\ disk\ s'= disk\ s)$

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

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

definition $Next :: state \Rightarrow state \Rightarrow bool$
where
$Next\ s\ s' = (\exists\, p.$
$StartBallot\ s\ s'\ p$
$\vee\ (\exists\, d.\quad Phase0Read\ s\ s'\ p\ d$
$\vee\ Phase1or2Write\ s\ s'\ p\ d$
$\vee\ (\exists\, q.\ q \neq p \wedge Phase1or2Read\ s\ s'\ p\ d\ q))$
$\vee\ EndPhase1or2\ s\ s'\ p$
$\vee\ Fail\ s\ s'\ p$
$\vee\ EndPhase0\ s\ s'\ p)$

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 $HNextPart :: state \Rightarrow state => bool$
where
$HNextPart\ s\ s' =$
$(chosen\ s' =$
$(if\ chosen\ s \neq NotAnInput \vee (\forall\, p.\ outpt\ s'\ p = NotAnInput\ )$
$then\ chosen\ s$
$else\ outpt\ s'\ (SOME\ p.\ outpt\ s'\ p \neq NotAnInput))$
$\wedge\ allInput\ s' = allInput\ s \cup (range\ (inpt\ s')))$

definition $HNext :: state \Rightarrow state \Rightarrow bool$
where
$HNext\ s\ s' =$

$(Next\ s\ s'$
$\wedge\ HNextPart\ s\ s')$

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

definition
$HPhase1or2ReadThen :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool$ **where**
$HPhase1or2ReadThen\ s\ s'\ p\ d\ q = (Phase1or2ReadThen\ s\ s'\ p\ d\ q \wedge HNextPart\ s\ s')$

definition
$HEndPhase1 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$ **where**
$HEndPhase1\ s\ s'\ p = (EndPhase1\ s\ s'\ p \wedge HNextPart\ s\ s')$

definition
$HStartBallot :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$ **where**
$HStartBallot\ s\ s'\ p = (StartBallot\ s\ s'\ p \wedge HNextPart\ s\ s')$

definition
$HPhase1or2Write :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow bool$ **where**
$HPhase1or2Write\ s\ s'\ p\ d = (Phase1or2Write\ s\ s'\ p\ d \wedge HNextPart\ s\ s')$

definition
$HPhase1or2ReadElse :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow Proc \Rightarrow bool$ **where**
$HPhase1or2ReadElse\ s\ s'\ p\ d\ q = (Phase1or2ReadElse\ s\ s'\ p\ d\ q \wedge HNextPart\ s\ s')$

definition
$HEndPhase2 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$ **where**
$HEndPhase2\ s\ s'\ p = (EndPhase2\ s\ s'\ p \wedge HNextPart\ s\ s')$

definition
$HFail :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$ **where**
$HFail\ s\ s'\ p = (Fail\ s\ s'\ p\ \wedge HNextPart\ s\ s')$

definition
$HPhase0Read :: state \Rightarrow state \Rightarrow Proc \Rightarrow Disk \Rightarrow bool$ **where**
$HPhase0Read\ s\ s'\ p\ d = (Phase0Read\ s\ s'\ p\ d \wedge HNextPart\ s\ s')$

definition
$HEndPhase0 :: state \Rightarrow state \Rightarrow Proc \Rightarrow bool$ **where**
$HEndPhase0\ s\ s'\ p = (EndPhase0\ s\ s'\ p \wedge HNextPart\ s\ s')$

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 $EndPhase1$.

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

lemma *HNextPart-Inv1*: ⟦ *HInv1 s*; *HNextPart s s′*; *Inv1 s′* ⟧ ⟹ *HInv1 s′*
  ⟨*proof*⟩

theorem *HInit-HInv1*: *HInit s* ⟶ *HInv1 s*
  ⟨*proof*⟩

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*⟩

theorem *HPhase1or2ReadThen-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HPhase1or2ReadThen s s′ p d q*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HEndPhase1-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HEndPhase1 s s′ p*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HStartBallot-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HStartBallot s s′ p*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HPhase1or2Write-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HPhase1or2Write s s′ p d*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv1*:
assumes *act*: *HPhase1or2ReadElse s s′ p d q*
and *inv1*: *HInv1 s*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HEndPhase2-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HEndPhase2 s s′ p*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HFail-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HFail s s′ p*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HPhase0Read-HInv1*:
assumes *inv1*: *HInv1 s*
and *act*: *HPhase0Read s s′ p d*
shows *HInv1 s′*
⟨*proof*⟩

theorem *HEndPhase0-HInv1*:

assumes *inv1*: *HInv1 s*
and *act*: *HEndPhase0 s s' p*
shows *HInv1 s'*
⟨*proof*⟩

declare *HInv1-def* [*simp del*]

$HInv1$ is an invariant of $HNext$

lemma *I2a*:
assumes *nxt*: *HNext s s'*
and *inv*: *HInv1 s*
shows *HInv1 s'*
⟨*proof*⟩

end

theory *DiskPaxos-Inv2* **imports** *DiskPaxos-Inv1* **begin**

C.2 Invariant 2

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

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

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

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

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

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

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

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

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

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

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 $]\!] \Longrightarrow$ *blocksOf* s' $r \subseteq$ *blocksOf* s r
⟨*proof*⟩

theorem *HPhase1or2ReadThen-Inv2a*:
assumes *inv*: *Inv2a* s
and *act*: *HPhase1or2ReadThen* s s' p d q
shows *Inv2a* s'
⟨*proof*⟩

lemma *InitializePhase-rdBy*:
InitializePhase s s' $p \Longrightarrow$ *rdBy* s' *pp qq dd* $\subseteq$ *rdBy* s *pp qq dd*
⟨*proof*⟩

lemma *HStartBallot-blocksOf*:
HStartBallot s s' $p \Longrightarrow$ *blocksOf* s' $q \subseteq$ *blocksOf* s $q \cup \{$*dblock* s' $q\}$
⟨*proof*⟩

lemma *HStartBallot-Inv2a-dblock*:
assumes *act*: *HStartBallot* s s' p
and *inv2a*: *Inv2a-innermost* s p (*dblock* s p)
shows *Inv2a-innermost* s' p (*dblock* s' p)
⟨*proof*⟩

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*⟩

theorem *HStartBallot-Inv2a*:
assumes *inv*: *Inv2a* s
and *act*: *HStartBallot* s s' p
shows *Inv2a* s'
⟨*proof*⟩

lemma *HPhase1or2Write-blocksOf*:
$[\![$ *HPhase1or2Write* s s' p d $]\!] \Longrightarrow$ *blocksOf* s' $r \subseteq$ *blocksOf* s r
⟨*proof*⟩

theorem *HPhase1or2Write-Inv2a*:
assumes *inv*: *Inv2a* s
and *act*: *HPhase1or2Write* s s' p d

shows *Inv2a s'*
⟨*proof*⟩

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

lemma *HEndPhase2-blocksOf*:
$[\![$ *HEndPhase2 s s' p* $]\!] \Longrightarrow$ *blocksOf s' q* $\subseteq$ *blocksOf s q*
⟨*proof*⟩

theorem *HEndPhase2-Inv2a*:
assumes *inv*: *Inv2a s*
and *act*: *HEndPhase2 s s' p*
shows *Inv2a s'*
⟨*proof*⟩

lemma *HFail-blocksOf*:
HFail s s' p $\Longrightarrow$ *blocksOf s' q* $\subseteq$ *blocksOf s q* $\cup$ {*dblock s' q*}
⟨*proof*⟩

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*⟩

theorem *HFail-Inv2a*:
assumes *inv*: *Inv2a s*
and *act*: *HFail s s' p*
shows *Inv2a s'*
⟨*proof*⟩

lemma *HPhase0Read-blocksOf*:
HPhase0Read s s' p d $\Longrightarrow$ *blocksOf s' q* $\subseteq$ *blocksOf s q*
⟨*proof*⟩

theorem *HPhase0Read-Inv2a*:
assumes *inv*: *Inv2a s*
and *act*: *HPhase0Read s s' p d*
shows *Inv2a s'*
⟨*proof*⟩

lemma *HEndPhase0-blocksOf*:
HEndPhase0 s s' p $\Longrightarrow$ *blocksOf s' q* $\subseteq$ *blocksOf s q* $\cup$ {*dblock s' q*}
⟨*proof*⟩

lemma *HEndPhase0-blocksRead*:
assumes *act*: *HEndPhase0 s s′ p*
shows $\exists$ *d*. *blocksRead s p d* $\neq$ {}
⟨*proof*⟩

*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* $\Longrightarrow$ *?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*⟩

lemma *HEndPhase0-dblock-allBlocksRead*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv1*: *Inv1 s*
shows *dblock s′ p* ∈ (λ*x*. *x* (|*mbal*:= *mbal*(*dblock s′ p*)|)) ‘ *allBlocksRead s p*
⟨*proof*⟩

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*⟩

lemma *HEndPhase0-Inv2a-allBlocksRead*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv2a*: *Inv2a-inner s p*
and *inv2c*: *Inv2c-inner s p*
shows ∀ *t* ∈ (λ*x*. *x* (|*mbal*:= *mbal* (*dblock s′ p*)|)) ‘ *allBlocksRead s p*.
Inv2a-innermost s p t
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *HEndPhase1-blocksOf*:
$HEndPhase1\ s\ s'\ p \Longrightarrow blocksOf\ s'\ q \subseteq blocksOf\ s\ q \cup \{dblock\ s'\ q\}$
⟨*proof*⟩

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*⟩

lemma *blocksOf-nonInitBlks*:
$(\forall\, p\ bk.\ bk \in blocksOf\ s\ p \longrightarrow P\ bk)$
$\Longrightarrow bk \in nonInitBlks\ s\ p \longrightarrow P\ bk$
⟨*proof*⟩

lemma *maxBlk-allInput*:
assumes *inv*: *Inv2a s*
and *mblk*: $maxBlk\ s\ p \in nonInitBlks\ s\ p$
shows $inp\ (maxBlk\ s\ p) \in allInput\ s$
⟨*proof*⟩

lemma *HEndPhase1-dblock-allInput*:
assumes *act*: *HEndPhase1 s s′ p*
and *inv1*: *HInv1 s*
and *inv2*: *Inv2a s*
shows *inp′*: $inp\ (dblock\ s'\ p) \in allInput\ s'$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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* $\longrightarrow$ *Inv2b* *s*
⟨*proof*⟩

theorem *HPhase1or2ReadThen-Inv2b*:
$[\![$ *Inv2b* *s*; *HPhase1or2ReadThen* *s* s' *p* *d* *q* $]\!]$
$\Longrightarrow$ *Inv2b* s'
⟨*proof*⟩

theorem *HStartBallot-Inv2b*:
$[\![$ *Inv2b* *s*; *HStartBallot* *s* s' *p* $]\!]$
$\Longrightarrow$ *Inv2b* s'
⟨*proof*⟩

theorem *HPhase1or2Write-Inv2b*:
$[\![$ *Inv2b* *s*; *HPhase1or2Write* *s* s' *p* *d* $]\!]$
$\Longrightarrow$ *Inv2b* s'
⟨*proof*⟩

theorem *HPhase1or2ReadElse-Inv2b*:
$[\![$ *Inv2b* *s*; *HPhase1or2ReadElse* *s* s' *p* *d* *q* $]\!]$
$\Longrightarrow$ *Inv2b* s'
⟨*proof*⟩

theorem *HEndPhase1-Inv2b*:
$[\![$ *Inv2b* *s*; *HEndPhase1* *s* s' *p* $]\!]$ $\Longrightarrow$ *Inv2b* s'
⟨*proof*⟩

theorem *HFail-Inv2b*:
$\llbracket$ *Inv2b s*; *HFail s s′ p* $\rrbracket$
$\Longrightarrow$ *Inv2b s′*
⟨*proof*⟩

theorem *HEndPhase2-Inv2b*:
$\llbracket$ *Inv2b s*; *HEndPhase2 s s′ p* $\rrbracket \Longrightarrow$ *Inv2b s′*
⟨*proof*⟩

theorem *HPhase0Read-Inv2b*:
$\llbracket$ *Inv2b s*; *HPhase0Read s s′ p d* $\rrbracket \Longrightarrow$ *Inv2b s′*
⟨*proof*⟩

theorem *HEndPhase0-Inv2b*:
$\llbracket$ *Inv2b s*; *HEndPhase0 s s′ p* $\rrbracket \Longrightarrow$ *Inv2b s′*
⟨*proof*⟩

C.2.3 Proofs of Invariant 2 c

theorem *HInit-Inv2c*: *HInit s* $\longrightarrow$ *Inv2c s*
⟨*proof*⟩

lemma *HNextPart-Inv2c-chosen*:
assumes *hnp*: *HNextPart s s′*
and *inv2c*: *Inv2c s*
and *outpt′*: $\forall p$. *outpt s′ p* = (*if phase s′ p* = *3*
then inp(*dblock s′ p*)
else NotAnInput)
and *inp-dblk*: $\forall p$. *inp* (*dblock s′ p*) $\in$ *allInput s′* $\cup$ {*NotAnInput*}
shows *chosen s′* $\in$ *allInput s′* $\cup$ {*NotAnInput*}
⟨*proof*⟩

lemma *HNextPart-chosen*:
assumes *hnp*: *HNextPart s s′*
shows *chosen s′* = *NotAnInput* $\longrightarrow$ ($\forall p$. *outpt s′ p* = *NotAnInput*)
⟨*proof*⟩

lemma *HNextPart-allInput*:
$\llbracket$ *HNextPart s s′*; *Inv2c s* $\rrbracket \Longrightarrow \forall p$. *inpt s′ p* $\in$ *allInput s′*
⟨*proof*⟩

theorem *HPhase1or2ReadThen-Inv2c*:
assumes *inv*: *Inv2c s*
and *act*: *HPhase1or2ReadThen s s′ p d q*
and *inv2a*: *Inv2a s*
shows *Inv2c s′*
⟨*proof*⟩

theorem *HStartBallot-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HStartBallot s s′ p*
  and *inv2a*: *Inv2a s*
  shows *Inv2c s′*
⟨*proof*⟩

theorem *HPhase1or2Write-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HPhase1or2Write s s′ p d*
  and *inv2a*: *Inv2a s*
  shows *Inv2c s′*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-Inv2c*:
  $\llbracket$ *Inv2c s*; *HPhase1or2ReadElse s s′ p d q*; *Inv2a s* $\rrbracket \Longrightarrow$ *Inv2c s′*
  ⟨*proof*⟩

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*⟩

theorem *HEndPhase2-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HEndPhase2 s s′ p*
  and *inv2a*: *Inv2a s*
  shows *Inv2c s′*
⟨*proof*⟩

theorem *HFail-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HFail s s′ p*
  and *inv2a*: *Inv2a s*
  shows *Inv2c s′*
⟨*proof*⟩

theorem *HPhase0Read-Inv2c*:
  assumes *inv*: *Inv2c s*
  and *act*: *HPhase0Read s s′ p d*
  and *inv2a*: *Inv2a s*
  shows *Inv2c s′*
⟨*proof*⟩

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*⟩

theorem *HInit-HInv2*:
HInit s ⟹ *HInv2 s*
⟨*proof*⟩

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

lemma *I2b*:
assumes *nxt*: *HNext s s′*
and *inv*: *HInv1 s* ∧ *HInv2 s*
shows *HInv2 s′*
⟨*proof*⟩

end

theory *DiskPaxos-Inv3* **imports** *DiskPaxos-Inv2* **begin**

C.3 Invariant 3

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

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

⟨*proof*⟩

lemma *InitPhase-HInv3-p*:
⟦ *InitializePhase s s′ p*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *InitPhase-HInv3-q*:
⟦ *InitializePhase s s′ q* ; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *HInv3-L-sym*: *HInv3-L s p q d* ⟹ *HInv3-L s q p d*
⟨*proof*⟩

lemma *HInv3-R-sym*: *HInv3-R s p q d* ⟹ *HInv3-R s q p d*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

lemma *StartBallot-HInv3-p*:
⟦ *StartBallot s s′ p*; *HInv3-L s′ p q d* ⟧
⟹ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *StartBallot-HInv3-q*:
⟦ *StartBallot s s′ q*; *HInv3-L s′ p q d* ⟧
⟹ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *StartBallot-HInv3-nL*:
  ⟦ *StartBallot s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* ⟧
  ⟹ ¬*HInv3-L s′ p q d*
  ⟨*proof*⟩

lemma *StartBallot-HInv3-R*:
  ⟦ *StartBallot s s′ t*; *HInv3-R s p q d*; *t*≠*p*; *t*≠ *q* ⟧
  ⟹ *HInv3-R s′ p q d*
  ⟨*proof*⟩

lemma *StartBallot-HInv3-t*:
  ⟦ *StartBallot s s′ t*; *HInv3-inner s p q d*; *t*≠*p*; *t*≠ *q* ⟧
  ⟹ *HInv3-inner s′ p q d*
  ⟨*proof*⟩

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*⟩

theorem *HStartBallot-HInv3*:
  ⟦ *HStartBallot s s′ p*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  ⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv3*:
  ⟦ *HPhase1or2ReadElse s s′ p d q*; *HInv3 s* ⟧ ⟹ *HInv3 s′*
  ⟨*proof*⟩

theorem *HPhase1or2Write-HInv3*:
  assumes *act*: *HPhase1or2Write s s′ p d*
  and *inv*: *HInv3 s*
  shows *HInv3 s′*
⟨*proof*⟩

lemma *EndPhase1-HInv3-p*:
  ⟦ *EndPhase1 s s′ p*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *EndPhase1-HInv3-q*:
  ⟦ *EndPhase1 s s′ q*; *HInv3-L s′ p q d* ⟧ ⟹ *HInv3-R s′ p q d*
  ⟨*proof*⟩

lemma *EndPhase1-HInv3-nL*:
  ⟦ *EndPhase1 s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* ⟧
  ⟹ ¬*HInv3-L s′ p q d*
  ⟨*proof*⟩

lemma *EndPhase1-HInv3-R*:

$[\![$ *EndPhase1 s s' t*; *HInv3-R s p q d*; $t{\neq}p$; $t{\neq}$ *q* $]\!]$
$\Longrightarrow$ *HInv3-R s' p q d*
⟨*proof*⟩

lemma *EndPhase1-HInv3-t*:
$[\![$ *EndPhase1 s s' t*; *HInv3-inner s p q d*; $t{\neq}p$; $t{\neq}$ *q* $]\!]$
$\Longrightarrow$ *HInv3-inner s' p q d*
⟨*proof*⟩

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*⟩

theorem *HEndPhase1-HInv3*:
$[\![$ *HEndPhase1 s s' p*; *HInv3 s* $]\!]$ $\Longrightarrow$ *HInv3 s'*
⟨*proof*⟩

lemma *EndPhase2-HInv3-p*:
$[\![$ *EndPhase2 s s' p*; *HInv3-L s' p q d* $]\!]$ $\Longrightarrow$ *HInv3-R s' p q d*
⟨*proof*⟩

lemma *EndPhase2-HInv3-q*:
$[\![$ *EndPhase2 s s' q*; *HInv3-L s' p q d* $]\!]$ $\Longrightarrow$ *HInv3-R s' p q d*
⟨*proof*⟩

lemma *EndPhase2-HInv3-nL*:
$[\![$ *EndPhase2 s s' t*; ¬*HInv3-L s p q d*; $t{\neq}p$; $t{\neq}$ *q* $]\!]$
$\Longrightarrow$ ¬*HInv3-L s' p q d*
⟨*proof*⟩

lemma *EndPhase2-HInv3-R*:
$[\![$ *EndPhase2 s s' t*; *HInv3-R s p q d*; $t{\neq}p$; $t{\neq}$ *q* $]\!]$
$\Longrightarrow$ *HInv3-R s' p q d*
⟨*proof*⟩

lemma *EndPhase2-HInv3-t*:
$[\![$ *EndPhase2 s s' t*; *HInv3-inner s p q d*; $t{\neq}p$; $t{\neq}$ *q* $]\!]$
$\Longrightarrow$ *HInv3-inner s' p q d*
⟨*proof*⟩

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*⟩

theorem *HEndPhase2-HInv3*:

$\llbracket$ *HEndPhase2 s s′ p*; *HInv3 s* $\rrbracket \Longrightarrow$ *HInv3 s′*
⟨*proof*⟩

lemma *Fail-HInv3-p*:
$\llbracket$ *Fail s s′ p*; *HInv3-L s′ p q d* $\rrbracket \Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *Fail-HInv3-q*:
$\llbracket$ *Fail s s′ q*; *HInv3-L s′ p q d* $\rrbracket \Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *Fail-HInv3-nL*:
$\llbracket$ *Fail s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\Longrightarrow$ ¬*HInv3-L s′ p q d*
⟨*proof*⟩

lemma *Fail-HInv3-R*:
$\llbracket$ *Fail s s′ t*; *HInv3-R s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *Fail-HInv3-t*:
$\llbracket$ *Fail s s′ t*; *HInv3-inner s p q d*; *t*≠*p*; *t*≠ *q* $\rrbracket$
$\Longrightarrow$ *HInv3-inner s′ p q d*
⟨*proof*⟩

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*⟩

theorem *HFail-HInv3*:
$\llbracket$ *HFail s s′ p*; *HInv3 s* $\rrbracket \Longrightarrow$ *HInv3 s′*
⟨*proof*⟩

theorem *HPhase0Read-HInv3*:
assumes *act*: *HPhase0Read s s′ p d*
and *inv*: *HInv3 s*
shows *HInv3 s′*
⟨*proof*⟩

lemma *EndPhase0-HInv3-p*:
$\llbracket$ *EndPhase0 s s′ p*; *HInv3-L s′ p q d* $\rrbracket$
$\Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *EndPhase0-HInv3-q*:
$\llbracket$ *EndPhase0 s s′ q*; *HInv3-L s′ p q d* $\rrbracket$

$\Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *EndPhase0-HInv3-nL*:
⟦ *EndPhase0 s s′ t*; ¬*HInv3-L s p q d*; *t*≠*p*; *t*≠ *q* ⟧
$\Longrightarrow$ ¬*HInv3-L s′ p q d*
⟨*proof*⟩

lemma *EndPhase0-HInv3-R*:
⟦ *EndPhase0 s s′ t*; *HInv3-R s p q d*; *t*≠*p*; *t*≠ *q* ⟧
$\Longrightarrow$ *HInv3-R s′ p q d*
⟨*proof*⟩

lemma *EndPhase0-HInv3-t*:
⟦ *EndPhase0 s s′ t*; *HInv3-inner s p q d*; *t*≠*p*; *t*≠ *q* ⟧
$\Longrightarrow$ *HInv3-inner s′ p q d*
⟨*proof*⟩

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*⟩

theorem *HEndPhase0-HInv3*:
⟦ *HEndPhase0 s s′ p*; *HInv3 s* ⟧ $\Longrightarrow$ *HInv3 s′*
⟨*proof*⟩

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

lemma *I2c*:
assumes *nxt*: *HNext s s′*
and *inv*: *HInv1 s* ∧ *HInv2 s* ∧ *HInv3 s*
shows *HInv3 s′* ⟨*proof*⟩

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*
  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*)
  ∧ *bal*(*disk s d p*) ≤ *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*
  ⟨*proof*⟩

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*⟩

lemma *HStartBallot-HInv4a2*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4a2 s p*
shows *HInv4a2 s′ p*
⟨*proof*⟩

lemma *HStartBallot-HInv4a-p*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4a s p*
and *inv2a*: *Inv2a-inner s′ p*
shows *HInv4a s′ p*
⟨*proof*⟩

lemma *HStartBallot-HInv4a-q*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4a s q*
and *pnq*: $p \neq q$
shows *HInv4a s′ q*
⟨*proof*⟩

theorem *HStartBallot-HInv4a*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4a s q*
and *inv2a*: *Inv2a s′*
shows *HInv4a s′ q*
⟨*proof*⟩

lemma *Phase1or2Write-HInv4a1*:
⟦ *Phase1or2Write s s′ p d*; *HInv4a1 s q* ⟧ ⟹ *HInv4a1 s′ q*
⟨*proof*⟩

lemma *Phase1or2Write-HInv4a2*:
⟦ *Phase1or2Write s s′ p d*; *HInv4a2 s q* ⟧ ⟹ *HInv4a2 s′ q*
⟨*proof*⟩

theorem *HPhase1or2Write-HInv4a*:
assumes *act*: *HPhase1or2Write s s′ p d*
and *inv*: *HInv4a s q*
shows *HInv4a s′ q*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-HInv4a1-p*:
assumes *act*: *HPhase1or2ReadThen s s′ p d q*
and *inv*: *HInv4a1 s p*
shows *HInv4a1 s′ p*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-HInv4a2*:
  ⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4a2 s q* ⟧ ⟹ *HInv4a2 s′ q*
  ⟨*proof*⟩

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*⟩

lemma *HPhase1or2ReadThen-HInv4a-q*:
  assumes *act*: *HPhase1or2ReadThen s s′ p d r*
  and *inv*: *HInv4a s q*
  and *pnq*: $p \neq q$
  shows *HInv4a s′ q*
⟨*proof*⟩

theorem *HPhase1or2ReadThen-HInv4a*:
  ⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4a s q*; *Inv2b s* ⟧ ⟹ *HInv4a s′ q*
  ⟨*proof*⟩

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*⟩

lemma *HEndPhase1-HInv4a1*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a1 s p*
  shows *HInv4a1 s′ p*
⟨*proof*⟩

lemma *HEndPhase1-HInv4a2*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a2 s p*
  and *inv2a*: *Inv2a s*
  shows *HInv4a2 s′ p*
⟨*proof*⟩

lemma *HEndPhase1-HInv4a-p*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4a s p*
  and *inv2a*: *Inv2a s*
  shows *HInv4a s′ p*
⟨*proof*⟩

lemma *HEndPhase1-HInv4a-q*:
assumes *act*: *HEndPhase1 s s′ p*
and *inv*: *HInv4a s q*
and *pnq*: $p \neq q$
shows *HInv4a s′ q*
⟨*proof*⟩

theorem *HEndPhase1-HInv4a*:
$[\![$ *HEndPhase1 s s′ p*; *HInv4a s q*; *Inv2a s* $]\!] \Longrightarrow$ *HInv4a s′ q*
⟨*proof*⟩

theorem *HFail-HInv4a*:
$[\![$ *HFail s s′ p*; *HInv4a s q* $]\!] \Longrightarrow$ *HInv4a s′ q*
⟨*proof*⟩

theorem *HPhase0Read-HInv4a*:
$[\![$ *HPhase0Read s s′ p d*; *HInv4a s q* $]\!] \Longrightarrow$ *HInv4a s′ q*
⟨*proof*⟩

theorem *HEndPhase2-HInv4a*:
$[\![$ *HEndPhase2 s s′ p*; *HInv4a s q* $]\!] \Longrightarrow$ *HInv4a s′ q*
⟨*proof*⟩

lemma *allSet*:
assumes *aPQ*: $\forall\, a.\ \forall\, r \in P\ a.\ Q\ r$ **and** *rb*: $rb \in P\ d$
shows *Q rb*
⟨*proof*⟩

lemma *EndPhase0-44*:
assumes *act*: *EndPhase0 s s′ p*
and *bk*: $bk \in$ *blocksOf s p*
and *inv4d*: *HInv4d s p*
and *inv2c*: *Inv2c-inner s p*
shows $\exists\, d.\ \exists\, rb \in$ *blocksRead s p d*. *bal bk* $\leq$ *mbal*(*block rb*)
⟨*proof*⟩

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*⟩

lemma *hasRead-allBlks*:
assumes *inv2c*: *Inv2c-inner s p*
and *phase*: *phase s p* = *0*
shows ($\forall\, d \in \{d.$ *hasRead s p d p*$\}$. *disk s d p* $\in$ *allBlocksRead s p*)
⟨*proof*⟩

lemma *HEndPhase0-41*:
assumes *act*: $HEndPhase0\ s\ s'\ p$
and *inv1*: $Inv1\ s$
and *inv2c*: $Inv2c\text{-}inner\ s\ p$
shows $\exists\, D{\in}MajoritySet.\ \forall\, d{\in}D.\quad mbal(disk\ s\ d\ p) \le mbal(dblock\ s'\ p)$
$\wedge\ bal(disk\ s\ d\ p) \le bal(dblock\ s'\ p)$
⟨*proof*⟩

lemma *Majority-exQ*:
assumes *asm1*: $\exists\, D \in MajoritySet.\ \forall\, d{\in}D.\ P\ d$
shows $\forall\, D{\in}MajoritySet.\exists\, d{\in}D.\ P\ d$
⟨*proof*⟩

lemma *HEndPhase0-HInv4a2-p*:
assumes *act*: $HEndPhase0\ s\ s'\ p$
and *inv1*: $Inv1\ s$
and *inv2c*: $Inv2c\text{-}inner\ s\ p$
shows $HInv4a2\ s'\ p$
⟨*proof*⟩

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*⟩

lemma *HEndPhase0-HInv4a-q*:
assumes *act*: $HEndPhase0\ s\ s'\ p$
and *inv*: $HInv4a\ s\ q$
and *pnq*: $p{\ne}q$
shows $HInv4a\ s'\ q$
⟨*proof*⟩

theorem *HEndPhase0-HInv4a*:
$[\![\ HEndPhase0\ s\ s'\ p;\ HInv4a\ s\ q;\ HInv4d\ s\ p;$
$Inv2a\ s;\ \ Inv1\ s;\ \ \ Inv2a\ s;\ Inv2c\ s]\!]$
$\Longrightarrow HInv4a\ s'\ q$
⟨*proof*⟩

C.4.2 Proofs of Invariant 4b

lemma *blocksRead-allBlocksRead*:
$rb \in blocksRead\ s\ p\ d \Longrightarrow block\ rb \in allBlocksRead\ s\ p$

⟨*proof*⟩

lemma *HEndPhase0-dblock-mbal*:
$\llbracket$ *HEndPhase0 s s′ p* $\rrbracket$
$\Longrightarrow \forall\, br \in allBlocksRead\ s\ p.\ mbal\ br < mbal(dblock\ s'\ p)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *HEndPhase0-HInv4b-q*:
assumes *act*: *HEndPhase0 s s′ p*
and *pnq*: $p \neq q$
and *inv*: *HInv4b s q*
shows *HInv4b s′ q*
⟨*proof*⟩

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*⟩

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*⟩

lemma *HStartBallot-HInv4b-q*:
  assumes *act*: *HStartBallot s s′ p*
  and *pnq*: *p*≠*q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

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*
⟨*proof*⟩

theorem *HPhase1or2Write-HInv4b*:
  ⟦ *HPhase1or2Write s s′ p d*; *HInv4b s q* ⟧ ⟹ *HInv4b s′ q*
  ⟨*proof*⟩

lemma *HPhase1or2ReadThen-HInv4b-p*:
  assumes *act*: *HPhase1or2ReadThen s s′ p d q*
  and *inv*: *HInv4b s p*
  shows *HInv4b s′ p*
⟨*proof*⟩

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*
  ⟨*proof*⟩

theorem *HPhase1or2ReadThen-HInv4b*:
  ⟦ *HPhase1or2ReadThen s s′ p d q*; *HInv4b s r*⟧ ⟹ *HInv4b s′ r*
  ⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv4b*:
  ⟦ *HPhase1or2ReadElse s s′ p d q*; *HInv4b s r*;
    Inv2a s; *HInv4a s p* ⟧
  ⟹ *HInv4b s′ r*

⟨*proof*⟩

lemma *HEndPhase1-HInv4b-p*:
  HEndPhase1 s s′ p ⟹ *HInv4b s′ p*
  ⟨*proof*⟩

lemma *HEndPhase1-HInv4b-q*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *pnq*: *p≠q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

theorem *HEndPhase1-HInv4b*:
  assumes *act*: *HEndPhase1 s s′ p*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

lemma *HEndPhase2-HInv4b-p*:
  HEndPhase2 s s′ p ⟹ *HInv4b s′ p*
  ⟨*proof*⟩

lemma *HEndPhase2-HInv4b-q*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *pnq*: *p≠q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

theorem *HEndPhase2-HInv4b*:
  assumes *act*: *HEndPhase2 s s′ p*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

lemma *HFail-HInv4b-p*:
  HFail s s′ p ⟹ *HInv4b s′ p*
  ⟨*proof*⟩

lemma *HFail-HInv4b-q*:
  assumes *act*: *HFail s s′ p*
  and *pnq*: *p≠q*
  and *inv*: *HInv4b s q*
  shows *HInv4b s′ q*
⟨*proof*⟩

theorem *HFail-HInv4b*:
  assumes *act*: *HFail s s′ p*

and *inv*: *HInv4b s q*
shows *HInv4b s′ q*
⟨*proof*⟩

lemma *HPhase0Read-HInv4b-p*:
HPhase0Read s s′ p d ⟹ *HInv4b s′ p*
⟨*proof*⟩

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*⟩

theorem *HPhase0Read-HInv4b*:
assumes *act*: *HPhase0Read s s′ p d*
and *inv*: *HInv4b s q*
shows *HInv4b s′ q*
⟨*proof*⟩

C.4.3 Proofs of Invariant 4c

lemma *HStartBallot-HInv4c-p*:
⟦ *HStartBallot s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

lemma *HStartBallot-HInv4c-q*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4c s q*
and *pnq*: *p*≠*q*
shows *HInv4c s′ q*
⟨*proof*⟩

theorem *HStartBallot-HInv4c*:
⟦ *HStartBallot s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

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*⟩

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*⟩

theorem *HPhase1or2Write-HInv4c*:
⟦ *HPhase1or2Write s s′ p d*; *HInv4c s q*; *Inv2c s* ⟧
⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-HInv4c-p*:
⟦ *HPhase1or2ReadThen s s′ p d q*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

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*⟩

theorem *HPhase1or2ReadThen-HInv4c*:
⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4c s q*⟧
⟹ *HInv4c s′ q*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv4c*:
⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HEndPhase1-HInv4c-p*:
assumes *act*: *HEndPhase1 s s′ p*
and *inv2b*: *Inv2b s*
shows *HInv4c s′ p*
⟨*proof*⟩

lemma *HEndPhase1-HInv4c-q*:
assumes *act*: *HEndPhase1 s s′ p*
and *inv*: *HInv4c s q*
and *pnq*: *p*≠*q*
shows *HInv4c s′ q*
⟨*proof*⟩

theorem *HEndPhase1-HInv4c*:
⟦ *HEndPhase1 s s′ p*; *HInv4c s q*; *Inv2b s*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HEndPhase2-HInv4c-p*:
⟦ *HEndPhase2 s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

lemma *HEndPhase2-HInv4c-q*:
assumes *act*: *HEndPhase2 s s′ p*
and *inv*: *HInv4c s q*
and *pnq*: *p≠q*
shows *HInv4c s′ q*
⟨*proof*⟩

theorem *HEndPhase2-HInv4c*:
⟦ *HEndPhase2 s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HFail-HInv4c-p*:
⟦ *HFail s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

lemma *HFail-HInv4c-q*:
assumes *act*: *HFail s s′ p*
and *inv*: *HInv4c s q*
and *pnq*: *p≠q*
shows *HInv4c s′ q*
⟨*proof*⟩

theorem *HFail-HInv4c*:
⟦ *HFail s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HPhase0Read-HInv4c-p*:
⟦ *HPhase0Read s s′ p d*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

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*⟩

theorem *HPhase0Read-HInv4c*:
⟦ *HPhase0Read s s′ p d*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

lemma *HEndPhase0-HInv4c-p*:
⟦ *HEndPhase0 s s′ p*; *HInv4c s p*⟧ ⟹ *HInv4c s′ p*
⟨*proof*⟩

lemma *HEndPhase0-HInv4c-q*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv*: *HInv4c s q*
and *pnq*: *p≠q*

shows *HInv4c s′ q*
⟨*proof*⟩

theorem *HEndPhase0-HInv4c*:
⟦ *HEndPhase0 s s′ p*; *HInv4c s q*⟧ ⟹ *HInv4c s′ q*
⟨*proof*⟩

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*⟩

lemma *HStartBallot-HInv4d-q*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv4d s q*
and *pnq*: *p*≠*q*
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HStartBallot-HInv4d*:
⟦ *HStartBallot s s′ p*; *HInv4d s q*⟧ ⟹ *HInv4d s′ q*
⟨*proof*⟩

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*⟩

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*⟩

theorem *HPhase1or2Write-HInv4d*:
⟦ *HPhase1or2Write s s′ p d*; *HInv4d s q*; *HInv4a s p*⟧ ⟹ *HInv4d s′ q*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-HInv4d-p*:
assumes *act*: *HPhase1or2ReadThen s s′ p d q*
and *inv*: *HInv4d s p*
shows *HInv4d s′ p*
⟨*proof*⟩

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*⟩

theorem *HPhase1or2ReadThen-HInv4d*:
⟦ *HPhase1or2ReadThen s s′ p d r*; *HInv4d s q*⟧ ⟹ *HInv4d s′ q*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv4d*:
⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv4d s q*⟧ ⟹ *HInv4d s′ q*
⟨*proof*⟩

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*⟩

lemma *HEndPhase1-HInv4d-q*:
assumes *act*: *HEndPhase1 s s′ p*
and *inv*: *HInv4d s q*
and *pnq*: *p≠q*
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HEndPhase1-HInv4d*:
⟦ *HEndPhase1 s s′ p*; *HInv4d s q*; *Inv2b s*; *HInv4c s p*⟧
⟹ *HInv4d s′ q*
⟨*proof*⟩

lemma *HEndPhase2-HInv4d-p*:
assumes *act*: *HEndPhase2 s s′ p*
and *inv*: *HInv4d s p*
shows *HInv4d s′ p*
⟨*proof*⟩

lemma *HEndPhase2-HInv4d-q*:
assumes *act*: *HEndPhase2 s s′ p*
and *inv*: *HInv4d s q*
and *pnq*: *p≠q*
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HEndPhase2-HInv4d*:
$\llbracket$ *HEndPhase2 s s′ p*; *HInv4d s q*$\rrbracket \Longrightarrow$ *HInv4d s′ q*
⟨*proof*⟩

lemma *HFail-HInv4d-p*:
assumes *act*: *HFail s s′ p*
and *inv*: *HInv4d s p*
shows *HInv4d s′ p*
⟨*proof*⟩

lemma *HFail-HInv4d-q*:
assumes *act*: *HFail s s′ p*
and *inv*: *HInv4d s q*
and *pnq*: $p \neq q$
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HFail-HInv4d*:
$\llbracket$ *HFail s s′ p*; *HInv4d s q* $\rrbracket \Longrightarrow$ *HInv4d s′ q*
⟨*proof*⟩

lemma *HPhase0Read-HInv4d-p*:
assumes *act*: *HPhase0Read s s′ p d*
and *inv*: *HInv4d s p*
shows *HInv4d s′ p*
⟨*proof*⟩

lemma *HPhase0Read-HInv4d-q*:
assumes *act*: *HPhase0Read s s′ p d*
and *inv*: *HInv4d s q*
and *pnq*: $p \neq q$
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HPhase0Read-HInv4d*:
$\llbracket$ *HPhase0Read s s′ p d*; *HInv4d s q*$\rrbracket \Longrightarrow$ *HInv4d s′ q*
⟨*proof*⟩

lemma *HEndPhase0-blocksOf2*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv2c*: *Inv2c-inner s p*
shows *allBlocksRead s p* $\subseteq$ *blocksOf s p*
⟨*proof*⟩

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*⟩

lemma *HEndPhase0-HInv4d-q*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv*: *HInv4d s q*
and *pnq*: *p≠q*
shows *HInv4d s′ q*
⟨*proof*⟩

theorem *HEndPhase0-HInv4d*:
⟦ *HEndPhase0 s s′ p*; *HInv4d s q*;
Inv2c s; *Inv1 s*⟧ ⟹ *HInv4d s′ q*
⟨*proof*⟩

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

lemma *I2d*:
assumes *nxt*: *HNext s s′*
and *inv*: *HInv1 s* ∧ *HInv2 s* ∧ *HInv2 s′* ∧ *HInv4 s*
shows *HInv4 s′*
⟨*proof*⟩

end

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

C.5 Invariant 5

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

definition *maxBalInp* :: *state* ⇒ *nat* ⇒ *InputsOrNi* ⇒ *bool*
where *maxBalInp s b v* = (∀ *bk*∈*allBlocks s*. *b* ≤ *bal bk* ⟶ *inp bk* = *v*)

definition *HInv5-inner-R* :: *state* ⇒ *Proc* ⇒ *bool*
where
HInv5-inner-R s p =
(*maxBalInp s* (*bal*(*dblock s p*)) (*inp*(*dblock s p*))
∨ (∃ *D*∈*MajoritySet*. ∃ *q*. (∀ *d*∈*D*. *bal*(*dblock s p*) < *mbal*(*disk s d q*)
∧ ¬*hasRead s p d q*)))

definition *HInv5-inner* :: *state* ⇒ *Proc* ⇒ *bool*
where *HInv5-inner s p* = (*phase s p* = *2* ⟶ *HInv5-inner-R s p*)

definition *HInv5* :: *state* ⇒ *bool*
where *HInv5 s* = (∀ *p*. *HInv5-inner s p*)

C.5.1 Proof of Invariant 5

The initial state implies Invariant 5.

theorem *HInit-HInv5*: *HInit s* ⟹ *HInv5 s*
⟨*proof*⟩

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* ⟨*proof*⟩

lemma *HStartBallot-blocksOf-q*:
assumes *act*: *HStartBallot s s′ p*
and *pnq*: *p*≠*q*
shows *blocksOf s′ q* ⊆ *blocksOf s q* ⟨*proof*⟩

lemma *HStartBallot-allBlocks*:
assumes *act*: *HStartBallot s s′ p*
shows *allBlocks s′* ⊆ *allBlocks s* ∪ {*dblock s′ p*}
⟨*proof*⟩

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*⟩

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*⟩

lemma *HStartBallot-HInv5-q*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv5-inner s q*
and *pnq*: $p \neq q$
shows *HInv5-inner s′ q*
⟨*proof*⟩

theorem *HStartBallot-HInv5*:
⟦ *HStartBallot s s′ p*; *HInv5-inner s q* ⟧ ⟹ *HInv5-inner s′ q*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*⟩

lemma *HPhase1or2Write-HInv5-p*:
assumes *act*: *HPhase1or2Write s s′ p d*
and *inv*: *HInv5-inner s p*
and *inv4*: *HInv4c s p*
shows *HInv5-inner s′ p*
⟨*proof*⟩

lemma *HPhase1or2Write-allBlocks*:
assumes *act*: *HPhase1or2Write s s′ p d*
shows *allBlocks s′* ⊆ *allBlocks s*
⟨*proof*⟩

lemma *HPhase1or2Write-HInv5-q2*:
assumes *act*: *HPhase1or2Write s s′ p d*
and *pnq*: $p \neq q$
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 $\exists\, D \in MajoritySet.\ \exists\, qq.\ (\forall\, d \in D.\quad bal(dblock\ s'\ q) < mbal(disk\ s'\ d\ qq)$
$\wedge\ \neg hasRead\ s'\ q\ d\ qq)$
$\langle proof \rangle$

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*
$\langle proof \rangle$

theorem *HPhase1or2Write-HInv5*:
⟦ *HPhase1or2Write s s′ p d*; *HInv5-inner s q*;
HInv4c s p; *HInv4a s p* ⟧ $\Longrightarrow$ *HInv5-inner s′ q*
$\langle proof \rangle$

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 $maxBalInp\ s'\ (bal(dblock\ s'\ q))\ (inp(dblock\ s'\ q))$
$\langle proof \rangle$

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

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*
$\langle proof \rangle$

lemma *HPhase1or2ReadThen-allBlocks*:
assumes *act*: *HPhase1or2ReadThen s s′ p d r*
shows $allBlocks\ s' \subseteq allBlocks\ s$
$\langle proof \rangle$

lemma *HPhase1or2ReadThen-HInv5-q2*:
assumes *act*: *HPhase1or2ReadThen s s′ p d r*

and *pnq*: $p \neq q$
and *inv4a*: *HInv4a s p*
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*⟩

lemma *HPhase1or2ReadThen-HInv5-q*:
assumes *act*: *HPhase1or2ReadThen s s′ p d r*
and *inv*: *HInv5-inner s q*
and *inv4a*: *HInv4a s p*
and *pnq*: $p \neq q$
shows *HInv5-inner s′ q*
⟨*proof*⟩

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*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-HInv5*:
⟦ *HPhase1or2ReadElse s s′ p d r*; *HInv5-inner s q* ⟧
⟹ *HInv5-inner s′ q*
⟨*proof*⟩

lemma *HEndPhase2-HInv5-p*:
HEndPhase2 s s′ p ⟹ *HInv5-inner s′ p*
⟨*proof*⟩

lemma *HEndPhase2-allBlocks*:
assumes *act*: *HEndPhase2 s s′ p*
shows $allBlocks\ s' \subseteq allBlocks\ s$
⟨*proof*⟩

lemma *HEndPhase2-HInv5-q1*:
assumes *act*: *HEndPhase2 s s′ p*
and *pnq*: $p \neq q$
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*⟩

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*⟩

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*
⟨*proof*⟩

theorem *HEndPhase2-HInv5*:
$[\![\ HEndPhase2\ s\ s'\ p;\ HInv5\text{-}inner\ s\ q\ ]\!] \Longrightarrow HInv5\text{-}inner\ s'\ q$
⟨*proof*⟩

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*⟩

lemma *union-inclusion*:
$[\![\ A \subseteq A';\ B{\subseteq}\ B'\ ]\!] \Longrightarrow A{\cup}B \subseteq A'{\cup}B'$
⟨*proof*⟩

lemma *HEndPhase1-blocksOf-q*:
assumes *act*: *HEndPhase1 s s′ p*
and *pnq*: $p{\neq}q$
shows $blocksOf\ s'\ q \subseteq blocksOf\ s\ q$
⟨*proof*⟩

lemma *HEndPhase1-allBlocks*:
assumes *act*: *HEndPhase1 s s′ p*
shows $allBlocks\ s' \subseteq allBlocks\ s \cup \{dblock\ s'\ p\}$
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *HFail-HInv5-p*:
HFail s s' p ⟹ *HInv5-inner s' p*
⟨*proof*⟩

lemma *HFail-blocksOf-q*:
assumes *act*: *HFail s s' p*
and *pnq*: *p*≠*q*
shows *blocksOf s' q* ⊆ *blocksOf s q*
⟨*proof*⟩

lemma *HFail-allBlocks*:
assumes *act*: *HFail s s' p*
shows *allBlocks s'* ⊆ *allBlocks s* ∪ {*dblock s' p*}
⟨*proof*⟩

lemma *HFail-HInv5-q1*:
assumes *act*: *HFail s s' p*
and *pnq*: *p*≠*q*
and *inv2a*: *Inv2a-inner s' q*
and *inv5-1*: *maxBalInp s* (*bal*(*dblock s q*)) (*inp*(*dblock s q*))
shows *maxBalInp s'* (*bal*(*dblock s' q*)) (*inp*(*dblock s' q*))
⟨*proof*⟩

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*)

$\wedge \neg hasRead\ s\ q\ d\ qq)$
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*⟩

lemma *HFail-HInv5-q*:
assumes *act*: *HFail s s′ p*
and *inv*: *HInv5-inner s q*
and *pnq*: $p{\neq}q$
and *inv2a*: *Inv2a s′*
shows *HInv5-inner s′ q*
⟨*proof*⟩

theorem *HFail-HInv5*:
$[\![\ HFail\ s\ s'\ p;\ HInv5\text{-}inner\ s\ q;\ Inv2a\ s'\ ]\!] \Longrightarrow HInv5\text{-}inner\ s'\ q$
⟨*proof*⟩

lemma *HPhase0Read-HInv5-p*:
$HPhase0Read\ s\ s'\ p\ d \Longrightarrow HInv5\text{-}inner\ s'\ p$
⟨*proof*⟩

lemma *HPhase0Read-allBlocks*:
assumes *act*: *HPhase0Read s s′ p d*
shows $allBlocks\ s' \subseteq allBlocks\ s$
⟨*proof*⟩

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 $maxBalInp\ s'\ (bal(dblock\ s'\ q))\ (inp(dblock\ s'\ q))$
⟨*proof*⟩

lemma *HPhase0Read-HInv5-q2*:
assumes *act*: *HPhase0Read s s′ p d*
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*⟩

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*⟩

theorem *HPhase0Read-HInv5*:
⟦ *HPhase0Read s s′ p d*; *HInv5-inner s q* ⟧ $\Longrightarrow$ *HInv5-inner s′ q*
⟨*proof*⟩

lemma *HEndPhase0-HInv5-p*:
HEndPhase0 s s′ p $\Longrightarrow$ *HInv5-inner s′ p*
⟨*proof*⟩

lemma *HEndPhase0-blocksOf-q*:
assumes *act*: *HEndPhase0 s s′ p*
and *pnq*: *p*≠*q*
shows *blocksOf s′ q* ⊆ *blocksOf s q*
⟨*proof*⟩

lemma *HEndPhase0-allBlocks*:
assumes *act*: *HEndPhase0 s s′ p*
shows *allBlocks s′* ⊆ *allBlocks s* ∪ {*dblock s′ p*}
⟨*proof*⟩

lemma *HEndPhase0-HInv5-q1*:
assumes *act*: *HEndPhase0 s s′ p*
and *pnq*: *p*≠*q*
and *inv1*: *Inv1 s*
and *inv5-1*: *maxBalInp s* (*bal*(*dblock s q*)) (*inp*(*dblock s q*))
shows *maxBalInp s′* (*bal*(*dblock s′ q*)) (*inp*(*dblock s′ q*))
⟨*proof*⟩

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*⟩

lemma *HEndPhase0-HInv5-q*:
assumes *act*: *HEndPhase0 s s′ p*
and *inv*: *HInv5-inner s q*
and *inv1*: *Inv1 s*
and *pnq*: *p*≠*q*
shows *HInv5-inner s′ q*
⟨*proof*⟩

theorem *HEndPhase0-HInv5*:
⟦ *HEndPhase0 s s′ p*; *HInv5-inner s q*; *Inv1 s* ⟧ $\Longrightarrow$ *HInv5-inner s′ q*
⟨*proof*⟩

$HInv1 \wedge HInv2 \wedge HInv3 \wedge HInv4 \wedge HInv5$ 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'*
⟨*proof*⟩

end

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

C.6 Lemma I2f

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

definition *valueChosen* :: *state* ⇒ *InputsOrNi* ⇒ *bool*
where
valueChosen s v =
(∃ *b*∈ (*UN p. Ballot p*).
maxBalInp s b v
∧ (∃ *p*. ∃ *D*∈*MajoritySet*.(∀ *d*∈*D*. *b* ≤ *bal*(*disk s d p*)
∧(∀ *q*.(*phase s q* = *1*
∧ *b* ≤*mbal*(*dblock s q*)
∧ *hasRead s q d p*
) ⟶ (∃ *br*∈*blocksRead s q d*. *b* ≤ *bal*(*block br*))
))))

lemma *HEndPhase1-valueChosen-inp*:
assumes *act*: *HEndPhase1 s s' q*
and *inv2a*: *Inv2a s*
and *asm1*: *b* ∈ (*UN p. Ballot p*)
and *bk-blocksOf*: *bk*∈*blocksOf s r*
and *bk*: *bk*∈ *blocksSeen s q*
and *b-bal*: *b* ≤ *bal bk*
and *asm3*: *maxBalInp s b v*
and *inv1*: *Inv1 s*
shows *inp*(*dblock s' q*) = *v*
⟨*proof*⟩

lemma *HEndPhase1-maxBalInp*:
assumes *act*: *HEndPhase1 s s' q*
and *asm1*: *b* ∈ (*UN p. Ballot p*)
and *asm2*: *D*∈*MajoritySet*
and *asm3*: *maxBalInp s b v*
and *asm4*: ∀ *d*∈*D*. *b* ≤ *bal*(*disk s d p*)

$\land(\forall\, q.(\quad phase\ s\ q = 1$
$\land\ b \le mbal(dblock\ s\ q)$
$\land\ hasRead\ s\ q\ d\ p$
$) \longrightarrow (\exists\, br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$

and *inv1*: *Inv1 s*
and *inv2a*: *Inv2a s*
and *inv2b*: *Inv2b s*
shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HEndPhase1-valueChosen2*:
assumes *act*: *HEndPhase1 s s′ q*
and *asm4*: $\forall\, d \in D.\quad b \le bal(disk\ s\ d\ p)$
$\land(\forall\, q.(\quad phase\ s\ q = 1$
$\land\ b \le mbal(dblock\ s\ q)$
$\land\ hasRead\ s\ q\ d\ p$
$) \longrightarrow (\exists\, br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$ (**is** *?P s*)
shows *?P s′*
⟨*proof*⟩

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*⟩

lemma *HStartBallot-maxBalInp*:
assumes *act*: *HStartBallot s s′ q*
and *asm3*: *maxBalInp s b v*
shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HStartBallot-valueChosen2*:
assumes *act*: *HStartBallot s s′ q*
and *asm4*: $\forall\, d \in D.\quad b \le bal(disk\ s\ d\ p)$
$\land(\forall\, q.(\quad phase\ s\ q = 1$
$\land\ b \le mbal(dblock\ s\ q)$
$\land\ hasRead\ s\ q\ d\ p$
$) \longrightarrow (\exists\, br \in blocksRead\ s\ q\ d.\ b \le bal(block\ br)))$ (**is** *?P s*)
shows *?P s′*
⟨*proof*⟩

theorem *HStartBallot-valueChosen*:
assumes *act*: *HStartBallot s s′ q*

and *vc*: *valueChosen s v*
and *v-input*: $v \in$ *Inputs*
shows *valueChosen s′ v*
⟨*proof*⟩

lemma *HPhase1or2Write-maxBalInp*:
assumes *act*: *HPhase1or2Write s s′ q d*
and *asm3*: *maxBalInp s b v*
shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HPhase1or2Write-valueChosen2*:
assumes *act*: *HPhase1or2Write s s′ pp d*
and *asm2*: *D*∈*MajoritySet*
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*)
and *inv4*: *HInv4a s pp*
shows *?P s′*
⟨*proof*⟩

theorem *HPhase1or2Write-valueChosen*:
assumes *act*: *HPhase1or2Write s s′ q d*
and *vc*: *valueChosen s v*
and *v-input*: $v \in$ *Inputs*
and *inv4*: *HInv4a s q*
shows *valueChosen s′ v*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-maxBalInp*:
assumes *act*: *HPhase1or2ReadThen s s′ q d p*
and *asm3*: *maxBalInp s b v*
shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HPhase1or2ReadThen-valueChosen2*:
assumes *act*: *HPhase1or2ReadThen s s′ q d pp*
and *asm4*: ∀ *d*∈*D*. *b* ≤ *bal*(*disk s d p*)
∧(∀ *q*.(*phase s q* = *1*
∧ *b* ≤*mbal*(*dblock s q*)
∧ *hasRead s q d p*
) ⟶ (∃ *br*∈*blocksRead s q d*. *b* ≤ *bal*(*block br*))) (**is** *?P s*)
shows *?P s′*
⟨*proof*⟩

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*⟩

theorem *HPhase1or2ReadElse-valueChosen*:
⟦ *HPhase1or2ReadElse s s′ p d r*; *valueChosen s v*; *v*∈ *Inputs* ⟧
⟹ *valueChosen s′ v*
⟨*proof*⟩

lemma *HEndPhase2-maxBalInp*:
assumes *act*: *HEndPhase2 s s′ q*
and *asm3*: *maxBalInp s b v*
shows *maxBalInp s′ b v*
⟨*proof*⟩

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*⟩

theorem *HEndPhase2-valueChosen*:
assumes *act*: *HEndPhase2 s s′ q*
and *vc*: *valueChosen s v*
and *v-input*: *v* ∈ *Inputs*
shows *valueChosen s′ v*
⟨*proof*⟩

lemma *HFail-maxBalInp*:
assumes *act*: *HFail s s′ q*
and *asm1*: *b* ∈ (*UN p*. *Ballot p*)
and *asm3*: *maxBalInp s b v*
shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HFail-valueChosen2*:
assumes *act*: *HFail 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*⟩

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*⟩

lemma *HPhase0Read-maxBalInp*:
  assumes *act*: *HPhase0Read s s′ q d*
    and *asm3*: *maxBalInp s b v*
  shows *maxBalInp s′ b v*
⟨*proof*⟩

lemma *HPhase0Read-valueChosen2*:
  assumes *act*: *HPhase0Read s s′ qq dd*
    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*⟩

theorem *HPhase0Read-valueChosen*:
  assumes *act*: *HPhase0Read s s′ q d*
  and *vc*: *valueChosen s v*
  and *v-input*: $v \in Inputs$
  shows *valueChosen s′ v*
⟨*proof*⟩

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*⟩

lemma *HEndPhase0-valueChosen2*:
  assumes *act*: *HEndPhase0 s s′ q*
    and *asm4*: $\forall d{\in}D.\ \ b \leq bal(disk\ s\ d\ p)$
      $\wedge(\forall q.(\ \ phase\ s\ q = 1$
        $\wedge\ b \leq mbal(dblock\ s\ q)$
        $\wedge\ hasRead\ s\ q\ d\ p$
        $) \longrightarrow (\exists\, br{\in}blocksRead\ s\ q\ d.\ b \leq bal(block\ br)))$ (**is** *?P s*)
  shows *?P s′*
⟨*proof*⟩

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*⟩

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*
  ⟨*proof*⟩

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*⟩

lemma *valueChosen-equal-case*:
  assumes *max-v*: *maxBalInp s b v*
  and *Dmaj*: *D* ∈ *MajoritySet*
  and *asm-v*: ∀ *d*∈*D*. *b* ≤ *bal* (*disk s d p*)
  and *max-w*: *maxBalInp s ba w*
  and *Damaj*: *Da* ∈ *MajoritySet*
  and *asm-w*: ∀ *d*∈*Da*. *ba* ≤ *bal* (*disk s d pa*)
  and *b-ba*: *b*≤*ba*
  shows *v*=*w*
⟨*proof*⟩

lemma *valueChosen-equal*:
assumes *v*: *valueChosen s v*
and *w*: *valueChosen s w*
shows $v{=}w$ ⟨*proof*⟩

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*⟩

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*⟩

lemma *outpt-chosen*:
assumes *outpt*: *outpt s* = *outpt s′*
and *inv2c*: *Inv2c s*
and *nextp*: *HNextPart s s′*
shows *chosen s′* = *chosen s*
⟨*proof*⟩

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*}
⟨*proof*⟩

theorem *HStartBallot-Inv6*:
assumes *act*: *HStartBallot s s′ p*
and *inv*: *HInv6 s*
and *inv2c*: *Inv2c s*
shows *HInv6 s′*
⟨*proof*⟩

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*⟩

theorem *HPhase1or2ReadThen-Inv6*:
assumes *act*: *HPhase1or2ReadThen s s' p d q*
and *inv*: *HInv6 s*
and *inv2c*: *Inv2c s*
shows *HInv6 s'*
⟨*proof*⟩

theorem *HPhase1or2ReadElse-Inv6*:
assumes *act*: *HPhase1or2ReadElse s s' p d q*
and *inv*: *HInv6 s*
and *inv2c*: *Inv2c s*
shows *HInv6 s'*
⟨*proof*⟩

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*⟩

lemma *outpt-chosen-2*:
assumes *outpt*: $outpt\ s' = (outpt\ s)\ (p{:=}\ NotAnInput)$
and *inv2c*: *Inv2c s*
and *nextp*: *HNextPart s s'*
shows $chosen\ s = chosen\ s'$
⟨*proof*⟩

lemma *outpt-HInv6-2*:
assumes *outpt*: $outpt\ s' = (outpt\ s)\ (p{:=}\ NotAnInput)$
and *inv*: $\forall\, p.\ outpt\ s\ p \in \{chosen\ s,\ NotAnInput\}$
and *inv2c*: *Inv2c s*
and *nextp*: *HNextPart s s'*
shows $\forall\, p.\ outpt\ s'\ p \in \{chosen\ s',\ NotAnInput\}$
⟨*proof*⟩

theorem *HFail-Inv6*:
assumes *act*: *HFail s s' p*
and *inv*: *HInv6 s*
and *inv2c*: *Inv2c s*
shows *HInv6 s'*
⟨*proof*⟩

theorem *HPhase0Read-Inv6*:
  assumes *act*: *HPhase0Read s s′ p d*
  and *inv*: *HInv6 s*
  and *inv2c*: *Inv2c s*
  shows *HInv6 s′*
⟨*proof*⟩

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*⟩

$HInv1 \wedge HInv2 \wedge HInv2' \wedge HInv3 \wedge HInv4 \wedge HInv5 \wedge HInv6$ 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′* ⟨*proof*⟩

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*
  ⟨*proof*⟩

theorem *I2*:
  assumes *inv*: *HInv s*
  and *nxt*: *HNext s s′*
  shows *HInv s′*
  ⟨*proof*⟩

end

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

C.9 Inner Module

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

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

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

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

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

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

theorem *R1*:
$\llbracket$ *HInit s*; *is* = *s2is s*$\rrbracket \Longrightarrow$ *IInit is*
$\langle$*proof*$\rangle$

theorem *R2b*:
assumes *inv*: *HInv s*
and *inv'*: *HInv s'*
and *nxt*: *HNext s s'*
and *srel*: *is*=*s2is s* $\wedge$ *is'*=*s2is s'*
shows ($\exists$ *p*. *IFail is is' p* $\vee$ *IChoose is is' p*) $\vee$ *is* = *is'*
$\langle$*proof*$\rangle$

end