# Proving the Correctness of Disk Paxos in Isabelle/HOL 

Mauro Jaskelioff

Stephan Merz
September 13, 2023


#### Abstract

Disk Paxos [GL00] is an algorithm for building arbitrary faulttolerant 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 $\mathrm{TLA}^{+}$specifications.


## Contents

1 Introduction ..... 2
2 The Disk Paxos Algorithm ..... 3
2.1 Informal description of the algorithm. ..... 4
2.2 Disk Paxos and its TLA ${ }^{+}$Specification ..... 4
3 Translating from TLA ${ }^{+}$to Isabelle/HOL ..... 6
3.1 Typed vs. Untyped ..... 6
3.2 Primed Variables ..... 8
3.3 Restructuring the specification ..... 8
4 Structure of the Correctness Proof ..... 9
4.1 Going from Informal Proofs to Formal Proofs ..... 10
5 Conclusion ..... 11
A TLA ${ }^{+}$correctness specification ..... 12
B Disk Paxos Algorithm Specification ..... 13
C Proof of Disk Paxos' Invariant ..... 19
C. 1 Invariant 1 ..... 19
C. 2 Invariant 2 ..... 21
C.2.1 Proofs of Invariant 2 a ..... 22
C.2.2 Proofs of Invariant 2 b ..... 27
C.2.3 Proofs of Invariant 2 c ..... 28
C. 3 Invariant 3 ..... 30
C.3.1 Proofs of Invariant 3 ..... 30
C. 4 Invariant 4 ..... 35
C.4.1 Proofs of Invariant 4a ..... 36
C.4.2 Proofs of Invariant 4b ..... 40
C.4.3 Proofs of Invariant 4 c ..... 44
C.4.4 Proofs of Invariant 4d ..... 47
C. 5 Invariant 5 ..... 50
C.5.1 Proof of Invariant 5 ..... 51
C. 6 Lemma I2f ..... 59
C. 7 Invariant 6 ..... 64
C. 8 The Complete Invariant ..... 67
C. 9 Inner Module ..... 68

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

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

### 2.1 Informal description of the algorithm.

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

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

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

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

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

In each block, processors maintain three values:
mbal The current ballot number.
bal The largest ballot number for which the processor entered phase 2 .
inp The value the processor tried to commit in ballot number bal.
For a complete description of the algorithm, see [GL00].

### 2.2 Disk Paxos and its TLA ${ }^{+}$Specification

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

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

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

## Processors



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:

$$
\text { HDiskSynodSpec } \triangleq \text { HInit } \wedge \square[H N e x t]_{\langle v a r s, c h o s e n, a l l I n p u t\rangle}
$$

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:

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

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:

$$
\begin{array}{lll}
\text { THEOREM } & R 1 & \text { HInit } \Rightarrow \text { IInit } \\
\text { THEOREM } & R 2 & \text { HInit } \wedge \square[H N e x t]_{\langle\text {vars,chosen,allInput }\rangle} \Rightarrow \square[\text { INext }]_{i v a r s}
\end{array}
$$

The proof of $R 1$ is trivial. For $R 2$, we use TLA proof rules [Lam02] that show that to prove $R 2$, it suffices to find a state predicate HInv for which we can prove:

A predicate satisfying HInv is said to be an invariant of HDiskSynodSpec. To prove $R 2 a$, we make HInv strong enough to satisfy:

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

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

THEOREM $I 1 \quad H I n i t \Rightarrow H I n v$
THEOREM I2 HInv $\wedge H N e x t \Rightarrow H I n v{ }^{\prime}$
Again, we have TLA proof rules that say that $I 1$ and $I 2$ imply $R 2 a$. In summary, $R 2 b, I 1$, and $I 2$ together imply $H D i s k S y n o d S p e c ~ \Rightarrow I S p e c$.

Finding a predicate HInv that is strong enough can be rather difficult. Fortunately, Gafni and Lamport give this predicate. In their paper, they present HInv as a conjunction of 6 predicates HInv $1, \ldots$, HInv6, where HInv 1 is a simple "type invariant" and the higher-numbered predicates add more and more information. The proof is structured such that the preservation of HInv $i$ by the algorithm's next-state relation relies on all HInv $j$ (for $j \leq i$ ) being true in the state before the transition. In our proofs we are going to use exactly the same tactic.

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

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

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

### 3.1 Typed vs. Untyped

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

[^0]```
TLA+
CONSTANT Inputs
NotAnInput }\triangleq CHOOSE c:c\not\inInput
DiskBlock \triangleq [mbal : (UNION Ballot (p) : p\inProc)\cup{0},
    bal : (UNION Ballot (p):p\inProc)\cup{0},
    inp : Inputs \cup{NotAnInput}]
Isabelle/HOL:
typedecl InputsOrNi
consts
    Inputs :: InputsOrNi set
    NotAnInput :: InputsOrNi
axioms
    NotAnInput: NotAnInput & Inputs
    InputsOrNi:(UNIV :: InputsOrNi set) = Inputs \cup {NotAnInput }
record
    DiskBlock =
        mbal:: nat
        bal :: nat
        inp :: InputsOrNi
```

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

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

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

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

```
TLA \({ }^{+}\):
Phase1or 2 Write \((p, d) \triangleq\)
    \(\wedge\) phase \([p] \in\{1,2\}\)
    \(\wedge\) disk \({ }^{\prime}=[\) disk EXCEPT \(![d][p]=\) dblock \([p]]\)
    \(\wedge\) disks Written \(^{\prime}=[\) disksWritten EXCEPT \(![p]=@ \cup\{d\}]\)
    \(\wedge\) UNCHANGED \(\langle\) input, output, phase, dblock, blocksRead \(\rangle\)
```

Isabelle/HOL:

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

Figure 3: Translation of an action
lating from an untyped formalism to a typed one. Another solution to this matter that is being currently investigated is the implementation of native (untyped) support for $\mathrm{TLA}^{+}$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, Pss will be true iff executing an action $P$ in the $s$ state could result in the $s^{\prime}$ state. In figure 3 we can see how the action Phase1or $2 W$ rite 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-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, Phase1or 2 Read is mainly a big if-then-else. We break it down into two simpler actions:

$$
\text { Phase1or2Read } \triangleq \text { Phase1or } 2 \text { ReadThen } \vee \text { Phase1or } 2 \text { ReadElse }
$$

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:

$$
\operatorname{HInv} 2 \triangleq \operatorname{Inv} 2 a \wedge \operatorname{Inv} 2 b \wedge \operatorname{Inv} 2 c
$$

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 $\operatorname{Inv} 2 a$, and after translating to Isabelle/HOL, instead of writing:
Inv2a $s \equiv \forall p . \forall b k \in$ blocksOf s $p . \ldots$
we write:

```
Inv2a-innermost :: state \(\Rightarrow\) Proc \(\Rightarrow\) DiskBlock \(\Rightarrow\) bool
Inv2a-innermost s \(p b k \equiv \ldots\)
Inv2a-inner :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
Inv2a-inner s \(p \equiv \forall b k \in b l o c k s O f\) s \(p\). Inv2a-innermost s \(p b k\)
Inv2a \(::\) state \(\Rightarrow\) bool
Inv2a \(s \equiv \forall p\). 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 HInv 4 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



## 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 }=>\mathrm{ nat set and
    IsMajority :: Disk set }=>\mathrm{ bool
where
    NotAnInput: NotAnInput }\ddagger\mathrm{ Inputs and
    InputsOrNi:(UNIV :: InputsOrNi set) = Inputs \cup {NotAnInput } and
    Ballot-nzero: }\forall\mathrm{ p. 0 # Ballot p and
    Ballot-disj: }\forallp\mathrm{ q. p}\not=q\longrightarrow(\mathrm{ Ballot p) }\cap(\mathrm{ Ballot q) ={} and
    Disk-isMajority:IsMajority(UNIV) and
    majorities-intersect:
        \forall T. IsMajority (S)^IsMajority (T)\longrightarrowS\capT\not={}
lemma ballots-not-zero [simp]:
    b\in Ballot p\Longrightarrow0<b
<proof\rangle
lemma majority-nonempty [simp]: IsMajority (S)\LongrightarrowS\not={}
<proof\rangle
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 }=>\mathrm{ Proc }=>\mathrm{ DiskBlock
dblock :: Proc = DiskBlock
phase :: Proc }=>\mathrm{ nat
disksWritten :: Proc }=>\mathrm{ Disk set
blocksRead :: Proc = Disk => BlockProc set
allInput :: InputsOrNi set
chosen :: InputsOrNi
```

```
definition hasRead \(::\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where hasRead spd \(q=(\exists\) br \(\in\) blocksRead spd. proc \(b r=q)\)
definition allRdBlks :: state \(\Rightarrow\) Proc \(\Rightarrow\) BlockProc set
    where allRdBlks s \(p=(U N d\). blocksRead spd)
definition allBlocksRead \(::\) state \(\Rightarrow\) Proc \(\Rightarrow\) DiskBlock set
    where allBlocksRead s \(p=\) block' (allRdBlks s \(p\) )
definition Init :: state \(\Rightarrow\) bool
    where
    Init \(s=\)
        (range (inpt s) \(\subseteq\) Inputs
        \(\&\) outpt \(s=(\lambda p\). NotAnInput \()\)
        \(\&\) disk \(s=(\lambda d\) p. InitDB \()\)
        \& phase \(s=(\lambda p .0)\)
        \& dblock \(s=(\lambda p\). InitDB \()\)
        \(\&\) disksWritten \(s=(\lambda p .\{ \})\)
        \& blocksRead \(s=(\lambda p d .\{ \}))\)
definition InitializePhase :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where
    InitializePhase s \(s^{\prime} p=\)
        (disksWritten \(s^{\prime}=(\) disksWritten \(s)(p:=\{ \})\)
    \& blocksRead \(s^{\prime}=(\) blocksRead \(\left.s)(p:=(\lambda d .\{ \}))\right)\)
definition StartBallot \(::\) state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    StartBallot s s \(s^{\prime} p=\)
    (phase s \(p \in\{1,2\}\)
    \(\&\) phase \(s^{\prime}=(\) phase \(s)(p:=1)\)
    \(\&(\exists b \in\) Ballot \(p\).
    mbal (dblock s p) <b
        \& dblock \(s^{\prime}=(\) dblock \(s)(p:=(\) dblock s \(p) \bigvee\) mbal \(:=b\) ) \(\left.)\right)\)
    \& InitializePhase s \(s^{\prime} p\)
    \& inpt \(s^{\prime}=\) inpt \(s \&\) outpt \(s^{\prime}=\) outpt \(s \& d i s k s^{\prime}=\) disk \(\left.s\right)\)
```

```
definition Phase1or2Write :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) bool
where
    Phase1or2Write s \(s^{\prime} p d=\)
    (phase s \(p \in\{1,2\}\)
    \(\wedge\) disk \(s^{\prime}=(\) disk \(s)(d:=(\) disk s d) \((p:=\) dblock s \(p))\)
    \(\wedge\) disksWritten \(s^{\prime}=(\) disksWritten \(s)(p:=(\) disksWritten \(s p) \cup\{d\})\)
    \(\wedge\) inpt \(s^{\prime}=\) inpt \(s \wedge\) outpt \(s^{\prime}=\) outpt \(s\)
    \(\wedge\) phase \(s^{\prime}=\) phase \(s \wedge\) dblock \(s^{\prime}=\) dblock \(s\)
    \(\wedge\) blocksRead \(s^{\prime}=\) blocksRead \(s\) )
definition Phase1or2ReadThen \(::\) state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    Phase1or2ReadThen s \(s^{\prime} p d q=\)
        ( \(d \in\) disksWritten s \(p\)
    \& mbal \((\) disk s d q) \(<\operatorname{mbal}(d b l o c k s p)\)
    \& blocksRead \(s^{\prime}=(\) blocksRead \(s)(p:=(\) blocksRead s \(p)(d:=\)
                            (blocksRead spd) \(\cup\{0\) block \(=\) disk s \(d q\),
                                proc \(=q D\})\) )
    \(\&\) inpt \(s^{\prime}=\) inpt \(s \&\) outpt \(s^{\prime}=\) outpt \(s\)
    \& disk \(s^{\prime}=\) disk \(s\) \& phase \(s^{\prime}=\) phase \(s\)
    \& dblock \(s^{\prime}=\) dblock \(s\) \& disksWritten \(s^{\prime}=\) disksWritten \(\left.s\right)\)
definition Phase1or2ReadElse :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    Phase1or2ReadElse s \(s^{\prime} p d q=\)
    ( \(d \in\) disksWritten sp
    \(\wedge\) StartBallot s s'p)
definition Phase1or2Read \(::\) state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    Phase1or2Read s \(s^{\prime} p d q=\)
        (Phase1or2ReadThen \(s s^{\prime} p d q\)
        \(\vee\) Phase1or2ReadElse \(s s^{\prime} p d q\) )
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
    wherenonInitBlks s \(p=\{b s . b s \in\) 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 \(c \leq b a l b))\)
definition EndPhase1 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    EndPhase1s s' \(p=\)
    (IsMajority \(\{d . d \in\) disks Written s \(p\)
```

```
                            \wedge(\forallq\inUNIV - {p}. hasRead s pd d)}
    ^phase s p=1
    \dblock s'=(dblock s) (p:= dblock s p
        \ bal := mbal(dblock s p),
            inp :=
            (if nonInitBlks s p={}
                then inpt s p
                else inp (maxBlk s p))
    D)
    \outpt s' = outpt s
    ^phase s'=(phase s) (p:= phase s p + 1)
    \ InitializePhase s s' p
    ^inpt s'= inpt s ^disk s'= disks)
definition EndPhase2 :: state }=>\mathrm{ state }=>\mathrm{ Proc }=>\mathrm{ bool
where
    EndPhase2 s s'p=
    (IsMajority {d.d\in disksWritten s p
                                    \wedge(\forallq\inUNIV - {p}. hasRead s p d q)}
    ^phase s p=2
    \outpt s' = (outpt s) (p:= inp (dblock s p))
    ^dblock s' = dblock s
    ^phase s' = (phase s) ( p:= phase s p + 1)
    InitializePhase s s'p
    ^inpt s'= inpt s ^ disk s'= disk s)
definition EndPhase1or2 :: state }=>\mathrm{ state }=>\mathrm{ Proc }=>\mathrm{ bool
    where EndPhase1or2 s s'p}=(\mathrm{ EndPhase1 s s'p
definition Fail :: state }=>\mathrm{ state }=>\mathrm{ Proc }=>\mathrm{ bool
where
    Fail s s'p=
    (\existsip\in Inputs. inpt s' = (inpt s) (p:= ip)
    ^phase s' = (phase s) (p:=0)
    ^dblock s'= (dblock s) (p:= InitDB)
    \outpt s}\mp@subsup{s}{}{\prime}=(\mathrm{ outpt s) (p:= NotAnInput)
    \InitializePhase s s' p
    ^ disk s'= disk s)
definition Phase0Read :: state }=>\mathrm{ state }=>\mathrm{ Proc }=>\mathrm{ Disk }=>\mathrm{ bool
where
    Phase0Read s s' p d=
    (phase s p=0
    ^blocksRead s'=(blocksRead s) (p:= (blocksRead s p) (d:= blocksRead s p d
\cup \{ 0 ~ b l o c k = ~ d i s k ~ s ~ d ~ p , ~ p r o c ~ = ~ p ~ D \} ) ) ~
    inpt s}\mp@subsup{s}{}{\prime}=\mathrm{ inpt s& outpt s}\mp@subsup{s}{}{\prime}=\mathrm{ outpt s
    ^disk s}\mp@subsup{s}{}{\prime}=\mathrm{ disk s & phase }\mp@subsup{s}{}{\prime}=\mathrm{ phase s
    ^dblock s'=dblock s & disksWritten s}\mp@subsup{}{}{\prime}=\mathrm{ disksWritten s)
```

```
definition EndPhase0 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    EndPhase0 s \(s^{\prime} p=\)
        (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 \(s^{\prime}=(\) dblock \(s)(p:=\)
                            (SOME r. \(\quad r \in\) allBlocksRead s \(p\)
                                    \(\wedge(\forall s \in\) allBlocksRead sp.bal \(s \leq\) bal \(r))(\) mbal \(:=b \mid))\)
    \(\wedge\) InitializePhase \(s s^{\prime} p\)
    \(\wedge\) phase \(s^{\prime}=(\) phase \(s)(p:=1)\)
    \(\wedge\) inpt \(s^{\prime}=\) inpt \(s \wedge\) outpt \(s^{\prime}=\) outpt \(s \wedge\) disk \(s^{\prime}=\) disk \(\left.s\right)\)
definition Next \(::\) state \(\Rightarrow\) state \(\Rightarrow\) bool
where
    Next s \(s^{\prime}=(\exists p\).
        StartBallot s \(s^{\prime} p\)
    \(\vee\left(\exists d\right.\). Phase0Read \(s s^{\prime} p d\)
            \(\vee\) Phase1or2Write \(s s^{\prime} p d\)
            \(\vee\left(\exists q . q \neq p \wedge\right.\) Phase1or2Read \(\left.\left.s s^{\prime} p d q\right)\right)\)
    \(\checkmark\) EndPhase1or2 s s'p
    \(\checkmark\) Fail \(s s^{\prime} p\)
    \(\vee\) EndPhase0 \(s s^{\prime} p\) )
```

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 $s=\operatorname{range}($ inpt $s))$
HNextPart is the part of the Next action that is concerned with history variables.

```
definition HNextPart :: state \(\Rightarrow\) state \(=>\) bool
where
    HNextPart \(s s^{\prime}=\)
    (chosen \(s^{\prime}=\)
        (if chosen \(s \neq\) NotAnInput \(\vee\left(\forall\right.\) p. outpt \(s^{\prime} p=\) NotAnInput \()\)
            then chosen \(s\)
            else outpt \(s^{\prime}\left(S O M E\right.\) p. outpt \(s^{\prime} p \neq\) NotAnInput))
    \(\wedge\) allInput \(s^{\prime}=\) allInput \(s \cup\left(\right.\) range \(\left(\right.\) inpt \(\left.\left.\left.s^{\prime}\right)\right)\right)\)
```

definition HNext :: state $\Rightarrow$ state $\Rightarrow$ bool
where
HNext s $s^{\prime}=$

```
    (Next s s'
^HNextPart s s')
```

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

## definition

```
    HPhase1or2ReadThen \(::\) state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) Proc \(\Rightarrow\) bool where
    HPhase1or2ReadThen s s'pd \(q=(\) Phase1or2ReadThen s s' \(p\) d \(q \wedge\) HNextPart
\(s s^{\prime}\) )
```


## definition

HEndPhase1 $::$ state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ bool where HEndPhase1 s $s^{\prime} p=\left(\right.$ EndPhase1 s s'p 1 HNextPart s $\left.s^{\prime}\right)$

## definition

HStartBallot $::$ state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ bool where
HStartBallot s $s^{\prime} p=\left(\right.$ StartBallot s s $s^{\prime} p \wedge$ HNextPart s $\left.s^{\prime}\right)$

## definition

HPhase1or2Write :: state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ Disk $\Rightarrow$ bool where
HPhase1or2Write $s s^{\prime} p d=\left(\right.$ Phase1or2Write $\left.s s^{\prime} p d \wedge H N e x t P a r t s s^{\prime}\right)$

## definition

HPhase1or2ReadElse :: state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ Disk $\Rightarrow$ Proc $\Rightarrow$ bool where
HPhase1or2ReadElse s s'pdq=(Phase1or2ReadElse s s'pd q $s^{\prime}$ HNextPart $s$
$\left.s^{\prime}\right)$

## definition

HEndPhase2 :: state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ bool where
HEndPhase2 $s s^{\prime} p=\left(\right.$ EndPhase2 $s s^{\prime} p \wedge$ HNextPart $\left.s s^{\prime}\right)$

## definition

HFail $::$ state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ bool where
HFail s $s^{\prime} p=\left(\right.$ Fail s s $s^{\prime} p \wedge$ HNextPart $\left.s s^{\prime}\right)$

## definition

HPhaseORead $::$ state $\Rightarrow$ state $\Rightarrow$ Proc $\Rightarrow$ Disk $\Rightarrow$ bool where
HPhase0Read s s $s^{\prime} p=\left(\right.$ Phase0Read $s s^{\prime} p d \wedge$ HNextPart s $\left.s^{\prime}\right)$

## definition

```
HEndPhase0 :: state \(\Rightarrow\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool where
HEndPhase0 s \(s^{\prime} p=\left(\right.\) EndPhase0 s s \(s^{\prime} p \wedge\) HNextPart \(\left.s s^{\prime}\right)\)
```

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 $\Rightarrow$ bool where
Inv1 $s=(\forall p$.
inpt $s p \in$ Inputs
$\wedge$ phase s $p \leq 3$
$\wedge$ finite (allRdBlks sp))

```
definition HInv1 :: state \(\Rightarrow\) bool
where
    HInv1 \(s=\)
        (Inv1 s
    \(\wedge\) allInput \(s \subseteq\) Inputs)
```

declare HInv1-def [simp]

We added the assertion that the set all $R d B l k s p$ is finite for every process $p$; one may therefore choose a block with a maximum ballot number in action EndPhase1.

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

```
lemma HNextPart-Inv1: \llbracketHInv1 s;HNextPart s s'; Inv1 s'\rrbracket \LongrightarrowHInv1 s
    <proof\rangle
theorem HInit-HInv1: HInit s\longrightarrowHInv1 s
    <proof>
lemma allRdBlks-finite:
    assumes inv: HInv1 s
    and asm: }\forallp.allRdBlks s' p\subseteqinsert bk (allRdBlks s p
    shows }\forallp\mathrm{ . finite (allRdBlks s'p)
<proof>
```

```
theorem HPhase1or2ReadThen-HInv1:
    assumes inv1: HInv1 s
    and act: HPhase1or2ReadThen s s' pd q
    shows HInv1 s'
\langleproof\rangle
theorem HEndPhase1-HInv1:
    assumes inv1: HInv1 s
    and act:HEndPhase1 s s'p
    shows HInv1 s'
\langleproof\rangle
theorem HStartBallot-HInv1:
    assumes inv1: HInv1s
    and act:HStartBallot s s'p
    shows HInv1 s'
    <proof>
theorem HPhase1or2Write-HInv1:
    assumes inv1: HInv1 s
    and act:HPhase1or2Write s s' pd
    shows HInv1 s'
<proof\rangle
theorem HPhase1or2ReadElse-HInv1:
    assumes act: HPhase1or2ReadElse s s' p d q
    and inv1: HInv1s
    shows HInv1 s'
    <proof>
theorem HEndPhase2-HInv1:
    assumes inv1: HInv1 s
    and act:HEndPhase2 s s'p
    shows HInv1 s'
<proof\rangle
theorem HFail-HInv1:
    assumes inv1: HInv1 s
    and act:HFail s s'p
    shows HInv1 s'
<proof\rangle
theorem HPhase0Read-HInv1:
    assumes inv1: HInv1 s
    and act:HPhase0Read s s' p d
    shows HInv1 s'
<proof\rangle
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\rangle
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 \(r d B y::\) state \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) BlockProc set
where
    \(r d B y s p q d=\)
    \(\{b r . b r \in\) blocksRead s \(q d \wedge\) proc \(b r=p\}\)
definition blocksOf :: state \(\Rightarrow\) Proc \(\Rightarrow\) DiskBlock set
where
    blocksOf s \(p=\)
        \(\{\) dblock s \(p\}\)
    \(\cup\{\) disk s \(d p \mid d . d \in U N I V\}\)
    \(\cup\{\) block \(b r \mid b r . b r \in(U N q d . r d B y s p q d)\}\)
definition allBlocks :: state \(\Rightarrow\) DiskBlock set
    where allBlocks \(s=(U N p\). blocksOf \(s p)\)
definition Inv2a-innermost :: state \(\Rightarrow\) Proc \(\Rightarrow\) DiskBlock \(\Rightarrow\) bool
where
    Inv2a-innermost s \(p\) bk \(=\)
    \((\) mbal bk \(\in(\) Ballot \(p) \cup\{0\}\)
    \(\wedge\) bal bk \(\in(\) Ballot \(p) \cup\{0\}\)
    \(\wedge(b a l b k=0)=(\) inp \(b k=\) NotAnInput \()\)
    \(\wedge\) bal bk \(\leq\) mbal bk
    \(\wedge\) inp \(b k \in(\) allInput \(s) \cup\{\) NotAnInput \(\})\)
```

```
definition Inv2a-inner :: state }=>\mathrm{ Proc }=>\mathrm{ bool
    where Inv2a-inner s p = (\forallbk\inblocksOf s p. Inv2a-innermost s p bk)
definition Inv2a :: state }=>\mathrm{ bool
    where Inv2a s = ( }\forall\mathrm{ p. Inv2a-inner s p)
definition Inv2b-inner :: state }=>\mathrm{ Proc }=>\mathrm{ Disk }=>\mathrm{ bool
where
    Inv2b-inner s p d=
        ((d disksWritten s p}
            (phase s p}\in{1,2} ^ disk s d p= dblock s p))
        \wedge(phase s p }\in{1,2}
            ( (blocksRead s p d}\not={}\longrightarrow| < disksWritten s p)
            \wedge ᄀ hasRead s p d p)))
definition Inv2b :: state }=>\mathrm{ bool
    where Inv2b s = ( }\forallpd\mathrm{ . Inv2b-inner s p d)
definition Inv2c-inner :: state }=>\mathrm{ Proc }=>\mathrm{ bool
where
    Inv2c-inner s p=
        ((phase s p=0\longrightarrow
            ( dblock s p = InitDB
            \disksWritten s p = {}
                \wedge ( }\forall\mathrm{ d. }\forall\textrm{br}\in\mathrm{ blocksRead s p d.
                    proc br = p ^ block br = disk s d p)))
    \wedge(phase s p\not=0\longrightarrow
            ( mbal(dblock s p) E Ballot p
                \bal(dblock s p) \in Ballot p\cup{0}
                \wedge ( }\foralld.\forallbr\inblocksRead s p d.
                    mbal(block br) < mbal(dblock s p))))
    \wedge(phase s p \in{2,3} \longrightarrow 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\in allInput }s\cup{\mathrm{ NotAnInput}
    \wedge ( }\forall\textrm{p}.\quad\mathrm{ inpt s p G allInput s
            \wedge ( \text { chosen s = NotAnInput } \longrightarrow \text { outpt s p = NotAnInput )))}
definition Inv2c :: state }=>\mathrm{ bool
    where Inv2c s = ( }\forall\textrm{p}\mathrm{ . Inv2c-inner s p)
definition HInv2 :: state }=>\mathrm{ bool
    where HInv2 s = (Inv2a s ^ Inv2b s ^ Inv2c s)
```


## C.2.1 Proofs of Invariant 2 a

theorem HInit-Inv2a: HInit $s \longrightarrow$ Inv2a $s$ $\langle p r o o f\rangle$

For every action we define a action-blocks $O f$ lemma. We have two cases: ei-
ther the new blocks $O f$ is included in the old blocks $O f$, or the new blocks $O f$ is included in the old blocks $O f$ union the new dblock. In the former case the assumption $i n v$ 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:
    \llbracketHPhase1or2ReadThen s s'pd q\rrbracket\Longrightarrow blocksOf s'r}\subseteq\mathrm{ blocksOf s r
    <proof>
theorem HPhase1or2ReadThen-Inv2a:
    assumes inv: Inv2a s
    and act: HPhase1or2ReadThen s s' pd q
    shows Inv2a s'
<proof\rangle
lemma InitializePhase-rdBy:
    InitializePhase s s'}p\LongrightarrowrdBy s'ppqqdd\subseteqrdBysppqq dd
<proof\rangle
lemma HStartBallot-blocksOf:
    HStartBallot s s'}p\Longrightarrow\mathrm{ blocksOf s' q}\subseteq\mathrm{ blocksOf s q U{dblock s' q}
<proof\rangle
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)
\langleproof\rangle
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\rangle
theorem HStartBallot-Inv2a:
    assumes inv: Inv2a s
    and act:HStartBallot s s'p
    shows Inv2a s'
<proof\rangle
lemma HPhase1or2Write-blocksOf:
    \llbracketHPhase1or2Write s s'p d\rrbracket\Longrightarrow blocksOf s'r\subseteqblocksOf s r
    <proof\rangle
```

theorem HPhase1or2Write-Inv2a:
assumes inv: Inv2a s
and act: HPhase1or2Write $s s^{\prime} p d$

```
    shows Inv2a s'
<proof\rangle
theorem HPhase1or2ReadElse-Inv2a:
    assumes inv: Inv2a s
    and act: HPhase1or2ReadElse s s' p d q
    shows Inv2a s'
<proof>
lemma HEndPhase2-blocksOf:
    \llbracket HEndPhase2 s s' p\rrbracket\Longrightarrow blocksOf s' q\subseteqblocksOf s q
    <proof>
```

theorem HEndPhase2-Inv2a:
assumes inv: Inv2a $s$
and act: HEndPhase2 $s s^{\prime} p$
shows Inv2a $s^{\prime}$
〈proof〉
lemma HFail-blocksOf:
HFail s s'p blocksOf $s^{\prime} q \subseteq$ blocksOf $s q \cup\left\{\right.$ dblock $\left.s^{\prime} q\right\}$
$\langle$ proof $\rangle$

```
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\rangle
```

```
theorem HFail-Inv2a:
assumes inv: Inv2a s
    and act:HFail s s'p
    shows Inv2a s'
<proof\rangle
```

lemma HPhase0Read-blocksOf:
HPhase0Read s s'pd blocksOf $s^{\prime} q \subseteq$ blocksOf s $q$
〈proof〉

```
theorem HPhase0Read-Inv2a:
    assumes inv: Inv2a s
    and act: HPhase0Read s s' p d
    shows Inv2a s'
<proof\rangle
lemma HEndPhase0-blocksOf:
    HEndPhase0 s s'}p\Longrightarrow\mathrm{ blocksOf s'q}\subseteq\mathrm{ blocksOf s q U{dblock s'q}
    <proof\rangle
```

```
lemma HEndPhase0-blocksRead:
    assumes act: HEndPhase0 s s'p
    shows \existsd. blocksRead s p d\not={}
<proof\rangle
```

EndPhase0 has the additional difficulty of having a choose expression．We prove that there exists an $x$ such that the predicate of the choose expression holds，and then apply someI：？P ？$x \Longrightarrow$ ？P（Eps ？P）．
lemma HEndPhase0－some：
assumes act：HEndPhase0 s s＇p
and inv1：Inv1 $s$
shows（SOME b．$\quad b \in$ allBlocksRead $s p$
$\wedge(\forall t \in$ allBlocksRead s p．bal $t \leq$ bal b）
$) \in$ allBlocksRead s $p$
$\wedge(\forall t \in$ allBlocksRead s $p$ ．
bal $t \leq$ bal（SOME $b . \quad b \in$ allBlocksRead $s p$
$\wedge(\forall t \in$ allBlocksRead s p．bal $t \leq$ bal b）$))$
$\langle p r o o f\rangle$
lemma HEndPhase0－dblock－allBlocksRead：
assumes act：HEndPhase0 s s＇p
and inv1：Inv1 $s$
shows dblock $s^{\prime} p \in\left(\lambda x . x\left(m b a l:=m b a l\left(d b l o c k s^{\prime} p\right) \mid\right)\right.$＇allBlocksRead $s$ p
〈proof〉
lemma HNextPart－allInput－or－NotAnInput：
assumes act：HNextPart $s s^{\prime}$
and inv2a：Inv2a－innermost s $p$（dblock $s^{\prime} p$ ）
shows inp（dblock $\left.s^{\prime} p\right) \in$ allInput $s^{\prime} \cup\{$ 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 $\forall t \in\left(\lambda x . x \ m b a l:=m b a l\left(d b l o c k s^{\prime} p\right) D\right)$＇allBlocksRead s $p$ ．
Inv2a－innermost spt
$\langle p r o o f\rangle$
lemma HEndPhase0－Inv2a－dblock：
assumes act：HEndPhase0 $s s^{\prime} p$
and inv1：Inv1 s
and inv2a：Inv2a－inner s $p$
and inv2c：Inv2c－inner s $p$
shows Inv2a－innermost $s^{\prime} p\left(d b l o c k ~ s^{\prime} p\right)$
$\langle p r o o f\rangle$
lemma HEndPhase0－Inv2a－dblock－q：

```
    assumes act: HEndPhase0 s s'p
    and inv1: Inv1s
    and inv2a: Inv2a-inner s q
    and inv2c: Inv2c-inner s p
    shows Inv2a-innermost s' q(dblock s' q)
<proof\rangle
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\rangle
lemma HEndPhase1-blocksOf:
    HEndPhase1 s s'
\langleproof\rangle
lemma maxBlk-in-nonInitBlks:
    assumes b:b\in nonInitBlks s p
    and inv1: Inv1 s
    shows maxBlk s p}\in\mathrm{ nonInitBlks s p
        \wedge(\forallc\in nonInitBlks s p.bal c \leq bal (maxBlk s p))
<proof\rangle
lemma blocksOf-nonInitBlks:
    (\forallp bk.bk \in blocksOf s p\longrightarrowPbk)
        \Longrightarrow b k \in ~ n o n I n i t B l k s ~ s ~ p \longrightarrow P b k
    <proof\rangle
lemma maxBlk-allInput:
    assumes inv: Inv2a s
    and mblk: maxBlk s p n nonInitBlks s p
    shows inp (maxBlk s p)\in allInput s
<proof\rangle
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:HEndPhase1s 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\rangle
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\rangle
theorem HEndPhase1-Inv2a:
    assumes act:HEndPhase1 s s'p
    and inv1:HInv1s
    and inv: Inv2a s
    and inv2c: Inv2c-inner s p
    shows Inv2a s'
<proof\rangle
```


## 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\longrightarrowInv2b s
<proof\rangle
theorem HPhase1or2ReadThen-Inv2b:
    \llbracketInv2b s; HPhase1or2ReadThen s s'pd q\rrbracket
    Inv2b s'
\langleproof\rangle
```

theorem HStartBallot-Inv2b:
【Inv2b $s$; HStartBallot $s s^{\prime} p \rrbracket$
$\Longrightarrow$ Inv2b $s^{\prime}$
$\langle p r o o f\rangle$
theorem HPhase1or2Write-Inv2b:
【Inv2b $s$; HPhase1or2Write $s s^{\prime} p d \rrbracket$
$\Longrightarrow$ Inv2b $s^{\prime}$
$\langle p r o o f\rangle$

```
theorem HPhase1or2ReadElse-Inv2b:
    【Inv2b s; HPhase1or2ReadElse s \(s^{\prime} p d q \rrbracket\)
        \(\Longrightarrow\) Inv2b \(s^{\prime}\)
\(\langle p r o o f\rangle\)
```

theorem HEndPhase1-Inv2b:
【Inv2b $s ;$ HEndPhase1 $s s^{\prime} p \rrbracket \Longrightarrow$ Inv2b $s^{\prime}$
$\langle p r o o f\rangle$

```
theorem HFail-Inv2b:
    \llbracketInv2b s; HFail s s' p\rrbracket
    Cnv2b s'
\langleproof\rangle
```

theorem HEndPhase2-Inv2b:
【Inv2b $s$; HEndPhase2 $s s^{\prime} p \rrbracket \Longrightarrow$ Inv2b $s^{\prime}$
$\langle p r o o f\rangle$

## theorem HPhase0Read－Inv2b：

【Inv2b $s$ ；HPhase0Read $s s^{\prime} p d \rrbracket \Longrightarrow$ Inv2b $s^{\prime}$ $\langle p r o o f\rangle$

## theorem HEndPhase0－Inv2b：

【Inv2b $s$ ；HEndPhase0 $s s^{\prime} p \rrbracket \Longrightarrow$ Inv2b $s^{\prime}$ $\langle p r o o f\rangle$

## C．2．3 Proofs of Invariant 2 c

theorem HInit－Inv2c：HInit $s \longrightarrow$ Inv2c $s$
$\langle p r o o f\rangle$
lemma HNextPart－Inv2c－chosen：
assumes hnp：HNextPart s $s^{\prime}$
and inv2c：Inv2c s
and outpt ${ }^{\prime}: \forall p$ ．outpt $s^{\prime} p=\left(\right.$ if phase $s^{\prime} p=3$

$$
\text { then inp (dblock } \left.s^{\prime} p\right)
$$

else NotAnInput)
and inp－dblk：$\forall p$ ．inp（dblock $\left.s^{\prime} p\right) \in$ allInput $s^{\prime} \cup\{$ NotAnInput $\}$
shows chosen $s^{\prime} \in$ allInput $s^{\prime} \cup\{$ NotAnInput $\}$
$\langle p r o o f\rangle$

## lemma HNextPart－chosen：

assumes hnp：HNextPart s s＇
shows chosen $s^{\prime}=$ NotAnInput $\longrightarrow\left(\forall\right.$ p．outpt $s^{\prime} p=$ NotAnInput $)$
$\langle p r o o f\rangle$
lemma HNextPart－allInput：
$\llbracket H N e x t P a r t s s^{\prime} ; \operatorname{Inv2c} s \rrbracket \Longrightarrow \forall p$ ．inpt $s^{\prime} p \in$ allInput $s^{\prime}$〈proof〉
theorem HPhase1or2ReadThen－Inv2c：
assumes inv：Inv2cs
and act：HPhase1or2ReadThen $s s^{\prime} p d q$
and inv2a：Inv2a s
shows Inv2c s＇
$\langle p r o o f\rangle$

```
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' pd
    and inv2a: Inv2a s
    shows Inv2c s'
<proof\rangle
theorem HPhase1or2ReadElse-Inv2c:
    \llbracketInv2c s; HPhase1or2ReadElse s s'pd q; Inv2a s \rrbracket\Longrightarrow Inv2c s'
    <proof\rangle
theorem HEndPhase1-Inv2c:
    assumes inv: Inv2c s
    and act:HEndPhase1 s s'p
    and inv2a: Inv2a s
    and inv1:HInv1s
    shows Inv2c s'
<proof\rangle
theorem HEndPhase2-Inv2c:
    assumes inv: Inv2c s
    and act:HEndPhase2 s s'p
    and inv2a:Inv2a s
    shows Inv2c s'
<proof\rangle
theorem HFail-Inv2c:
    assumes inv: Inv2c s
    and act:HFail s s'p
    and inv2a: Inv2a s
    shows Inv2c s'
<proof\rangle
theorem HPhase0Read-Inv2c:
    assumes inv: Inv2c s
    and act: HPhase0Read s s' p d
    and inv2a:Inv2a s
    shows Inv2c s'
<proof\rangle
theorem HEndPhase0-Inv2c:
```

```
    assumes inv: Inv2c s
    and act:HEndPhase0 s s'p
    and inv2a: Inv2a s
    and inv1: Inv1s
    shows Inv2c s'
<proof>
```

theorem HInit-HInv2:
HInit $s \Longrightarrow$ HInv2 $s$
$\langle p r o o f\rangle$
$H \operatorname{Inv} 1 \wedge H I n v 2$ is an invariant of $H N e x t$.
lemma $I 2 b$ :
assumes nxt: HNext $s s^{\prime}$
and inv: HInv1 $s \wedge$ HInv2 $s$
shows HInv2 $s^{\prime}$
$\langle p r o o f\rangle$
end

## theory DiskPaxos-Inv3 imports DiskPaxos-Inv2 begin

## C. 3 Invariant 3

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

```
definition HInv3-L :: state \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) bool
where
    HInv3-L s p qd \(=\) (phase s \(p \in\{1,2\}\)
    \(\wedge\) phase s \(q \in\{1,2\}\)
    \(\wedge\) hasRead spdq
    \(\wedge\) hasRead s q d p)
definition HInv3-R :: state \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) bool
where
    HInv3-R s p q d \(=(0\) block \(=\) dblock \(s q\), proc \(=q) \in\) blocksRead spd
    \(\vee(\) block \(=\) dblock s \(p\), proc \(=p) \in\) blocksRead s q d)
definition HInv3-inner :: state \(\Rightarrow\) Proc \(\Rightarrow\) Proc \(\Rightarrow\) Disk \(\Rightarrow\) bool
    where HInv3-inner s p qd=(HInv3-L s p qd \(\longrightarrow\) HInv3-R s p q d)
definition HInv3 :: state \(\Rightarrow\) bool
    where HInv3 \(s=(\forall p q d\). HInv3-inner sp \(q d)\)
```


## C.3.1 Proofs of Invariant 3

theorem HInit-HInv3: HInit $s \Longrightarrow H I n v 3 s$

```
    <proof\rangle
lemma InitPhase-HInv3-p:
    \llbracketInitializePhase s s' p; HInv3-L s'p qd | \LongrightarrowHInv3-R s'p qd
    <proof\rangle
lemma InitPhase-HInv3-q:
    \llbracketInitializePhase s s'q;HInv3-L s'pqd\rrbracket\LongrightarrowHInv3-R s'pqd
    <proof\rangle
lemma HInv3-L-sym: HInv3-L s p qd \LongrightarrowHInv3-L s q p d
    <proof\rangle
lemma HInv3-R-sym:HInv3-R s p qd \LongrightarrowHInv3-R s q pd
    <proof>
lemma Phase1or2ReadThen-HInv3-pq:
    assumes act: Phase1or2ReadThen s s' p d q
    and inv-L': HInv3-L s' p qd
    and pq: p\not=q
    and inv2b: Inv2b s
    shows HInv3-R s' p q d
<proof\rangle
lemma Phase1or2ReadThen-HInv3-hasRead:
    | \neghasRead s pp dd qq;
        Phase1or2ReadThen s s'p d q;
        pp\not=p\veeqq\not=q\veedd\not=d\rrbracket
    \Longrightarrow ~ \neg h a s R e a d ~ s ' p p ~ d d ~ q q ~
    <proof\rangle
theorem HPhase1or2ReadThen-HInv3:
    assumes act: HPhase1or2ReadThen s s'p d q
    and inv: HInv3 s
    and pq: p\not=q
    and inv2b: Inv2b s
    shows HInv3 s'
<proof\rangle
lemma StartBallot-HInv3-p:
    \llbracketStartBallot s s' p; HInv3-L s'pqd\rrbracket
    #HInv3-R s' p qd
\langleproof\rangle
lemma StartBallot-HInv3-q:
    \llbracketStartBallot s s' q; HInv3-L s'pqd\rrbracket
    #HInv3-R s'pqd
    <proof>
```

```
lemma StartBallot-HInv3-nL:
    \StartBallot s s't;\negHInv3-L s p qd;t\not=p;t\not=q\rrbracket
        #HInv3-L s'pqd
    <proof\rangle
lemma StartBallot-HInv3-R:
    【StartBallot s s
        HInv3-R s' pqd
    <proof>
lemma StartBallot-HInv3-t:
    【StartBallot s s't; HInv3-inner s p qd;t\not=p;t\not=q\rrbracket
        HInv3-inner s}\mp@subsup{s}{}{\prime}pq
    <proof\rangle
lemma StartBallot-HInv3:
    assumes act: StartBallot s s
    and inv: HInv3-inner s p qd
    shows HInv3-inner s'pqd
<proof\rangle
theorem HStartBallot-HInv3:
    \llbracketHStartBallot s s' p;HInv3 s\rrbracket\LongrightarrowHInv3 s'
    <proof>
theorem HPhase1or2ReadElse-HInv3:
    \llbracket HPhase1or2ReadElse s s' pd q; HInv3 s\rrbracket\LongrightarrowHInv3 s'
    <proof\rangle
theorem HPhase1or2Write-HInv3:
    assumes act: HPhase1or2Write s s' p d
    and inv: HInv3 s
    shows HInv3 s'
<proof\rangle
lemma EndPhase1-HInv3-p:
    \llbracket EndPhase1 s s'p;HInv3-L s'pqd\rrbracket\LongrightarrowHInv3-R s'pqd
\langleproof\rangle
lemma EndPhase1-HInv3-q:
    \llbracketEndPhase1 s s'q;HInv3-L s'pqd\rrbracket\LongrightarrowHInv3-R s'p qd
    <proof\rangle
lemma EndPhase1-HInv3-nL:
    \llbracketEndPhase1 s s't;\negHInv3-L s p qd;t\not=p;t\not=q\rrbracket
                        \squareHInv3-L s'pqd
    <proof\rangle
lemma EndPhase1-HInv3-R:
```

```
    \llbracketEndPhase1 s s't; HInv3-R s p qd; t\not=p;t\not=q\rrbracket
    HInv3-R s'p qd
    <proof\rangle
lemma EndPhase1-HInv3-t:
    \llbracket EndPhase1 s s't; HInv3-inner s p qd; t\not=p;t\not=q\rrbracket
            HInv3-inner s' p qd
    <proof>
lemma EndPhase1-HInv3:
    assumes act: EndPhase1 s s't
    and inv: HInv3-inner s p qd
    shows HInv3-inner s'p qd
<proof\rangle
theorem HEndPhase1-HInv3:
    \llbracketHEndPhase1 s s'p;HInv3 s \rrbracket\LongrightarrowHInv3 s'
    <proof\rangle
lemma EndPhase2-HInv3-p:
    \llbracketEndPhase2 s s' p; HInv3-L s'p qd\rrbracket\LongrightarrowHInv3-R s' p qd
<proof\rangle
lemma EndPhase2-HInv3-q:
    \llbracketEndPhase2 s s'q;HInv3-L s'p qd\rrbracket\LongrightarrowHInv3-R s' p qd
    <proof\rangle
lemma EndPhase2-HInv3-nL:
    \llbracket EndPhase2 s s't;\negHInv3-L s p qd;t\not=p;t\not=q\rrbracket
                        \squareHInv3-L s'p qd
    <proof\rangle
lemma EndPhase2-HInv3-R:
    \llbracketEndPhase2 s s't; HInv3-R s p qd; t\not=p;t\not=q\rrbracket
                        HInv3-R s' p qd
    <proof\rangle
lemma EndPhase2-HInv3-t:
    \llbracket EndPhase2 s s't; HInv3-inner s p qd; t\not=p;t\not=q\rrbracket
                        HInv3-inner s' p qd
    <proof>
lemma EndPhase2-HInv3:
    assumes act: EndPhase2 s s
    and inv: HInv3-inner s p qd
    shows HInv3-inner s'pqd
<proof\rangle
theorem HEndPhase2-HInv3:
```

```
    \llbracketHEndPhase2 s s' p; HInv3 s\rrbracket\LongrightarrowHInv3 s'
    <proof\rangle
lemma Fail-HInv3-p:
    \llbracket Fail s s' p; HInv3-L s'pqd\rrbracket\LongrightarrowHInv3-R s'pqd
<proof\rangle
lemma Fail-HInv3-q:
    \llbracket Fail s s' q; HInv3-L s'pqd\rrbracket\LongrightarrowHInv3-R s'pqd
    <proof>
```

```
lemma Fail-HInv3-nL:
    \llbracket Fail s s
            \LongrightarrowHInv3-L s'pqd
    <proof\rangle
lemma Fail-HInv3-R:
    \llbracket Fail s s't; HInv3-R s p qd; t\not=p;t\not=q\rrbracket
            HInv3-R s' p qd
    <proof\rangle
lemma Fail-HInv3-t:
    \llbracket Fail s s't; HInv3-inner s p qd; t\not=p;t\not=q\rrbracket
            HInv3-inner s' p qd
        <proof>
lemma Fail-HInv3:
    assumes act: Fail s s'}
    and inv: HInv3-inner s p qd
    shows HInv3-inner s'pqd
<proof\rangle
theorem HFail-HInv3:
    \llbracketHFail s s' p;HInv3 s\rrbracket\LongrightarrowHInv3 s'
    <proof>
theorem HPhase0Read-HInv3:
    assumes act:HPhase0Read s s' p d
    and inv: HInv3 s
    shows HInv3 s'
<proof\rangle
lemma EndPhase0-HInv3-p:
    \llbracket EndPhase0 s s'p;HInv3-L s' p q d\rrbracket
    HInv3-R s'pqd
<proof\rangle
lemma EndPhase0-HInv3-q:
    \llbracketEndPhase0 s s' q; HInv3-L s' p q d \rrbracket
```

$$
\langle\text { proof }\rangle \quad \Longrightarrow H I n v 3-R s^{\prime} p q d
$$

lemma EndPhase0－HInv3－nL：
【EndPhase0 s $s^{\prime} t ; \neg H I n v 3-L$ s p $q d ; t \neq p ; t \neq q \rrbracket$
$\Longrightarrow \neg H I n v 3-L s^{\prime} p q d$
$\langle p r o o f\rangle$
lemma EndPhase0-HInv3-R:
【EndPhase0 s s't; HInv3-R spqd;t申p;tキq】
$\Longrightarrow$ HInv3-R $s^{\prime} p q d$
$\langle$ proof $\rangle$
lemma EndPhase0-HInv3-t:
$\llbracket$ EndPhase0 s s' $t$ HInv3-inner s p $q d ; t \neq p ; t \neq q \rrbracket$
$\Longrightarrow H I n v 3-i n n e r s^{\prime} p q d$
$\langle p r o o f\rangle$
lemma EndPhase0-HInv3:
assumes act: EndPhase0 $s s^{\prime} t$
and inv: HInv3-inner s $p q d$
shows HInv3-inner $s^{\prime} p q d$
$\langle p r o o f\rangle$
theorem HEndPhase0-HInv3:
$\llbracket$ HEndPhase0 s s'p; HInv3 s $\Longrightarrow H I n v 3 ~ s '$
〈proof〉
$H \operatorname{Inv} 1 \wedge H I n v 2 \wedge H I n v 3$ is an invariant of $H N e x t$.
lemma I2c：
assumes $n x t$ : HNext $s s^{\prime}$
and inv: HInv1 $s \wedge$ HInv2 $s \wedge H I n v 3 s$
shows HInv3 $s^{\prime}\langle$ proof $\rangle$
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．HInv $4 a$ 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．HInv $4 b$ conjunct asserts that，in phase 1 ，its $m b a l$ value is actually greater than the bal field of any of its blocks．HInv $4 c$ asserts that， in phase 2 ，its bal value is the mbal field of all its blocks on some majority
set of disks．HInv4d asserts that the bal field of any of its blocks is at most as large as the mbal field of all its disk blocks on some majority set of disks．

```
definition MajoritySet :: Disk set set
    where MajoritySet \(=\{D\). IsMajority \((D)\}\)
definition HInv4a1 :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv4a1 s \(p=(\forall b k \in\) blocksOf s \(p\). bal bk \(\leq\) mbal \((\) dblock s \(p))\)
definition HInv4a2 :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    HInv4a2 s \(p=(\forall D \in \operatorname{MajoritySet.} .(\exists d \in D . \operatorname{mbal}(\operatorname{disk} s d p) \leq m b a l(d b l o c k s\)
p)
                                    \(\wedge \operatorname{bal}(\operatorname{disk} s d p) \leq b a l(d b l o c k s p)))\)
definition HInv \(4 a\) :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv孔a s \(p=(\) phase s \(p \neq 0 \longrightarrow\) HInv4a1 s \(p \wedge\) HInv孔a2 s \(p)\)
definition HInv \(4 b\) :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv 4 b s \(p=(\) phase s \(p=1 \longrightarrow(\forall b k \in\) blocksOf s \(p\). bal bk \(<\operatorname{mbal}(\) dblock
\(s p)\) )
definition HInv4c :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv4c s \(p=(\) phase s \(p \in\{2,3\} \longrightarrow(\exists\) D MajoritySet. \(\forall d \in D\). mbal (disk
s d \(p)=\) bal (dblock sp)))
definition HInv4d \(::\) state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv d s \(p=(\forall b k \in\) blocksOf s \(p\). \(\exists D \in\) MajoritySet. \(\forall d \in D\). bal \(b k \leq\)
mbal (disk s d p))
definition HInv4 :: state \(\Rightarrow\) bool
    where HInv孔 \(s=(\forall p\). HInv孔a s \(p \wedge\) HInv孔b s \(p \wedge\) HInv孔c s \(p \wedge\) HInv孔d s \(p)\)
```

The initial state implies Invariant 4.

```
theorem HInit-HInv4: HInit \(s \Longrightarrow\) HInv4 \(s\)
```

    \(\langle p r o o f\rangle\)
    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 $H I n v 4 x s^{\prime} p$ ，we prove two lemmas．The first lemma action－HInv4x－p proves the case of $p=q$ ， while lemma action－H Inv4x－q proves the other case．

## C．4．1 Proofs of Invariant 4a

lemma HStartBallot－HInv4a1：
assumes act：HStartBallot $s s^{\prime} p$
and inv：HInv $4 a 1$ s $p$
and inv2a：Inv2a－inner $s^{\prime} p$

```
        shows HInv4a1 s'p
<proof\rangle
lemma HStartBallot-HInv4a2:
    assumes act:HStartBallot s s'p
    and inv:HInv4a2 s p
    shows HInv{a2 s' p
<proof>
lemma HStartBallot-HInv4a-p:
    assumes act:HStartBallot s s'p
    and inv: HInv&a s p
    and inv2a: Inv2a-inner s' p
    shows HInv4a s'p
<proof\rangle
lemma HStartBallot-HInv4a-q:
    assumes act: HStartBallot s s'p
    and inv:HInv_a s q
    and pnq: p\not=q
    shows HInv4a s' q
<proof\rangle
theorem HStartBallot-HInv4a:
    assumes act: HStartBallot s s'p
    and inv: HInv{a s q
    and inv2a: Inv2a s'
    shows HInv4a s' q
<proof\rangle
lemma Phase1or2Write-HInv_a1:
    \llbracketPhase1or2Write s s'pd; HInv4a1s q \rrbracket\LongrightarrowHInv&a1 s' q
    <proof\rangle
lemma Phase1or2Write-HInv_a2:
    \llbracketPhase1or2Write s s'pd; HInv&a2 s q\rrbracket\LongrightarrowHInv&a2 s' q
    <proof>
theorem HPhase1or2Write-HInv4a:
    assumes act: HPhase1or2Write s s}\mp@subsup{s}{}{\prime}p
    and inv:HInv_a s q
    shows HInv4a s' q
<proof\rangle
lemma HPhase1or2ReadThen-HInv4a1-p:
    assumes act: HPhase1or2ReadThen s s' p d q
    and inv: HInv&a1 s p
    shows HInv4a1 s' p
<proof\rangle
```

```
lemma HPhase1or2ReadThen-HInv&a2:
    \llbracketHPhase1or2ReadThen s s'pdr;HInv&a2 s q \rrbracket\LongrightarrowHInv&a2 s' q
    <proof\rangle
lemma HPhase1or2ReadThen-HInv4a-p:
    assumes act: HPhase1or2ReadThen s s' p dr
    and inv:HInv_a s p
    and inv2b: Inv2b s
    shows HInv4a s'p
\langleproof\rangle
lemma HPhase1or2ReadThen-HInv4a-q:
    assumes act: HPhase1or2ReadThen s s' p d r
    and inv: HInv_a s q
    and pnq: p\not=q
    shows HInv4a s' q
<proof\rangle
theorem HPhase1or2ReadThen-HInv4a:
    \llbracketHPhase1or2ReadThen s s'pdr; HInv4a s q;Inv2b s\rrbracket\LongrightarrowHInv&a s'q
    <proof\rangle
```

```
theorem HPhase1or2ReadElse-HInv{a:
    assumes act: HPhase1or2ReadElse s s'pdr
    and inv:HInv4a s q and inv2a: Inv2a s
    shows HInv4a s' q
<proof\rangle
```

lemma HEndPhase1-HInv孔a1:
assumes act: HEndPhase1 s s'p
and inv: HInv4a1 s p
shows HInv4a1 s' p
$\langle p r o o f\rangle$
lemma HEndPhase1-HInv孔a2:
assumes act: HEndPhase1 s s'p
and inv: HInv4a2 s $p$
and inv2a: Inv2a s
shows HInv4a2 s'p
$\langle p r o o f\rangle$
lemma HEndPhase1-HInv4a-p:
assumes act: HEndPhase1 s s'p
and inv: HInv孔a s p
and inv2a: Inv2a s
shows HInv4a s $s^{\prime} p$
〈proof〉

```
lemma HEndPhase1-HInv4a-q:
    assumes act:HEndPhase1 s s'p
    and inv:HInv_a s q
    and pnq: p\not=q
    shows HInv4a s' q
<proof\rangle
theorem HEndPhase1-HInv4a:
    \llbracketHEndPhase1 s s'p;HInv4a s q; Inv2a s\rrbracket\LongrightarrowHInv&a s'q
    <proof\rangle
theorem HFail-HInv4a:
    \llbracketHFail s s' p; HInv&as q\rrbracket\LongrightarrowHInv&a s'q
    <proof>
theorem HPhase0Read-HInv4a:
    \llbracketHPhase0Read s s'pd; HInv4as q q\rrbracket\LongrightarrowHInv4a s
    <proof\rangle
theorem HEndPhase2-HInv{a:
    \llbracketHEndPhase2 s s' p; HInv&a s q \ # HInv&a s'q
    <proof\rangle
lemma allSet:
    assumes aPQ: \foralla.\forallr\inPa.Qr and rb: rb\inPd
    shows Q rb
\langleproof\rangle
lemma EndPhase0-44:
    assumes act: EndPhase0 s s
    and bk:bk\in blocksOf s p
    and inv4d: HInv4d s p
    and inv2c: Inv2c-inner s p
    shows \existsd.\existsrb\inblocksRead s p d. bal bk\leqmbal(block rb)
<proof\rangle
lemma HEndPhase0-HInv4a1-p:
    assumes act:HEndPhase0 s s' p
    and inv2a':Inv2a s'
    and inv2c: Inv2c-inner s p
    and inv&d:HInv4dsp
    shows HInv4a1 s'p
<proof\rangle
lemma hasRead-allBlks:
    assumes inv2c: Inv2c-inner s p
    and phase: phase s p=0
    shows (\foralld\in{d. hasRead s p d p}. disk s d p \in allBlocksRead s p)
<proof\rangle
```

```
lemma HEndPhase0-41:
    assumes act: HEndPhase0 s s'p
    and inv1: Inv1 s
    and inv2c: Inv2c-inner s p
    shows \existsD\inMajoritySet.}\foralld\inD. mbal(disk s d p)\leqmbal(dblock s'p
                                \wedge bal(disk s d p) \leqbal(dblock s'p)
\langleproof\rangle
lemma Majority-exQ:
    assumes asm1: \existsD G MajoritySet. }\foralld\inD.P
    shows }\forallD\inMajoritySet.\existsd\inD.P
<proof\rangle
lemma HEndPhase0-HInv4a2-p:
    assumes act: HEndPhase0 s s'p
    and inv1: Inv1 s
    and inv2c: Inv2c-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: HInv_a s p
    shows HInv4a s'p
\langleproof\rangle
lemma HEndPhase0-HInv4a-q:
    assumes act:HEndPhase0 s s'p
    and inv: HInv_a s q
    and pnq: p\not=q
    shows HInv4a s' q
\langleproof\rangle
theorem HEndPhase0-HInv_a:
    \llbracketHEndPhase0 s s' p; HInv&a s q; HInv&d s p;
        Inv2a s; Inv1 s; Inv2a s; Inv2c s\rrbracket
    HInv4a s'q
    \langleproof\rangle
```


## C.4.2 Proofs of Invariant 4b

lemma blocksRead-allBlocksRead: $r b \in$ blocksRead spd block $r b \in$ allBlocksRead s $p$

```
    <proof\rangle
lemma HEndPhase0-dblock-mbal:
    \llbracketHEndPhase0 s s'p\rrbracket
            \Longrightarrow\forallbr\inallBlocksRead s p.mbal br < mbal(dblock s' p)
    \langleproof\rangle
lemma HEndPhase0-HInv{b-p-dblock:
    assumes act: HEndPhase0 s s'p
    and inv1: Inv1s
    and inv2a:Inv2a s
    and inv2c: Inv2c-inner s p
    shows bal(dblock s'p)<mbal(dblock s' p)
<proof\rangle
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\rangle
lemma HEndPhase0-HInv&b-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\rangle
lemma HEndPhase0-HInv4b-q:
    assumes act: HEndPhase0 s s'p
    and pnq: p\not=q
    and inv: HInv{b s q
    shows HInv4b s' q
\langleproof\rangle
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\rangle
```

```
lemma HStartBallot-HInv4b-p:
    assumes act: HStartBallot s s'p
    and inv2a: Inv2a-innermost s p (dblock s p)
    and inv&b: HInv4b s p
    and inv&a: HInv_a s p
    shows HInv4b s' p
<proof\rangle
lemma HStartBallot-HInv4b-q:
    assumes act:HStartBallot s s'p
    and pnq: p\not=q
    and inv: HInv{b s q
    shows HInv4b s' q
<proof>
theorem HStartBallot-HInv4b:
    assumes act: HStartBallot s s'p
    and inv2a:Inv2a s
    and inv4b: HInv{b s q
    andinv&a: HInv&a s p
    shows HInv4b s' q
<proof>
theorem HPhase1or2Write-HInv4b:
    \llbracketHPhase1or2Write s s' pd;HInv4b s q \ \LongrightarrowHInv4b s' q
    <proof\rangle
lemma HPhase1or2ReadThen-HInv{b-p:
    assumes act: HPhase1or2ReadThen s s' pd q
    and inv: HInv<b sp
    shows HInv4b s'p
<proof\rangle
lemma HPhase1or2ReadThen-HInv4b-q:
    assumes act: HPhase1or2ReadThen s s' p d r
    and inv: HInv4b s q
    and pnq: p\not=q
    shows HInv4b s' q
    <proof\rangle
theorem HPhase1or2ReadThen-HInv4b:
    \llbracketHPhase1or2ReadThen s s'p d q; HInv4b s r\rrbracket\Longrightarrow HInv4b s'r
    <proof\rangle
```

theorem HPhase1or2ReadElse-HInv4b:
【HPhase1or2ReadElse s s'pd $q$; HInv孔b s r;
Inv2a s; HInv孔a s p
$\Longrightarrow H I n v 4 b s^{\prime} r$

```
\langleproof\rangle
lemma HEndPhase1-HInv&b-p:
    HEndPhase1 s s' p\LongrightarrowHInv4b s'p
    <proof\rangle
lemma HEndPhase1-HInv{b-q:
    assumes act:HEndPhase1 s s'p
    and pnq: p\not=q
    and inv:HInv&b s q
    shows HInv4b s' q
<proof\rangle
theorem HEndPhase1-HInv{b:
    assumes act: HEndPhase1 s s'p
    and inv: HInv4b s q
    shows HInv4b s' q
\langleproof\rangle
lemma HEndPhase2-HInv{b-p:
    HEndPhase2 s s' p\LongrightarrowHInv4b s'p
    <proof\rangle
lemma HEndPhase2-HInv{b-q:
    assumes act:HEndPhase2 s s'p
    and pnq: p\not=q
    and inv: HInv4b s q
    shows HInv4b s' q
<proof\rangle
theorem HEndPhase2-HInv4b:
    assumes act:HEndPhase2 s s'p
    and inv: HInv4b s q
    shows HInv4b s' q
<proof\rangle
lemma HFail-HInv4b-p:
    HFail s s'}p\LongrightarrowHInv4b s'
    \langleproof\rangle
lemma HFail-HInv4b-q:
    assumes act: HFail s s'p
    and pnq: p\not=q
    and inv: HInv4b s q
    shows HInv4b s' q
<proof\rangle
theorem HFail-HInv4b:
    assumes act:HFail s s'p
```

```
    and inv: HInv4b s q
    shows HInv&b s' q
<proof\rangle
lemma HPhase0Read-HInv4b-p:
    HPhase0Read s s' p d \LongrightarrowHInv4b s'p
    \langleproof\rangle
lemma HPhase0Read-HInv&b-q:
    assumes act:HPhase0Read s s'pd
    and pnq: p\not=q
    and inv: HInv4b s q
    shows HInv4b s'q
<proof\rangle
theorem HPhase0Read-HInv4b:
    assumes act:HPhase0Read s s'pd
    and inv: HInv{b sq
    shows HInv4b s' q
<proof\rangle
```


## C.4.3 Proofs of Invariant 4c

```
lemma HStartBallot-HInv4c-p:
    \llbracketHStartBallot s s' p; HInv4c s p\rrbracket\Longrightarrow HInv4c s' p
    <proof\rangle
lemma HStartBallot-HInv4c-q:
    assumes act: HStartBallot s s'p
    and inv: HInv4c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HStartBallot-HInv4c:
    \llbracketHStartBallot s s' p; HInv4c s q\rrbracket\Longrightarrow HInv4c s' q
    <proof\rangle
lemma HPhase1or2Write-HInv4c-p:
    assumes act:HPhase1or2Write s s' pd
        and inv: HInv4c s p
        and inv2c: Inv2c s
    shows HInv4c s' p
<proof\rangle
lemma HPhase1or2Write-HInv4c-q:
    assumes act:HPhase1or2Write s s'p d
    and inv: HInv4c s q
    and pnq: p\not=q
```

```
    shows HInv4c s'q
<proof\rangle
theorem HPhase1or2Write-HInv4c:
    \llbracketHPhase1or2Write s s'pd; HInv4c s q;Inv2c s\rrbracket
        #Hnv4c s'q
    <proof>
lemma HPhase1or2ReadThen-HInv4c-p:
    \llbracketHPhase1or2ReadThen s s'pdq;HInv4c s p\rrbracket\LongrightarrowHInv4c s'p
    <proof\rangle
lemma HPhase1or2ReadThen-HInv4c-q:
    assumes act: HPhase1or2ReadThen s s
    and inv: HInv4c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HPhase1or2ReadThen-HInv4c:
    \llbracketHPhase1or2ReadThen s s' p d r; HInv4c s q\rrbracket
        \Longrightarrow H I n v 4 c ~ s ' q
    <proof\rangle
theorem HPhase1or2ReadElse-HInv4c:
    \llbracketHPhase1or2ReadElse s s'pdr; HInv4c s q\rrbracket\LongrightarrowHInv4c s'q
<proof\rangle
lemma HEndPhase1-HInv4c-p:
    assumes act:HEndPhase1 s s'p
    and inv2b: Inv2b s
    shows HInv4c s'p
<proof\rangle
lemma HEndPhase1-HInv4c-q:
    assumes act: HEndPhase1 s s'p
    and inv: HInv4c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HEndPhase1-HInv4c:
    \llbracketHEndPhase1 s s' p; HInv4c s q; Inv2b s\rrbracket\LongrightarrowHInv4c s' q
    <proof\rangle
lemma HEndPhase2-HInv4c-p:
    \llbracketHEndPhase2 s s' p;HInv4c s p\rrbracket\LongrightarrowHInv4c s' p
    <proof\rangle
```

```
lemma HEndPhase2-HInv4c-q:
    assumes act: HEndPhase2 s s'p
    and inv: HInv<c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HEndPhase2-HInv4c:
    \llbracketHEndPhase2 s s'p; HInv&c s q\rrbracket\LongrightarrowHInv&c s'q
    <proof\rangle
lemma HFail-HInv4c-p:
    |HFail s s' p;HInv4c s p\rrbracket\LongrightarrowHInv4c s'p
    <proof>
lemma HFail-HInv4c-q:
    assumes act:HFail s s'p
    and inv: HInv4c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HFail-HInv4c:
    \llbracketHFail s s' p;HInv4c s q\rrbracket \LongrightarrowHInv4c s' q
    \langleproof\rangle
lemma HPhase0Read-HInv4c-p:
    \llbracketHPhaseORead s s'pd; HInv4c s p\rrbracket\LongrightarrowHInv{c s'p
    <proof>
lemma HPhase0Read-HInv4c-q:
    assumes act:HPhase0Read s s'pd
    and inv: HInv4c s q
    and pnq: p\not=q
    shows HInv4c s' q
<proof\rangle
theorem HPhase0Read-HInv4c:
    \llbracketHPhase0Read s s'pd;HInv4c s q\rrbracket \LongrightarrowHInv4c s'q
    <proof>
lemma HEndPhase0-HInv4c-p:
    \llbracketHEndPhase0 s s' p;HInv4c s p\rrbracket\LongrightarrowHInv4c s' p
    <proof\rangle
lemma HEndPhase0-HInv4c-q:
    assumes act: HEndPhase0 s s'p
    and inv: HInv4c s q
    and pnq: p\not=q
```

```
    shows HInv4c s' q
<proof>
theorem HEndPhase0-HInv4c:
    \llbracketHEndPhase0 s s' p;HInv4c s q\rrbracket\Longrightarrow HInv4c s' q
    <proof>
```


## C．4．4 Proofs of Invariant 4d

lemma HStartBallot－HInv4d－p：
assumes act：HStartBallot $s s^{\prime} p$
and inv：HInv4d sp
shows HInv4d s＇p
$\langle p r o o f\rangle$
lemma HStartBallot－HInv4d－q：
assumes act：HStartBallot s $s^{\prime} p$
and inv：HInv4d s $q$
and $p n q: p \neq q$
shows HInv4d s＇$q$
$\langle p r o o f\rangle$
theorem HStartBallot－HInv4d：
$\llbracket$ HStartBallot s s＇p；HInv4d s q】 $\Longrightarrow$ HInv4d $s^{\prime} q$〈proof〉
lemma HPhase1or2Write－HInv4d－p： assumes act：HPhase1or2Write s $s^{\prime} p d$
and inv：HInv4d sp
and inv4a：HInvza s $p$
shows HInv4d $s^{\prime} p$
$\langle p r o o f\rangle$
lemma HPhase1or2Write－HInv4d－q：
assumes act：HPhase1or2Write $s s^{\prime} p d$
and inv：HInv4d s $q$
and $p n q: p \neq q$
shows HInv 4 d $s^{\prime} q$
〈proof〉

```
theorem HPhase1or2Write-HInv4d:
    \llbracketHPhase1or2Write s s' p d; HInv4d s q; HInv4a s p\rrbracket\Longrightarrow HInv4d s' q
    <proof\rangle
```

lemma HPhase1or2ReadThen－HInv4d－p：
assumes act：HPhase1or2ReadThen $s s^{\prime} p d q$
and inv：HInv4d s p
shows HInv4d s＇$p$
〈proof〉

```
lemma HPhase1or2ReadThen-HInv4d-q:
    assumes act: HPhase1or2ReadThen s s'pdr
    and inv: HInv{d s q
    and pnq: p\not=q
    shows HInv4d s' q
\langleproof\rangle
theorem HPhase1or2ReadThen-HInv4d:
    \llbracketHPhase1or2ReadThen s s'p dr; HInv4d s q\rrbracket\LongrightarrowHInv4d s' q
    <proof\rangle
theorem HPhase1or2ReadElse-HInv4d:
    \llbracketHPhase1or2ReadElse s s' p d r; HInv4d s q\rrbracket\LongrightarrowHInv4d s'q
<proof\rangle
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\rangle
lemma HEndPhase1-HInv4d-q:
    assumes act: HEndPhase1 s s'p
    and inv: HInv4d s q
    and pnq: p\not=q
    shows HInv4d s' q
<proof\rangle
theorem HEndPhase1-HInv4d:
    \llbracket HEndPhase1 s s' p; HInv4d s q; Inv2b s; HInv4c s p\rrbracket
        \Longrightarrow H I n v 4 d ~ s ' ~ q ~
    <proof\rangle
lemma HEndPhase2-HInv4d-p:
    assumes act:HEndPhase2 s s'p
    and inv:HInv4d s p
    shows HInv4d s'p
<proof\rangle
lemma HEndPhase2-HInv4d-q:
    assumes act: HEndPhase2 s s'p
    and inv: HInv{d s q
    and pnq: p\not=q
    shows HInv4d s' q
<proof>
```

```
theorem HEndPhase2-HInv{d:
    \llbracketHEndPhase2 s s' p; HInv&d s q\rrbracket\LongrightarrowHInv&d s'q
    <proof\rangle
lemma HFail-HInv4d-p:
    assumes act: HFail s s'p
    and inv: HInv4d s p
    shows HInv4d s'p
<proof\rangle
lemma HFail-HInv4d-q:
    assumes act:HFail s s'p
    and inv: HInv{d s q
    and pnq: p\not=q
    shows HInv4d s' q
\langleproof\rangle
theorem HFail-HInv4d:
    |HFail s s' p; HInv4d s q\rrbracket\Longrightarrow HInv{d s' q
    <proof\rangle
lemma HPhase0Read-HInv4d-p:
    assumes act:HPhase0Read s s' p d
    and inv: HInv4d s p
    shows HInv&d s'p
<proof\rangle
lemma HPhase0Read-HInv4d-q:
    assumes act:HPhase0Read s s'pd
    and inv: HInv4d s q
    and pnq: p\not=q
    shows HInv4d s' q
<proof\rangle
theorem HPhase0Read-HInv4d:
    \llbracketHPhase0Read s s' p d; HInv4d s q\rrbracket\Longrightarrow HInv4d s' q
    <proof\rangle
lemma HEndPhase0-blocksOf2:
    assumes act:HEndPhase0 s s'p
    and inv2c: Inv2c-inner s p
    shows allBlocksRead s p\subseteqblocksOf s p
<proof\rangle
lemma HEndPhase0-HInv4d-p:
    assumes act: HEndPhase0 s s'p
    and inv: HInv{d s p
    and inv2c: Inv2c s
    and inv1:Inv1 s
```

```
        shows HInv4d s' p
<proof\rangle
lemma HEndPhase0-HInv{d-q:
    assumes act: HEndPhase0 s s'p
    and inv: HInv4d s q
    and pnq: p\not=q
    shows HInv4d s' q
<proof\rangle
theorem HEndPhase0-HInv4d:
    \llbracket HEndPhase0 s s' p; HInv&d s q;
        Inv2c s; Inv1 s\rrbracket\Longrightarrow HInv4d s}\mp@subsup{s}{}{\prime}
    <proof>
```

Since we have already proved HInv2 is an invariant of $H N e x t, H I n v 1 \wedge$ $H I n v 2 \wedge H I n v 4$ 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 $\operatorname{maxBalInp}$, or else $p$ must eventually abort its current ballot. Processor $p$ will eventually abort its ballot if there is some processor $q$ and majority set $D$ such that $p$ has not read $q$ 's block on any disk $D$, and all of those blocks have mbal values greater than bal(dblocksp).

```
definition maxBalInp :: state \(\Rightarrow\) nat \(\Rightarrow\) InputsOrNi \(\Rightarrow\) bool
    where maxBalInp s b \(v=(\forall b k \in\) allBlocks \(s . b \leq b a l b k \longrightarrow i n p b k=v)\)
definition HInv5-inner- \(R\) :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
where
    HInv5-inner-R s \(p=\)
            (maxBalInp \(s(b a l(d b l o c k s p))(\) inp \((d b l o c k s p))\)
            \(\vee(\exists D \in\) MajoritySet. \(\exists q .(\forall d \in D\). bal (dblock s \(p)<\operatorname{mbal}(\) disk s d \(q)\)
                        \(\wedge \neg\) hasRead s p d q)))
definition HInv5-inner :: state \(\Rightarrow\) Proc \(\Rightarrow\) bool
    where HInv5-inner s \(p=(\) phase s \(p=2 \longrightarrow\) HInv5-inner-R s \(p)\)
```

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

## C．5． 1 Proof of Invariant 5

The initial state implies Invariant 5.

```
theorem HInit-HInv5: HInit s CHInv5 s
    <proof\rangle
```

We will use the notation used in the proofs of invariant 4，and prove the lemma action－H Inv5－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 $s^{\prime} p$
and inv：HInv5－inner s $p$
shows HInv5－inner s＇$p\langle p r o o f\rangle$
lemma HStartBallot－blocksOf－q：
assumes act：HStartBallot s $s^{\prime} p$
and $p n q: p \neq q$
shows blocksOf $s^{\prime} q \subseteq$ blocksOf s $q\langle p r o o f\rangle$
lemma HStartBallot－allBlocks： assumes act：HStartBallot s s $s^{\prime} p$ shows allBlocks $s^{\prime} \subseteq$ allBlocks $s \cup\left\{\right.$ dblock $\left.s^{\prime} p\right\}$
〈proof〉
lemma HStartBallot－HInv5－q1：
assumes act：HStartBallot s s $s^{\prime} p$
and $p n q: p \neq q$
and inv5－1：maxBalInp $s($ bal（dblock s q））（inp（dblock $s q))$
shows maxBalInp $s^{\prime}\left(\right.$ bal（dblock $\left.\left.s^{\prime} q\right)\right)\left(\right.$ inp（dblock $\left.\left.s^{\prime} q\right)\right)$
〈proof〉
lemma HStartBallot－HInv5－q2：
assumes act：HStartBallot s s $s^{\prime} p$
and $p n q$ ：$p \neq q$
and inv5－2：$\exists D \in$ MajoritySet．$\exists q q .(\forall d \in D . \quad b a l(d b l o c k s q)<m b a l(d i s k s d$ $q q)$ $\wedge \neg$ hasRead s q d qq）
shows $\exists D \in$ MajoritySet．$\exists q q .\left(\forall d \in D . \quad\right.$ bal $\left(\right.$ dblock $\left.s^{\prime} q\right)<\operatorname{mbal}\left(d i s k s^{\prime} d q q\right)$ $\wedge \neg$ hasRead $\left.s^{\prime} q d q q\right)$

```
\langleproof\rangle
lemma HStartBallot-HInv5-q:
    assumes act: HStartBallot s s'p
    and inv: HInv5-inner s q
    and pnq: p\not=q
    shows HInv5-inner s' q
    <proof\rangle
theorem HStartBallot-HInv5:
    \llbracket HStartBallot s s' p; HInv5-inner s q\rrbracket\Longrightarrow 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\rangle
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: \exists D\inMajoritySet. \exists q. (\foralld\inD. bal(dblock s p)<mbal(disk s d q)
                        \wedge ᄀhasRead s p d q)
    shows }\exists\mathrm{ D MajoritySet. }\exists\textrm{q}.(\foralld\inD.\quadbal(dblock s'p)<mbal(disk s'd q
                        \wedge \neghasRead s'pd q)
<proof\rangle
lemma HPhase1or2Write-HInv5-p:
    assumes act: HPhase1or2Write s s'pd
    and inv: HInv5-inner s p
    and inv4:HInv4c s p
    shows HInv5-inner s'p
<proof\rangle
lemma HPhase1or2 Write-allBlocks:
    assumes act: HPhase1or2Write s s' p d
    shows allBlocks s'\subseteqallBlocks s
    <proof\rangle
lemma HPhase1or2Write-HInv5-q2:
    assumes act: HPhase1or2Write s s'pd
    and pnq: p\not=q
    and inv&a: HInv_a s p
    and inv5-2: \existsD\inMajoritySet. \existsqq. ( }\foralld\inD.\quadbal(dblock s q)<mbal(disk s d
qq)
    \wedge \neghasRead s q d qq)
```

```
        shows }\existsD\inMajoritySet. \existsqq. (\foralld\inD.\quadbal(dblock s' q)< mbal(disk s'd qq
                        \wedge \neghasRead s' q d qq)
<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\not=q
    shows HInv5-inner s' q
<proof>
theorem HPhase1or2Write-HInv5:
    \llbracketHPhase1or2Write s s' pd; HInv5-inner s q;
        HInv4c s p; HInv&a s p \\Longrightarrow HInv5-inner s' q
    <proof\rangle
lemma HPhase1or2ReadThen-HInv5-1:
    assumes act: HPhase1or2ReadThen s s' p dr
    and inv5-1: maxBalInp s (bal(dblock s q)) (inp(dblock s q))
    shows maxBalInp s'(bal(dblock s'q)) (inp(dblock s' q))
    \langleproof\rangle
lemma HPhase1or2ReadThen-HInv5-p2:
    assumes act: HPhase1or2ReadThen s s' p dr
    and inv4c: HInv4c s p
    and inv2c: Inv2c-inner s p
    and phase: phase s p=2
    and inv5-2: \exists D\inMajoritySet. \exists q. (\foralld\inD. bal(dblock s p)<mbal(disk s d q)
                    \neghasRead s p d q)
    shows }\exists\mathrm{ D MajoritySet. }\exists\textrm{q}.(\foralld\inD.\quadbal(dblock s'p)<mbal(disk s'd q
                        \wedge hasRead s' pd q)
<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
<proof\rangle
lemma HPhase1or2ReadThen-allBlocks:
    assumes act: HPhase1or2ReadThen s s'p d r
    shows allBlocks s'\subseteqallBlocks s
    <proof>
lemma HPhase1or2ReadThen-HInv5-q2:
    assumes act: HPhase1or2ReadThen s s'p dr
```

and $p n q: p \neq q$
and inv4a：HInv4a s $p$
and inv5－2：$\exists D \in$ MajoritySet．$\exists q q .(\forall d \in D . \quad$ bal $(d b l o c k ~ s q)<m b a l(d i s k s d$ $q q)$

## $\wedge \neg h a s R e a d s q d q q)$

shows $\exists$ D $\in$ MajoritySet．$\exists q q .\left(\forall d \in D . \quad b a l\left(d b l o c k s^{\prime} q\right)<\operatorname{mbal}\left(d i s k s^{\prime} d q q\right)\right.$ $\wedge \neg$ hasRead s＇q d qq）
$\langle p r o o f\rangle$
lemma HPhase1or2ReadThen－HInv5－q：
assumes act：HPhase1or2ReadThen s s＇p dr
and inv：HInv5－inner s $q$
and inv4a：HInv4a s $p$
and $p n q$ ：$p \neq q$
shows HInv5－inner s＇$q$
$\langle p r o o f\rangle$
theorem HPhase1or2ReadThen－HInv5：
【HPhase1or2ReadThen $s s^{\prime} p d r$ ；HInv5－inner $s q$ ； Inv2c s；HInv4c s $p$ ；HInv孔a s $p \rrbracket \Longrightarrow H I n v 5-i n n e r s^{\prime} q$ $\langle p r o o f\rangle$
theorem HPhase1or2ReadElse－HInv5：
【HPhase1or2ReadElse $s s^{\prime} p d r$ ；HInv5－inner $s q \rrbracket$
$\Longrightarrow H I n v 5-i n n e r s^{\prime} q$
$\langle p r o o f\rangle$
lemma HEndPhase2－HInv5－p： HEndPhase2 $s s^{\prime} p \Longrightarrow$ HInv5－inner $s^{\prime} p$〈proof〉
lemma HEndPhase2－allBlocks： assumes act：HEndPhase2 $s s^{\prime} p$ shows allBlocks $s^{\prime} \subseteq$ allBlocks $s$〈proof〉
lemma HEndPhase2－HInv5－q1： assumes act：HEndPhase2 $s s^{\prime} p$ and $p n q: p \neq q$ and inv5－1：maxBalInp $s(b a l(d b l o c k s q))(i n p(d b l o c k s q))$ shows maxBalInp $s^{\prime}\left(b a l\left(\right.\right.$ dblock $\left.\left.s^{\prime} q\right)\right)\left(\operatorname{inp}\left(\right.\right.$ dblock $\left.\left.s^{\prime} q\right)\right)$
$\langle p r o o f\rangle$
lemma HEndPhase2－HInv5－q2：
assumes act：HEndPhase2 $s s^{\prime} p$
and $p n q: p \neq q$
and inv5－2：$\exists D \in$ MajoritySet．$\exists q q .(\forall d \in D . \quad$ bal $(d b l o c k s q)<m b a l(d i s k s d$ $q q)$

$$
\wedge \neg h a s R e a d s q d q q)
$$

```
    shows }\exists\textrm{D}\in\mathrm{ MajoritySet. }\existsqq.(\foralld\inD.\quadbal(dblock s' q)<mbal(disk s'd qq
                        \wedge \neghasRead s' q d qq)
<proof\rangle
lemma HEndPhase2-HInv5-q:
    assumes act:HEndPhase2 s s'p
    and inv: HInv5-inner s q
    and pnq: p\not=q
    shows HInv5-inner s' q
    <proof\rangle
theorem HEndPhase2-HInv5:
    \llbracketHEndPhase2 s s' p; HInv5-inner s q\rrbracket\LongrightarrowHInv5-inner s' q
    <proof\rangle
lemma HEndPhase1-HInv5-p:
    assumes act:HEndPhase1 s s' p
    and inv4:HInv4 s
    and inv2a:Inv2a s
    and inv2a': Inv2a s s
    and inv2c: Inv2c s
    and asm4:\negmaxBalInp s'(bal(dblock s' p)) (inp(dblock s' p))
    shows ( }\exists\textrm{D}\in\textrm{MajoritySet. \exists q. ( }\foralld\inD.\quadbal(dblock s' p)<mbal(disk s'd q
        \wedge ᄀhasRead s' p d q))
<proof\rangle
lemma union-inclusion:
\llbracketA\subseteq\mp@subsup{A}{}{\prime};B\subseteq\mp@subsup{B}{}{\prime}\rrbracket\LongrightarrowA\cupB\subseteq\mp@subsup{A}{}{\prime}\cup\mp@subsup{B}{}{\prime}
<proof\rangle
lemma HEndPhase1-blocksOf-q:
    assumes act:HEndPhase1 s s'p
    and pnq: p\not=q
    shows blocksOf s' q\subseteqblocksOf s q
<proof\rangle
lemma HEndPhase1-allBlocks:
    assumes act: HEndPhase1 s s'p
    shows allBlocks s'\subseteq allBlocks s\cup{dblock s'p}
<proof\rangle
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=
    and pnq: p\not=q
    and asm4: \negmaxBalInp s' (bal(dblock s' q)) (inp(dblock s' q))
    shows (\existsD\inMajoritySet. \existsqq. (\foralld\inD. bal(dblock s' q) < mbal(disk s'd qq)
        \wedge \neghasRead s' q d qq))
\langleproof\rangle
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\LongrightarrowHInv5-inner s}\mp@subsup{s}{}{\prime}
    <proof>
lemma HFail-blocksOf-q:
    assumes act: HFail s s'p
    and pnq: p\not=q
    shows blocksOf s' q\subseteq blocksOf s q
    <proof\rangle
lemma HFail-allBlocks:
    assumes act:HFail s s'p
    shows allBlocks s'\subseteqallBlocks s\cup{dblock s'p}
<proof\rangle
lemma HFail-HInv5-q1:
    assumes act:HFail s s'p
    and pnq: p\not=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\rangle
lemma HFail-HInv5-q2:
    assumes act:HFail s s'p
    and pnq: p\not=q
    and inv5-2: \existsD\inMajoritySet. \existsqq. ( }\foralld\inD.\quadbal(dblock s q)<mbal(disk s d
qq)
```

```
                        \wedge \neghasRead s q d qq)
    shows }\existsD\inMajoritySet. \existsqq. (\foralld\inD.\quadbal(dblock s'q)<mbal(disk s'd qq
                        \wedge \neghasRead s' q d qq)
<proof\rangle
lemma HFail-HInv5-q:
    assumes act:HFail s s' p
    and inv: HInv5-inner s q
    and pnq: p\not=q
    and inv2a: Inv2a s'
    shows HInv5-inner s' q
<proof\rangle
theorem HFail-HInv5:
    \llbracketHFail s s' p; HInv5-inner s q;Inv2a s'\rrbracket \LongrightarrowHInv5-inner s}\mp@subsup{s}{}{\prime}
\langleproof\rangle
lemma HPhase0Read-HInv5-p:
    HPhase0Read s s' pd\LongrightarrowHInv5-inner s}\mp@subsup{s}{}{\prime}
    <proof\rangle
lemma HPhase0Read-allBlocks:
    assumes act:HPhase0Read s s' pd
    shows allBlocks s'\subseteqallBlocks 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\rangle
lemma HPhase0Read-HInv5-q2:
    assumes act:HPhase0Read s s'pd
    and pnq: p\not=q
    and inv5-2: }\exists\textrm{D}\in\mathrm{ MajoritySet. }\exists\textrm{qq.}(\foralld\inD.\quadbal(dblock s q) < mbal(disk s d
qq)
    \wedge \neghasRead s q d qq)
    shows }\exists\mathrm{ D MajoritySet. }\exists\mathrm{ qq. ( }\foralld\inD.\quadbal(dblock s' q)< mbal(disk s'd qq
                        \wedge \neghasRead s' q d qq)
<proof\rangle
lemma HPhase0Read-HInv5-q:
    assumes act:HPhase0Read s s' p d
    and inv: HInv5-inner s q
    and pnq: p\not=q
    shows HInv5-inner s' q
<proof>
```

```
theorem HPhase0Read-HInv5:
    |HPhase0Read s s'pd;HInv5-inner s q\rrbracket\LongrightarrowHInv5-inner s' q
<proof\rangle
lemma HEndPhase0-HInv5-p:
    HEndPhase0 s s'}p\LongrightarrowHInv5-inner s'
    <proof\rangle
lemma HEndPhase0-blocksOf-q:
    assumes act:HEndPhase0 s s'p
    and pnq: p\not=q
    shows blocksOf s' q\subseteqblocksOf s q
<proof\rangle
lemma HEndPhase0-allBlocks:
    assumes act: HEndPhase0 s s'p
    shows allBlocks s'\subseteq allBlocks s \cup{dblock s'p}
<proof\rangle
lemma HEndPhase0-HInv5-q1:
    assumes act: HEndPhase0 s s'p
    and pnq: p\not=q
    and inv1:Inv1s
    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\not=q
    and inv5-2: \existsD\inMajoritySet. \exists qq. ( }\foralld\inD.\quadbal(dblock s q) < mbal(disk s d
qq)
                        \wedge ᄀhasRead s q d qq)
shows }\exists\textrm{D}\in\mathrm{ MajoritySet. }\exists\mathrm{ qq. ( }\foralld\inD.\quadbal(dblock s'q)<mbal(disk s'd qq
                        \wedge \neghasRead s' q d qq)
\langleproof\rangle
lemma HEndPhase0-HInv5-q:
    assumes act: HEndPhase0 s s'p
    and inv: HInv5-inner s q
    and inv1: Inv1s
    and pnq: p\not=q
    shows HInv5-inner s' q
    <proof>
theorem HEndPhase0-HInv5:
    \llbracketHEndPhase0 s s' p; HInv5-inner s q;Inv1 s\rrbracket\LongrightarrowHInv5-inner s' q
    <proof\rangle
```

$H \operatorname{Inv} 1 \wedge H \operatorname{Inv} 2 \wedge H \operatorname{Inv} 3 \wedge H \operatorname{Inv} 4 \wedge H \operatorname{Inv} 5$ is an invariant of HNext.

## lemma $22 e$ :

```
    assumes nxt: HNext s s'
    and inv:HInv1 s ^ HInv2 s ^ HInv2 s'^ HInv3 s ^ HInv4 s ^ HInv5 s
    shows HInv5 s'
    <proof\rangle
```

end
theory DiskPaxos-Chosen imports DiskPaxos-Inv5 begin

## C. 6 Lemma I2f

To prove the final conjunct we will use the predicate valueChosen $(v)$. This predicate is true if $v$ is the only possible value that can be chosen as output. It also asserts that, for every disk $d$ in $D$, if $q$ has already read disksdp, then it has read a block with bal field at least $b$.
definition valueChosen :: state $\Rightarrow$ InputsOrNi $\Rightarrow$ bool where
valueChosen $s v=$
$(\exists b \in(U N p$. Ballot $p)$. maxBalInp s b $v$
$\wedge(\exists p . \exists D \in$ MajoritySet. $(\forall d \in D . \quad b \leq \operatorname{bal}($ disk $s d p)$ $\wedge(\forall q \cdot(\quad$ phase s $q=1$
$\wedge b \leq m b a l($ dblock s $q$ )
$\wedge$ hasRead s q d p
$) \longrightarrow(\exists b r \in b l o c k s R e a d s q d . b \leq b a l($ block $b r))$
))))
lemma HEndPhase1-valueChosen-inp:
assumes act: HEndPhase1 $s s^{\prime} q$
and inv2a: Inv2a s
and asm1: $b \in(U N p$. Ballot $p)$
and bk-blocksOf: bk $\in$ blocksOf s r
and $b k: b k \in$ blocksSeen s $q$
and $b$-bal: $b \leq b a l b k$
and asm3: maxBalInp s bv
and inv1: Inv1 $s$
shows inp(dblock $\left.s^{\prime} q\right)=v$
〈proof〉
lemma HEndPhase1-maxBalInp:
assumes act: HEndPhase1 $s s^{\prime} q$
and asm1: $b \in(U N p$. Ballot $p)$
and asm2: $D \in$ MajoritySet
and asm3: maxBalInp s bv
and asm4: $\forall d \in D . \quad b \leq b a l(d i s k s d p)$

```
\wedge(\forallq.( phase s q = 1
    \wedgeb\leqmbal(dblock s q)
    hasRead s q d p
    )\longrightarrow(\existsbr\inblocksRead s q d. b \leq bal(block br)))
```

and inv1: Inv1 $s$
and inv2a: Inv2a $s$
and inv2b: Inv2b $s$
shows maxBalInp $s^{\prime} b v$
$\langle p r o o f\rangle$
lemma HEndPhase1-valueChosen2:
assumes act: HEndPhase1 s s $s^{\prime} q$
and asm4: $\forall d \in D . \quad b \leq \operatorname{bal}($ disk s d $p)$

$$
\begin{aligned}
& \wedge\left(\forall q \cdot \left(\begin{array}{c}
\text { phase s } q=1 \\
\quad \wedge b \leq \operatorname{mbal}(\text { dblock } s q)
\end{array}\right.\right.
\end{aligned}
$$

$\wedge$ hasRead s q d p
$) \longrightarrow(\exists b r \in$ blocksRead $s q d . b \leq b a l(b l o c k ~ b r)))($ is ?P $s)$
shows ?P $s^{\prime}$
$\langle p r o o f\rangle$

```
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\rangle
lemma HStartBallot-maxBalInp:
    assumes act:HStartBallot s s' q
    and asm3: maxBalInp s b v
    shows maxBalInp s'bv
<proof>
lemma HStartBallot-valueChosen2:
    assumes act: HStartBallot s s' q
    and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
        \wedge(\forallq.( phase s q=1
        \wedge b\leqmbal(dblock s q)
        ^hasRead s q d p
        )\longrightarrow(\existsbr\inblocksRead s q d. b \leq bal(block br)))(is ?P s)
    shows ?P s'
<proof\rangle
theorem HStartBallot-valueChosen:
    assumes act: HStartBallot s s' q
```

```
    and vc: valueChosen s v
    and v-input:v\inInputs
    shows valueChosen s'v
<proof\rangle
lemma HPhase1or2Write-maxBalInp:
    assumes act: HPhase1or2Write s s}\mp@subsup{s}{}{\prime}q
    and asm3: maxBalInp s bv
    shows maxBalInp s'bv
<proof\rangle
lemma HPhase1or2Write-valueChosen2:
    assumes act: HPhase1or2Write s s' ppd
        and asm2: D\inMajoritySet
    and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
        \wedge(\forallq.) phase s q=1
            \wedge b\leqmbal(dblock s q)
            hasRead s q d p
                            )\longrightarrow(\existsbr\inblocksRead s q d. b \leqbal(block br)))(is ?P s)
    and inv4:HInv4a s pp
    shows ?P s'
<proof\rangle
theorem HPhase1or2Write-valueChosen:
    assumes act: HPhase1or2Write s s' qd
    and vc: valueChosen s v
    and v-input:v\in Inputs
    and inv4:HInv4a s q
    shows valueChosen s'v
<proof\rangle
lemma HPhase1or2ReadThen-maxBalInp:
    assumes act: HPhase1or2ReadThen s s' qd p
    and asm3: maxBalInp s b v
    shows maxBalInp s' b v
<proof>
lemma HPhase1or2ReadThen-valueChosen2:
    assumes act: HPhase1orQReadThen s s'q d pp
        and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
            \wedge(\forallq.( phase s q = 1
            \wedge b\leqmbal(dblocksq)
            ^ hasRead s q d p
            )\longrightarrow(\existsbr\inblocksRead s q d. b \leqbal(block br)))(is ?P s)
    shows ?P s'
<proof\rangle
theorem HPhase1or2ReadThen-valueChosen:
```

```
assumes act: HPhase1or2ReadThen s s' q d p
and vc: valueChosen s v
and v-input:}v\in\mathrm{ Inputs
shows valueChosen s'v
<proof\rangle
theorem HPhase1or2ReadElse-valueChosen:
    \llbracketHPhase1or2ReadElse s s' pdr;valueChosen s v;v\in Inputs \rrbracket
        \Longrightarrow \text { valueChosen s'v}
    <proof\rangle
lemma HEndPhase2-maxBalInp:
    assumes act:HEndPhase2 s s'q
        and asm3: maxBalInp s b v
    shows maxBalInp s'bv
\langleproof\rangle
lemma HEndPhase2-valueChosen2:
    assumes act: HEndPhase2 s s'q
        and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
            \wedge(\forallq.( phase s q=1
                            \wedge b\leqmbal(dblock s q)
                            \hasRead s q d p
                            )\longrightarrow(\existsbr\inblocksRead s q d.b\leqbal(block br)))(is ?P s)
    shows ?P s'
<proof\rangle
theorem HEndPhase2-valueChosen:
    assumes act:HEndPhase2 s s'q
    and vc: valueChosen s v
    and v-input:v\in Inputs
    shows valueChosen s'v
<proof\rangle
lemma HFail-maxBalInp:
    assumes act: HFail s s'q
        and asm1:b\in(UN p. Ballot p)
        and asm3: maxBalInp s b v
    shows maxBalInp s' b v
<proof\rangle
lemma HFail-valueChosen2:
    assumes act:HFail s s'q
        and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
    \wedge(\forallq.( phase s q=1
        \wedge b\leqmbal(dblock s q)
        hasRead s qd p
        )\longrightarrow(\existsbr\inblocksRead s q d. b \leq bal(block br)))(is ?P s)
        shows ?P s'
```

```
\langleproof\rangle
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\rangle
lemma HPhase0Read-maxBalInp:
    assumes act:HPhase0Read s s' q d
    and asm3: maxBalInp s b v
    shows maxBalInp s'bv
<proof\rangle
lemma HPhase0Read-valueChosen2:
    assumes act: HPhase0Read s s' qq dd
    and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
        \wedge(\forallq.( phase s q=1
            \wedge b\leqmbal(dblock s q)
            \asRead s q d p
                            )\longrightarrow(\existsbr\inblocksRead s q d.b\leqbal(block br)))(is ?P s)
    shows ?P s'
<proof\rangle
theorem HPhase0Read-valueChosen:
    assumes act:HPhase0Read s s' q d
    and vc: valueChosen s v
    and v-input:v\inInputs
    shows valueChosen s'v
\langleproof\rangle
lemma HEndPhase0-maxBalInp:
    assumes act: HEndPhase0 s s'q
        and asm3: maxBalInp s b v
        and inv1: Inv1 s
    shows maxBalInp s'b v
\langleproof\rangle
lemma HEndPhase0-valueChosen2:
    assumes act: HEndPhase0 s s'q
        and asm4: }\foralld\inD.\quadb\leqbal(disk s d p
            \wedge(\forallq.( phase s q=1
        ^ b\leqmbal(dblock s q)
    hasRead s q d p
    )\longrightarrow(\existsbr\inblocksRead s q d. b\leqbal(block br)))(is ?P s)
    shows ?P s'
<proof\rangle
```

```
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\rangle
end
```

theory DiskPaxos-Inv6 imports DiskPaxos-Chosen begin

## C． 7 Invariant 6

The final conjunct of HInv asserts that，once an output has been cho－ sen，valueChosen（chosen）holds，and each processor＇s output equals either chosen or NotAnInput．
definition HInv6 ：：state $\Rightarrow$ bool where

```
    HInv6 \(s=((\) chosen \(s \neq\) NotAnInput \(\longrightarrow\) valueChosen \(s(\) chosen \(s))\)
```

                        \(\wedge(\forall\) p. outpt s \(p \in\{\) chosen \(s\), NotAnInput \(\}))\)
    theorem HInit-HInv6: HInit $s \Longrightarrow H I n v 6 s$
〈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 $s^{\prime} \neq$ NotAnInput
shows valueChosen $s^{\prime}$ (chosen $s^{\prime}$ )
〈proof〉
lemma valueChosen-equal-case:
assumes max-v: maxBalInp s b v
and Dmaj: $D \in$ MajoritySet
and asm-v: $\forall d \in D . b \leq b a l(d i s k$ s $d p)$
and max-w: maxBalInp s ba w
and Damaj: Da $\in$ MajoritySet
and asm-w: $\forall d \in D a . b a \leq b a l(d i s k s d p a)$
and $b-b a$ : $b \leq b a$
shows $v=w$
$\langle p r o o f\rangle$

```
lemma valueChosen-equal:
    assumes v: valueChosen s v
    and w: valueChosen s w
    shows v=w\langleproof\rangle
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}\not==\mathrm{ NotAnInput
    shows outpt s'r}=\mathrm{ chosen s}\mp@subsup{}{}{\prime
<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\rangle
lemma outpt-chosen:
    assumes outpt: outpt s= outpt s'
    and inv2c: Inv2c s
    and nextp: HNextPart s s'
    shows chosen s'}=\mathrm{ chosen s
<proof\rangle
lemma outpt-Inv6:
    \llbracket outpt s= outpt s';}\forallp\mathrm{ . outpt s p { {chosen s,NotAnInput};
        Inv2c s;HNextPart s s'\ \Longrightarrow \forallp.outpt s' p { {chosen s',NotAnInput}
    <proof\rangle
theorem HStartBallot-Inv6:
    assumes act:HStartBallot s s'p
    and inv: HInv6 s
    and inv2c: Inv2c s
    shows HInv6 s'
\langleproof\rangle
theorem HPhase1or2Write-Inv6:
    assumes act: HPhase1or2Write s s' pd
    and inv: HInv6 s
    and inv4:HInv4 a s p
```

```
    and inv2c:Inv2c s
    shows HInv6 s'
<proof\rangle
theorem HPhase1or2ReadThen-Inv6:
    assumes act: HPhase1or2ReadThen s s'p d q
    and inv: HInv6 s
    and inv2c: Inv2c s
    shows HInv6 s'
<proof\rangle
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:HEndPhase1s 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'
\langleproof\rangle
lemma outpt-chosen-2:
    assumes outpt: outpt s'=(outpt s) (p:= NotAnInput)
    and inv2c: Inv2c s
    and nextp:HNextPart s s'
    shows chosen s}=\mathrm{ chosen s'
<proof\rangle
lemma outpt-HInv6-2:
    assumes outpt:outpt s' = (outpt s) (p:= NotAnInput)
    and inv: }\forallp\mathrm{ . outpt s p G{chosen s,NotAnInput}
    and inv2c: Inv2c s
    and nextp: HNextPart s s'
    shows }\forallp\mathrm{ . outpt s' p}\in{\mathrm{ chosen }\mp@subsup{s}{}{\prime},NotAnInput
\langleproof\rangle
theorem HFail-Inv6:
    assumes act:HFail s s' p
    and inv: HInv6 s
    and inv2c: Inv2c s
    shows HInv6 s'
<proof\rangle
```

```
theorem HPhase0Read-Inv6:
    assumes act: HPhase0Read s s'pd
    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:Inv1s
    and inv2c: Inv2c s
    shows HInv6 s'
<proof\rangle
HInv 1^HInv2^HInv2'^HInv 3^HInv 4^HInv5^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\rangle
end
```

theory DiskPaxos-Invariant imports DiskPaxos-Inv6 begin

## C. 8 The Complete Invariant

```
definition HInv :: state }=>\mathrm{ bool
```

where
HInv $s=($ HInv1 $s$
$\wedge$ HInv2 s
$\wedge$ HInv3 s
$\wedge$ HInv4 s
$\wedge$ HInv5 s
$\wedge$ HInv6 s)
theorem I1:
HInit $s \Longrightarrow$ HInv s
〈proof〉
theorem I2:
assumes inv: HInv s
and $n x t$ : HNext s s ${ }^{\prime}$
shows HInv s ${ }^{\prime}$
$\langle p r o o f\rangle$
end
theory DiskPaxos imports DiskPaxos-Invariant begin

## C. 9 Inner Module

record
Istate $=$
iinput :: Proc $\Rightarrow$ InputsOrNi
ioutput :: Proc $\Rightarrow$ InputsOrNi
ichosen :: InputsOrNi
iallInput :: InputsOrNi set
definition IInit :: Istate $\Rightarrow$ bool where
IInit $s=($ range $($ iinput $s) \subseteq$ Inputs
$\wedge$ ioutput $s=(\lambda p$. NotAnInput $)$
$\wedge$ ichosen $s=$ NotAnInput
$\wedge$ iallInput $s=\operatorname{range}($ iinput $s))$
definition IChoose :: Istate $\Rightarrow$ Istate $\Rightarrow$ Proc $\Rightarrow$ bool
where
IChoose s s' $p=($ ioutput $s p=$ NotAnInput
$\wedge$ (if (ichosen $s=$ NotAnInput)
then $\left(\exists\right.$ ip $\in$ iallInput $s$. ichosen $s^{\prime}=i p$
$\wedge$ ioutput $s^{\prime}=($ ioutput $\left.s)(p:=i p)\right)$
else $\left(\right.$ ioutput $s^{\prime}=($ ioutput $s)(p:=$ ichosen $s)$
$\wedge$ ichosen $s^{\prime}=$ ichosen $\left.s\right)$ )
$\wedge$ iinput $s^{\prime}=$ iinput $s \wedge$ iallInput $s^{\prime}=$ iallInput $\left.s\right)$
definition IFail :: Istate $\Rightarrow$ Istate $\Rightarrow$ Proc $\Rightarrow$ bool where

```
    IFail \(s s^{\prime} p=\left(\right.\) ioutput \(s^{\prime}=(\) ioutput \(s)(p:=\) NotAnInput \()\)
            \(\wedge\left(\exists\right.\) ip \(\in\) Inputs. iinput \(s^{\prime}=(\) iinput \(s)(p:=\) ip \()\)
                            \(\wedge\) iallInput \(s^{\prime}=\) iallInput \(\left.s \cup\{i p\}\right)\)
                            \(\wedge\) ichosen \(s^{\prime}=\) ichosen \(s\) )
definition INext :: Istate \(\Rightarrow\) Istate \(\Rightarrow\) bool
    where INext \(s s^{\prime}=\left(\exists p\right.\). IChoose \(s s^{\prime} p \vee\) IFail \(\left.s s^{\prime} p\right)\)
definition s2is :: state \(\Rightarrow\) Istate
where
    s2is \(s=\\) iinput \(=\) inpt \(s\),
        ioutput \(=\) outpt \(s\),
        ichosen=chosen \(s\),
        iallInput \(=\) allInput \(s\) )
```


## theorem R1:

$\llbracket$ HInit $s$; is $=$ s2is $s \rrbracket \Longrightarrow$ Init is $\langle$ proof $\rangle$

## theorem $R 2 b$ :

assumes inv: HInv s and $i n v{ }^{\prime}$ : HInv $s^{\prime}$ and nxt: HNext $s s^{\prime}$ and srel: is=s2is $s \wedge i s^{\prime}=s 2 i s s^{\prime}$
shows $\left(\exists p\right.$. IFail is is' $p \vee$ IChoose is is $\left.s^{\prime} p\right) \vee i s=i s^{\prime}$ $\langle p r o o f\rangle$
end


[^0]:    ${ }^{1}$ 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.

