Specifying and Verifying Fault-Tolerant Systems

Leslie Lamport and Stephan Merz
lamport@src.dec.com merz@src.dec.com

Digital Equipment Corporation
Systems Research Center

25 July 1994
minor correction: 14 October 1994

To appear in Proceedings of the Third International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, held 19–23 September 1994 in Lübeck, Germany.

Specifying and Verifying Fault-Tolerant Systems

Leslie Lamport and Stephan Merz

Digital Equipment Corporation, Systems Research Center

Abstract. We formally specify a well known solution to the Byzantine generals problem and give a rigorous, hierarchically structured proof of its correctness. We demonstrate that this is an engineering exercise, requiring no new scientific ideas.

1 Introduction

Assertional verification of concurrent systems began almost twenty years ago with the work of Ashcroft [4]. By the early 1980's, the basic principles of formal specification and verification of concurrent systems were known [10, 12, 19]. More precisely, we had learned how to specify and verify those aspects of a system that can be expressed as the correctness of an individual execution. Fault-tolerant systems are just one class of concurrent systems; they require no special techniques.

The most important problems that remain are in the realm of engineering, not science. Scientific ideas must be translated into engineering practice. We describe here what we believe to be a suitable framework for an engineering discipline of formal specification and verification.

The limited space provided by these proceedings, and the limited time and patience of the authors, have forced us to choose a simple example—the specification and hierarchical verification of a well known fault-tolerant algorithm. Our example is OM(1), the one-traitor "oral-message" solution to the Byzantine generals problem [16]. In this problem, there is a collection of generals—a commander and a set of lieutenants—who communicate with one another by message. Any of the generals, including the commander, may be a traitor. The commander must send an order to all the lieutenants so that (i) all loyal lieutenants agree on the same order o, and (ii) if the commander is loyal, then o is the order that she issued. Algorithm OM(1) satisfies these conditions if at most one of the generals is a traitor. We augment the traditional statement of the problem by requiring that all loyal generals choose their order within some fixed time after the start of the algorithm. A solution to the Byzantine generals problem lies at the heart of a fault-tolerant system in which faulty processors can exhibit completely arbitrary behavior.

This algorithm was formally specified and verified in 1983, in an appendix to a final report [20]. Even then, it was considered too straightforward an exercise to be worth writing up separately for publication. The specification and verification of fault-tolerant algorithms is not rocket science, but it is still not standard engineering practice. Most of the literature on verification concentrates on the underlying formalism and ignores the problem of defining a language for specifying real systems [11, 17, 18]. The literature on specification languages generally ignores the problem of reasoning formally about specifications of real systems [8, 9].

We address these practical issues by using existing tools: a precisely defined specification language and a hierarchical proof method. Section 2 contains the formal specifications of the problem and the algorithm. There are three specifications, a high-level problem specification, a mid-level specification of the algorithm at roughly the level of detail provided in [16], and a low-level specification that more realistically models message passing. Section 3 proves that each specification implements the next higher-level one. A correctness property of the high-level specification is also proved. Section 4 discusses the specifications and proofs.

2 The Formal Specifications

Our specifications are written in TLA$^+$, a complete specification language based on TLA, the Temporal Logic of Actions. The semantics of TLA is defined in terms of states and behaviors. A state is an assignment of values to variables, and a behavior is an infinite sequence of states. A TLA formula is interpreted as a boolean function on behaviors.

In TLA, a system is modeled by choosing variables whose values describe the system's state, and an execution of the system is represented by a behavior. The system is specified by a TLA formula that is true of a behavior iff (if and only if) that behavior represents a correct execution of the system. A specification is a mathematical formula with precisely defined semantics. The correspondence between the real system and the mathematical formula lies in the interpretation of the formula's variables. The free variables of the specification represent the system's interface—the part of the system that is being specified. A description of TLA and its proof rules can be found in [13]. However, we try to explain the meaning of the TLA formulas in our specification well enough so they can be read with no prior knowledge of TLA.

Some formalisms describe systems in terms of events (often called actions) rather than states. An event in such a formalism corresponds to a change to the value of an interface variable in a TLA specification. The basic method of writing and reasoning about specifications is the same for event-based and state-based formalisms.

TLA$^+$ provides a language for writing TLA specifications. In addition to the operators of TLA, it contains operators for defining and manipulating data structures and syntactic structures for handling large specifications. The first published description of TLA$^+$ was in [15]. Since then, the following changes have been made to the language: (i) explicit specification of sorts is no longer required for definitions, (ii) the EXCEPT construct (described below) has replaced the earlier syntax for the same operator, (iii) single square brackets have replaced double square brackets for record operators. (Record operators are not used here.) All of these changes preceded work on this particular example.

Some formulas in the specifications have been annotated with boxed numbers, such as $\boxed{98}$. A boxed number refers to the corresponding number in the margin of the text, such as the one on this line, which marks the point where the formula 98
is explained.

Our specifications provide a crash course in TLA$^+$, since they use almost all the basic syntactic features of the language and many of its predefined operators. Figures 21–23 at the end of this paper list all the syntactic constructs and predefined operators of TLA$^+$. Ones that are used in the specifications are annotated with pointers to the places in the text where they are explained.

2.1 The High-Level Specification

The high-level specification appears in Figure 1. It begins with a module named
SpecParams. The module is the basic unit of a TLA$^+$ specification. It is a col- 1
lection of declarations, definitions, assumptions, and theorems.

The **import** statement imports the contents of the modules *FiniteSets* and 2
Reals. This statement has almost the same effect as inserting the text of these modules into module *SpecParams*. The only difference is that when another module imports or includes module *SpecParams*, it does not obtain the definitions that *SpecParams* imported from modules *FiniteSets* and *Reals*.

Module *FiniteSets* defines $IsFiniteSet(S)$ to equal TRUE iff S is a finite set. 3
The module *Reals* defines the set *Real* of real numbers, with the zero element 0 and the usual operators $+$, $*$, $<$, and $\leq$ on real numbers. Our specification can be understood without knowing precisely how these imported operators are defined. For the reader who wants to see the formal definitions, we include module *FiniteSets* without further explanation in Figure 20, at the end of the paper. We omit module *Reals*. (Starting with the predefined module *Naturals* that defines the natural numbers, it takes fewer than fifty lines to define the set *Real* and all the operators on real numbers used in our specification.)

Returning to module *SpecParams*, we next encounter a **parameters** state- 4
ment, which declares the module's parameters. The parameters are the free symbols of a module. All formulas defined in a module can be expressed in terms of its parameters and the built-in operators of TLA$^+$. There are two classes of
module parameters, constants and variables. Constants are the rigid variables of 5
temporal logic; they represent quantities that are unchanged during a behavior. Variables are the flexible variables of temporal logic; they represent quantities that can change during the course of a behavior. The constant parameters of the specification are:

Cmdr	The commander.	6
Lt	The set of lieutenants.	
Order	The set of all possible orders.	
Now_0	The time at which the algorithm starts.	
Δ	The maximum length of time it can take for the loyal generals to decide on their values.	

module *SpecParams*[1]

import *FiniteSets, Reals*[3]

parameters $Cmdr, Lt, Order, Now_0, \Delta$[6] : CONSTANT
$status, ord, now$[7] : VARIABLE

assumption $HiAssump$[11] $\triangleq$

$$\begin{array}{l}
\land\ (IsFiniteSet(Lt)) \land (Cmdr \notin Lt) \\
\land\ \text{“?”} \notin Order \\
\land\ Now_0 \in Real \\
\land\ (\Delta \in Real) \land (0 < \Delta)
\end{array}$$

Gen[13] $\triangleq\ Lt \cup \{Cmdr\}$

$Loyal(g)$[13] $\triangleq\ status[g] = \text{“Loyal”}$

module *HiLevel*

import *SpecParams, Reals, RealTime*

$$\begin{array}{lcl}
LInit(g)^{[20]} & \triangleq & \land\ status[g] \in \{\text{“Loyal”}, \text{“Traitor”}\} \\
 & & \land\ Loyal(g) \Rightarrow (ord[g] = \text{“?”}) \\
CInit^{[18]} & \triangleq & \land\ status[Cmdr] \in \{\text{“Loyal”}, \text{“Traitor”}\} \\
 & & \land\ Loyal(Cmdr) \Rightarrow (ord[Cmdr] \in Order) \\
Fail(g)^{[19]} & \triangleq & status'[g] = \text{“Traitor”} \\
Choose(g)^{[21]} & \triangleq & \land\ Loyal(g) \land (ord[g] = \text{“?”}) \\
 & & \land\ ord'[g] \in Order \\
 & & \land\ \forall h \in Gen : Loyal(h)' \land (ord'[h] \neq \text{“?”}) \Rightarrow (ord'[g] = ord'[h]) \\
 & & \land\ \text{UNCHANGED } status[g] \\
var(g)^{[18]} & \triangleq & \langle ord[g], status[g] \rangle \\
CSpec^{[15]} & \triangleq & CInit \land \Box[Fail(Cmdr)]_{var(Cmdr)} \\
LSpec(g)^{[20]} & \triangleq & \land\ LInit(g) \land \Box[Choose(g) \lor Fail(g)]_{var(g)} \\
 & & \land\ \Box(Loyal(g) \land (ord[g] = \text{“?”}) \Rightarrow (now \leq Now_0 + \Delta)) \\
Spec^{[23]} & \triangleq & \land\ CSpec \\
 & & \land\ \forall g \in Lt : LSpec(g) \\
 & & \land\ (now = Now_0) \\
 & & \land\ RT(\langle ord, status \rangle)
\end{array}$$

theorem

$$\begin{array}{l}
SpecGood^{[25]} \triangleq Spec \Rightarrow \Box \forall g \in Gen : \\
\qquad Loyal(g) \land (Now_0 + \Delta < now) \\
\qquad\quad \Rightarrow \land\ ord[g] \in Order \\
\qquad\qquad\ \ \land\ \forall h \in Gen : Loyal(h) \Rightarrow (ord[g] = ord[h])
\end{array}$$

Fig. 1. The high-level specification.

5

The variable parameters are:

status For each general g, the value of $status[g]$ will be the string "Loyal" or "Traitor", denoting whether or not the general is loyal. (Strings, denoted by expressions of the form "... ", are a predefined data type of TLA$^+$.) The Byzantine generals problem is expressed in terms of which generals are loyal. In the formal specification, the state must describe which generals are loyal. Hence, we introduce the variable *status*. 7

ord For each general g, the value of $ord[g]$ denotes the order chosen by general g, or the string "?" if g has not yet chosen an order.

now The variable *now* will be a real number that denotes the current time.

In our informal discussion, we describe the values that variables will have in a behavior that satisfies the specification. TLA is a typeless logic, which means that a variable can assume any value. (More precisely, for any value v and variable x, there is a state in which the value of x is v.) Thus, when we say that the value of $status[g]$ will be a string, we mean that its value will be a string in every state of every behavior that satisfies the specification we are about to describe.

The ⊢———⊣ lines are primarily decorative, though the second one serves 8
to delimit the **assumption** section.

An **assumption** section asserts assumptions about constant parameters. In 9
this module, there is a single assumption named *HiAssump*, which is defined to be the expression to the right of the $\triangleq$. In general, the symbol $\triangleq$ denotes 10
that the identifier to its left is defined to equal the expression to its right. The formula *HiAssump* is the conjunction of four assertions: 11

1. *Lt* (the set of lieutenants) is a finite set that does not contain *Cmdr*.
2. "?" is not a possible order.
3. Now_0 is a real number.
4. Δ is a positive real number.

TLA$^+$ uses the notation that a list of expressions bulleted by $\wedge$ denotes their 12
conjunction, and a list of expressions bulleted by $\vee$ denotes their disjunction. Indentation is used to eliminate parentheses.

Following the **assumption** section is a section consisting of two definitions. The constant operator *Gen* is defined to be the set of all generals—both the 13
lieutenants and the commander. The boolean operator *Loyal* is defined so that $Loyal(x)$ equals TRUE iff $status[x]$ equals "Loyal", for any x. (We care about the value of $Loyal(x)$ only when x is an element of *Gen*.) Round parentheses denote application of an operator to its argument (or arguments). The symbol *Loyal* by itself, without an argument, is not a syntactically complete expression. Square brackets denote function application. Both *status* and $status[e]$ are syntactically complete expressions that denote values, for any expression e.

The └———┘ line ends the module. 14

The actual specification is contained in module *HiLevel*, which comes next. This module imports the modules *SpecParams*, *Reals*, and *RealTime*.

The *RealTime* module is used to express real-time properties. It is essentially the same module as in [15], which in turn used the definitions from [2].[1] Specifying real-time properties is an engineering problem that is solved by applying standard methods. We will explain the operators in the *RealTime* module as they appear in our specification. The module is given in Figure 20.

Importing *SpecParams* provides module *HiLevel* with the specification parameters and the definitions of *Gen* and *Loyal*. Importing *SpecParams* does not import the modules *FiniteSets* and *Reals* that it in turn imports.

Instead of reading module *HiLevel* in sequence, we examine next the temporal
15 formula *CSpec*, which specifies the commander. This formula has the canonical form $Init \wedge \Box[Next]_v$ for a process specification[2], where

Init is a *predicate*—a Boolean expression made from constants and variables. It describes the initial state of the process's variables.

Next is an *action*—a boolean expression made from constants, variables, and primed variables. It describes all steps (pairs of successive states) that can change the process's variables. In an action, unprimed variables denote values in the first (starting) state and primed variables denote values in the second (final) state.

v is a *state function*—a nonboolean expression made from constants and variables. It is usually a tuple that describes all components of the process's state.

16 The formula $[Next]_v$ denotes $Next \vee (v' = v)$, so it represents a step that is either a *Next* step or one that leaves v (and hence all variables in the tuple v)
17 unchanged. The temporal operator $\Box$ means *always*, so $Init \wedge \Box[Next]_v$ is true for a behavior (infinite sequence of states) iff the first state satisfies *Init* and every successive pair of states is either a *Next* step or leaves v unchanged.

The commander's state is described by $ord[Cmdr]$ and $status[Cmdr]$, which
18 are the components of the tuple $var(Cmdr)$. (In TLA$^+$, tuples are enclosed in angle brackets $\langle \ldots \rangle$.) The initial condition *CInit* asserts that

1. The commander's status is either "Loyal" or "Traitor".
2. If her status is "Loyal", then her order is an element of *Order*. The symbol $\Rightarrow$ denotes logical implication. It has lower precedence (binds more loosely) than any other boolean operator.

19 The commander's next-state relation is $Fail(Cmdr)$, which asserts that the new value of $status[Cmdr]$ is "Traitor". This allows any step that ends in a state with $status[Cmdr]$ equal to "Traitor". Such a step can change $ord[Cmdr]$ to any value. Thus, the formula *CSpec* is satisfied by any behavior that either (i) consists of an infinite number of states with $status[Cmdr] =$ "Loyal" and $ord[g] = o$, for some o in *Order*, or (ii) starts with a finite (possibly empty) sequence of such

[1] The definition of *VTimer* in the *RealTime* module of [15] contains a typographical error; the correct definition appears in [2] and Figure 20.

[2] Since real-time conditions are used to specify progress, there are no fairness conditions in our processes.

states and ends with an infinite sequence of states in which *status*[*Cmdr*] equals "Traitor" and *ord*[*Cmdr*] assumes completely arbitrary values.

The formula *CSpec* describes only the values assumed by *ord*[*Cmdr*] and *status*[*Cmdr*]. It makes no assertion about any other part of the state—such as the value of the variable *now*, or of *ord*[g] for g different from *Cmdr*, or of a variable *foo* that might be introduced later.

The formula *LSpec*(g) is the specification of lieutenant g. It is the conjunction 20
of two formulas. The first asserts that *LInit*(g) holds initially and that each step is a *Choose*(g) step, a *Fail*(g) step, or a step that leaves lieutenant g's state unchanged. The initial predicate *LInit*(g) asserts that (i) *status*[g] has a correct value, and (ii) if g is loyal, then *ord*[g] equals "?", denoting that g has not yet chosen an order. A *Choose*(g) step is one in which 21

1. In the starting a state, g is loyal and has not yet chosen an order. (Any step taken by a traitorous general can be interpreted as a *Fail* step.)
2. In the final state, g has chosen an order.
3. For every general h that, in the final state, is loyal and has chosen an order, the order that h has chosen is the same as the one that g has chosen.
4. g's status does not change, which by (1) implies that he is still loyal in the final state. The formula UNCHANGED f is an abbreviation for $f' = f$. 22

The second conjunct of *LSpec*(g) is of the form $\Box P$, for a predicate P. Such a formula asserts of a behavior that P is true in every state. In this case, the formula asserts that if g is loyal and has not chosen an order, then *now* is at most $Now_0 + \Delta$. In other words, it asserts that if *now* is greater than $Now_0 + \Delta$ and g is loyal, then he must have chosen an order. Since we interpret *now* as the current time, the second conjunct of *LSpec*(g) asserts that if g is loyal, then he must choose an order by time $Now_0 + \Delta$.

The formula *Spec* is the complete high-level specification. It asserts 23

1. The commander's specification *CSpec*.
2. The specification *LSpec*(g), for each lieutenant g.
3. $now = Now_0$, which means that the current time initially (at the start of the algorithm) equals Now_0.
4. Formula $RT(\langle ord, status \rangle)$. Module *RealTime* defines formula $RT(v)$ to assert that (a) *now* is a monotonically nondecreasing real number and (b) steps that change *now* leave v unchanged. Thus, $RT(\langle ord, status \rangle)$ asserts that (a) *now* changes the way we expect time to change, and (b) *ord* and *status* do not change when *now* does. Thus, in steps that change *ord* and *status*, the value of *now* is the same in the initial and final state. Intuitively, this means that we are considering changes to *ord* and *status* to happen instantaneously. 24

Formula *Spec* is the high-level specification. This means that we consider a behavior to represent a correct execution of the algorithm iff it satisfies *Spec*. A specification is therefore a definition, and a definition cannot be right or wrong. However, a specifications can fail to capture our intent. To gain confidence in a specification, we can prove theorems about it. Such theorems usually have the

form $Spec \Rightarrow Prop$, which asserts that every behavior satisfying the specification *Spec* also satisfies the property *Prop*. Module *HiLevel* asserts such a theorem,
25 named *SpecGood*. This theorem states that, in any behavior satisfying *Spec*, it is always the case that, for any general g, if g is loyal and the time is later than $Now_0 + \Delta$, then g has chosen an order and his order is the same as that of every other loyal general. In other words, every loyal general chooses an order by time $Now_0 + \Delta$, and that order is the same as any other loyal general's order. All theorems are proved in Section 3.

2.2 The Mid-Level Specification

The mid-level specification describes Algorithm OM(1) of [16], an "oral-message" Byzantine agreement algorithm that works in the presence of at most one traitor. It is a two-round algorithm. In the first round, the commander sends her message to all lieutenants. In the second round, each lieutenant relays the message he received to all other lieutenants. A lieutenant chooses his order by applying a *majority* function to the values that he has received. The only requirements on this majority function are (i) if the same order o is received from all but one of the other generals, then o is chosen, and (ii) all lieutenants use the same majority function.

This informal description is essentially the one given in [16]. It contains several tacit assumptions—for example, that every lieutenant receives a value in the first round, even if the commander is a traitor. The formal specification makes these assumptions explicit.

26 Module *MidLevelParams* of Figure 2 declares the following new parameters that are used in the mid-level specification.

- *rcvd* For each lieutenant g, the order that g receives directly from the commander is recorded in $rcvd[g][g]$, and the relayed order that g receives from each other lieutenant h is recorded in $rcvd[g][h]$. Thus, for each lieutenant g, the value of $rcvd[g]$ will be a function whose domain is the set Lt of lieutenants. If lieutenant g is loyal, then before any orders have been sent, $rcvd[g][h]$ will equal "?", for all lieutenants h.
- *Majority* The majority function. More precisely, *Majority* is an operator with a single argument. We care about the value of $Majority(f)$ only when f is a function from Lt to the set *Order* of orders.
- δ The maximum delay between when an order is sent and when it is received. This delay applies both to the sending of orders by the commander and to the relaying of orders by the lieutenants. (The delay δ includes the time needed to decide to send an order.)
- ϵ The maximum time it takes a lieutenant to make a decision. In this specification, the only decision he has to make is to choose an order once he has received all the relayed orders.

27 Assumption *MidAssump* makes the following assertion about the constant parameters.

module *MidLevelParams*

import *SpecParams, Reals*
export[29] *MidLevelParams, SpecParams, Reals*

parameters *rcvd*[26] : VARIABLE
$Majority(\_), \delta, \epsilon$[26] : CONSTANT

assumption

$$
\begin{array}{lll}
MidAssump^{[27]} & \triangleq & \wedge\ (\delta \in Real) \wedge (\epsilon \in Real) \wedge (2 * \delta + \epsilon \leq \Delta) \\
& & \wedge\ \forall f \in [Lt \rightarrow Order]^{[28]} : \\
& & \quad \wedge\ Majority(f) \in Order \\
& & \quad \wedge\ \forall\, o \in Order : (\exists\, h \in Lt : \forall\, g \in Lt \setminus \{h\} : f[g] = o) \\
& & \qquad\qquad \Rightarrow (Majority(f) = o)
\end{array}
$$

Fig. 2. Module *MidLevelParams*.

1. δ and ϵ are real numbers such that $2 * \delta + \epsilon$ is at most Δ.
2. For any function f that maps lieutenants to orders:
   1. $Majority(f)$ is an order.
   2. If o is an order such that $f[g]$ equals o for every lieutenant except some lieutenant h, then $Majority(f)$ equals o.

   The expression $[Lt \rightarrow Order]$ denotes the set of all functions f with domain 28
   Lt such that $f[g] \in Order$, for all $g \in Lt$.

The **export** statement causes the named definitions to be imported by any mod- 29
ule that imports *MidLevelParams*. A module name stands for all definitions from that module. (Omitting the **export** statement would be equivalent to writing **export** *MidLevelParams*.)

Module *MidLevel* in Figure 3 defines the formula *Spec* to be the mid-level specification. It first imports two other modules. Importing *MidLevelParams* imports all its parameters, including the ones it imported from *SpecParams*.

The **include** statement incorporates the definitions from module *HiLevel*, 30
with all defined symbols prefixed by "*Hi.*"; for example, a definition of $Hi.var$ is included that makes $Hi.var(h)$ equal to $\langle ord[h], status[h] \rangle$. Parameters of an included module are instantiated by expressions. In the absence of explicit instantiation (described below), a parameter is instantiated by the symbol of the same name. Thus, the parameter *Cmdr* of *HiLevel* is instantiated by *Cmdr* (a parameter of *MidLevel*, obtained via the **import** statement); the parameter *ord* of *HiLevel* is instantiated by *ord*; etc. When a module is included, its assumptions (with the appropriate instantiations) must be provable from the assumptions and definitions of the including module. Thus, assumption $Hi.HiAssump$, the assumption of the included module *HiLevel*, must be provable from the assumptions and definitions of module *MidLevel*. This assumption follows trivially from assumption *HiLevel*, which is imported from module *MidLevelParams*, which in turn imports it from module *SpecParams*.

module *MidLevel*

import *MidLevelParams, RealTime*

include *HiLevel* **as** *Hi*[30]

$$
\begin{array}{lll}
LInit(g)^{\boxed{32}} & \triangleq & \land\ Hi.LInit(g) \\
 & & \land\ Loyal(g) \Rightarrow (rcvd[g] = [h \in Lt \mapsto \text{“?”}])
\end{array}
$$

$$
\begin{array}{lll}
Issue(g)^{\boxed{34}} & \triangleq & \land\ Loyal(g) \land rcvd[g][g] = \text{“?”} \\
 & & \land\ \exists\, o \in Order : \begin{array}[t]{l} \land\ rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![g] = o] \\ \land\ Loyal(Cmdr) \Rightarrow (o = ord[Cmdr]) \end{array} \\
 & & \land\ \text{UNCHANGED } \langle ord[g], status[g] \rangle
\end{array}
$$

$$
\begin{array}{lll}
Relay(g, h)^{\boxed{35}} & \triangleq & \land\ Loyal(g) \land rcvd[g][h] = \text{“?”} \\
 & & \land\ \exists\, o \in Order : \begin{array}[t]{l} \land\ rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![h] = o] \\ \land\ Loyal(h) \Rightarrow (o = rcvd[h][h]) \end{array} \\
 & & \land\ \text{UNCHANGED } \langle ord[g], status[g] \rangle
\end{array}
$$

$$
\begin{array}{lll}
Choose(g) & \triangleq & \land\ Loyal(g) \land ord[g] = \text{“?”} \\
 & & \land\ \forall\, h \in Lt : rcvd[g][h] \neq \text{“?”} \\
 & & \land\ ord'[g] = Majority(rcvd[g]) \\
 & & \land\ \text{UNCHANGED } \langle rcvd[g], status[g] \rangle
\end{array}
$$

$$
Next(g)^{\boxed{33}} \triangleq Issue(g) \lor (\exists\, h \in Lt \setminus \{g\} : Relay(g, h)) \lor Choose(g) \lor Hi.Fail(g)^{\boxed{36}}
$$

$$
var(g) \triangleq \langle ord[g], rcvd[g], status[g] \rangle
$$

$$
\begin{array}{lll}
LSpec(g) & \triangleq & \land\ LInit(g) \land \Box[Next(g)]^{\boxed{32}}_{var(g)} \\
 & & \land\ \Box(Loyal(g) \land (rcvd[g][g] = \text{“?”}) \Rightarrow (now \leq Now_0 + \delta))^{\boxed{37}} \\
 & & \land\ \forall\, h \in Lt \setminus \{g\} : \\
 & & \qquad \Box(Loyal(g) \land (rcvd[g][h] = \text{“?”}) \Rightarrow (now \leq Now_0 + 2 * \delta))^{\boxed{37}} \\
 & & \land\ \boldsymbol{\exists}\, t : \begin{array}[t]{l} \land\ VTimer(t, Choose(g), \epsilon, var(g)) \\ \land\ MaxTimer(t)^{\boxed{38}} \end{array}
\end{array}
$$

$$
\begin{array}{lll}
Spec^{\boxed{31}} & \triangleq & \land\ Hi.CSpec \\
 & & \land\ \forall\, g \in Lt : LSpec(g) \\
 & & \land\ now = Now_0 \\
 & & \land\ RT(\langle ord, rcvd, status \rangle)
\end{array}
$$

$$
OneTraitor^{\boxed{40}} \triangleq \exists\, h \in Gen : \forall\, g \in Gen \setminus \{h\} : Loyal(g)
$$

theorem

$$
MidCorrect^{\boxed{40}} \triangleq (\Box OneTraitor) \land Spec \Rightarrow Hi.Spec
$$

Fig. 3. Module *MidLevel*

Formula *Spec* is the specification of the mid-level algorithm. It is similar to 31
the specification *Spec* of module *HiLevel*, consisting of the conjunction of four formulas that assert: (i) the specification *Hi.CSpec* of the commander included from module *HiLevel*, (ii) formula $LSpec(g)$, for every lieutenant g, (iii) *now* is initially equal to Now_0, and (iv) formula $RT(\langle ord, rcvd, status \rangle)$. As explained in Section 2.1, $RT(\langle ord, rcvd, status \rangle)$ asserts that *now* behaves the way we expect it to and that *ord*, *rcvd*, and *status* change instantaneously. The rest of the specification involves the definition of $LSpec(g)$, the specification of lieutenant g. Formula $LSpec(g)$ is the conjunction of four formulas, which we describe in turn.

The first conjunct of $LSpec(g)$ has the canonical form $Init \wedge \Box[Next]_v$ ex- 32
plained in Section 2.1. The initial predicate $LInit(g)$ asserts (i) the initial condition $Hi.LInit(g)$ on $status[g]$ and $ord[g]$ and (ii) that $rcvd[g]$ is a function with domain Lt such that $rcvd[g][h]$ equals "?" for every h in Lt. In general, the construct $[x \in S \mapsto exp(x)]$ denotes the function f with domain S such that
$f[x] = exp(x)$ for all x in S. The next-state relation $Next(g)$ is the disjunction 33
of four actions:

$Issue(g)$ Represents the sending of an order by the commander to lieutenant g. 34
It is enabled iff g is loyal and $rcvd[g][g]$ equals "?", denoting that g has not yet received an order from the commander. The action sets $rcvd[g][g]$ to an order that, if the commander is loyal, is actually her order. The notation $[f \text{ EXCEPT } ![x] = exp]$ denotes a function $\widehat{f}$ that is the same as f except that $\widehat{f}[x]$ equals exp.

$\exists\, h \in Lt \backslash \{g\} : Relay(g, h)$ Asserts that a $Relay(g, h)$ step occurs, for some lieu- 35
tenant h other than g. Such a step represents the relaying of an order from h to g. If h is loyal, then the relayed order is $rcvd[h][h]$.

$Hi.Fail(g)$ As described above, an action taken when g is or becomes a traitor. 36
It allows arbitrary changes to $ord[g]$ and $rcvd[g]$ (and all variables other than $status[g]$).

$Choose(g)$ The action in which g chooses his order by applying *Majority* to his array $rcvd[g]$ of relayed values.

The final three conjuncts of $LSpec(g)$ place timing bounds on when g's actions
must occur. The second conjunct asserts that (it is always true that) if g is loyal 37
and $rcvd[g][g]$ equals "?", then *now* is at most $Now_0 + \delta$. If g is loyal, $rcvd[g][g]$ equals "?" iff an $Issue(g)$ step has not occurred. Thus, this conjunct asserts that, if g is loyal, then an $Issue(g)$ step must occur by time $Now_0 + \delta$. Similarly, the third conjunct asserts that, if g is loyal, then a $Relay(g, h)$ step must occur by time $Now_0 + 2 * \delta$.

The final conjunct of $LSpec(g)$ places a timing bound on the $Choose(g)$ ac-
tion, using the temporal formulas *VTimer* and *MaxTimer*, defined in module 38
RealTime. These formulas were introduced in [2] and used again in [15] as a general method for specifying real-time bounds. If A is an action and v a state function such that any A step changes v, and if t is a variable that does not occur in A or v, then the formula $VTimer(t, A, \epsilon, v) \wedge MaxTimer(t)$ asserts that A cannot be enabled for more than ϵ time units before the next A step occurs.

The temporal existential quantifier $\exists\!\!\!\exists\, t$ essentially hides the variable t. Thus, the fourth conjunct of $LSpec(g)$ asserts that, if lieutenant g is loyal, then a $Choose(g)$ step must occur within ϵ time units of when he has received an order from the commander and from all other lieutenants. 39

40 Finally, the module asserts the correctness of the mid-level algorithm. The predicate *OneTraitor* asserts of a state that there is at most one traitorous general. Theorem *MidCorrect* asserts that, for any behavior, if (i) there is always at most one traitorous general and (ii) formula *Spec* holds, then formula *Hi.Spec* holds. In other words, this theorem asserts that, in the presence of at most one traitor, the mid-level specification implements the high-level specification.

2.3 The Low-Level Specification

In the mid-level specification, a value is transferred from the commander to a lieutenant in a single step, and is relayed from one lieutenant to another in a single step. In the low-level specification, we model the way values are transmitted over communication channels. This requires adding timeouts to detect if a traitorous general fails to send a message.

41 The specification uses module *TimedChannel* of Figure 4, which provides generic definitions for describing the transmission of values over a channel. This module imports module *Sequences* to define operators on sequences. In TLA$^+$,
42 an n-tuple $\langle v_1, \ldots, v_n \rangle$ is a function whose domain is the set $\{1, \ldots, n\}$ of natural numbers, where $\langle v_1, \ldots, v_n \rangle[i]$ equals v_i, for $1 \leq i \leq n$. The *Sequences* module represents sequences as tuples. The module is given without explanation in Figure 20. It defines the usual operators *Head*, *Tail*, and $\circ$ (concatenation) on sequences.

43 Module *TimedChannel* has two variable parameters:

src The interface at the sender's end of the channel. It will be a pair whose first element is a sequence of values and whose second element is either 0 or 1.

dest The interface at the receiver's end of the channel. It will be a pair whose first element is a sequence of values and whose second element is either 0 or 1.

(The purpose of the second components of *src* and *dest* is explained below.) The module has a single constant parameter τ, which is a real number that represents the maximum time required to transmit one value.

44 The sending of a value v is initiated by a $Send(v)$ step, which appends v to the tail of $src[1]$ and complements $src[2]$—changing its value from 0 to 1 or vice-versa. Receipt of the value v occurs with a $Rcv(v)$ step, which is enabled iff v is the head of the sequence $dest[1]$. A $Rcv(v)$ step removes v from the head of $dest[1]$ and complements $dest[2]$.

45 The transmission of a value from one end of the channel to the other is modeled by a *Tmt* step, which moves an element from the head of $src[1]$ to the tail of $dest[1]$.

module *TimedChannel*[41]

import *Sequences, Reals, RealTime*

parameters $src, dest$[43] : VARIABLE
τ[47] : CONSTANT

assumption $Assump \triangleq (\tau \in Real) \wedge (0 < \tau)$

$$Send(v)\boxed{44} \triangleq src' = \langle src[1] \circ \langle v \rangle, 1 - src[2] \rangle$$

$$Rcv(v)\boxed{44} \triangleq \begin{array}{l} \wedge\ (dest[1] \neq \langle\,\rangle) \wedge (v = Head(dest[1])) \\ \wedge\ dest' = \langle Tail(dest[1]), 1 - dest[2] \rangle \end{array}$$

$$Tmt\boxed{45} \triangleq \begin{array}{l} \wedge\ src[1] \neq \langle\,\rangle \\ \wedge\ src' = \langle Tail(src[1]), src[2] \rangle \\ \wedge\ dest' = \langle dest[1] \circ \langle Head(src[1]) \rangle, dest[2] \rangle \end{array}$$

$$ext\boxed{48} \triangleq \langle src[2], dest[2] \rangle$$

$$Spec\boxed{46} \triangleq \begin{array}{l} \wedge\ (src \ = \langle \langle\,\rangle, 0 \rangle) \wedge \Box[Tmt \vee \exists\, v : Send(v)]_{src} \\ \wedge\ (dest = \langle \langle\,\rangle, 0 \rangle) \wedge \Box[Tmt \vee \exists\, v : Rcv(v)]_{dest} \\ \wedge\ \boldsymbol{\exists}\, t : VTimer(t, Tmt, \tau, \langle src, dest \rangle) \wedge MaxTimer(t)\boxed{47} \end{array}$$

Fig. 4. Module *TimedChannel*.

46 Formula *Spec* is the specification of the timed channel. A behavior satisfies this formula iff the variables *src* and *dest* behave the way they should for a timed channel. The formula has three conjuncts. The first describes the sequence of values assumed by *src*. Initially, $src[1]$ is the empty sequence and $src[2]$ equals 0; every step that changes *src* is a *Tmt* step or a $Send(v)$ step, for some v. The second conjunct similarly describes the sequence of values assumed by *dest*. The third conjunct asserts the real-time requirement. As explained in Section 2.2, the conjunct asserts that *Tmt* cannot remain enabled for more than τ time units 47
before the next *Tmt* step occurs. Thus, this conjunct asserts that, when $src[1]$ is nonempty, values are moved from it to $dest[1]$ at the rate of at least one every τ time units.

We now come to *ext*, the pair consisting of the second components of *src* and 48
dest, and the explanation of what those second components are for. The channel specification *Spec* allows arbitrary values to be sent "spontaneously", and it allows those values to be received at arbitrary times. This specification is used by conjoining it with specifications of a sender and receiver that constrain when *Send* and *Rcv* actions occur. The sender's specification will describe when *Send* actions can occur and what values they can send; the receiver's specification will describe when *Rcv* actions can occur. The sender's and receiver's specifications must also allow the channel's internal *Tmt* steps. They allow such steps by allowing any step that leave *ext* unchanged; *Spec* implies that any such step must be a *Tmt* step.

Module *LowLevelParams* in Figure 5 begins the specification of the low-level

module *LowLevelParams*

import *SpecParams, MidLevelParams, Reals*
export *SpecParams, MidLevelParams, Reals, LowLevelParams, Hi, Mid, TC* 52

parameters $in, out, sent$ 49 : VARIABLE
τ 50 : CONSTANT

---

assumption $LowAssump \triangleq (\tau \in Real) \wedge (0 < \tau) \wedge (\tau + 3 * \epsilon \leq \delta)$

---

include *HiLevel* **as** *Hi*
include *MidLevel* **as** *Mid*
include *TimedChannel* **as** $TC(g, h)$ **with** $src \leftarrow out[g][h],\ dest \leftarrow in[h][g]$ 51

$AllBut(func, g)$ 53 $\triangleq [h \in Lt \setminus \{g\} \mapsto func[h]]$
$NotSent(g, h) \triangleq sent[g][h] =$ "No"
$ext(g)$ 54 $\triangleq [h \in Lt \mapsto TC(g, h).ext]$

Fig. 5. Module *LowLevelParams*.

algorithm itself. The module imports the two higher-level …*Params* modules and declares three new variable parameters:

49 *in* The state function $in[h][g]$ represents the receiver's end (*dest*) of the channel from g to h.
out The state function $out[g][h]$ represents the sender's end (*src*) of the channel from g to h.
sent The value of $sent[g][h]$, which will be either "Yes" or "No", records whether or not general g has sent a value to lieutenant h. For each general g, $sent[g]$ will be a function with domain Lt.

50 The constant parameter τ has the same meaning as in module *TimedChannel*.

The module next asserts assumption *LowAssump*, which relates τ to δ and ϵ. The module also asserts assumptions *HiAssump* and *MidAssump*, which are imported with modules *SpecParams* and *MidLevelParams*, respectively.

Module *LowLevelParams* then includes modules *HiLevel* and *MidLevel*, and
51 includes a parameterized copy of module *TimedChannel*. The latter **include** statement incorporates all the definitions from module *TimedChannel* prefixed with "$TC(g, h)$.", and with the indicated instantiations of the parameters *src* and *dest*. For example, it includes the definition

$$TC(g, h).Tmt \triangleq \begin{array}{l} \wedge\ out[g][h][1] \neq \langle\,\rangle \\ \wedge\ out[g][h]' = \langle\, Tail(out[g][h][1]), out[g][h][2] \,\rangle \\ \wedge\ in[h][g]' = \langle\, in[h][g][1] \circ \langle\, Head(out[g][h][1]) \,\rangle, in[h][g][2] \,\rangle \end{array}$$

52 The **export** statement exports all these included definitions, as well as the ones

from the imported modules and the definitions made in the *LowLevelParams* module itself.

Finally, the module makes three definitions. If *func* is a function with domain 53
Lt, then *AllBut*(*func*, g) is the restriction of *func* to the domain $Lt \setminus \{g\}$, the
set of all lieutenants other than g. The state function *ext*(g) is the array of *ext* 54
tuples for each channel interface at lieutenant g. Thus, a step that leaves *ext*(g) unchanged allows *Tmt* steps for all the channels to and from g, but allows no *Send* or *Rcv* actions on those channels.

The definitions of the predicates and actions used in the final specification appear in module *LowLevelActions* of Figure 6. (Normally, one would combine this module with module *LowLevel*; we have split the specification to keep each module less than one page long.)

The section following the **import** statement specifies the initial predicate 55
CInit and next-state action *CNext* of the commander. (The heading is a comment, distinguished by its upright font.) The commander can either do a *Hi.Fail* step (any step that ends with *status*[*Cmdr*] = "Traitor") or send her order to some lieutenant g with a *CSend*(g) step. A *CSend*(g) step is enabled iff the commander is loyal and has not yet sent her order to g; it sends the order and changes *sent*[g] to indicate that the order was sent.

The next section defines the initial condition and the normal actions for a 56
lieutenant. The *Issue* and *Relay* actions represent the receipt of an order from the commander or another lieutenant, respectively. The *Send* action is the one in which a lieutenant sends an order on a channel.

The following section gives the definitions of a lieutenant's two timeout ac- 57
tions. An *IssueTimeout*(g) step can occur if lieutenant g has not received an order from the commander by time $Now_0 + \tau + 2 * \epsilon$. A *RelayTimeout*($g, h$) step can occur if g has not received a relayed order from lieutenant h by time $Now_0 + 2 * \tau + 5 * \epsilon$. These timeouts are needed to ensure progress if a traitorous general fails to send an order.

Action *LNext*(g) is the next-state action of lieutenant g. 58

Finally, module *LowLevel* in Figure 7 imports the preceding two modules and defines the complete low-level specification *Spec*.

The state functions *cvar* and *lvar*(g) are the tuples of variables of the com- 59
mander and lieutenant g, respectively.

The formula *EMax*(g, A) asserts that action A cannot be enabled for more 60
than ϵ time units before the next A step occurs (assuming that an A step changes *lvar*(g)).

Formula *CSpec* is the commander's specification. The second conjunct asserts 61
that, for every lieutenant g, a *CSend*(g) step must occur within ϵ seconds of when it becomes enabled. If the commander is loyal, then action *CSend*(g) is enabled initially and remains enabled until she sends g her order. Thus, the second conjunct asserts that a loyal commander must send her order to every lieutenant within ϵ time units of the start of the algorithm.

Formula *LSpec*(g) is the specification of lieutenant g. The last four conjuncts 62
express the requirements that, if loyal, he must execute each of his actions within

module *LowLevelActions*

import *LowLevelParams, RealTime*

The commander.[55]

$$
\begin{aligned}
CInit \;&\triangleq\; \land\ Hi.CInit \\
&\;\; \land\ Loyal(Cmdr) \Rightarrow (sent[Cmdr] = [h \in Lt \mapsto \text{“No”}]) \\
CSend(g) \;&\triangleq\; \land\ Loyal(Cmdr) \land NotSent(Cmdr, g) \\
&\;\; \land\ TC(Cmdr, g).Send(ord[Cmdr]) \\
&\;\; \land\ sent'[Cmdr] = [sent[Cmdr] \text{ EXCEPT } ![g] = \text{“Yes”}] \\
&\;\; \land\ \text{UNCHANGED } \langle in[Cmdr], AllBut(out[Cmdr], g), ord[Cmdr], \\
&\;\; \qquad\qquad status[Cmdr] \rangle \\
CNext \;&\triangleq\; Hi.Fail(Cmdr) \lor \exists\, g \in Lt : CSend(g)
\end{aligned}
$$

Lieutenant g.[56]

$$
\begin{aligned}
LInit(g) \;&\triangleq\; \land\ Mid.LInit(g) \\
&\;\; \land\ Loyal(g) \Rightarrow (sent[g] = [h \in Lt \mapsto \text{“No”}]) \\
Issue(g) \;&\triangleq\; \land\ Loyal(g) \land (rcvd[g][g] = \text{“?”}) \\
&\;\; \land\ \exists\, o \in Order : \land\ TC(Cmdr, g).Rcv(o) \\
&\;\; \qquad\qquad\qquad\quad \land\ rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![g] = o] \\
&\;\; \land\ \text{UNCHANGED } \langle ord[g], AllBut(in[g], Cmdr), out[g], status[g], sent[g] \rangle \\
Send(g, h) \;&\triangleq\; \land\ Loyal(g) \land (rcvd[g][g] \neq \text{“?”}) \land NotSent(g, h) \\
&\;\; \land\ TC(g, h).Send(rcvd[g][g]) \\
&\;\; \land\ sent'[g] = [sent[g] \text{ EXCEPT } ![h] = \text{“Yes”}] \\
&\;\; \land\ \text{UNCHANGED } \langle ord[g], rcvd[g], in[g], AllBut(out[g], h), status[g] \rangle \\
Relay(g, h) \;&\triangleq\; \land\ Loyal(g) \land (rcvd[g][h] = \text{“?”}) \\
&\;\; \land\ \exists\, o \in Order : \land\ TC(h, g).Rcv(o) \\
&\;\; \qquad\qquad\qquad\quad \land\ rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![h] = o] \\
&\;\; \land\ \text{UNCHANGED } \langle ord[g], AllBut(in[g], h), out[g], status[g], sent[g] \rangle \\
Choose(g) \;&\triangleq\; Mid.Choose(g) \land \text{UNCHANGED } \langle in[g], out[g] \rangle
\end{aligned}
$$

Timeout actions.[57]

$$
\begin{aligned}
IssueTimeout(g) \;&\triangleq\; \land\ Loyal(g) \land (rcvd[g][g] = \text{“?”}) \\
&\;\; \land\ Now_0 + \tau + 2 * \epsilon < now \\
&\;\; \land\ \exists\, o \in Order : rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![g] = o] \\
&\;\; \land\ \text{UNCHANGED } \langle ord[g], in[g], out[g], status[g], sent[g] \rangle \\
RelayTimeout(g, h) \;&\triangleq\; \land\ Loyal(g) \land (rcvd[g][h] = \text{“?”}) \\
&\;\; \land\ Now_0 + 2 * \tau + 5 * \epsilon < now \\
&\;\; \land\ \exists\, o \in Order : rcvd'[g] = [rcvd[g] \text{ EXCEPT } ![h] = o] \\
&\;\; \land\ \text{UNCHANGED } \langle ord[g], in[g], out[g], status[g] \rangle
\end{aligned}
$$

$$
\begin{aligned}
LNext(g)^{\boxed{58}} \;&\triangleq\; \lor\ Issue(g) \lor Choose(g) \lor IssueTimeout(g) \\
&\;\; \lor\ \exists\, h \in Lt \setminus \{g\} : Send(g, h) \lor Relay(g, h) \lor RelayTimeout(g, h) \\
&\;\; \lor\ Hi.Fail(g)
\end{aligned}
$$

Fig. 6. Module *LowLevelActions*.

module *LowLevel*

import *LowLevelParams, LowLevelActions*

State functions.[59]

$$
\begin{aligned}
cvar &\triangleq \langle ord[Cmdr], ext(Cmdr), status[Cmdr], sent[Cmdr] \rangle \\
lvar(g) &\triangleq \langle ord[g], rcvd[g], ext(g), status[g], sent[g] \rangle
\end{aligned}
$$

Temporal formulas.

$$EMax(g, A)^{\boxed{60}} \triangleq \exists\, t : VTimer(t, A, \epsilon, lvar(g)) \land MaxTimer(t)$$

$$
\begin{aligned}
CSpec^{\boxed{61}} \triangleq\ & \land\ CInit \land \Box[CNext]_{cvar} \\
& \land\ \forall\, g \in Lt : EMax(Cmdr, CSend(g))
\end{aligned}
$$

$$
\begin{aligned}
LSpec(g)^{\boxed{62}} \triangleq\ & \land\ LInit(g) \land \Box[LNext(g)]_{lvar(g)} \\
& \land\ EMax(g, Issue(g)) \\
& \land\ EMax(g, IssueTimeout(g)) \\
& \land\ EMax(g, Choose(g)) \\
& \land\ \forall\, h \in Lt \setminus \{g\} : \land\ EMax(g, Send(g, h)) \\
& \qquad\qquad\qquad\qquad\ \ \land\ EMax(g, Relay(g, h)) \\
& \qquad\qquad\qquad\qquad\ \ \land\ EMax(g, RelayTimeout(g, h))
\end{aligned}
$$

$$
\begin{aligned}
Spec^{\boxed{63}} \triangleq\ & \land\ CSpec \\
& \land\ \forall\, g \in Lt : LSpec(g) \\
& \land\ \forall\, g, h \in Gen : TC(g, h).Spec \\
& \land\ now = Now_0 \\
& \land\ RT(\langle ord, rcvd, in, out, status, sent \rangle)
\end{aligned}
$$

theorem $LowCorrect \triangleq Spec \Rightarrow Mid.Spec$

Fig. 7. Module *LowLevel*.

ϵ time units of when he can.

63 Finally, the complete specification *Spec* has five conjuncts: (i) the specification of the commander, (ii) the specifications of the lieutenants, (iii) the specifications of all the communication channels, (iv) the specification of the starting time, and (v) the usual *RT* formula.

3 The Proofs

We now describe how the theorems asserted in the specifications above are proved. The key to moving proofs from the realm of mathematics into engineering practice is hierarchical structuring. We use the method of structuring proofs introduced in [14]. The conventions used in this method are described as they appear.

A hierarchically structured proof is a sequence of steps, each with a proof. The proof of a step is either a short paragraph or calculation, or else a hierarchically structured proof. The idea is to make the unstructured "leaf" proofs sufficiently easy to check that they are highly unlikely to be wrong. We indicate how proofs of our theorems are carried down to the level at which each leaf proof consists of simple expansion of definitions and propositional logic. Such simple proofs are easy to check mechanically; most steps in our proofs can be checked with the TLP verification system [7].

3.1 Proof of Theorem *SpecGood*

Figure 8 is the high-level proof of theorem *SpecGood*, consisting of the level-one steps and the proof of the final step. The Let construct introduces definitions local to the proof. We use a hierarchical numbering convention for denoting parts of formulas, adding numbers to bulleted lists of conjuncts and disjuncts. Thus, $Inv(g, h).3$ is the formula $Loyal(g) \wedge (ord[g] \ldots \mathit{\Delta})$. We extend this convention to quantified formulas, so if F is the formula $\exists\, x : P(x)$, then $F(y)$ denotes the formula $P(y)$. We use this convention for formulas defined in the specifications even when the conjuncts and disjuncts are not explicitly numbered. Thus, with the definition if *Issue* from module *MidLevel* (Figure 3), $Issue(h).2(p).1$ denotes the formula $rcvd'[h] = [rcvd[h] \text{ EXCEPT } ![h] = p]$.

Theorem *SpecGood* is of the form $Spec \Rightarrow \Box P$, for a state predicate P. If *Spec* were of the canonical form $Init \wedge \Box[N]_v$, then this would be a completely standard proof using the method first described by Ashcroft [4]: find a state predicate I (the *invariant*) such that (i) *Init* implies I, (ii) I implies P, and (iii) $I \wedge [N]_v$ implies I'. (This TLA formulation of the proof method is more transparent than its original description as a method for reasoning about programs.) Since *Spec* is written as the conjunction of formulas in canonical form, along with formulas of the form $\Box Q$, our proof involves a simple generalization of Ashcroft's method.

The Q.E.D. in step $\langle 1 \rangle 5$ stands for the goal to be proved—in this case, the theorem itself. In the proof, curly braces enclose the justification of the implication or equivalence. Each step in this chain of implications and equivalence follows from simple substitution and application of standard rules.

LET: $Good(g) \triangleq Loyal(g) \land (Now_0 + \Delta < now)$
$\Rightarrow \land\ ord[g] \in Order$
$\land\ \forall h \in Gen : Loyal(h) \Rightarrow (ord[g] = ord[h])$

Theorem $Spec \Rightarrow \Box(\forall g \in Gen : Good(g))$

LET: $Inv(g,h) \triangleq$ 1.$\land\ Loyal(g) \Rightarrow ord[g] \in Order \cup \{\text{“?”}\}$
2.$\land\ Loyal(g) \land Loyal(h) \land (ord[g] \neq \text{“?”}) \land (ord[h] \neq \text{“?”})$
$\Rightarrow (ord[g] = ord[h])$
3.$\land\ Loyal(g) \land (ord[g] = \text{“?”}) \Rightarrow (now \leq Now_0 + \Delta)$

$CInv \triangleq Loyal(Cmdr) \Rightarrow (ord[Cmdr] \in Order)$

⟨1⟩1. ASSUME: $g, h \in Lt$
PROVE: $LSpec(g) \land LSpec(h) \Rightarrow \Box Inv(g,h)$

⟨1⟩2. ASSUME: $g \in Lt$
PROVE: $LSpec(g) \land CSpec \Rightarrow \Box Inv(g, Cmdr)$

⟨1⟩3. $CSpec \Rightarrow \Box CInv$

⟨1⟩4. $CInv \land (\forall g \in Lt : \forall h \in Gen : Inv(g,h)) \Rightarrow (\forall g \in Gen : Good(g))$

⟨1⟩5. Q.E.D.

PROOF: $Spec \Rightarrow$ {By the definition of $Spec$.}
$CSpec \land (\forall g, h \in Lt : LSpec(g) \land LSpec(h))$
$\Rightarrow$ {By ⟨1⟩1, ⟨1⟩2, and ⟨1⟩3.}
$\Box CInv \land (\forall g, h \in Lt : \Box Inv(g,h) \land \Box Inv(g, Cmdr))$
$\equiv$ {By the temporal logic rule that $\Box$ distributes over conjunction.}
$\Box(CInv \land (\forall g, h \in Lt : Inv(g,h) \land Inv(g, Cmdr))$
$\Rightarrow$ {⟨1⟩4, the definition of Gen, and the temporal logic rule
$(P \Rightarrow Q) \vdash (\Box P \Rightarrow \Box Q)$.}
$\Box(\forall g \in Gen : Good(g))$

Fig. 8. The high-level structure of the proof of theorem *SpecGood*

To finish the proof, we must now prove statements ⟨1⟩1–⟨1⟩4. The proof of ⟨1⟩4 involves simple predicate logic and will not be discussed. The proofs of ⟨1⟩1, ⟨1⟩2, and ⟨1⟩3 are similar; we consider only ⟨1⟩1.

The first-level proof of ⟨1⟩1 appears in Figure 9. The rule that underlies the proof is that proving $I \land A \Rightarrow I'$ allows us to infer $I \land \Box A \Rightarrow \Box I$, where I is a predicate and A an action. This is an RTLA [13] rule, where RTLA is a logic that is like TLA except that $\Box A$ is an RTLA formula for any action A, not just for actions A of the form $[N]_v$. In RTLA, temporal quantification (the operator $\exists$) can be applied only to a TLA formula, not to an arbitrary RTLA formula. All TLA proof rules are valid for RTLA.

To complete the proof of ⟨1⟩1, we must prove ⟨2⟩1 and ⟨2⟩2. Step ⟨2⟩1 follows from the definitions by simple predicate logic. The proof of ⟨2⟩2 is shown in Figure 10. The proof goal is first transformed into an ASSUME/PROVE form, so the new goal becomes simply $Inv(g,h)'$. Inside the proof, these four assumptions are referred to as assumption ⟨2⟩:1–⟨2⟩:4.

Since $Inv(g,h)$ is the conjunction of the formulas $Inv(g,h).1$, $Inv(g,h).2$, and $Inv(g,h).3$, the next level of the proof (steps ⟨3⟩1–⟨3⟩4) is immediate. The

$\langle 1\rangle 1.$ ASSUME: $g, h \in Lt$
PROVE: $LSpec(g) \land LSpec(h) \Rightarrow \Box Inv(g, h)$

LET: $T(g) \triangleq Loyal(g) \land (ord[g] = \text{“?”}) \Rightarrow (now \leq Now_0 + \Delta)$

$\langle 2\rangle 1.$ $LInit(g) \land LInit(h) \land T(g) \Rightarrow Inv(g, h)$

$\langle 2\rangle 2.$ $\land\ Inv(g, h)$
$\land\ [Choose(g) \lor Fail(g)]_{var(g)} \land [Choose(h) \lor Fail(h)]_{var(h)}$
$\land\ T(g)'$
$\Rightarrow Inv(g, h)'$

$\langle 2\rangle 3.$ Q.E.D.

PROOF:
$LSpec(g) \land LSpec(h)$
$\Rightarrow$ {By definition of $LSpec$ and T.}
$\land\ LInit(g) \land LInit(h)$
$\land\ \Box[Choose(g) \lor Fail(g)]_{var(g)} \land \Box[Choose(h) \lor Fail(h)]_{var(h)}$
$\land\ \Box T(g)$
$\Rightarrow$ {Using the RTLA rule $\vdash \Box P \equiv P \land \Box P'$, for any predicate P.}
$\land\ LInit(g) \land LInit(h) \land T(g)$
$\land\ \Box[Choose(g) \lor Fail(g)]_{var(g)} \land \Box[Choose(h) \lor Fail(h)]_{var(h)}$
$\land\ \Box T(g)'$
$\Rightarrow$ {By $\langle 2\rangle 1$.}
$\land\ Inv(g, h)$
$\land\ \Box[Choose(g) \lor Fail(g)]_{var(g)} \land \Box[Choose(h) \lor Fail(h)]_{var(h)}$
$\land\ \Box T(g)'$
$\Rightarrow$ {Using the rule that $\Box$ distributes over $\land$.}
$\land\ Inv(g, h)$
$\land\ \Box([Choose(g) \lor Fail(g)]_{var(g)} \land [Choose(h) \lor Fail(h)]_{var(h)} \land T(g)')$
$\Rightarrow$ {By $\langle 2\rangle 2$ and the RTLA Rule $(I \land A \Rightarrow I') \vdash (I \land \Box A \Rightarrow \Box I)$, for any predicate I and action A.}
$\Box Inv(g, h)$

Fig. 9. The high-level structure of the proof of step $\langle 1\rangle 1$ from Figure 8.

$\langle 2\rangle 2.\ \land\ Inv(g,h)$
$\quad \land\ [Choose(g) \lor Fail(g)]_{var(g)} \land [Choose(h) \lor Fail(h)]_{var(h)}$
$\quad \land\ T(g)'$
$\quad \Rightarrow Inv(g,h)'$

PROOF: By propositional logic, it suffices to:
ASSUME: 1. $Inv(g,h)$
2. $[Choose(g) \lor Fail(g)]_{var(g)}$
3. $[Choose(h) \lor Fail(h)]_{var(h)}$
4. $T(g)'$
PROVE: $Inv(g,h)'$

$\langle 3\rangle 1.$ $Inv(g,h).1'$
PROOF: By definition of $Inv(g,h).1'$ and propositional logic, it suffices to:
ASSUME: $Loyal(g)'$
PROVE: $ord'[g] \in Order \cup \{\text{"?"}\}$

$\langle 4\rangle 1.$ CASE: UNCHANGED $var(g)$
PROOF: Assumption $\langle 2\rangle$:1, Case Assumption $\langle 4\rangle$, and the definition of *Inv*, since UNCHANGED $var(g)$ implies $ord'[g] = ord[g]$.

$\langle 4\rangle 2.$ CASE: $Choose(g)$
PROOF: Case Assumption $\langle 4\rangle$, since $Choose(g).2 \triangleq ord'[g] \in Order$.

$\langle 4\rangle 3.$ CASE: $Fail(g)$
PROOF: Case Assumption $\langle 4\rangle$ and Assumption $\langle 3\rangle$ lead to a contradiction, since $Fail(g)$ implies $\neg Loyal(g)'$ by definition of *Fail* and *Loyal*.

$\langle 4\rangle 4.$ Q.E.D.
PROOF: By propositional logic from $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, and Assumption $\langle 2\rangle$:2, since $[A]_v \triangleq A \lor (\text{UNCHANGED } v)$.

$\langle 3\rangle 2.$ $Inv(g,h).2'$

$\langle 3\rangle 3.$ $Inv(g,h).3'$

$\langle 3\rangle 4.$ Q.E.D.
PROOF: $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, and the definition of *Inv*.

Fig. 10. The proof of step $\langle 2\rangle 2$ from Figure 9.

proof of $\langle 3\rangle 1$ is given; the proofs of $\langle 3\rangle 2$ and $\langle 3\rangle 3$ are analogous.

The goal, $Inv(g,h).1'$ is deduced from assumptions $\langle 2\rangle$:1 and $\langle 2\rangle$:2. Since assumption $\langle 2\rangle$:2 is a disjunction, we do a proof by cases. The statement CASE: S is equivalent to ASSUME: S, PROVE: Q.E.D.

3.2 Proof of Theorem *MidCorrect*

Theorem *MidCorrect* asserts that $(\Box OneTraitor) \land Spec$ implies $Hi.Spec$. It is trivial to prove that *Spec* implies $Hi.Spec.1$ and $Hi.Spec.3$, so the problem is proving $Hi.Spec.2$ and $Hi.Spec.4$. Proving $Hi.Spec.2$ requires proving $Hi.LSpec(g).1$ and $Hi.LSpec(g).2$ for each lieutenant g.

The high-level structure of the proof of theorem *MidCorrect* appears in Figure 11. Steps $\langle 1\rangle 2$ and $\langle 1\rangle 4$ prove $Hi.LSpec(g).1$ and $Hi.LSpec(g).2$, respectively, and $\langle 1\rangle 5$ proves $Hi.Spec.4$. Steps $\langle 1\rangle 1$ and $\langle 1\rangle 3$ establish useful invariants.

Theorem $(\Box \mathit{OneTraitor}) \land \mathit{Spec} \Rightarrow \mathit{Hi.Spec}$

LET: $\mathit{Inv}(g,l) \triangleq \mathit{Loyal}(g) \Rightarrow$
$$
\begin{aligned}
&1.\land\ (\textsc{Domain}\ \mathit{rcvd}[g] = \mathit{Lt})\\
&2.\land\ (\mathit{rcvd}[g][l] \neq \text{“?”}) \Rightarrow (\mathit{rcvd}[g][l] \in \mathit{Order})\\
&3.\land\ \mathit{Loyal}(\mathit{Cmdr}) \land (\mathit{rcvd}[g][g] \in \mathit{Order})\\
&\qquad \Rightarrow (\mathit{rcvd}[g][g] = \mathit{ord}[\mathit{Cmdr}])\\
&4.\land\ \mathit{Loyal}(l) \land (\mathit{rcvd}[g][l] \in \mathit{Order}) \Rightarrow (\mathit{rcvd}[g][l] = \mathit{rcvd}[l][l])\\
&5.\land\ (\mathit{ord}[g] \neq \text{“?”}) \Rightarrow \land\ \mathit{ord}[g] = \mathit{Majority}(\mathit{rcvd}[g])\\
&\qquad\qquad\qquad\qquad\quad \land\ \forall\, h \in \mathit{Lt} : \mathit{rcvd}[g][h] \neq \text{“?”}
\end{aligned}
$$

$\mathit{TInv}(g) \triangleq$
$$
\begin{aligned}
&1.\land\ \mathit{now} \in \mathit{Real}\\
&2.\land\ \mathit{Loyal}(g) \land (\mathit{ord}[g] = \text{“?”}) \land (\forall\, h \in \mathit{Lt} : \mathit{rcvd}[g][h] \neq \text{“?”})\\
&\qquad \Rightarrow (\mathit{now} \leq \mathit{Now}_0 + 2\delta + \epsilon)
\end{aligned}
$$

$\langle 1\rangle 1.$ ASSUME: $g, l \in \mathit{Lt}$
PROVE: $\mathit{Hi.CSpec} \land \mathit{LSpec}(g) \land \mathit{LSpec}(l) \Rightarrow \Box \mathit{Inv}(g,l)$

$\langle 1\rangle 2.$ ASSUME: $g \in \mathit{Lt}$
PROVE: $\Box \mathit{OneTraitor} \land \mathit{LSpec}(g) \land (\forall\, h, l \in \mathit{Lt} : \Box \mathit{Inv}(h,l)) \Rightarrow \mathit{Hi.LSpec}(g).1$

$\langle 1\rangle 3.$ ASSUME: $g \in \mathit{Lt}$
PROVE: $(\mathit{now} = \mathit{Now}_0) \land \mathit{RT}(\langle \mathit{ord}, \mathit{rcvd}, \mathit{status}\rangle) \land \mathit{LSpec}(g)$
$\land (\forall\, h \in \mathit{Lt} : \Box \mathit{Inv}(g,h)) \Rightarrow \Box \mathit{TInv}(g)$

$\langle 1\rangle 4.$ ASSUME: $g \in \mathit{Lt}$
PROVE: $\mathit{LSpec}(g) \land \Box \mathit{TInv}(g) \Rightarrow \mathit{Hi.LSpec}(g).2$

$\langle 1\rangle 5.$ $\mathit{RT}(\langle \mathit{ord}, \mathit{rcvd}, \mathit{status}\rangle) \Rightarrow \mathit{RT}(\langle \mathit{ord}, \mathit{status}\rangle)$

$\langle 1\rangle 6.$ Q.E.D.

$\langle 2\rangle 1.$ $(\Box \mathit{OneTraitor}) \land \mathit{Spec} \Rightarrow \mathit{Hi.Spec}.1$
PROOF: Trivial, since *Spec*.1 is the same as *Hi.Spec*.1.

$\langle 2\rangle 2.$ $(\Box \mathit{OneTraitor}) \land \mathit{Spec} \Rightarrow \mathit{Hi.Spec}.2$
PROOF: By $\langle 1\rangle 1$–$\langle 1\rangle 4$ and the definitions of *Spec* and *Hi.Spec*, since *Spec*.2 equals $\forall\, l \in \mathit{Lt} : \mathit{LSpec}(l)$, and *Hi.Spec*.2 equals $\forall\, g \in \mathit{Lt} : \mathit{Hi.LSpec}(g)$.

$\langle 2\rangle 3.$ $(\Box \mathit{OneTraitor}) \land \mathit{Spec} \Rightarrow \mathit{Hi.Spec}.3$
PROOF: Trivial, since *Spec*.3 is the same as *Hi.Spec*.3.

$\langle 2\rangle 4.$ $(\Box \mathit{OneTraitor}) \land \mathit{Spec} \Rightarrow \mathit{Hi.Spec}.4$
PROOF: By $\langle 1\rangle 5$ and the definitions of *Spec* and *Hi.Spec*.

$\langle 2\rangle 5.$ Q.E.D.
PROOF: $\langle 2\rangle 1$–$\langle 2\rangle 4$ and the definition of *Hi.Spec*.

Fig. 11. The high-level structure of the proof of theorem *MidCorrect*

$\langle 1 \rangle 2.$ ASSUME: $g \in Lt$
PROVE: $(\Box \mathit{OneTraitor}) \wedge \mathit{LSpec}(g) \wedge (\forall\, h, l \in Lt : \Box \mathit{Inv}(h, l))$
$\Rightarrow \mathit{Hi.LSpec}(g).1$

$\langle 2 \rangle 1.$ $\mathit{LInit}(g) \Rightarrow \mathit{Hi.LInit}(g)$
PROOF: By definition of $\mathit{LInit}(g)$.

$\langle 2 \rangle 2.$ $\mathit{OneTraitor}' \wedge (\forall\, h, l \in Lt : \mathit{Inv}(h, l) \wedge \mathit{Inv}(h, l)') \wedge [\mathit{Next}(g)]_{var(g)}$
$\Rightarrow [\mathit{Hi.Choose}(g) \vee \mathit{Hi.Fail}(g)]_{Hi.var(g)}$

...

$\langle 2 \rangle 3.$ Q.E.D.
PROOF:
$(\Box \mathit{OneTraitor}) \wedge \mathit{LSpec}(g) \wedge (\forall\, h, l \in Lt : \Box \mathit{Inv}(h, l))$
$\Rightarrow$ {By definition of $\mathit{LSpec}(g)$.}
$(\Box \mathit{OneTraitor}) \wedge \mathit{LInit}(g) \wedge \Box[\mathit{Next}(g)]_{var(g)} \wedge (\forall\, h, l \in Lt : \Box \mathit{Inv}(h, l))$
$\Rightarrow$ {By simple RTLA reasoning.}
$\mathit{LInit}(g) \wedge \Box(\mathit{OneTraitor}'$
$\wedge (\forall\, h, l \in Lt : \mathit{Inv}(h, l) \wedge \mathit{Inv}(h, l)') \wedge [\mathit{Next}(g)]_{var(g)})$
$\Rightarrow$ {By $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and simple RTLA reasoning.}
$\mathit{Hi.LInit}(g) \wedge \Box[\mathit{Hi.Choose}(g) \vee \mathit{Hi.Fail}(g)]_{Hi.var(g)}$
$\Rightarrow$ {By definition of $\mathit{Hi.LSpec}(g)$}
$\mathit{Hi.LSpec}(g).1$.

Fig. 12. The proof of step $\langle 1 \rangle 2$ from Figure 11, with the proof of $\langle 2 \rangle 2$ elided.

Step $\langle 1 \rangle 1$ is an invariance proof of the kind we have already seen in the proof of theorem *SpecGood*. Its proof is omitted.

We consider now the proof of $\langle 1 \rangle 2$, which appears in Figure 12. The key step is $\langle 2 \rangle 2$, whose proof is in Figure 13. Since $\mathit{Hi.var}(g)$ is a subtuple of $\mathit{var}(g)$, it is obvious that UNCHANGED $\langle \mathit{var}(g) \rangle$ implies UNCHANGED $\langle \mathit{Hi.var}(g) \rangle$. The proof demonstrates that every step of the mid-level specification, which is a $\mathit{Next}(g)$ step, is a $[\mathit{Hi.Choose}(g) \vee \mathit{Hi.Fail}(g)]_{Hi.var(g)}$ step—that is, a step allowed by the high-level specification. (This is sometimes called proving *step simulation*.)

Formally, the first level in the proof of $\langle 2 \rangle 2$ is a case split on the disjuncts of $[\mathit{Next}(g)]_{var(g)}$. The only hard case is $\mathit{Choose}(g)$, which we prove implies $\mathit{Hi.Choose}(g)$. (In other words, we prove that a mid-level $\mathit{Choose}(g)$ step implements a high-level $\mathit{Hi.Choose}(g)$ step.) The next-level proof of this case is obtained by separately proving $\mathit{Hi.Choose}(g).1$, ... , $\mathit{Hi.Choose}(g).4$. The only hard parts are steps $\langle 4 \rangle 2$ and $\langle 4 \rangle 3$. The proof of $\langle 4 \rangle 2$ is given in Figure 14; the proof of $\langle 4 \rangle 3$ is omitted.

Step $\langle 1 \rangle 3$ is a property of the same form as theorem *SpecGood*. However, there is a temporal quantifier $\boldsymbol{\exists}$ in $\mathit{LSpec}(g)$. Figure 15 indicates how this quantifier is handled. We first define $\mathit{LSpecT}(g, t)$ to be $\mathit{LSpec}(g)$ with the quantifier removed, and define the invariant $\mathit{TInvT}(g, t)$.[3] The heart of the proof, step $\langle 2 \rangle 1$, asserts an ordinary invariance property with no temporal quantifiers; its proof is omitted. Steps $\langle 2 \rangle 3$ and $\langle 2 \rangle 4$ show how the quantifier is "put back into

[3] If r and s are elements of *Real*, then $r \leq t \leq s$ means $(t \in \mathit{Real}) \wedge (r \leq t) \wedge (t \leq s)$.

$\langle 2\rangle 2.$ $\mathit{OneTraitor}' \land (\forall\, h, l \in \mathit{Lt} : \mathit{Inv}(h, l) \land \mathit{Inv}(h, l)') \land [\mathit{Next}(g)]_{\mathit{var}(g)}$
$\quad \Rightarrow [\mathit{Hi.Choose}(g) \lor \mathit{Hi.Fail}(g)]_{\mathit{Hi.var}(g)}$

PROOF: By propositional logic, it suffices to:

ASSUME: 1. $\mathit{OneTraitor}'$
2. $\forall\, h, l \in \mathit{Lt} : \mathit{Inv}(h, l) \land \mathit{Inv}(h, l)'$
3. $[\mathit{Next}(g)]_{\mathit{var}(g)}$

PROVE: $[\mathit{Hi.Choose}(g) \lor \mathit{Hi.Fail}(g)]_{\mathit{Hi.var}(g)}$

$\langle 3\rangle 1.$ CASE: $\mathit{Issue}(g)$
PROOF: By definition, $\mathit{Issue}(g).3$ equals UNCHANGED $\mathit{Hi.var}(g)$.

$\langle 3\rangle 2.$ CASE: $\exists\, h \in \mathit{Lt} \setminus \{g\} : \mathit{Relay}(g, h)$
PROOF: By propositional logic, it suffices to prove that $(h \in \mathit{Lt}) \land \mathit{Relay}(g, h)$ implies UNCHANGED $\mathit{Hi.var}(g)$, which follows from the definitions of $\mathit{Relay}(g, h)$ and $\mathit{Hi.var}(g)$.

$\langle 3\rangle 3.$ CASE: $\mathit{Choose}(g)$

$\langle 4\rangle 1.$ $\mathit{Loyal}(g) \land (\mathit{ord}[g] = \text{“?”})$
PROOF: $\mathit{Choose}(g).1$, which holds by Case Assumption $\langle 3\rangle$.

$\langle 4\rangle 2.$ $\mathit{ord}'[g] \in \mathit{Order}$
...

$\langle 4\rangle 3.$ $\forall\, h \in \mathit{Gen} : \mathit{Loyal}(h)' \land (\mathit{ord}'[h] \neq \text{“?”}) \Rightarrow (\mathit{ord}'[g] = \mathit{ord}'[h])$
...

$\langle 4\rangle 4.$ UNCHANGED $\mathit{status}[g]$
PROOF: $\mathit{Choose}(g).4$, which holds by Case Assumption $\langle 3\rangle$.

$\langle 4\rangle 5.$ Q.E.D.
PROOF: $\langle 4\rangle 1$–$\langle 4\rangle 4$ imply $\mathit{Hi.Choose}(g)$.

$\langle 3\rangle 4.$ CASE: $\mathit{Hi.Fail}(g)$
PROOF: Immediate.

$\langle 3\rangle 5.$ CASE: UNCHANGED $\mathit{var}(g)$
PROOF: By definition, $\mathit{Hi.var}(g)$ is a subsequence of $\mathit{var}(g)$.

$\langle 3\rangle 6.$ Q.E.D.
PROOF: $\langle 3\rangle 1$–$\langle 3\rangle 5$ and the definition of $\mathit{Next}(g)$.

Fig. 13. The proof of step $\langle 2\rangle 2$ from Figure 12, with the proofs of $\langle 4\rangle 2$ and $\langle 4\rangle 3$ elided.

$\langle 4\rangle 2.$ $ord'[g] \in Order$
  $\langle 5\rangle 1.$ $rcvd[g] \in [Lt \rightarrow Order]$
    $\langle 6\rangle 1.$ DOMAIN $rcvd[g] = Lt$
      PROOF: $\langle 4\rangle 1$ and $Inv(g, g).1$, which holds by Assumption $\langle 1\rangle$ and Assumption $\langle 2\rangle$:2.
    $\langle 6\rangle 2.$ $\forall h \in Lt : rcvd[g][h] \neq$ "?"
      PROOF: $Choose(g).2$, which holds by Assumption $\langle 3\rangle$.
    $\langle 6\rangle 3.$ $\forall h \in Lt : rcvd[g][h] \in Order$
      PROOF: $\langle 6\rangle 2$ and $Inv(g, h).2$, which holds by $\langle 4\rangle 1$, Assumption $\langle 1\rangle$, and Assumption $\langle 2\rangle$:2.
    $\langle 6\rangle 4.$ Q.E.D.
      PROOF: $\langle 6\rangle 1$ and $\langle 6\rangle 3$.
  $\langle 5\rangle 2.$ Q.E.D.
    PROOF: $Choose(g).3$ (which holds by Assumption $\langle 3\rangle$), $\langle 5\rangle 1$, and Assumption $MidAssump.2$.

Fig. 14. The proof of step $\langle 4\rangle 2$ from Figure 13.

the formula" using simple reasoning. Formally, we are using the following two rules, which are valid if x does not occur free in G:

$$\frac{F(x) \Rightarrow G}{(\exists x : F(x)) \Rightarrow G} \qquad \vdash (\exists x : F(x) \land G) \equiv (\exists x : F(x)) \land G$$

This proof strategy will be familiar to anyone who has done rigorous proofs in ordinary first-order logic.

The proof of step $\langle 1\rangle 4$ is a simple matter of deducing $(\Box P_1) \land (\Box P_2) \land (\Box P_3) \Rightarrow \Box Q$ from $P_1 \land P_2 \land P_3 \Rightarrow Q$, where $\Box P_1$ and $\Box P_2$ are $LSpec(g).2$ and $LSpec(g).3$, respectively. The proof of step $\langle 1\rangle 5$ uses the same basic strategy as the proof of $\langle 1\rangle 2$, but is much simpler. The formal proofs of these steps are omitted.

3.3 Proof of Theorem *LowCorrect*

The high-level proof of theorem *LowCorrect* appears in Figure 16. As in the proof of theorem *MidCorrect*, we first prove the implication with the quantifiers removed from the hypothesis, and then add the quantifiers in the final step.

The proof of the final step, $\langle 1\rangle 6$, is shown in Figure 17. To prove step $\langle 2\rangle 2$, we use the same reasoning as before to add the quantifiers. The proof of step $\langle 2\rangle 3$ makes use of a TLA proof rule that is analogous to the rule

$$\vdash (\forall y : \exists x : F(x)) \equiv (\exists x : \forall y : F(x(y)))$$

of higher-order logic. In a formalism like TLA$^+$ that includes set theory, functions are ordinary values, and this rule can be stated in first-order logic as

$$\vdash (\forall y \in S : \exists x : F(x)) \equiv (\exists x : \forall y \in S : F(x[y]))$$

$\langle 1 \rangle 3$. ASSUME: $g \in Lt$
PROVE: $(now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \wedge LSpec(g)$
$\quad \wedge (\forall h \in Lt : \Box Inv(g, h)) \Rightarrow \Box TInv(g)$

LET: $LSpecT(g, t) \triangleq LSpec(g).1 \wedge LSpec(g).2 \wedge LSpec(g).3 \wedge LSpec(g).4(t)$
$TInvT(g, t) \triangleq$
$\quad$ 1.$\wedge$ $now \in Real$
$\quad$ 2.$\wedge$ $Loyal(g) \wedge (ord[g] = \text{“?”}) \wedge (\forall h \in Lt : rcvd[g][h] \neq \text{“?”})$
$\quad\quad \Rightarrow (now \leq t \leq Now_0 + 2\delta + \epsilon)$

$\langle 2 \rangle 1$. $(now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \wedge LSpecT(g, t)$
$\quad \wedge (\forall h \in Lt : \Box Inv(g, h)) \Rightarrow \Box TInvT(g, t)$

...

$\langle 2 \rangle 2$. $(now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \wedge LSpecT(g, t)$
$\quad \wedge (\forall h \in Lt : \Box Inv(g, h)) \Rightarrow \Box TInv(g)$

PROOF: $\langle 2 \rangle 1$ and simple temporal reasoning, since $TInvT(g, t)$ implies $TInv(g)$.

$\langle 2 \rangle 3$. $(\exists t : (now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \wedge LSpecT(g, t)$
$\quad \wedge (\forall h \in Lt : \Box Inv(g, h))) \Rightarrow \Box TInv(g)$

PROOF: $\langle 2 \rangle 2$, since t does not occur free in $\Box TInv(g)$.

$\langle 2 \rangle 4$. Q.E.D.

PROOF: $\langle 2 \rangle 3$, since

$$
\begin{aligned}
&\exists t : \wedge (now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \\
&\qquad \wedge LSpecT(g, t) \\
&\qquad \wedge \forall h \in Lt : \Box Inv(g, h)) \\
&\equiv \{\text{By definition of } LSpecT.\} \\
&\quad \exists t : \wedge (now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \\
&\qquad\quad \wedge LSpec(g).1 \wedge LSpec(g).2 \wedge LSpec(g).3 \wedge LSpec(g).4(t) \\
&\qquad\quad \wedge \forall h \in Lt : \Box Inv(g, h) \\
&\equiv \{\text{Because } t \text{ occurs free only in } LSpecT(g).4.\} \\
&\quad \wedge (now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \\
&\quad \wedge LSpec(g).1 \wedge LSpec(g).2 \wedge LSpec(g).3 \wedge \exists t : LSpec(g).4(t) \\
&\quad \wedge \forall h \in Lt : \Box Inv(g, h) \\
&\equiv \{\text{By definition of } LSpec.\} \\
&\quad \wedge (now = Now_0) \wedge RT(\langle ord, rcvd, status \rangle) \\
&\quad \wedge LSpec(g) \\
&\quad \wedge \forall h \in Lt : \Box Inv(g, h)
\end{aligned}
$$

Fig. 15. The proof of $\langle 1 \rangle 3$ from Figure 11, with the proof of $\langle 2 \rangle 1$ elided.

Theorem $Spec \Rightarrow Mid.Spec$

LET: $qt, ct, t_1, t_2, t_3, t_4, t_5, t_6$: VARIABLE

$$TEMax(t, g, A) \triangleq VTimer(t, A, \epsilon, lvar(g)) \wedge MaxTimer(t)$$

$$CSpecT \triangleq \begin{array}{l} 1.\wedge\ CInit \wedge \Box[CNext]_{cvar} \\ 2.\wedge\ \forall\, g \in Lt\ :\ TEMax(ct[g], Cmdr, CSend(g)) \end{array}$$

$$LSpecT(g) \triangleq \begin{array}{l} 1.\wedge\ LInit(g) \wedge \Box[LNext(g)]_{lvar(g)} \\ 2.\wedge\ TEMax(t_1[g], g, Issue(g)) \\ 3.\wedge\ TEMax(t_2[g], g, IssueTimeout(g)) \\ 4.\wedge\ TEMax(t_3[g], g, Choose(g)) \\ 5.\wedge\ \forall\, h \in Lt \setminus \{g\}\ : \\ \qquad \wedge\ TEMax(t_4[g][h], g, Send(g, h)) \\ \qquad \wedge\ TEMax(t_5[g][h], g, Relay(g, h)) \\ \qquad \wedge\ TEMax(t_6[g][h], g, RelayTimeout(g, h)) \end{array}$$

$$TCSpecT(g, h) \triangleq \begin{array}{l} TC(g, h).Spec.1\ \wedge\ TC(g, h).Spec.2 \\ \quad \wedge\ TC(g, h).Spec.3(qt[g][h]) \end{array}$$

$$CInv(g) \triangleq \ \ldots$$

$$LInv(g, h) \triangleq \ \ldots$$

$\langle 1 \rangle 1.$ $CSpec \Rightarrow Mid.Hi.CSpec$

$\langle 1 \rangle 2.$ ASSUME: $g \in Lt$
PROVE: $\wedge\ CSpecT \wedge LSpecT(g)$
$\wedge\ TCSpecT(Cmdr, g)$
$\wedge\ (now = Now_0) \wedge RT(\langle\, ord, rcvd, in, out, status, sent\,\rangle)$
$\Rightarrow \Box CInv(g)$

$\langle 1 \rangle 3.$ ASSUME: 1. $g, h \in Lt$
2. $g \neq h$
PROVE: $\wedge\ LSpecT(g) \wedge LSpecT(h)$
$\wedge\ TCSpecT(g, h)$
$\wedge\ (now = Now_0) \wedge RT(\langle\, ord, rcvd, in, out, status, sent\,\rangle)$
$\wedge\ \Box CInv(g)$
$\Rightarrow \Box LInv(g, h)$

$\langle 1 \rangle 4.$ ASSUME: $g \in Lt$
PROVE: $LSpec(g)\ \wedge\ \Box CInv(g)\ \wedge\ \Box(\forall\, h \in Lt \setminus \{g\}\ :\ LInv(h, g))$
$\Rightarrow Mid.LSpec(g)$

$\langle 1 \rangle 5.$ $RT(\langle\, ord, rcvd, in, out, status\,\rangle) \Rightarrow RT(\langle\, ord, rcvd, status\,\rangle)$

$\langle 1 \rangle 6.$ Q.E.D.

Fig. 16. The high-level proof of theorem *LowCorrect*.

(The range S is needed because the domain of a function must be a set.) If the ordinary existential quantifier $\exists$ is replaced by the temporal quantifier $\boldsymbol{\exists}$, the rule remains sound in general only if S is finite.[4]

We now return to the high-level proof of the theorem. The proof of step $\langle 1 \rangle 1$ is simple and is omitted. The proofs of $\langle 1 \rangle 2$ and $\langle 1 \rangle 3$ are straightforward

[4] The rule is unsound for an infinite set S because $\boldsymbol{\exists}$ is defined so $\boldsymbol{\exists}\, x : F$ is invariant under stuttering. It is sound for the operator $\boldsymbol{\exists}$ of Manna and Pnueli [18], which does not preserve invariance under stuttering.

$\langle 1 \rangle 6$. Q.E.D.
$\langle 2 \rangle 1$. $\wedge$ *CSpec*
$\wedge$ *CSpecT*
$\wedge \; \forall g \in Lt : LSpecT(g)$
$\wedge \; \forall g, h \in Gen : TCSpecT(g, h)$
$\wedge \; RT(\langle ord, rcvd, in, out, status, sent \rangle)$
$\wedge \; now = Now_0$
$\Rightarrow Mid.Spec$
PROOF: $\langle 1 \rangle 1$–$\langle 1 \rangle 5$ and the definition of *Mid.Spec*.
$\langle 2 \rangle 2$. $\wedge$ *CSpec*
$\wedge \; \boldsymbol{\exists}\, ct : CSpecT$
$\wedge \; \boldsymbol{\exists}\, t_1, t_2, t_3, t_4, t_5, t_6 : \forall g \in Lt : LSpecT(g)$
$\wedge \; \boldsymbol{\exists}\, qt : \forall g, h \in Gen : TCSpecT(g, h)$
$\wedge \; RT(\langle ord, rcvd, in, out, status, sent \rangle)$
$\wedge \; now = Now_0$
$\Rightarrow Mid.Spec$
PROOF: $\langle 2 \rangle 1$, since ct occurs only in $CSpecT$, t_1, t_2, t_3, t_4, t_5, and t_6 occur only in $LSpecT(g)$, and qt occurs only in $TCSpecT(g, h)$.
$\langle 2 \rangle 3$. 1. $(\boldsymbol{\exists}\, ct : CSpecT) \equiv CSpec$
2. $(\boldsymbol{\exists}\, t_1, t_2, t_3, t_4, t_5, t_6 : \forall g \in Lt : LSpecT(g)) \equiv \forall g \in Lt : LSpec(g)$
3. $(\boldsymbol{\exists}\, qt : \forall g, h \in Gen : TCSpecT(g, h)) \equiv \forall g, h \in Gen : TC(g, h).Spec$
PROOF: Lt is finite by assumption *HiAssump*.1 from module *SpecParams* (imported via module *MidLevel*), and $\boldsymbol{\exists}\, x : \forall v \in S : F(x[v])$ is equivalent to $\forall v \in S : \boldsymbol{\exists}\, x : F(x)$, for any finite set S and temporal operator $F(\_)$.
$\langle 2 \rangle 4$. Q.E.D.
PROOF: $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, and the definition of *Spec*.

Fig. 17. The proof of step $\langle 1 \rangle 6$ from Figure 16.

invariance arguments. The invariants, $CInv(g)$ and $LInv(g, h)$, are defined in Figure 18. Because the invariants are a bit long, the proofs of $\langle 1 \rangle 2$ and $\langle 1 \rangle 3$ are rather tedious; they are omitted.

Step $\langle 1 \rangle 4$ introduces a new problem—proving an implication whose conclusion contains a $\boldsymbol{\exists}$. We now examine its proof, which is outlined in Figure 19.

Step $\langle 2 \rangle 1$ is an implication of the form we've seen before. The major part of its proof consists of proving step simulation. The proof is straightforward but tedious, taking about two pages.

Steps $\langle 2 \rangle 2$ and $\langle 2 \rangle 3$ have the form $\Box P \Rightarrow \Box Q$, which is proved by showing that P implies Q. These proofs are easy.

Step $\langle 2 \rangle 4$ is where the $\boldsymbol{\exists}$ appears in the conclusion. In first-order logic, one proves that $F(x)$ implies $\exists y : G(x, y)$ by instantiating y—that is, finding an expression $h(x)$ such that $F(x)$ implies $G(x, h(x))$. The same technique is used with the temporal quantifier $\boldsymbol{\exists}$, where the instantiation is called a *refinement mapping* [1]. To prove $\langle 2 \rangle 4$, we must instantiate the timer variable t used to express the timing constraint on *Mid.Choose*. The instantiation is simple—we instantiate t with the timer variable for the low-level *Choose* action. Step $\langle 3 \rangle 1$

$$
\begin{array}{ll}
\text{LET: } r(g,h) \triangleq & \textbf{if } \mathit{rcvd}[g][h] = \text{"?"} \textbf{ then } \langle\,\rangle \\
& \phantom{\textbf{if } \mathit{rcvd}[g][h] = \text{"?"}} \textbf{ else } \langle \mathit{rcvd}[g][h] \rangle \\
c(g,h) \triangleq & \mathit{in}[h][g][1] \circ \mathit{out}[g][h][1] \\
\mathit{cOrd}(g) \triangleq & r(g,g) \circ c(\mathit{Cmdr}, g) \\
\mathit{lOrd}(g,h) \triangleq & r(g,h) \circ c(h,g) \\
\mathit{CInv}(g) \triangleq &
\end{array}
$$

$$
\begin{array}{l}
1.\wedge\ \mathit{Loyal}(\mathit{Cmdr}) \wedge \mathit{Loyal}(g) \Rightarrow \\
\qquad \vee\ (\mathit{cOrd}(g) = \langle\,\rangle) \wedge \mathit{NotSent}(\mathit{Cmdr}, g) \\
\qquad \vee\ (\mathit{cOrd}(g) = \langle\, \mathit{ord}[\mathit{Cmdr}]\rangle) \wedge \neg \mathit{NotSent}(\mathit{Cmdr}, g) \\
2.\wedge\ \mathit{Loyal}(g) \wedge (\mathit{rcvd}[g][g] = \text{"?"}) \Rightarrow \forall\, h \in \mathit{Lt} \,:\, \mathit{NotSent}(g,h) \\
3.\wedge\ \mathit{Loyal}(\mathit{Cmdr}) \wedge \mathit{NotSent}(\mathit{Cmdr}, g) \Rightarrow (\mathit{now} \le \mathit{ct}[g] \le \mathit{Now}_0 + \epsilon) \\
4.\wedge\ \mathit{Loyal}(g) \wedge \mathit{Loyal}(\mathit{Cmdr}) \wedge (\mathit{out}[\mathit{Cmdr}][g][1] \neq \langle\,\rangle) \\
\qquad \Rightarrow (\mathit{now} \le \mathit{qt}[\mathit{Cmdr}][g] \le \mathit{Now}_0 + \tau + \epsilon) \\
5.\wedge\ \mathit{Loyal}(g) \wedge \mathit{Loyal}(\mathit{Cmdr}) \wedge (\mathit{in}[g][\mathit{Cmdr}][1] \neq \langle\,\rangle) \\
\qquad \Rightarrow (\mathit{now} \le t_1[g] \le \mathit{Now}_0 + \tau + 2 * \epsilon) \\
6.\wedge\ \mathit{Loyal}(g) \wedge (\mathit{rcvd}[g][g] = \text{"?"}) \wedge (\mathit{Now}_0 + \tau + 2 * \epsilon < \mathit{now}) \\
\qquad \Rightarrow (\mathit{now} \le t_2[g] \le \mathit{Now}_0 + \tau + 3 * \epsilon) \\
7.\wedge\ \mathit{now} \in \mathit{Real}
\end{array}
$$

$$
\begin{array}{l}
\mathit{LInv}(g,h) \triangleq \\
1.\wedge\ \mathit{Loyal}(g) \wedge \mathit{Loyal}(h) \\
\qquad \Rightarrow \vee\ (\mathit{lOrd}(h,g) = \langle\,\rangle) \wedge \mathit{NotSent}(g,h) \\
\qquad \vee\ (\mathit{lOrd}(h,g) = \langle \mathit{rcvd}[g][g]\rangle) \wedge \neg \mathit{NotSent}(g,h) \\
2.\wedge\ \mathit{Loyal}(g) \wedge (\mathit{rcvd}[g][g] \neq \text{"?"}) \wedge \mathit{NotSent}(g,h) \\
\qquad \Rightarrow (\mathit{now} \le t_4[g][h] \le \mathit{Now}_0 + \tau + 4 * \epsilon) \\
3.\wedge\ \mathit{Loyal}(g) \wedge \mathit{Loyal}(h) \wedge (\mathit{out}[g][h][1] \neq \langle\,\rangle) \\
\qquad \Rightarrow (\mathit{now} \le \mathit{qt}[g][h] \le \mathit{Now}_0 + 2 * \tau + 4 * \epsilon) \\
4.\wedge\ \mathit{Loyal}(g) \wedge \mathit{Loyal}(h) \wedge (\mathit{in}[h][g][1] \neq \langle\,\rangle) \\
\qquad \Rightarrow (\mathit{now} \le t_5[h][g] \le \mathit{Now}_0 + 2 * \tau + 5 * \epsilon) \\
5.\wedge\ \mathit{Loyal}(h) \wedge (\mathit{rcvd}[h][g] = \text{"?"}) \wedge (\mathit{now} > \mathit{Now}_0 + 2 * \tau + 5 * \epsilon) \\
\qquad \Rightarrow (\mathit{now} \le t_6[h][g] \le \mathit{Now}_0 + 2 * \tau + 6 * \epsilon)
\end{array}
$$

Fig. 18. The invariants for the proof of theorem *LowCorrect*.

$\langle 1\rangle 4.$ ASSUME: $g \in Lt$
PROVE: $LSpec(g) \wedge \Box CInv(g) \wedge \Box(\forall h \in Lt \setminus \{g\} : LInv(h, g)) \Rightarrow Mid.LSpec(g)$

$\langle 2\rangle 1.$ $LSpec(g).1 \wedge \Box CInv(g) \wedge \Box(\forall h \in Lt \setminus \{g\} : LInv(h, g)) \Rightarrow Mid.Init(g) \wedge \Box[Mid.Next(g)]_{Mid.var(g)}$

...

$\langle 2\rangle 2.$ $\Box CInv(g) \Rightarrow \Box(Loyal(g) \wedge (rcvd[g][g] = \text{"?"}) \Rightarrow (now \leq Now_0 + \delta))$

...

$\langle 2\rangle 3.$ $\Box LInv(h, g) \Rightarrow \Box(Loyal(g) \wedge (rcvd[g][h] = \text{"?"}) \Rightarrow (now \leq Now_0 + 2 * \delta))$

...

$\langle 2\rangle 4.$ $\Box[LNext(g)]_{lvar(g)} \wedge EMax(g, Choose(g)) \Rightarrow \exists\, t : VTimer(t, Mid.Choose(g), \epsilon, Mid.var(g)) \wedge MaxTimer(t)$

$\langle 3\rangle 1.$ $\Box[LNext(g)]_{lvar(g)} \wedge VTimer(t, Choose(g), \epsilon, lvar(g)) \Rightarrow VTimer(t, Mid.Choose(g), \epsilon, Mid.var(g))$

...

$\langle 3\rangle 2.$ $\wedge\ \Box[LNext(g)]_{lvar(g)}$ $\wedge\ VTimer(t, Choose(g), \epsilon, lvar(g)) \wedge MaxTimer(t)$ $\Rightarrow \exists\, t : VTimer(t, Mid.Choose(g), \epsilon, Mid.var(g)) \wedge MaxTimer(t)$

PROOF: $\langle 3\rangle 1$.

$\langle 3\rangle 3.$ $\wedge\ \Box[LNext(g)]_{lvar(g)}$ $\wedge\ \exists\, t : VTimer(t, Choose(g), \epsilon, lvar(g)) \wedge MaxTimer(t)$ $\Rightarrow \exists\, t : VTimer(t, Mid.Choose(g), \epsilon, Mid.var(g)) \wedge MaxTimer(t)$

PROOF: $\langle 3\rangle 2$, since t does not occur free in $\Box[LNext(g)]_{lvar(g)}$ or $\exists\, t : \ldots$.

$\langle 3\rangle 4.$ Q.E.D.

PROOF: $\langle 3\rangle 2$ and the definition of $EMax(g, Choose(g))$.

$\langle 2\rangle 5.$ Q.E.D.

PROOF: $\langle 2\rangle 1$–$\langle 2\rangle 4$ and the definition of $Mid.LSpec(g)$.

Fig. 19. The proof of step $\langle 1\rangle 4$ from Figure 16, with most steps elided.

has a familiar form, since $VTimer(\ldots)$ has the form $Init \wedge \Box[Next]_v$; its proof is simple.

The one remaining step in the high-level proof of the theorem is $\langle 1\rangle 5$. It is easy, and is omitted.

4 Discussion

Because the algorithm and its informal specification were well understood, writing the formal specifications was a straightforward exercise. As expected, we discovered small mistakes in the initial versions while writing the proofs. Our specifications were not subject to mechanical checking, so they probably still contain typographical errors. We would like to say that any such errors are minor, but with modern text editors and typesetting systems, one careless keystroke can produce major mistakes.

We have used a specification style different from our customary one. We usually write *interleaving* specifications, in which events in different processes are represented by separate steps [3]. Here, we have written *noninterleaving* specifications that allow individual steps which represent actions in two or more processes. For example, the formula *Spec* of module *MidLevel* allows behaviors in which a single step is both an $Issue(g)$ step of lieutenant g and a $Relay(h, g)$ step of a different lieutenant h. Writing noninterleaving specifications introduced no new problems.

Writing the proofs was also a straightforward exercise. As is typical when reasoning about real time, our specifications are safety properties; there are no liveness properties. When proving safety properties, creativity is required only in finding the invariants. With practice, writing invariants becomes second nature. The rest of the proof is a standard process of applying simple TLA proof rules and using the structure of the formulas to decompose the resulting proof obligations.

Writing this kind of proof is an exercise in organizing a complex structure. It is very much like programming; it is completely different from what mathematicians do when they write proofs. All the steps in our proof are mathematically trivial. Hierarchically structured proofs are long and tedious, but they are the only kind of hand proofs that can be trusted. There are no shortcuts. Short proofs are short because they gloss over details that have to be checked to avoid errors.

These proofs are amenable to mechanical verification. Most steps can be checked with the TLP verification system [7]. However, for this type of reasoning, which cannot be checked by finite-state methods, mechanical theorem proving still seems to be considerably more work than writing a hand proof. The hierarchical proof style makes it possible to reduce the probability of errors in hand proofs to an acceptable level.

Our specifications are written in TLA$^+$. The flexibility of TLA$^+$ is indicated by the ease with which real-time properties are expressed, even though the language has no special primitives for time. We use the same *RealTime* module for specifying Byzantine generals that was used in [15] for specifying a gas burner. This kind of flexibility and modularity are characteristic of an engineering discipline.

Our proofs use the logic TLA. They are completely formal. Although most of the lower-level steps were omitted, the parts that we did present in detail show that all the proofs can be carried out to the level of simple propositional reasoning. The proofs are seamless. The theorems to be proved are mathematical formulas, and at each step we are proving a mathematical formula. There is no switching from programs to logic; there is no appeal to semantic understanding. Hierarchically decomposing a large problem into smaller ones by the use of simple mathematical rules is characteristic of an engineering discipline.

We believe that formal specification and proof is now feasible for high-level designs of real systems. They are not yet feasible for reasoning at the level of executable code, except in special applications or for small parts of a system. It may appear that it is difficult to reason about code because our specifications are logical formulas rather than programs. However, the primary issue is not one of language but of complexity. It is hard to reason about real programs because they are complicated. Formal reasoning is generally applied only to concurrent programs written in toy languages like CSP [8] and Unity [5]. A program in a toy language is no closer to a real program than is a TLA formula. Further work is needed before formal reasoning about executable code becomes routine.

References

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
2. Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. Research Report 91, Digital Equipment Corporation, Systems Research Center, 1992. An earlier version, without proofs, appeared in [6, pages 1–27].
3. Martín Abadi and Leslie Lamport. Conjoining specifications. Research Report 118, Digital Equipment Corporation, Systems Research Center, 1993. To appear in *ACM Transactions on Programming Languages and Systems*.
4. E. A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10:110–135, February 1975.
5. K. Mani Chandy and Jayadev Misra. *Parallel Program Design*. Addison-Wesley, Reading, Massachusetts, 1988.
6. J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992. Proceedings of a REX Real-Time Workshop, held in The Netherlands in June, 1991.
7. Urban Engberg, Peter Grønning, and Leslie Lamport. Mechanical verification of concurrent systems with TLA. In *Computer-Aided Verification*, Lecture Notes in Computer Science, Berlin, Heidelberg, New York, June 1992. Springer-Verlag. Proceedings of the Fourth International Conference, CAV'92.
8. C. A. R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, London, 1985.
9. Reino Kurki-Suonio. Operational specification with joint actions: Serializable databases. *Distributed Computing*, 6(1):19–37, 1992.
10. Simon S. Lam and A. Udaya Shankar. Protocol verification via projections. *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984.
11. Simon S. Lam and A. Udaya Shankar. Specifying modules to satisfy interfaces: A state transition system approach. *Distributed Computing*, 6(1):39–63, 1992.
12. Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, April 1983.
13. Leslie Lamport. The temporal logic of actions. Research Report 79, Digital Equipment Corporation, Systems Research Center, December 1991. To appear in *ACM Transactions on Programming Languages and Systems*.
14. Leslie Lamport. How to write a proof. Research Report 94, Digital Equipment Corporation, Systems Research Center, February 1993. To appear in *American Mathematical Monthly*.
15. Leslie Lamport. Hybrid systems in TLA$^+$. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 77–102, Berlin, Heidelberg, 1993. Springer-Verlag.

16. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
17. Nancy Lynch and Mark Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Symposium on the Principles of Distributed Computing*, pages 137–151. ACM, August 1987.
18. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York, 1991.
19. Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.
20. Peter G. Neumann and Leslie Lamport. Highly dependable distributed systems. Technical report, SRI International, June 1983. Contract Number DAEA18-81-G-0062, SRI Project 4180.

module *FiniteSets*[3]

import *Naturals*

$$
\begin{array}{l}
\mathit{HasCardinality}(n, S) \;\triangleq\; \textbf{let } Q \triangleq \{i \in \mathit{Nat} : i < n\} \\
\qquad\qquad\qquad\qquad\quad \textbf{in } \;\land\; n \in \mathit{Nat} \\
\qquad\qquad\qquad\qquad\qquad\;\; \land\; \exists f \in [Q \to S] : \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\; \forall s \in S : \exists q \in Q : f[q] = s \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\; \forall q1, q2 \in Q : (f[q1] = f[q2]) \Rightarrow (q1 = q2) \\
\mathit{IsFiniteSet}(S) \;\triangleq\; \exists n \in \mathit{Nat} : \mathit{HasCardinality}(n, S)
\end{array}
$$

module *Sequences*[42]

import *Naturals*

$$
\begin{array}{l}
\mathit{OneTo}(n) \;\triangleq\; \{i \in \mathit{Nat} : (1 \leq i) \land (i \leq n)\} \\
\mathit{Len}(s) \;\triangleq\; \textsc{choose } n : (n \in \mathit{Nat}) \land (\textsc{Domain } s) = \mathit{OneTo}(n) \\
\mathit{Head}(s) \;\triangleq\; s[1] \\
\mathit{Tail}(s) \;\triangleq\; [i \in \mathit{OneTo}(\mathit{Len}(s) - 1) \mapsto s[i+1]] \\
(s) \circ (t) \;\triangleq\; [i \in \mathit{OneTo}(\mathit{Len}(s) + \mathit{Len}(t)) \mapsto \textbf{if } i \leq \mathit{Len}(s) \textbf{ then } s[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } t[i - \mathit{Len}(s)]]
\end{array}
$$

module *RealTime*

import *Reals*

parameters $\mathit{now} : \textsc{variable}$

$\infty : \textsc{constant}$

assumption

$$\mathit{InfinityUnReal} \;\triangleq\; \infty \notin \mathit{Real}$$

$$
\begin{array}{l}
\mathit{RT}(v)^{[24]} \;\triangleq\; \land\; \mathit{now} \in \mathit{Real} \\
\qquad\qquad\quad\;\; \land\; \Box[\land\; \mathit{now}' \in \{r \in \mathit{Real} : \mathit{now} < r\} \\
\qquad\qquad\qquad\quad\;\; \land\; v' = v\;]_{\mathit{now}} \\
\mathit{VTimer}(x, A, \delta, v) \;\triangleq\; \land\; x = \textbf{if } \textsc{Enabled}\,\langle A\rangle_v \textbf{ then } \mathit{now} + \delta \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } \infty \\
\qquad\qquad\qquad\qquad\quad \land\; \Box[x' = \textbf{if } (\textsc{Enabled}\,\langle A\rangle_v)' \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{then if } \langle A\rangle_v \lor \neg\textsc{Enabled}\,\langle A\rangle_v \textbf{ then } \mathit{now}' + \delta \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\;\; \textbf{else } x \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } \infty \;]_{\langle x, v\rangle} \\
\mathit{MaxTimer}(x) \;\triangleq\; \Box[(x \neq \infty) \Rightarrow (\mathit{now}' \leq x)]_{\mathit{now}} \\
\mathit{MinTimer}(x, A, v) \;\triangleq\; \Box[A \Rightarrow (\mathit{now} \geq x)]_v
\end{array}
$$

Fig. 20. The modules *FiniteSets*, *Sequences*, and *RealTime*.

assumption Begins a sequence of assumptions that must be true of the constant parameters when the module is included in another module.[9]

CONSTANT Specifies a parameter's sort.[5]

definitions Begins a sequence of definitions. This keyword may be omitted or replaced by one of the keywords **action**, **boolean**, **constant**, **predicate**, **state function**, **temporal**, or **transition function** to denote the sort of the symbols being defined.

export Specifies definitions visible to importing and including modules.[29]

import Appends parameters, assumptions, definitions, and theorems of another module.[2]

include ... **as** ... **with** Appends definitions, assumptions (as theorems), and theorems of another module, with instantiations for its parameters.[30] [51]

module Begins a module.[1]

parameters Declares the free parameters of a module.[4]

theorem Begins a sequence of theorems. (They must be provable from the module's assumptions and the rules of first-order logic, set theory, and TLA.)

VARIABLE Specifies a parameter's sort.[5]

$\triangleq$ Defines an operator.[10]

A mostly meaningless decoration that ends the scope of an **assumption** or **theorem** section.[8]

Marks the end of a module.[14]

Fig. 21. The syntactic keywords and symbols of TLA$^+$.

Predicate and Action Operators

p'	[p true in final state of step]
$[A]_e$	[$A \vee (e' = e)$][16]
$\langle A \rangle_e$	[$A \wedge (e' \neq e)$]
ENABLED A	[An A step is possible]
UNCHANGED e	[$e' = e$][22]
$A \cdot B$	[Composition of actions]

Temporal Operators

$\Box F$	[F is always true][17]
$\Diamond F$	[Eventually: $\neg\Box\neg F$]
$\mathrm{WF}_e(A)$	[Weak fairness: $\Box\Diamond\langle A \rangle_e \vee \Box\Diamond\neg$ENABLED A]
$\mathrm{SF}_e(A)$	[Strong fairness: $\Box\Diamond\langle A \rangle_e \vee \Diamond\Box\neg$ENABLED A]
$F \leadsto G$	[Leads to: $\Box(F \Rightarrow \Diamond G)$]
$\boldsymbol{\exists}\, x : F$	[Temporal existential quantification (hiding).][39]

Fig. 22. The nonconstant operators of TLA$^+$.

Logic

$\textsc{true} \quad \textsc{false} \quad \land \quad \lor \quad \neg \quad \Rightarrow \quad \equiv$

$\forall x : p \qquad \exists x : p \qquad \forall x \in S : p \qquad \exists x \in S : p$

$\textsc{choose}\ x : p$ [Equals some x satisfying p]

Sets

$= \quad \neq \quad \in \quad \notin \quad \cup \quad \cap \quad \subseteq \quad \setminus$ [set difference]

$\{e_1, \ldots, e_n\}$	[Set consisting of elements e_i]
$\{x \in S : p\}$	[Set of elements x in S satisfying p]
$\{e : x \in S\}$	[Set of elements e such that x in S]
$\textsc{Subset}\ S$	[Set of subsets of S]
$\textsc{union}\ S$	[Union of all elements of S]

Functions

$f[e]$	[Function application][13]
$\textsc{Domain}\ f$	[Domain of function f]
$[x \in S \mapsto e]$	[Function f such that $f[x] = e$ for $x \in S$][32]
$[S \rightarrow T]$	[Set of functions f with $f[x] \in T$ for $x \in S$][28]
$[f\ \textsc{except}\ ![e_1] = e_2]$	[Function $\widehat{f}$ equal to f except $\widehat{f}[e_1] = e_2$][34]
$[f\ \textsc{except}\ ![e] \in S]$	[Set of functions $\widehat{f}$ equal to f except $\widehat{f}[e] \in S$]

Records

$e.x$	[The x-component of record e]
$[x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]$	[The record whose x_i component is e_i]
$[x_1 : S_1, \ldots, x_n : S_n]$	[Set of all records with x_i component in S_i]
$[r\ \textsc{except}\ !.x = e]$	[Record $\widehat{r}$ equal to r except $\widehat{r}.x = e$]
$[r\ \textsc{except}\ !.x \in S]$	[Set of records $\widehat{r}$ equal to r except $\widehat{r}.x \in S$]

Tuples

$e[i]$	[The i^{th} component of tuple e]
$\langle e_1, \ldots, e_n \rangle$	[The n-tuple whose i^{th} component is e_i][42]
$S_1 \times \ldots \times S_n$	[The set of all n-tuples with i^{th} component in S_i]

Miscellaneous

"$c_1 \ldots c_n$"	[A literal string of n characters][7]
$\textbf{if}\ p\ \textbf{then}\ e_1\ \textbf{else}\ e_2$	[Equals e_1 if p true, else e_2]
$\textbf{case}\ p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n$	[Equals e_i if p_i true]
$\textbf{let}\ x_1 \triangleq e_1 \ldots x_n \triangleq e_n\ \textbf{in}\ e$	[Equals e in the context of the definitions]

$\land\ p_1$ [the conjunction $p_1 \land \ldots \land p_n$] $\quad \lor\ p_1$ [the disjunction $p_1 \lor \ldots \lor p_n$][12]

$\quad \ldots \qquad\qquad\qquad\qquad \ldots$

$\land\ p_n \qquad\qquad\qquad\qquad \lor\ p_n$

Fig. 23. The constant operators of TLA$^+$.