Formal Verification of Algorithms for Critical Systems*

Reprint of paper in IEEE Transactions on Software Engineering, Volume 19, Number 1, pp 13–23, January 1993; a preliminary version was presented at *SIGSOFT '91: Software for Critical Systems*, December 4–6, New Orleans, LA (proceedings published as ACM SIGSOFT Engineering Notes, Volume 16, Number 5, pp. 1–15).

John Rushby and Friedrich von Henke†

Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA

Abstract

We describe our experience with formal, machine-checked verification of algorithms for critical applications, concentrating on a Byzantine fault-tolerant algorithm for synchronizing the clocks in the replicated computers of a digital flight control system.

First, we explain the problems encountered in unsynchronized systems and the necessity, and criticality, of fault-tolerant synchronization. We give an overview of one such algorithm, and of the arguments for its correctness.

Next, we describe a verification of the algorithm that we performed using our EHDM system for formal specification and verification. We indicate the errors we found in the published analysis of the algorithm, and other benefits that we derived from the verification.

Based on our experience, we derive some key requirements for a formal specification and verification system adequate to the task of verifying algorithms of the type considered.

Finally, we summarize our conclusions regarding the benefits of formal verification in this domain, and the capabilities required of verification systems in order to realize those benefits.

Index terms: Formal specification, formal verification, clock synchronization, fault-tolerant systems.

*This work was performed for the National Aeronautics and Space Administration Langley Research Center under contract NAS1 17067

†Current main affiliation: Fakultät für Informatik, Universität Ulm, Germany

1 Introduction

Use of formal methods is often advocated, and sometimes required, in the construction of software and digital hardware for critical systems [7,33]. Formal methods can range from pencil-and-paper descriptions and analyses in the style of conventional mathematical discourse, through the use of formalized specification languages, to truly formal specifications and mechanically-checked verification.

In this paper we describe our experience at one particular point in this spectrum of formal methods: we focus on formal verification, rather than specification alone, on algorithms rather than more concrete descriptions such as programs and circuits, and on fully mechanized methods, rather than those used only by hand.

We draw on the experiences of ourselves and colleagues in performing a number of substantial formal verifications of algorithms to support fault tolerance in digital flight control systems [25, 30]. In order to keep the description focussed, we concentrate on one particular application that we first undertook some years ago: a mechanically-checked formal verification of the Interactive Convergence Algorithm for Byzantine fault-tolerant clock synchronization [17, 27].

The paper is organized as follows. In the next section, we describe our problem domain in general, and fault-tolerant clock synchronization in particular. We describe how attempts to avoid synchronization have been unsuccessful, and hence the necessity and criticality of fault-tolerant synchronization algorithms. In the third section we outline the Interactive Convergence Algorithm, and the arguments for its correctness in the manner of a conventional mathematical presentation.

Our experience in performing a mechanically-checked verification of the algorithm, and the benefits we perceive to have derived from that effort, are described in the fourth section. In the fifth section we identify the capabilities required in a specification and verification system in order to realize those benefits most effectively. Our conclusions make up the sixth and final section.

2 Flight Control Systems

Increasingly, modern aircraft rely on Digital Flight-Control Systems (DFCS): computer systems that interpret the pilot's control inputs and send appropriate commands to the control surfaces and engines.

The perceived advantages of DFCS over analog or direct mechanical and hydraulic control include lower manufacturing costs, improved performance, efficiency, and handling, reduced pilot workload and, for military aircraft, improved agility, expanded flight envelope (e.g., extreme angles of attack), and the ability to exploit unstable airframes. Depending on the aircraft design, DFCS may manage all, or merely some, of the control surfaces and may or may not have back-up systems comprising either analog computers or direct mechanical and hydraulic connections between the pilot and control surfaces.

The Airbus A320 is the only passenger aircraft in current service with a full DFCS—that is, one controlling the primary control surfaces and all three axes [23], but forthcoming aircraft such as the Boeing 777 will also employ comprehensive DFCS.[1]

It is clear that extreme reliability must be required of a flight-critical DFCS. A much-quoted figure is a requirement for passenger aircraft that the probability of catastrophic failure during a 10 hour flight should be less than 10^{-9} per hour [9]. Such reliabilities are beyond those that can be guaranteed for the computers and other digital devices comprising the DFCS hardware. Not only must occasional latent manufacturing defects and the effects of aging be considered, but also environmental hazards such as power-supply disturbances, lightning strikes, and cosmic rays. These factors conspire to yield an overall reliability well below that required. It follows that some form of fault tolerance based on replication and redundancy is needed in order to achieve an underlying "hardware platform" of the required reliability.

In an N-modularly redundant (or N-plex) system, all calculations are performed by N identical computer systems and the results are submitted to some form of averaging or voting before being sent to the actuators. Great care is taken to eliminate single-point failures, so the separate computer systems (or "channels," as they are often called in fault-tolerant systems) will generally use different power supplies and be otherwise electrically and physically isolated as far as possible. It is then reasonable to assume that failures of the separate channels will be statistically independent, so that the probability of overall system failure is orders of magnitude better than that of the individual channels.

Notice that the purpose of this design is simply to tolerate random hardware faults; there is no protection against design faults. Any such faults in either the hardware or the software will be common to all members of the N-plex and all will fail together. In this paper, we do not address the issue of design faults in the hardware, nor in the application software that it runs. We are, however, very much concerned with the possibility of design faults in the redundancy-management software that harnesses the failure-prone individual components together as a fault-tolerant N-plex. Instead of a single computer executing the DFCS software, there will be several, which must manage redundant sensors, coordinate and vote actuator commands, and tolerate faults among their own members.

Complexity is a source of design faults, and there is a distinct possibility that a large quantity of redundancy-management software may lessen, rather than enhance, overall reliability. Consequently, some designers have sought simple solutions to the redundancy management problem. A plausible and simple approach uses an "asynchronous" design in which the computers run fairly independently of each other: each computer samples sensors independently, evaluates the control laws independently, and sends its actuator commands to an averaging or selection component

[1]The Concorde, which received FAA certification in 1969, has an analog flight control system with mechanical backup in all three primary axes.

that chooses the value to send to the actuator concerned. The triplex-redundant DFCS of the experimental AFTI-F16 airplane was built this way, and its flight tests reveal some of the shortcomings of the approach [20].

Because they are unsynchronized, individual channels can sample sensors at slightly different times, and thereby obtain readings that differ quite appreciably from one another. The gain in the control laws can amplify these input differences to provide even larger differences in the results submitted to the output selection algorithm. During ground qualification of the AFTI-F16, it was found that these differences sometimes resulted in a channel being declared failed when no real failure had occurred [19, p. 478]. An even more serious shortcoming of asynchronous systems arises when the control laws contain decision points. Here, sensor noise and sampling skew may cause independent channels to take different paths at the decision points and to produce widely divergent outputs. This occurred on Flight 44 of the AFTI-F16 flight tests [20, p. 44]. Each channel declared the others failed; the analog back-up was not selected because the simultaneous failure of two channels had not been anticipated and the aircraft was flown home on a single digital channel. Notice that all protective redundancy had been lost, and the aircraft was flown home in a mode for which it had not been designed—*yet no hardware failure had occurred.*

The AFTI-F16 flight tests revealed numerous other problems of a similar nature, to the extent that redundancy management became the *primary source of unreliability* in the DFCS and an impediment to testing [20, pp. 40–41]. Overcoming the various problems involved modifications to the application tasks implementing the control laws. These serious difficulties in plausibly simple designs have stimulated research into principled approaches to redundancy management for DFCS that will yield predictable behavior.

The favored approach uses synchronized channels, distribution of sensor data to all channels, and exact-match voting [5, 8, 11, 13, 15, 35]. Distribution of sensor data ensures that each channel performs its computations on the same inputs, and should therefore produce the same outputs. Channel failures can therefore be masked by exact-match majority voting of outputs sent to the actuators. Synchronization ensures that the clocks of the individual channels are kept within some bounded skew of each other, so that each channel will perform the same computations and will be ready to participate in votes and the distribution of sensor data at approximately the same time as all the other channels.

Of course, it is crucial to this approach that the algorithms and protocols for clock synchronization, sensor distribution, and voting are themselves fault tolerant. Prior to the investigations of the SIFT project [17, 22], the subtlety and delicacy of the required algorithms and protocols were not properly understood, and the notion of Byzantine faults had not been fully articulated.[2] Consequently, early synchronization protocols were seriously flawed: all were vulnerable to Byzantine faults, and many were incapable of tolerating less severe classes of faults. For example, the

[2] A Byzantine fault is one where a faulty component provides conflicting information to other components, potentially causing non-faulty components to declare each other failed.

failure of the first attempt to launch the Space Shuttle was due to a synchronization problem [10], and some observations were lost when the heavy radiation environment at Jupiter caused one of the clocks on the Voyager spacecraft to jump several seconds [1].

Some of the difficulties in synchronizing clocks in the presence of faults can be appreciated by considering a simple system with three components: A, B, and C. A and C have good clocks, but B's clock is faulty. A's clock indicates 2.00 pm, C's 2.01 pm, and B's clock indicates 1:58 pm to A but 2.03 pm to C. A sees that C's clock is ahead of its own, and that B's is behind by a somewhat greater amount; it would be natural therefore for A to set its own clock back a little. This situation is reversed, however, when considered from C's perspective. C sees that A's clock is a little behind its own and that B's is ahead by a rather greater amount; it will be natural for C to set its own clock *forward* a little. Thus the faulty clock B has the effect of driving the good clocks A and C further apart. The behavior of B's clock may seem actively malicious and therefore implausible. This is not so, however. A failed clock may act as a random number generator and thereby distribute very different values to different components at slightly different times.

Byzantine fault-tolerant clock synchronization algorithms are required to maintain synchronization despite the occurrence of any kind of fault whatever, provided there are not too many of them. Lamport and Melliar-Smith's paper [17] was a landmark in the field. They not only introduced three Byzantine fault-tolerant clock synchronization algorithms, but they developed formalizations of the assumptions and desired properties that made it possible to give a precise statement and proof for the correctness of such algorithms. Other authors subsequently proposed several additional clock synchronization protocols, but most of these can be seen as variations on one of those introduced by Lamport and Melliar-Smith, namely the Interactive Convergence Algorithm (ICA) (see [24] for a brief survey). Schneider [29] makes this observation precise and shows that most algorithms for fault-tolerant clock synchronization are refinements of a single common paradigm, exemplified by ICA.[3]

In the following section we outline the Interactive Convergence Algorithm and its analysis. The outline includes precise statements of all the assumptions, constraints, and key lemmas needed for the analysis. We include this level of detail because it is necessary to convey the intricacy of the argument, and the challenge it poses to those who require complete confidence that all the details and boundary cases are handled correctly.

3 The Interactive Convergence Clock-Synchronization Algorithm and its Analysis

The goal of ICA is to maintain the clocks of redundant channels within some bounded skew δ of each other. All channels have reasonably accurate clocks with

[3]Our colleague Shankar has formally verified Schneider's general paradigm [30].

a maximum rate of drift from real time given by ρ, and are synchronized within some bound δ_0 initially. The slight differences in their running rates will cause the clocks gradually to drift apart, so that they must be resynchronized periodically. Each channel resynchronizes by determining the differences between its own clock and those of other channels, forming a "fault-tolerant average" of those differences, and adjusting its own clock by that amount. Trivial solutions, such as those in which all clocks are reset to zero at every resynchronization must be excluded, so there is a small bound Σ on the size of the adjustment that may be made at each resynchronization. Determining the differences between their clock readings may require cooperation among the channels, so they must all engage in the synchronization protocol at approximately the same time. To do this, each channel engages in the synchronization protocol every R seconds, and for a duration of S seconds, according to its own clock. Channels may not be able to determine the differences between their clocks with absolute accuracy: the quantity ϵ is assumed to bound the error in $\Delta_{q\,p}^{(i)}$, which denotes the difference between the clocks of channels q and p, as determined by p during its ith resynchronization.

The key to making a clock synchronization algorithm resistant to Byzantine failures among its components is the use of a "fault-tolerant average" in the adjustment step [29]. ICA is characterized by use of the *egocentric mean* as its averaging function. To compute the egocentric mean, a channel replaces all differences $\Delta_{q\,p}^{(i)}$ larger than a fixed quantity Δ by zero, and then calculates the arithmetic mean of the resulting set of differences.

To gain an idea of why this works, consider two nonfaulty channels p and q. For simplicity, assume that these channels perform their synchronization calculations simultaneously and instantaneously. If r is also a nonfaulty channel, then the estimates that p and q form of r's clock value can differ by at most 2ϵ. If r is a faulty channel, however, p and q could form estimates of its clock value that differ by as much as $2\Delta + \delta$ (since r could indicate a value as large as Δ different from each of p and q without being disregarded, and these channels could themselves have clocks that are δ apart). Assuming there are n channels, of which m are faulty, the egocentric means formed by p and q can therefore differ from each other by as much as

$$\frac{(n-m)2\epsilon + m(\delta + 2\Delta)}{n}.$$

Thus, provided

$$\delta \geq 2\epsilon + \frac{2m\Delta}{n-m},$$

this procedure will maintain the clocks of p and q within δ of each other, as required.

The sketch we have just given neglects many important details (for example, the channels do not perform the algorithm simultaneously and instantaneously, and their clocks may drift further apart in the periods between resynchronizations). Since the clock synchronization algorithm is a potential source of single point failure in DFCS, we require an unusually high degree of confidence in the analysis of ICA. In the

next subsection we outline the fully detailed analysis of ICA that we have subjected to mechanically-checked verification.

3.1 Formalizing the Arguments

We need to distinguish two notions of time: *clock time* is the internal estimate of time that a channel obtains from its clock, while *real time* is an external, abstract notion of time that provides a common frame of reference. Real time need not be directly observable, since clocks are only to be synchronized with respect to each other, not to an external reference. Clocks are modeled as functions from clock to real time: $c_p(T)$ denotes the real time at which channel p's physical clock reads T. Clocks are adjusted by subtracting a correction from their readings: suppose p's clock reads T' when we would like it to read T, then $C = T' - T$ is the *correction* that should be subtracted from the physical clock reading to yield the value to be reported as the "logical" clock reading. Since the corrected value T is reported when the physical clock reads T' (i.e., $T + C$), we see that the logical clock time T is reported at real time $c(T + C)$.

Corrections are computed once every resynchronization period. The correction for channel p in period i is denoted $C_p^{(i)}$, and we define

$$c_p^{(i)}(T) = c_p(T + C_p^{(i)})$$

as the *logical* clock for p during period i. The ith period is denoted $R^{(i)}$ and runs from clock time $T^{(i)}$ to $T^{(i+1)}$, where $T^{(i)} = T^0 + iR$ $(i \geq 0)$ and T^0 is an arbitrary constant. Synchronization takes place during the last S seconds of each period; $S^{(i)}$ denotes the interval $[T^{(i+1)} - S, T^{(i+1)}]$.

Now we can formalize the informal description given earlier. First we define a good clock to be one that keeps reasonably accurate time.

Good clock: *A clock c is a* good clock *during the clock time interval $[T_0, T_N]$ if*

$$\left| \frac{c(T_1) - c(T_2)}{T_1 - T_2} - 1 \right| \leq \frac{\rho}{2}$$

whenever T_1 and T_2 $(T_1 \neq T_2)$ are clock times in $[T_0, T_N]$.[4]

All channels start off with their clocks approximately synchronized.

A0: *For all channels p and q,*

$$|c_p^{(0)}(T^{(0)}) - c_q^{(0)}(T^{(0)})| < \delta_0.$$

Channels and their clocks can develop faults, so we need to say what it means for a channel to be nonfaulty.

[4]This definition implies that good clocks are monotonic—a fact that is proved in our formal verification.

A1: *We say that a channel is* nonfaulty *through period i if its clock is a good clock in the clock time interval $[T^{(0)} + C_p^{(0)}, T^{(i+1)} + C_p^{(i)}]$.*

Now we can state the conditions that ICA is to maintain. The first says that the skew between the clocks of nonfaulty channels must be bounded; the second says that corrections must be bounded.

Clock Synchronization Conditions: For all channels p and q, if all but at most m channels (out of n) are nonfaulty through period i, then

S1: *If p and q are nonfaulty through period i, then for all T in $R^{(i)}$*

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta.$$

S2: *If channel p is nonfaulty through period i, then*

$$|C_p^{(i+1)} - C_p^{(i)}| < \Sigma.$$

ICA requires that each nonfaulty channel can read the difference between its own clock and that of another nonfaulty channel with a bounded error. In order to do this, the channels may already need to be synchronized, and so we require S1 and S2 to hold.

A2: *If conditions S1 and S2 hold for the i'th period, and channel p is nonfaulty through period i, then for each other channel q, p obtains a value $\Delta_{qp}^{(i)}$ during the synchronization period $S^{(i)}$. If q is also nonfaulty through period i, then*

$$|\Delta_{qp}^{(i)}| \leq S$$

and

$$|c_p^{(i)}(T' + \Delta_{qp}^{(i)}) - c_q^{(i)}(T')| < \epsilon$$

for some time T' in $S^{(i)}$.

If $p = q$, we take $\Delta_{qp}^{(i)} = 0$ so that A2 holds in this case also.

Finally, we can give a formal description of ICA.

Algorithm ICA: *For all channels p:*

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p^{(i)},$$

where

$$\Delta_p^{(i)} = \left(\frac{1}{n}\right) \sum_{r=1}^{n} \bar{\Delta}_{rp}^{(i)}, \quad \textbf{and}$$

$$\bar{\Delta}_{rp}^{(i)} = \textbf{if } |\Delta_{rp}^{(i)}| < \Delta \textbf{ then } \Delta_{rp}^{(i)} \textbf{ else } 0.$$

Note that $C_p^{(0)}$ is constrained by A0.

In order to establish that ICA satisfies S1 and S2, the following constraints among its parameters must be satisfied.

C0: $0 \leq m < n$

C1: $R \geq 3S$

C2: $S \geq \Sigma$

C3: $\Sigma \geq \Delta$

C4: $\Delta \geq \delta + \epsilon + \frac{\rho}{2} S$

C5: $\delta \geq \delta_0 + \rho R$

C6: $\delta \geq 2(\epsilon + \rho S) + \frac{2m\Delta}{n-m} + \frac{n\rho R}{n-m} + \frac{n\rho\Sigma}{n-m} + \rho\Delta$

Lemma 1 *If the clock synchronization conditions S1 and S2 hold for i, and channels p and q are nonfaulty through period $i+1$, then*

$$|\Delta_{q\,p}^{(i)}| < \Delta.$$

Lemma 2 *If channel p is nonfaulty through period $i+1$, and T and Π are such that $T + C_p^{(i)}$ and $T + \Pi + C_p^{(i)}$ are both in the interval $[T^{(0)} + C_p^{(0)}, T^{(i+2)} + C_p^{(i+1)}]$, then*

$$|c_p^{(i)}(T+\Pi) - [c_p^{(i)}(T) + \Pi]| \leq \frac{\rho}{2} |\Pi|.$$

Lemma 3 *If the clock synchronization conditions S1 and S2 hold for i, channels p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T + \Delta_{q\,p}^{(i)}) - c_q^{(i)}(T)| < \epsilon + \rho S.$$

Lemma 4 *If the clock synchronization conditions S1 and S2 hold for i, channels p, q, and r are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < 2(\epsilon + \rho S) + \rho\Delta.$$

Lemma 5 *If the clock synchronization condition S1 holds for i, channels p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < \delta + 2\Delta.$$

Figure 1: Statements of the Principal Lemmas used in The Proof

The proof that A0, A1, A2, A3, and C0 through C6 are sufficient to ensure that ICA achieves S1 and S2 depends on the 5 lemmas shown in Figure 1. The motivation for these lemmas can be described as follows.

We are interested in the skew between two nonfaulty processors during the $i+1$'st period—that is, in the quantity

$$|c_p^{(i+1)}(T) - c_q^{(i+1)}(T)|$$

where $T \in R^{(i+1)}$. By the Algorithm,

$$|c_p^{(i+1)}(T) - c_q^{(i+1)}(T)| = |c_p^{(i)}(T + \Delta_p^{(i)}) - c_q^{(i)}(T + \Delta_q^{(i)})|, \tag{1}$$

and since good clocks run at approximately the correct rate, $c_p^{(i)}(T + \Delta_p^{(i)})$ and $c_q^{(i)}(T + \Delta_q^{(i)})$ are close to $c_p^{(i)}(T) + \Delta_p^{(i)}$ and to $c_q^{(i)}(T) + \Delta_q^{(i)}$, respectively. From this it follows that the right hand side of (1) can be approximated by

$$|c_p^{(i)}(T) + \Delta_p^{(i)} - [c_q^{(i)}(T) + \Delta_q^{(i)}]|.$$

A major step in the proof, identified as Lemma 2, is concerned with bounding the error introduced by this approximation. Then, since $\Delta_p^{(i)}$ and $\Delta_q^{(i)}$ are the averages of $\bar{\Delta}_{rp}^{(i)}$ and $\bar{\Delta}_{rq}^{(i)}$, it is natural to consider the individual components

$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]|. \tag{2}$$

There are two cases to consider. The first, in which only p and q are assumed nonfaulty, is the focus of Lemma 5, while the second, in which r is also assumed nonfaulty, is considered in Lemma 4. The first case is quite easy—the Algorithm ensures that $\bar{\Delta}_{rp}^{(i)}$ and $\bar{\Delta}_{rq}^{(i)}$ can be no larger than Δ, while $c_p^{(i)}(T)$ and $c_q^{(i)}(T)$ can differ by no more than δ (by the inductive hypothesis). For the second case, Lemma 1 provides the result $|\Delta_{rp}^{(i)}| < \Delta$, so that the Algorithm will establish $\bar{\Delta}_{rp}^{(i)} = \Delta_{rp}^{(i)}$ and $\bar{\Delta}_{rq}^{(i)} = \Delta_{rq}^{(i)}$. The quantity (2) is then rewritten as

$$|c_p^{(i)}(T) + \Delta_{rp}^{(i)} - c_r^{(i)}(T) - [c_q^{(i)}(T) + \Delta_{rq}^{(i)} - c_r^{(i)}(T)]|.$$

Regarding this as the absolute difference of two similar expressions, we are led to consider values of the form

$$|c_p^{(i)}(T) + \Delta_{rp}^{(i)} - c_r^{(i)}(T)|$$

which, using Lemma 2, can be approximated by

$$|c_p^{(i)}(T + \Delta_{rp}^{(i)}) - c_r^{(i)}(T)|.$$

Lemma 3 is concerned with quantities of this form.

In the next section we describe our formal verification of this analysis, and the benefits we derived.

4 Formal Verification

Although a broad understanding of why ICA works can be obtained fairly readily, detailed proof of its lemmas and of its main theorem requires attention to a mass of details and an astonishingly intricate argument. The journal proof by Lamport and Melliar-Smith [17] is unusually precise and detailed, yet it is hard to internalize: it makes use of approximate arithmetic and neglect of insignificant terms, and when we examined it we were unable to convince ourselves of all the details after several days study. We needed to be convinced of the details, and of all the assumptions that underlie the proof, because we had the goal of designing a combination of hardware and software to implement the algorithm. ICA assumptions such as A2 become specifications for the hardware that will perform the exchange of clock values, and constraints such as C6 determine the closeness of the synchronization that can be assumed by the rest of the DFCS redundancy management software.

In order to resolve our doubts concerning the analysis of ICA, we embarked on a formal verification of ICA in 1988, using an early version of our EHDM formal verification system [28]. Our goal was to obtain a complete understanding of the argument for the synchronization bound maintained by ICA, and a complete enumeration of the assumptions on which the argument depends.

As we performed the formal specification and verification, we discovered that the presentation given by Lamport and Melliar-Smith was flawed in several details. One of the principal sources of error and difficulty was their use of approximations—i.e., approximate equality ($\approx$) and inequalities ($\lesssim$ and $\gtrsim$)—in order to "simplify the calculations." We eventually found that elimination of the approximations not only removed one class of errors, but actually simplified the analysis and presentation. We also found and corrected several other technical flaws in the published proof of Lamport and Melliar-Smith.

In total, we found that four of the five lemmas in Lamport and Melliar-Smith's proof were false, or flawed in some other way, and that the main induction was incorrect. Some of these errors are painfully obvious once they have been spotted: for example, the problem in the main induction is that it seeks to establish S1, but the inductive step provides only

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| \lesssim \delta;$$

that is strict inequality is assumed, but only an approximation is delivered. Other flaws include missing, or insufficiently tight constraints in the statements of some lemmas, and typographical errors in two of them. Our corrections required slight modifications to the assumptions underlying the algorithm, and to the constraints on its parameters, and thus changed the external specifications of the algorithm. The presentation in the previous section used our corrected statements of assumptions, constraints, and lemmas. A full discussion of the flaws in Lamport and Melliar-Smith's original presentation, and of our revisions, is available in our report on the verification [27, chapter 3].

In addition to identifying and correcting flaws in the published analysis of ICA, we were able to extract a journal-level description of our revised analysis fairly directly from the text of our formal specification and proofs. The EHDM theorem prover is driven from "proof descriptions" that explicitly list the instantiations of all lemmas and axioms to be used in the proof concerned. While tedious to construct, these descriptions provide the key information needed for a journal-level proof. For example, the EHDM proof description for Lemma 2 lists the definition of a good clock, the definition of the logical clock values $c_p^{(i)}(T)$ and $c_p^{(i)}(T+\Pi)$, and the assumption A1 as the key ingredients of the proof. It is easy to construct the journal-level proof from these facts.

Another benefit that we derived from our formal specification and verification of ICA was a complete enumeration of all the assumptions, definitions, and constraints employed. This enumeration is a by-product of the EHDM "proof-chain checker," which examines the macroscopic structure of a verification in order to ensure that there are no undischarged proof obligations and no circularities in the argument. It is important to those contemplating the construction of hardware support for ICA to have this enumeration of assumptions available, since it comprises the specification for their part of the overall enterprise.

Yet another benefit that we derived from our formal verification of ICA was the ability to explore the consequences of changed assumptions. For example, the journal proof of Lamport and Melliar-Smith employs an assumption that the initial clock corrections $C_p^{(0)}$ are all zero. We incorporated this assumption into our formal verification. Later, when we were contemplating implementation of the algorithm, we recognized that this was a very inconvenient constraint and wondered if it could be eliminated. We explored this possibility by simply eliminating the constraint from the formal specification and re-running all the proofs. We found that the proofs of a few internal lemmas needed to be adjusted, but that the rest of the verification was unaffected.

We found the formal verification of ICA to be quite challenging. The initial effort took a little over a man-month, with the properties required of summations and other "supporting theories" treated as "temporary axioms" (i.e., unproven lemmas) at that stage. We gradually developed satisfactorily primitive axiomatizations—or constructive definitions—for the supporting theories and developed proofs for all our temporary axioms. The current version of the verification uses 19 axioms (most of which are used to specify the ICA and its assumptions and constraints), and requires mechanical checking of a shade under 200 proofs.

In the following section we draw on our experiences in verifying ICA and other critical algorithms to identify the capabilities required or desirable in a formal verification system to undertake these verifications with maximum benefit and ease.

5 Capabilities Required

Our verification of ICA helped us identify shortcomings in the implementation, specification language, and theorem prover and other tools of EHDM—as well as confirming a number of its good features. In the years since we first completed this verification, we and our colleagues have worked to improve EHDM, and have tested the benefits by undertaking several other verifications of difficult fault-tolerant algorithms [25, 30], as well as a number of moderate test-pieces including the Oral Messages algorithm for Byzantine Agreement [26], the finite Ramsey Theorem, and several examples in hardware verification and software specification. We have revised our specification and verification of ICA several times in order to take advantage of new capabilities in EHDM, or new insight into how best to formulate certain properties.

In this section we draw on the experience gained in performing these verifications and identify what we consider to be important issues in the design of specification languages and verification systems. We will use ICA to supply concrete illustrations of our points, and suggest that the ability to specify and verify ICA in reasonable style could be considered a minimum benchmark for mechanized formal verification systems.[5]

The first point to note is that mechanically-checked verification is the very essence of exercises we have performed, and the chief source of the benefits that accrued. In some domains, formal specifications, possibly augmented by pencil and paper analysis, can provide worthwhile benefits on their own—that is, without undertaking mechanically-checked verification [12]. That is not the case with most of the examples we have undertaken: we have been concerned mainly with algorithms and the theorems that sustain them, and these were adequately specified already—albeit in the quasi-formal style of traditional mathematical presentations [17, 18, 29]. Merely casting these journal-level descriptions into a formal specification language is unlikely to reap large dividends.

But although we have primarily been concerned with verification, this does not mean that we regard formal specification as unimportant. On the contrary, we consider human review essential for certification of truly critical systems, and it is therefore crucial that the formal specifications should be accessible to anyone familiar with traditional mathematical presentations; similarly, the verification should lend itself to the extraction of a genuine proof that can be followed by anyone willing to devote modest effort.

A formal specification language to be used with mechanically-checked verification must strike a very delicate balance between its convenience and expressiveness as a specification medium and the automation and effectiveness of the mechanical support that can be provided. Thus, the most powerful and automatic theorem provers tend to be associated with the most limited facilities for specification (for

[5] Our original verification in EHDM [27, unrevised edition] has been duplicated using the Boyer-Moore prover [4] (with the CONSTRAIN and DEFN-SK extensions) by Bill Young [36]. We invite others to try it using their own favorite formal specification and verification system.

example, the very powerful Boyer-Moore theorem prover [4] uses a highly constructive, untyped, quantifier-free first order logic as its specification language), whereas some more attractive and expressive specification notations (for example, Z [32]) lack mechanical support for verifications as difficult as that of ICA. In the next two subsections we identify some requirements on specification languages and theorem provers adequate to the task of verifying ICA, and suggest how the competing requirements for expressiveness and mechanization can be reconciled.

5.1 Specification Languages

The purpose of specification is communication, and we believe that communication with those who will review or implement our specifications is best served by a formalism that is as close as possible to that of conventional mathematical presentations in the field concerned. For our verification of ICA, we had a readily available benchmark in the form of a conventional mathematical presentation of that very algorithm [17]. Our goal was to formally specify and verify ICA in a manner that would lend itself to a journal-level presentation similar to that of the original.

Examination of the original presentation reveals that it uses, at the very least, first-order logic, with arithmetic and full quantification. (Most statements are, implicitly, universally quantified in their free variables, but assumption A2, for example, requires an explicit, nested, existential quantifier.)

Specification of the ICA algorithm uses a finite summation, which suggests that it will be desirable to have available a facility for recursive or iterative definitions. Summation is most naturally regarded as a higher-order functional (i.e., it takes the function to be summed as an argument), which suggests that the specification language should admit at least that fragment of higher-order logic that allows functions to take functions as arguments and to return functions as values. The journal proof for the correctness of ICA is by induction, and it is easy to anticipate that establishing the properties of summations will also require proofs by induction. Thus, the verification system must either provide a repertoire of built-in induction schemes, or the specification language must permit higher-order quantification (i.e., quantification over predicates and functions), so that induction schemes can be manipulated as ordinary formulas.

Next, we can observe that at least two types of numerical quantities are required for specification and verification of ICA. Clocktime and realtime are assumed to be dense, whereas synchronizing periods are numbered by the naturals. It follows that our specification language should be typed, and should provide interpretations for (at least) the rational (or the real) numbers, and the naturals. In general, integers are required as well.

Considered as numerical types, clocktime and realtime share the same properties, yet they cannot be combined freely: for example, a formula that adds a term of type clocktime to one of type realtime is likely to be erroneous. Thus there is a distinction between clocktime and realtime akin to the notion of dimension, which distinguishes length from velocity even though both have the same numerical properties. A good

specification language should make it possible to catch simple errors early, and by simple means; hence, strict typing, possibly augmented by dimensions, is very desirable.

Although not needed for ICA, type-constructors such as records, enumerations, and tuples are invaluable in most specifications, as is the ability to introduce new, uninterpreted types.

Not only is it important for a specification language to provide a familiar and reasonably rich logic, we consider it important that it should, to the extent possible, express that logic in familiar notation. For example, a rich set of propositional connectives can enhance the readability of the specification. Thus, specification of ICA uses a polymorphic *if-then-else* in the definition

$$\bar{\Delta}^{(i)}_{r\,p} = \textbf{if } |\Delta^{(i)}_{r\,p}| < \Delta \textbf{ then } \Delta^{(i)}_{r\,p} \textbf{ else } 0.$$

This is equivalent to the formula

$$(|\Delta^{(i)}_{r\,p}| < \Delta \supset \bar{\Delta}^{(i)}_{r\,p} = \Delta^{(i)}_{r\,p}) \wedge (|\Delta^{(i)}_{r\,p}| \geq \Delta \supset \bar{\Delta}^{(i)}_{r\,p} = 0)$$

and there is no reason why the verification system should not silently perform this "if-lifting" transformation and allow the specifier to use the more convenient *if-then-else* form.

Similarly, we should expect to be able to write arithmetic operators and relations in their familiar infix form, and to be able to overload infix and other symbols in a reasonably natural way—so that the $+$ in $x{+}y$ is interpreted appropriately according to the types of its arguments, and the context of its use. Maintaining such a natural notation, while providing a rigorous yet straightforward semantics and supplying the most effective information to the arithmetic reasoning component of a theorem prover, can involve some sophistication.

For example, notice that the constraint C6 involves several division operations, with divisor $n-m$. Clearly, the logic must make some provision for partial functions such as division (which is undefined if the divisor is zero). There are several ways to do this. One, which we favor, avoids partial functions (which rapidly complicate matters and can require a three-valued logic [6] or a logic of partial terms [3, Section 5]) and instead defines the signature of division as $\mathcal{Q} \times \mathcal{Q}_z \to \mathcal{Q}$ where $\mathcal{Q}$ denotes the rationals and $\mathcal{Q}_z$ denotes the nonzero rationals: a subtype of the rationals associated with the predicate $(\lambda r : r \neq 0)$. We allow a term of a supertype to appear where one of a subtype is required, provided the term can be proved to satisfy the defining predicate of the subtype concerned—thus theorem proving can be required during typechecking. To see how this works in EHDM, consider a simplified fragment from C6: $\frac{\rho}{n-m}$, where ρ is a rational, n and m are naturals, and $n > m$. There is no subtraction operation defined on the naturals, so n and m are promoted to integers (a supertype of naturals), and $n - m$ is interpreted as integer subtraction, yielding an integer result. We are now supplying an integer as the second argument to the division operator, which requires a nonzero rational in this position. The rationals are a common supertype to both the integers and the nonzero rationals, so $n - m$

can be promoted to a rational, and then reduced to the nonzero rational required for type-correctness if we can prove the theorem $n - m \neq 0$ (which in this case follows obviously from the constraint $n > m$).

Correctness of ICA is contingent upon a number of assumptions that relate the values of several quantities to each other. Some of these can be regarded as definitions—for example:

$$\Delta_p^{(i)} = \left(\frac{1}{n}\right) \sum_{r=1}^{n} \bar{\Delta}_{r\,p}^{(i)}.$$

It is best if the specification language has some definitional principle that admits such definitions by conservative extension—that is, in a way that is guaranteed not to introduce inconsistencies. The definitional principle should be strong enough to admit recursive definitions, such as that underlying the recurrence

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p^{(i)}.$$

Ensuring the "termination" of recursive definitions requires theorem proving in general.

Not all the assumptions underlying ICA are simple definitions, however: constraints C0 to C6, and A0 and A2, are inequalities. Some specification languages are strictly constructive and prohibit the direct introduction of axioms. In such cases, one must either provide a construction that satisfies the constraints, or make the constraints into explicit hypotheses of the theorems to be proved. The first approach overspecifies the problem: the sense surely intended by the conventional mathematical presentation is that any implementation that satisfies the constraints is considered acceptable. It is not the job of this level of specification to describe or overly constrain possible implementations.

The second approach makes the specification cumbersome without adding any useful security. The concern of those who advocate totally constructive specifications is that unrestricted addition of axioms can introduce inconsistencies and thereby render the specification meaningless and any verifications worthless. But moving the constraints into the hypothesis of the theorem provides no advantage, since although

$$\vdash A0 \wedge \ldots \wedge C0 \ldots \wedge C6 \supset S1$$

is guaranteed to be sound, it is useless if the antecedent cannot be satisfied.

We argue that a useful specification language should permit the introduction of axioms, but should also assist (or require) demonstration of their consistency (i.e., the existence of a model). This differs from the purely constructive approach in that exhibition of a constructively defined model merely serves to demonstrate the consistency of the axiomatization, it is not a prescriptive part of the specification. EHDM supports this through the mechanism of "theory interpretation," which is also used in other applications to verify hierarchical development (i.e., that one level of refinement correctly "implements" another).[6]

[6]Our original specification used strict inequality in the definition of a good clock; Bill Young pointed out that this definition is unsatisfiable if the clocks are perfect (i.e., if $\rho = 0$). We have since

Another glance at the journal presentation of ICA shows that the verification uses several properties of the absolute value function ($|A|$) and of finite summations. It will clearly improve the structure and presentation of the overall development if specification and proof of these subsidiary properties can be presented separately from those of ICA itself. This argues for a module structure for specifications; the modules for "absolute value" and of "finite summations," for example, should contain the statement and proof of generally useful properties, such as the triangle inequality $|x + y| \leq |x| + |y|$, and identities such as

$$\sum_{i=m}^{n} (F(i) + G(i)) = \sum_{i=m}^{n} F(i) + \sum_{i=m}^{n} G(i).$$

It is desirable that the module mechanism should permit parameterization; it is then often necessary to place semantic constraints on the actual parameters that can be supplied to parameterized specification modules. For example, a module specifying the general principle of Noetherian induction should only be applied to a well-founded ordering. As we have noted several times before, it can require theorem proving to check constraints such as this.

The mechanical support required for a specification language obviously includes a parser and typechecker. In addition, since specifications can get quite large (1,300 lines of specification and 472 distinct identifiers are used in our verification of ICA), browsers, cross-reference generators, and other similar support tools for navigating a large body of material are highly desirable.

Finally, if formal specifications are to be used for effective communication, we consider it highly desirable that its mechanical support should include tools that can reproduce the typographical conventions of normal mathematical discourse. EHDM, for example, provides a LaTeX-printer: a table-driven tool that converts specification text such as

```
abs(c(p, i, T) - c(q, i, T))
```

into the notation of conventional mathematics used by Lamport and Melliar-Smith:

$$|c_p^{(i)}(T) - c_q^{(i)}(T)|.$$

Attractively typeset specifications are much easier to compare with arguments presented in normal mathematical notation, and much easier for persons not directly involved in the specification effort to read and study. This latter point is particularly important if formal specifications are to be subjected to useful peer review and scrutiny.

5.2 Verification and Theorem Proving

Mechanically-checked verification requires a theorem prover or proof checker. (A proof checker is simply a theorem prover that requires more human guidance.) It

verified the consistency of the axioms in our specification of ICA using the theory interpretation mechanism of EHDM.

might seem that the more powerful and automatic the theorem prover the better. This is a view that needs severe qualification however.

First, for the purposes for which we are undertaking verification, we want to obtain a genuine proof—that is a chain of argument that will convince a human reviewer—rather than a mere grunt of assent from a mechanical theorem prover. Many powerful theorem proving techniques (for example, resolution) work in ways that do not lend themselves to the extraction of a proof.

Second, many of the theorems that we attempt to prove will turn out to be incorrect—i.e., they will not be theorems. It is precisely because the details of unverified proofs can be missing or flawed that mechanical verification is of value: but this means that it is at least as important for the theorem prover to provide assistance in the discovery of error, as that it should be able to prove true theorems with aplomb. Highly automatic provers can waste a lot of time in fruitless search when given non-theorems; furthermore, the user has to figure out whether a failed proof attempt is due to a non-theorem or inadequate heuristics.

These limitations on the utility of powerful automatic theorem proving in the large do not apply in the small. Routine manipulations of arithmetic, equalities, disequalities, and inequalities can, and should, be completely mechanized. This is so because a human reviewer does not need to examine the argument for elementary deductions such as $x \geq y \geq z \supset x+1 > z$, and because *complete* decision procedures are available (at least for the ground case) [31], so that their behavior is predictable. It is hard to overestimate the contribution of the decision procedures in EHDM towards the successful completion of our verification of ICA. Arithmetic reasoning is fundamental to the analysis of ICA, and it would be enormously time consuming to undertake its verification with a theorem prover that lacks facility in arithmetic.[7]

We experienced a small indication of the tedium occasioned by absence of arithmetic competence when we first tackled the final manipulation required in the verification of the ICA synchronization condition S1. Here, it is necessary to prove the lemma:

$$\delta \geq 2(\epsilon + \rho S) + \frac{2m\Delta}{n-m} + \frac{n\rho R}{n-m} + \frac{n\rho\Sigma}{n-m} + \rho\Delta$$

$$\supset \delta \geq (\delta + 2\Delta) \times \frac{m}{n} + 2(\epsilon + \rho S + \frac{\rho}{2}\Delta) \times \frac{n-m}{n} + \rho R + \rho\Sigma$$

When we first attempted this proof, EHDM lacked built-in heuristics for nonlinear multiplication and division[8] and it took us a whole day, required 13 intermediate lemmas, and occupied 5 pages in the listing. Later, when David Cyrluk had provided effective heuristics for division and nonlinear multiplication, the proof was reduced to only two intermediate lemmas and less than an hour's work.

[7] The Boyer-Moore prover, which has been used to duplicate our verification of ICA [36], has decision procedures for integer arithmetic, a powerful rational arithmetic package, and was used in a very controlled, non-automatic mode to duplicate our proof lemma by lemma.

[8] Nonlinear multiplication is undecidable.

In our opinion, the key to truly effective theorem proving will be a productive symbiosis between man and machine: the user should guide the overall strategy and provide the insights, and the machine should do the straightforward calculations. The goal should be to maximize the productivity of the human time spent in dialog with the theorem prover. Our preferred approach requires careful integration of powerful primitive inference mechanisms within a goal-directed (i.e., subgoaling, or backwards-chaining) proof search [21].

The outcome of a successful proof attempt should be two proof descriptions (or possibly one description that serves two purposes): one should be very close to a proof that can accompany a journal-style presentation of the verification; the other should be a description that guides the prover to repeat the proof without human guidance. The latter will be needed to rerun the proof at later stages in the overall verification, when many surrounding details may have changed slightly.

Ideally, this second proof description should be robust—describing a strategy rather than a line by line argument—so that unimportant changes to the specification of lemmas will not derail it. This rerunning of proofs is essential if we regard the purpose of mechanically-checked verification as the acquisition of insight rather than mere certification. As a verification develops, one discovers simplifications, improvements, and generalizations that should be assisted, not discouraged, by investment in an existing verification. Even those most interested in certification should recognize that a specification is seldom static: a change in external requirements or in the specification of an assumed service will require reverification.

The dynamic and creative nature of verification development argues strongly for the ability to perform proofs in any order. Some systems require, or strongly encourage, a bottom up development in which only previously proven lemmas can be cited in a new proof. This is not the way real mathematics is performed: one generally prefers to know whether a proposed lemma is adequate to its intended use before attempting to prove it. But if proofs can be attempted in any order, then it is necessary to provide an analysis tool that examines the macroscopic structure of a complete verification in order to ensure the absence of circularities and unproved lemmas. An additional benefit of such a tool is a complete enumeration of the assumptions, definitions, and axioms on which the verification ultimately depends.

6 Conclusions

The benefits that we obtained from our formal verification of ICA can be summarized as follows.

- Identification of errors in the published analysis and proof.
- A corrected and simplified journal-level proof for the synchronization bounds maintained by ICA.

Note that our corrections merely dot the i's and cross some important t's in the original; the substance of all the arguments is due to Lamport and Melliar-Smith [17].

- A formal specification of the algorithm and a mechanically-checked verification of its synchronization bounds.
- A complete enumeration of the assumptions and constraints underlying the analysis of ICA.
- The ability to rapidly and reliably explore the consequences of modifications to certain assumptions and constraints.

It is the traditional mathematical presentation of our revised proof for the synchronization bounds maintained by ICA that we consider one of the main contributions of our work; we hope that anyone contemplating using the algorithm will study our presentation and will convince *themselves* of the correctness of the algorithm and of the appropriateness of the assumptions (and of the ability of their implementation to satisfy those assumptions).

We and our colleagues have derived similar benefits, including the identification of errors in previous journal-level proofs [29] in other formal verifications that we have undertaken [25, 30]. We do not claim that these benefits can only be obtained through mechanically-checked formal verification. For example, the flaws in the published analysis of ICA are readily apparent. The fact remains, however, that these flaws were not, to our knowledge, identified by the "social process" of peer review and scrutiny to which Lamport and Melliar-Smith's paper has been subjected since its publication, but they were detected—and we claim were bound to be detected—by our formal verification. Contrary to parodies erected by some of the detractors to formal verification, a mechanical theorem prover does not act as an oracle that certifies bewildering arguments for inscrutable reasons, but as an implacable skeptic that insists on all assumptions being stated and all claims justified. It was the demands of formal verification that led us to scrutinize the analysis of ICA with the care required to identify its shortcomings. Similarly, it was the depth of understanding of the analysis that we acquired thereby that enabled us to simplify and improve the journal-level presentation of the analysis. Thus, we regard formal verification as an instrument of discovery as much as a means of certification [14]:

> *"The virtue of a logical proof is not that it compels belief but that it suggests doubts."* [16, page 48]

Formal specification and verification is, at present, an undeniably expensive way of dispelling ignorance and revealing error. The costs can be reduced by appropriate choice of specification language and theorem prover, so that effort is brought to bear on the substance of the problem, and not on incidental difficulties due to complexity of the notation or inadequacy of the theorem prover. We identified some of the important issues in these regards in the previous section. The two principal issues

are the need to reconcile the desire for expressiveness in the specification language with the ability to provide effective mechanical support, and the requirement for the theorem prover to be specialized towards the requirements of verification: that is, it must assist in the rapid identification of the sources of errors in incorrect theorems as well as in the certification of true theorems, it must produce a proof suitable for human review, it must have powerful capabilities for low-level reasoning in standard theories (e.g., arithmetic), and its high-level capabilities must allow productive interaction with a human user.

No current system for formal specification and verification satisfies all these requirements simultaneously, and we are not persuaded that many are even headed in the right direction. In our opinion, those that have neglected verification and mechanical support to concentrate on specification have not faced up to the hard language design problems that come with full mechanization, those that have not done large, hard verifications underestimate the theorem proving power that is needed, and those that concentrate on theorem proving have inadequately addressed the need to extract a genuine proof from the exercise, the need to interact productively with a human user, and the consequences of the fact that most putative theorems are false.

Although no current verification system satisfies our requirements, we believe it is feasible to construct one that will come close to doing so. We have found that one of the keys to reconciling the desire for expressiveness with that for effective mechanical support is to exploit the latter to assist the former: by being willing to use theorem proving during typechecking we have found that we can provide an attractive and expressive notation without substantial penalty. And we have found it possible to develop a very attractive and effective proof checker by combining powerful primitive inferences (including decision procedures) with effective user guidance [21].

Another source of expense in formal verification is the cost of developing the supporting theories. (In our verification of ICA, for example, more than half the proofs are concerned with the properties of the supporting theories of summation, absolute value, induction, and some fragments of arithmetic.) As more verifications are performed, we expect a library of reusable theories to be developed. The cost of developing the library will be amortized over many verifications, eventually reducing overall costs significantly.

Our verification of ICA, and most of the other verifications that we have performed, have been concerned with algorithms, and other abstractions. We have not proved the "correctness" of specific programs or hardware circuits, although we are currently in the process of verifying the design of a hardware circuit to support implementation of ICA. Verification of a physical artifact such as a circuit or a program is not absolute in the way that verification of an abstraction such as an algorithm may be. For example, the digital circuits that implement our ICA circuit may behave differently than we assume, or the specification to which we verify our circuit may differ from that assumed by the ICA implementation that will use it. These difficulties attend any use of applied mathematics in engineering: the underlying model may be incorrect, and the requirements to be satisfied may be mis-

understood [2]. Formal verification does not make these difficulties any more acute than informal verification, nor are they solved by avoiding all intellectual scrutiny of our designs. Formal verification does, however, make explicit the specification whose satisfaction is verified, and the assumptions on which the verification rests. It therefore identifies those properties whose satisfaction, or utility, in the physical world must be established empirically.

Critical systems such as DFCS require the highest degree of human skill and responsibility in their design, analysis and certification. The stochastic behavior of ultra-reliable fault-tolerant real-time systems such as DFCS cannot be fully validated by purely empirical means; intellectual scrutiny and mathematical analysis of the design are required as well. We hope to have shown that formal verification is a tool that can provide practical assistance in discharging some of these responsibilities.

Acknowledgments

EHDM is the collective work of many people; over the last few years, much of the effort and creativity have been provided by our colleagues Sam Owre and Natarajan Shankar. Ricky Butler of NASA Langley Research Center provided valuable encouragement and guidance in our study of fault-tolerant algorithms.

References

[1] Anonymous. Reprogramming capability proves key to extending Voyager 2's journey. *Aviation Week and Space Technology*, page 72, August 7, 1989.

[2] Jon Barwise. Mathematical proofs of computer system correctness. *Notices of the AMS*, 36:844–851, September 1989.

[3] Michael J. Beeson. Towards a computation system based on set theory. *Theoretical Computer Science*, 60:297–340, 1988.

[4] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, New York, NY, 1988.

[5] Ricky W. Butler, James L. Caldwell, and Ben L. Di Vito. Design strategy for a formally verified reliable computing platform. In *COMPASS '91 (Proceedings of the Sixth Annual Conference on Computer Assurance)*, pages 125–133, Gaithersburg, MD, June 1991. IEEE Washington Section.

[6] J. H. Cheng and C. B. Jones. On the usability of logics which handle partial functions. In Carroll Morgan and J. C. P. Woodcock, editors, *Proceedings of the Third Refinement Workshop*, pages 51–69. Springer-Verlag Workshops in Computing, 1990.

[7] *Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[8] Ben L. Di Vito, Ricky W. Butler, and James L. Caldwell. High level design proof of a reliable computing platform. In J. F. Meyer and R. D. Schlichting, editors, *Dependable Computing for Critical Applications—2*, volume 6 of *Dependable Computing and Fault-Tolerant Systems*, pages 279–306, Tucson, AZ, February 1991. Springer-Verlag, Wien, New York.

[9] *System Design Analysis*. Federal Aviation Administration, September 7, 1982. Advisory Circular 25.1309-1.

[10] John R. Garman. The "bug" heard 'round the world. *ACM Software Engineering Notes*, 6(5):3–10, October 1981.

[11] Richard E. Harper and Jaynarayan H. Lala. Fault-tolerant parallel processor. *AIAA Journal of Guidance, Control, and Dynamics*, 14(3):554–563, May–June 1991.

[12] Ian Hayes, editor. *Specification Case Studies*. Prentice-Hall International Ltd., Hemel Hempstead, UK, 1987.

[13] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37(4):398–405, April 1988.

[14] Israel Kleiner. Rigor and proof in mathematics: A historical perspective. *Mathematics Magazine*, 64(5):291–314, December 1991. Published by the Mathematical Association of America.

[15] Hermann Kopetz et al. Distributed fault-tolerant real-time systems: The Mars approach. *IEEE Micro*, 9(1):25–40, February 1989.

[16] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press, Cambridge, England, 1976.

[17] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[18] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM TOPLAS*, 4(3):382–401, July 1982.

[19] Dale A. Mackall. AFTI/F-16 digital flight control system experience. In Gary P. Beasley, editor, *NASA Aircraft Controls Research 1983*, pages 469–487. NASA Conference Publication 2296, 1984. Proceedings of workshop held at NASA Langley Research Center, October 25–27, 1983.

[20] Dale A. Mackall. Development and flight test experiences with a flight-crucial digital control system. NASA Technical Paper 2857, NASA Ames Research Center, Dryden Flight Research Facility, Edwards, CA, 1988.

[21] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, number 607 in Lecture Notes in Artificial Intelligence, pages 748–752, Saratoga, NY, 1992. Springer Verlag.

[22] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[23] Didier Puyplat. A320: First of the computer-age aircraft. *Aerospace America*, 29(5):28–30, May 1991.

[24] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.

[25] John Rushby. Formal specification and verification of a fault-masking and transient-recovery model for digital flight-control systems. In Vytopil [34], pages 237–257.

[26] John Rushby. Formal verification of an Oral Messages algorithm for interactive consistency. Technical Report SRI-CSL-92-1, Computer Science Laboratory, SRI International, Menlo Park, CA, June 1992.

[27] John Rushby and Friedrich von Henke. Formal verification of the Interactive Convergence clock synchronization algorithm using EHDM. Technical Report SRI-CSL-89-3R, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1989 (Revised August 1991). Original version also available as NASA Contractor Report 4239.

[28] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[29] Fred B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Department of Computer Science, Cornell University, Ithaca, NY, August 1987.

[30] Natarajan Shankar. Mechanical verification of a generalized protocol for Byzantine fault-tolerant clock synchronization. In Vytopil [34], pages 217–236.

[31] Robert E. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

[32] J. M. Spivey, editor. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, Hemel Hempstead, UK, 1989.

[33] *Interim Defence Standard 00-55: The procurement of safety critical software in defence equipment.* UK Ministry of Defence, April 1991. Part 1, Issue 1: Requirements; Part 2, Issue 1: Guidance.

[34] J. Vytopil, editor. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, Nijmegen, The Netherlands, January 1992. Springer Verlag.

[35] John H. Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

[36] William D. Young. Verifying the Interactive Convergence clock-synchronization algorithm using the Boyer-Moore prover. NASA Contractor Report 189649, NASA Langley Research Center, Hampton, VA, April 1992. (Work performed by Computational Logic Incorporated).