The Proof of Correctness of a Fault-Tolerant Circuit Design

William R. Bevier
William D. Young

Technical Report 57
August 1990

Computational Logic Inc.
1717 W. 6th St. Suite 290
Austin, Texas 78703
(512) 322-9951

This work was sponsored in part at Computational Logic, Inc. by National Aeronautics and Space Administration Langley Research Center (NAS1-18878). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., NASA Langley Research Center or the U.S. Government.

Abstract

We describe a formally verified implementation of the ‘‘Oral Messages’’ algorithm of Pease, Shostak, and Lamport [4, 5]. An abstract implementation of the algorithm has been verified to achieve interactive consistency in the presence of faults [1]. This abstract characterization is then mapped down to a hardware level implementation which inherits the fault-tolerant characteristics of the abstract version. The proof that the hardware level description is a correct implementation of the ‘‘Oral Messages’’ algorithm has been fully checked with a mechanical theorem prover. A significant result of this work is the demonstration of a fault-tolerant device that is formally specified and whose implementation is proved correct with respect to this specification.

Key words. Fault tolerance, mechanical theorem proving, program verification, specification.

1. Introduction

A key problem facing the designers of systems which attempt to ensure fault tolerance by redundant processing is how to guarantee that the processors reach agreement, even when one or more processing units are faulty. Pease, Shostak and Lamport [5, 4] have devised the notion of *interactive consistency*, which formally characterizes what it means for non-faulty processors to reach agreement. They prove that, given certain assumptions about the type of inter-process communication, interactive consistency can be achieved if and only if the total number of processors exceeds three times the number of faulty processors. They also provide a clever algorithm which achieves interactive consistency.

Our goal was a verified hardware implementation of the ‘‘Oral Messages’’ (*OM*) algorithm of Pease, Shostak, and Lamport. Our approach to achieving this was to proceed in several phases. In the first phase, we defined an ‘‘abstract implementation’’ of the algorithm in the Boyer-Moore logic and proved that this high-level formalization achieved interactive consistency. We then defined a low-level characterization of the algorithm and proved that our low-level version is a correct implementation of our high-level version. As a consequence of this proof, we are guaranteed that our low-level implementation achieves interactive consistency.

An earlier paper [1] reports on our work to describe our abstract implementation of the algorithm in the Boyer-Moore logic. We stated the interactive consistency conditions in the logic, and used the Boyer-Moore theorem prover to check a proof that our formalization of *OM* satisfies these conditions. We also mechanically checked the result presented by Lamport, Shostak and Pease that *OM* provides an optimal solution: no algorithm exists which achieves interactive consistency via an exchange of oral messages if the number of faulty processors is at least one third of the total.

In this paper we present the design of a hardware implementation of *OM*(1) — the instance of *OM* which tolerates one faulty process when there are at least three non-faulty processes. We have mechanically checked the proof that the hardware design implements *OM*(1), and therefore achieves interactive consistency.

The paper is organized as follows. The following section describes our formal specification of the Oral Messages algorithm and its correctness properties. Section 3 describes our implementation of the algorithm and Section 4 sketches the proof that our implementation satisfies its specification. Finally, Section 5 gives some of our conclusions and observations on this development process.

2. The Specification

2.1 Interactive Consistency & The Function *OM*

The problem addressed by the interactive consistency algorithm is the following: given a number of communicating processors, how can they arrive at a consistent view of the system if there are faulty processors among them which potentially send conflicting information to different parts of the system. Lamport, Shostak, and Pease [4] describe the problem in terms of the metaphor of Byzantine Generals attempting to arrive at a common battle plan through an exchange of messages. One or more of the generals may be traitorous and attempt to thwart the loyal generals by preventing them from reaching agreement.

It is straightforward to state the problem in terms of a single commanding general communicating with a number of lieutenant generals. In this case we desire an algorithm which guarantees the following.

> A commanding general must send an order to his n - 1 lieutenant generals such that
>
> IC1. All loyal lieutenants obey the same order;
>
> IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
>
> Conditions IC1 and IC2 are called the *interactive consistency* conditions. [4]

The interactive consistency conditions can be formalized fairly straightforwardly. Let n be some number of processes. Let $\bar{v}$ be a vector of length n, where $\bar{v}[i]$ is the local value of process i, $i \in \{0, ..., n-1\}$. Let L be the set $\{0, ..., n-1\}$ of indices into $\bar{v}$. This serves as the set of process names.

We assume a predicate $faulty : L \to \{T, F\}$ that identifies a potentially faulty process. A faulty process may or may not forward a message correctly. We also assume a function $faults : 2^L \to \mathbf{N}$ that counts the number of potentially faulty processes in a set of process names. The functions *faulty* and *faults* serve only in the specification of the problem; they are not computable.

Let g (the general) be a member of L and $\bar{w}_g$ be a vector of length n, where each entry $\bar{w}_g[i]$ is the value which process i concludes is process g's local value. $\bar{w}_g$ satisfies the interactive consistency conditions if for each $i, j \in L - \{g\}$ we have the following.

IC1. $\neg\, faulty(i)\ \&\ \neg\, faulty(j) \ \to\ \bar{w}_g[i] = \bar{w}_g[j]$

IC2. $\neg\, faulty(g)\ \&\ \neg\, faulty(i) \ \to\ \bar{w}_g[i] = \bar{v}[g]$

We have formalized in the Boyer-Moore logic a version of the Oral Messages algorithm of Pease, Shostak, and Lamport [5]. This algorithm, and our (abstract) implementation of it in the form of a function *OM* in the Boyer-Moore logic, produces a vector $\overline{w}$ which satisfies the interactive consistency conditions. The vector is computed after some number of rounds of *information exchange* among the processes.

Our Boyer-Moore function *OM* takes four parameters: n, the number of processes; g, the name of the general; $x = \overline{v}[g]$, the general's local value; and m, an integer which determines the number of rounds of information exchange to take place. Lamport, Shostak and Pease [4] prove that *OM* is guaranteed to achieve interactive consistency only if n is greater than three times the number of faulty processes. The number of rounds of information exchange m must be at least the number of faulty processes.

An accompanying paper [1] presents our formal definition of *OM*, and describes our mechanically checked proof that our formalization of the algorithm produces a vector which satisfies the interactive consistency conditions. Our formal statements of the two theorems that *OM* satisfies IC1 and IC2, respectively, are given below. We believe that these are an intuitive and straightforward formalization of the interactive consistency conditions.

$$
\begin{array}{ll}
 & \neg\, faulty(i) \\
 & \&\ \neg\, faulty(j) \\
 & \&\ 3 \cdot faults(L) < n \\
 & \&\ faults(L) \le m \\
\rightarrow & \\
 & OM(n, g, x, m)[i] = OM(n, g, x, m)[j],
\end{array}
$$

$$
\begin{array}{ll}
 & \neg\, faulty(g) \\
 & \&\ \neg\, faulty(i) \\
 & \&\ 3 \cdot faults(L) < n \\
 & \&\ faults(L) \le m \\
\rightarrow & \\
 & OM(n, g, x, m)[i] = x
\end{array}
$$

2.2 Multiple Applications of OM

To reach agreement, each process among a set of processes must act in turn as the general in an application of *OM*. We define the function *OML* recursively to apply *OM* to each member of a list of process names.[1]

---

[1]The notation $|\overline{v}|$ denotes the length of vector $\overline{v}$, which in this case gives the number of processes n. The notation $\overline{v}\,|_{x}^{i}$ denotes a vector with the following property.

$$\overline{v}\,|_{x}^{i}[j] \equiv if\ j = i\ then\ x\ else\ \overline{v}[j]$$

$OML(nil, \bar{v}, m) \;\equiv\; nil$

$OML(cons(g, l), \bar{v}, m) \;\equiv\; cons(OM(|\bar{v}|, g, \bar{v}[g], m) \mid^{g}_{\bar{v}[g]},\; OML(l, \bar{v}, m))$

The expression

$OM(|\bar{v}|, g, \bar{v}[g], m) \mid^{g}_{\bar{v}[g]}$

in this definition denotes the vector $\bar{w}$ in which, for each $i \neq g \in L$, $\bar{w}[i]$ records the value which process i concludes is process g's local value, after m rounds of information exchange. The value of $\bar{w}[g]$ is $\bar{v}[g]$, g's local value.

OML produces an $n \times n$ matrix in which the i^{th} row is a vector $\bar{w}_i$ of values such that $\bar{w}_i[j]$ is j's guess at i's local value. The i^{th} column of the matrix is the *interactive consistency vector* for process i. This vector contains the values which process i concludes is the local value for each process. $OML(L, \bar{v}, m)[i, g]$ is the value which process i concludes is process g's local value.

We can derive the following two facts about *OML* as a result of the interactive consistency conditions proved of *OM*.

1. In the matrix value of *OML*, any two non-faulty processes agree on the local value of all other processes.
2. Each non-faulty process has the correct value for a non-faulty general.

These facts correspond to IC1 and IC2, respectively. The formal versions are displayed below.

$$\begin{aligned}
&\quad\neg\, faulty(i)\\
&\& \neg\, faulty(j)\\
&\&\ 3 \cdot faults(L) < n\\
&\&\ faults(L) \leq m\\
\rightarrow&\\
&OML(L, \bar{v}, m)[i, g] = OML(L, \bar{v}, m)[j, g]
\end{aligned}$$

$$\begin{aligned}
&\quad\neg\, faulty(g)\\
&\& \neg\, faulty(i)\\
&\&\ 3 \cdot faults(L) < n\\
&\&\ faults(L) \leq m\\
\rightarrow&\\
&OML(L, \bar{v}, m)[i, g] \quad = \quad \bar{v}[g]
\end{aligned}$$

From these two properties, we can prove that two non-faulty processes have identical interactive consistency vectors. That is,

$$
\begin{aligned}
& \neg\, faulty(i) \\
& \&\ \neg\, faulty(j) \\
& \&\ 3 \cdot faults(L) < n \\
& \&\ faults(L) \leq m \\
\rightarrow\ & \\
& OML(L, \bar{v}, m)[i] = OML(L, \bar{v}, m)[j].
\end{aligned}
$$

2.3 Traces of *OM* Applications

The function *OML* formally describes a single instance of n processes reaching agreement through m rounds of information interchange. This formalization is not conducive to mapping down to a lower-level implementation which executes the algorithm in a number of ‘‘steps.’’ Therefore, we define a *trace function* O^* to model n processes attempting to reach agreement through time. The input to O^* is sequence of n-tuples of sensed values where element $\overline{s}$ of the sequence is a vector in which the i^{th} element represents the input to process i. O^* produces a sequence of output vectors. Each element $\overline{o}$ of the output sequence is a vector in which the i^{th} element represents the output from process i.

At each step, the trace function applies a *step function* O. The input to O is one of the input n-tuples $\overline{s}$, and its output is one of the vectors $\overline{o}$. The function O involves an application of *OML*, and of a *filter function* which computes an output value based on an interactive consistency vector. An example of such a filter function is *Majority*. O is defined by the following formula. For $j \in L = \{0, ..., n-1\}$,

$$O(\overline{s}, m)[j] \;\equiv\; filter(OML(L, \overline{s}, m)[j])$$

The trace function can be written as follows.

$$O^*(nil, m) \;\equiv\; nil$$

$$O^*(cons(\overline{s}, l), m) \;\equiv\; cons(O(\overline{s}, m), O^*(l, m))$$

An elementary theorem about O^* is that for an input sequence Σ, $O^*(\Sigma, m)[i] \;=\; O(\Sigma[i], m)$.

The Byzantine properties of *OML* are provably inherited by the trace version. In particular, we can prove that, given a sufficiently small number of faulty processes, two non-faulty processes always agree on their outputs. That is, for a trace of input n-tuples l, and for index k into that trace,

$$
\begin{aligned}
& \neg\, faulty(i) \\
& \&\ \neg\, faulty(j) \\
& \&\ 3 \cdot faults(L) < n \\
& \&\ faults(L) \leq m \\
\rightarrow\ & \\
& O^*(l, m)[k][i] \quad = \quad O^*(l, m)[k][j].
\end{aligned}
$$

This conclusion follows from the fact that processes i and j have identical interactive consistency vectors, and therefore *filter* must produce the same value for both processes.

Instantiating this trace function with $n = 4$ and $m = 1$ gives us a specification for a system of four redundant processes that achieve Byzantine agreement, and which can tolerate up to one faulty process. The architecure of this system is illustrated in Figure 1.

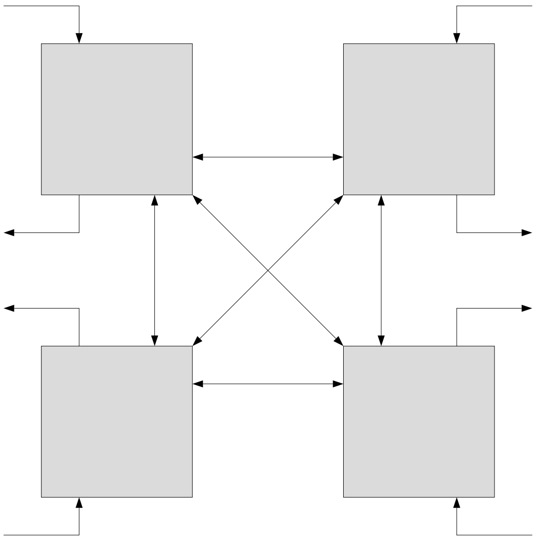

Figure 1: Four Redundant Processes

3. The Implementation

Implementing of our circuit entails describing the internal logic of each of the four processes represented by the boxes in Figure 1. These processes achieve agreement after exchanging messages. A goal of our design was for the four processes to be identical; this goal was achieved.

Each process has five inputs: an sensor value, clock, and data lines from each of the other three processes. Additionally, each process has four outputs: an actuator and data lines out to each of the other processes. These inputs and outputs are listed below. In our formal description of the circuit the widths of these data paths are not fixed. This leaves the implementor free to choose a data width.

- **sense**. A sensed value.
- **clock**. A clock waveform.
- **data_in**. Inputs from the three other processes.

- **data_out**. Outputs to the three other processes.
- **actuator**. Output to some actuator.

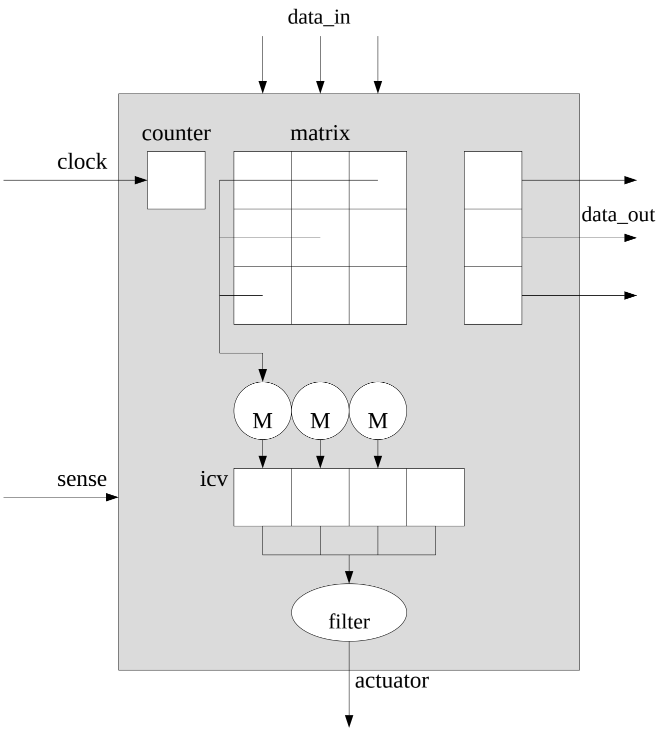

Figure 2: The Internal State of a Process

Figure 2 shows the *internal* state of a single process, along with some of the internal data paths. The internal state of a process contains the following components.

- **counter**. A 3-bit counter, used to cycle a process through 8 steps.
- **matrix**. A 3×3 matrix of data used to store values received during the information exchange.
- **icv**. The 1×4 interactive consistency vector for this process. `ICV[3]` holds the process's local value, derived from the sense input.

The inter-connection of the processes to accomplish information exchange is depicted in Figure 3. Each arrow represents one-way communication. For each $i \in \{0, 1, 2, 3\}$, and $j \in \{0, 1, 2\}$, `data_in`[j] for process i is connected to `data_out`[$2 - j$] of process $(i + j)$ *mod* 4. The interconnection scheme is designed to assure that all of the processes are identical.[2]

Each process cycles through the 8 steps described in Figure 4. The purpose of each step is described below. The steps are numbered by the value of the 3-bit counter. The four processes share the clock input and hence perform these steps synchronously.

0. Read the sensed input. Save this as the process's local value in `ICV[3]`. Also, place this value on the output lines to the other three processes. This begins the report of each process's local value to all of the other processes.
1. Receive the local values of the other three processes, and store them in row `0` of the matrix.

2,3. Fill the remaining rows of each matrix with the reports of each process's value. In steps 2 and 3 each process receives two values from each of the other three processes. At the end of step 3, the information exchange required for the four instances of *OM*(1) is complete, as depicted in Figure 5.

4. Compute the interactive consistency vector. This is accomplished by computing the majority of the three reported values for each of the other processes. (The circle labeled M in Figure 2 represents a 3-input majority circuit.)
5. Compute the actuator output based on the value of the interactive consistency vector. This is represented by a call to a function *filter*. In our specification *filter* is not defined, but is constrained to be invariant under rotation of its argument

6,7. No state change other than incrementing the counter.

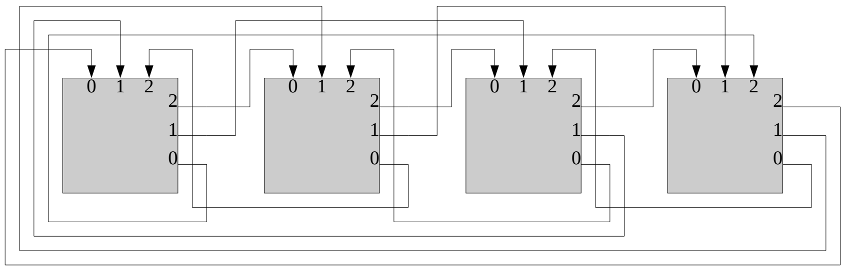

Figure 3: Process Interconnections

The behavior of this circuit can be summarized as follows. Each process senses its input simultaneously, goes through three steps of information exchange, determines an interactive consistency vector, and then produces an actuator value 5 steps after the input was sensed. The actuator value is fixed until a new

[2]A result is that the interactive consistency vectors computed by two non-faulty processes are not actually identical, but are, in fact, rotations of one another. This implies that the filter function defined on the interactive consistency vector must be invariant under rotations of its vector argument.

actuator value is computed on the next cycle.

```
Case Counter:

0: data_out[i] ← sense, i ∈ {0, 1, 2}
   icv[3]      ← sense
   clock       ← clock+1

1: matrix[0,i] ← input[i], i ∈ {0, 1, 2}
   data_out[0] ← input[1]
   data_out[1] ← input[0]
   data_out[2] ← input[0]
   clock       ← clock+1

2: matrix[1,i] ← input[i], i ∈ {0, 1, 2}
   data_out[0] ← matrix[0,2]
   data_out[1] ← matrix[0,2]
   data_out[2] ← matrix[0,1]
   clock       ← clock+1

3: matrix[2,i] ← input[i], i ∈ {0, 1, 2}
   clock       ← clock+1

4: icv[0] ← majority(matrix[0,0], matrix[1,2], matrix[2,1])
   icv[1] ← majority(matrix[0,1], matrix[1,0], matrix[2,2])
   icv[2] ← majority(matrix[0,2], matrix[1,1], matrix[2,0])
   clock  ← clock+1

5: Actuator    ← filter(icv)
   clock       ← clock+1

6: clock       ← clock+1

7: clock       ← clock+1
```

Figure 4: Process Steps

4. The Proof of Correctness

We want to be able to assert about our circuit design that the actuator values for all non-faulty processes agree, even in the presence of a single faulty process. How can we convince ourselves that this is true? Our design is fairly simple, but has some tricky details where mistakes can easily be made.

To convince ourselves of this assertion we have proved the design correct, and mechanically checked the proof with the Boyer-Moore theorem prover [2, 3]. The specification and proof of correctness consists of the following elements.

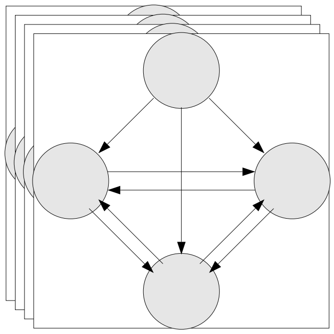

Figure 5: Rounds of Information Exchange for *OM(1)*

- **Correctness of the function** ***OM*****.** The function *OM* is defined in the Boyer-Moore logic, and is proved to satisfy interactive consistency conditions.
- **The Circuit Specification Function.** The trace function O^* described in Section 2 of this paper is defined in the logic. This function includes a call to *OM* to perform the information exchange. Because *OM* achieves interactive consistency, it is possible to prove that at any point in the trace all non-faulty processes agree if there are a sufficient number of non-faulty processes. The instance of this trace function with $n = 4$ and $m = 1$ serves as a specification function for our circuit design.
- **The Circuit Implementation Function.** The design of the circuit is formalized in the Boyer-Moore logic. A function named *LOCAL–STEP* is defined to formally express the state changes to a single process as described in Figure 4. A function *GLOBAL–STEP* applies *LOCAL–STEP* on each clock tick to each of the four processes. *GLOBAL–STEP* also formally describes the data flow among processes. A trace function C^* uses *GLOBAL–STEP* as its step function. C^* is proved to ‘‘correspond’’ to the function O^* (defined in Section 2). Describing the sense in which these functions correspond is the purpose of the remainder of this section.

The trace function C^* is defined as follows. Let $\overline{st}$ be a vector of four process states, and let G^* be the trace of these states. Each $G^*[k]$ contains a *4*-tuple with the state of each process after k applications of *GLOBAL–STEP*.

$$G^*(nil, \overline{st}) \equiv nil$$

$$G^*(cons(\overline{s}, l), \overline{st}) \equiv cons(\overline{st}, G^*(l, global\text{–}step(\overline{s}, \overline{st})))$$

The trace function C^* is defined in terms of G^* by projecting out of the state of each process the value of its actuator at each step of the trace. The relationship between G^* and C^* is depicted in Figure 6. For $i \in \{0, 1, 2, 3\}$ and for k an index into the trace,

$$C^*(l, \overline{st})[k][i] = actuator(G^*(l, \overline{st})[k])[i].$$

The time granularity of C^* is greater than that of O^*. It takes C^* five clock ticks to compute actuator values in response to a set of sense inputs. The intermediate steps of the trace are not of interest in the statement of interactive consistency. It is useful to define the notion of trace *selection*. The *n–selection* of a trace t is the trace consisting of successive $(n-1)^{st}$ elements of t.[3].

$$Select(n, t) \equiv if\ |t| \geq n\ then\ cons(t[n-1], Select(n, NthCdr(t, n)))\ else\ nil$$

The proof of correctness of the circuit design requires the proof that some selection on C^* equals the trace O^*. We have chosen $n = 7$ as the selector value in our proof. The following theorem formally relates the behavior of the circuit design to the specification function. Figure 6 depicts this relationship proved between C^* and O^*.

$$Select(7, C^*(l, \overline{st})) = O^*(l, 1)$$

Recall the following fact about O^* (discussed in Section 2.3) which says that two non-faulty processes agree on their outputs. For a trace of input n-tuples l, and for index k into that trace,

$$\begin{array}{l} \quad \neg\, faulty(i) \\ \&\ \neg\, faulty(j) \\ \&\ 3 \cdot faults(L) < n \\ \&\ faults(L) \leq m \\ \rightarrow \\ O^*(l, m)[k][i] \quad = \quad O^*(l, m)[k][j]. \end{array}$$

Substituting $Select(7, C^*(l, \overline{st}))$ into this lemma, with $n = 4$, $m = 1$ and $L = \{0, 1, 2, 3\}$ gives a theorem which says that the circuit design, as defined by C^*, achieves agreement every 7th ‘‘tick’’ of the clock.

$$\begin{array}{l} \quad \neg\, faulty(i) \\ \&\ \neg\, faulty(j) \\ \&\ 3 \cdot faults(L) < n \\ \&\ faults(L) \leq m \\ \rightarrow \\ Select(7, C^*(l, \overline{st}))[k][i] \quad = \quad Select(7, C^*(l, \overline{st}))[k][j]. \end{array}$$

We take the proof of this theorem as a satisfactory formal demonstration of the correctness of the circuit design.

[3]The function $NthCdr(t, n)$ returns the list of length $|l| - n$, where $NthCdr(l,n)[i] = l[i+n]$

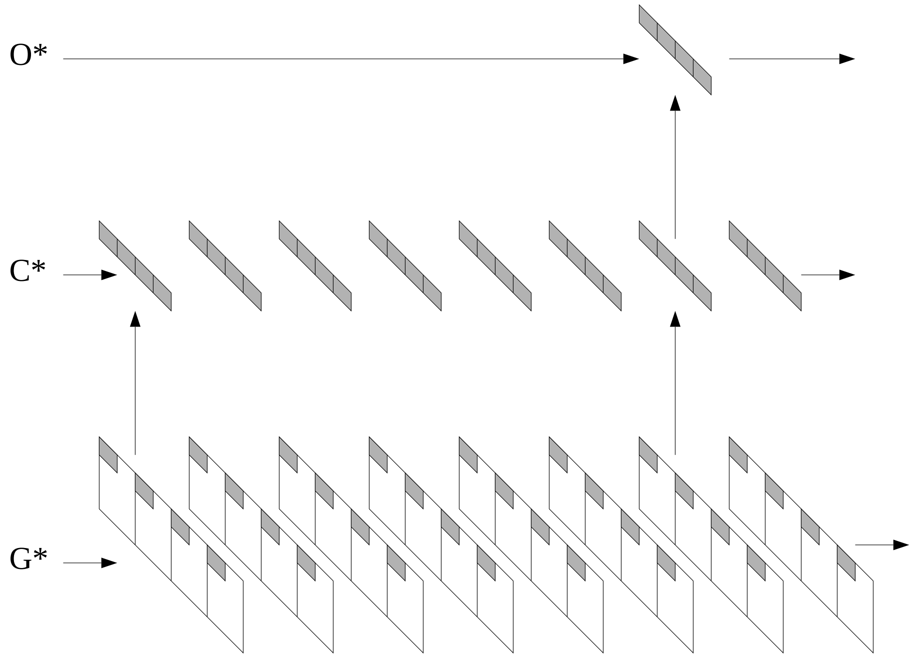

Figure 6: Correspondence among Trace Functions

5. Conclusion

We have verified a low-level hardware implementation of the Oral Messages algorithm of Pease, Shostak, and Lamport using a high-level abstract implementation as its specification. Because this abstract implementation has been formally proven to achieve interactive consistency, we are assured that our low-level implementation is fault-tolerant as well.

The main achievement of this work is the demonstration of a fault-tolerant device that can be formally specified, and whose implementation can be proved correct. We have shown how to formally relate an abstract algorithm like *OM* to a design which is implementable in hardware.

The main limitation of our device specification is that it does not explicitly account for distributed processes. Processes are described as operating synchronously. This simplifies the problem dramatically. Addressing this limitation is a future goal of our work.

All of the proofs, from the proof of correctness of the general Oral Messages Algorithm to the proof of the hardware implementation were fully machine checked. Proponents of the view that such fully formal and machine checked proofs do not contribute materially to mathematics or engineering may feel that our effort was superfluous.

From a mathematical perspective, we believe that two important goals of proof are to increase one’s understanding and intuition about the content and significance of a theorem, and to provide a convincing argument that it is, in fact, valid. Our proof efforts led us to develop a very clean and unambiguous statement of the algorithm and its correctness properties. We believe that we understand this quite subtle algorithm and the reason it works much better for the effort. Moreover, our success in convincing a congenitally skeptical mechanical proof checker of the validity of this theorem practically guarantees that we have eliminated any errors which the much touted ‘‘social process’’ might overlook. Such confidence is particularly comforting in domains such as fault-tolerant and real-time computing where a well-developed intuition is difficult to cultivate; the theorem prover is not subject to being misled by the urgings of a misguided or ill-informed intuition.

From an engineering perspective, we feel that our approach has several benefits. By proving properties such as the interactive consistency conditions with respect to our high-level abstract implementation, we retain the clarity and abstractness of the published algorithm and benefit from the intuitions derived from the published proof. By then mapping down to a more concrete characterization, but one which provably retains the fault-tolerant characteristics of the abstract version, we are able to derive a hardware level characterization of the algorithm which is trivial to implement. We suspect that an attempt to implement the Oral Messages algorithm directly from the published abstract presentation would be extremely error-prone.

References

1. W.R. Bevier and W.D. Young. Machine Checked Proofs of a Byzantine Agreement Algorithm. Technical Report 55, Computational Logic, Inc., June, 1990.

2. R. S. Boyer and J S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

3. R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Academic Press, Boston, 1988.

4. {Leslie Lamport, Robert Shostak, and Marshall Pease
. "The Byzantine Generals Problem". *ACM TOPLAS 4*, 3 (July 1982), 382-401.}

5. {Marshall Pease, Robert Shostak, and Leslie Lamport
. "Reaching Agreement in the Presence of Faults". *JACM 27*, 2 (April 1980), 228-234.}

Table of Contents

List of Figures