Exception handling in replicated systems with voting

L. Mancini
S.K. Shrivastava

Abstract

In a concurrent processing environment consisting of N-Modular Redundant (NMR) nodes communicating by message passing, exceptions which require detection and treatment to prevent subsequent failures are identified. These exceptions include 'sequence exceptions' caused by 'out of order' message processing and 'hardware exceptions' caused by processor failures. In particular, the paper describes standard and exceptional domains for majority voters and presents implementation strategies for the detection and handling of the discussed exceptions.

Series Editor: M. J. Elphick

© 1986 University of Newcastle upon Tyne. Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

MANCINI, Luigi

Exception handling in replicated systems with voting.

Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Laboratory, 1986.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 217.)

Abstract

In a concurrent processing environment consisting of N-Modular Redundant (NMR) nodes communicating by message passing, exceptions which require detection and treatment to prevent subsequent failures are identified. These exceptions include 'sequence exceptions' caused by 'out of order' message processing and 'hardware exceptions' caused by processor failures. In particular, the paper describes standard and exceptional domains for majority voters and presents implementation strategies for the detection and handling of the discussed exceptions.

About the author

Luigi Mancini has been at the Computing Laboratory since May, 1985 as a Research Associate.

Dr. Shrivastava joined the Computing Laboratory in August, 1975, where he is a lecturer.

Suggested keywords

DISTRIBUTED PROCESSING
EXCEPTION HANDLING
MAJORITY VOTING
RELIABILITY
REPLICATED PROCESSING

Suggested classmarks (primary classmark underlined)

Dewey (18th): 001.64404
U. D. C. 519.687
Exception Handling in Replicated Systems with voting

L. Mancini and S.K. Shrivastava
Computing Laboratory
The University
Newcastle upon Tyne, NE1 7RU, England.

Abstract
In a concurrent processing environment consisting of N-Modular Redundant (NMR) nodes communicating by message passing, exceptions which require detection and treatment to prevent subsequent failures are identified. These exceptions include 'sequence exceptions' caused by 'out of order' message processing and 'hardware exceptions' caused by processor failures. In particular, the paper describes standard and exceptional domains for majority voters and presents implementation strategies for the detection and handling of the discussed exceptions.

Index Terms - majority voting, replicated processing, distributed processing, exception handling, reliability.

1. Introduction

Modular redundancy in the form of replication of processing modules with majority voting is one of the best known techniques for tolerating failures of processing modules. In the present paper we will explore the application of this form of redundancy to distributed systems and study one particular aspect - that of exception handling. We will consider our system to be composed out of a number of nodes fully connected by means of redundant communication channels. A node will represent a functional processing module, composed out of a number of processors and voters in a classical NMR (N-Modular Redundant) configuration. For such a distributed system, it will be shown that the 'traditional' majority voting algorithm is not adequate and must be supplemented by exception handling algorithms to prevent subsequent failures. The provision of exception detection and its treatment is the main subject of the paper.

The paper is structured as follows. In Section 2 we present a distributed system architecture for replicated processing and discuss why exception handling is necessary to prevent failures. In Section 3 we present the basic assumptions made in our design, next we discuss principles of exception handling and apply those principles. In Section 5, we discuss algorithms for the handling of exceptions in voters. Section 6 contains methods of treating those exceptions in NMR nodes. Conclusions from our study are presented in the last section.

2. Distributed replicated processing

The architecture of the system under consideration is shown in Fig. 1, where the degree of replication shown per node is three, thus giving us the well known triple modular redundant (TMR) nodes. A processing module within a node consists of a voter-processor combination. In particular, the voting function \( V_K \) is performed by software that runs on processor \( P_K \). The system consists of a number of NMR nodes communicating only by message passing. The function of a voter \( V_K \) of a node \( N_i \) is to perform majority voting on incoming messages: messages that have been voted are queued for processing in the voted message queue (VMQ) \( K \). The function of a processor \( P_K \) of node \( N_i \) is to pick a message from VMQ \( K \) for processing and after processing, transmit n copies of the result to the next NMR node (n=3 in Fig. 1).

![Fig. 1. System structure](image)

In the figure shown, a failure of a single voter-processor combination in \( N_i \) can be masked by non-faulty voters of node \( N_j \) provided that: the non-faulty processors of node \( N_i \) process identical messages in identical order and result messages from non-faulty processors of node \( N_i \) arrive uncorrupted at \( N_j \). Assuming that the only communication paths are those implied by the figure, no assumption about the behaviour of a failed component need be made. For the case just considered, arbitrary behaviour (a commission failure [4]) by the failed voter-processor combination can be tolerated.

It should be clear that to tolerate a maximum number of component failures within a node, the following sequencing condition must be met:

SEQ: All non-faulty processors of an NMR node process voted messages in an identical order.

It is assumed that each processor maintains some state information which affects the subsequent processing. For example, if some non-faulty processor \( P_K \) of \( N_i \) processes messages \( m_1 \) followed by \( m_2 \) and some other non-faulty processor \( P_Q \) of \( N_i \) processes \( m_3 \) followed by \( m_2 \), then results obtained by \( P_K \) and \( P_Q \) for message \( m_2 \) need not to be identical.

Violation of the sequencing condition in an NMR node will be termed a sequencing failure.
The above sequencing condition is particularly hard to meet in a concurrent processing environment, as can be appreciated by considering a simple example. Let $N_k$ be an NMR node which can receive results from two different nodes $N_j$ and $N_i$ (Fig. 2). Suppose that $N_j$ and $N_i$ send their result messages at about the same time to $N_k$, and that messages can experience variable delays during transmission.

![Fig. 2. NMR node $N_k$ receives messages from nodes $N_j$, $N_i$.](image)

It is thus possible that voters of $N_k$ receive messages in different order: this can cause voted messages to be queued in different order in VMQs.

One possible way of preventing sequencing failures is by employing an atomic message broadcast facility as discussed in [2]. If all non-faulty processors use this facility for broadcasting their messages to their receivers, then all non-faulty voters of a node are guaranteed to receive messages in an identical order, thereby preventing the possibility of sequencing failures. In the absence of an atomic broadcast facility, we believe that there can be three different approaches for meeting the condition SEQ.

(i) Specific scheduling algorithms are designed to ensure that non-faulty voters of an NMR node insert voted messages in their VMQs in an identical order [3].

(ii) Non-faulty processors of an NMR node periodically execute a Byzantine agreement protocol [5] to ensure that VMQs are identical [6]. (This approach turns out to be an optimised version of the atomic broadcast based approach mentioned earlier which also employs a Byzantine agreement protocol).

(iii) Occurrence of sequencing failures are detected as exceptions and specific exception handlers are provided for recovering from such failures.

We draw the reader's attention to papers [3, 6] where algorithms are presented for the first two approaches. They have one feature in common: they prevent the occurrence of sequencing failures. In the third approach, sequencing failures are permitted but there is a provision for their detection and recovery.

To illustrate that recovery from such failures is desirable, consider the following specific case for TMR nodes (Fig. 1). Assume that for node $N_3$, VMQ1 and VMQ2 are identical with messages $m_2$ and $m_3$ (at the head), and that VMQ3 contains messages $m_2$ and $m_3$ (at the head). So P1 and P2 process $m_1$, while P3 will process $m_3$, thus causing a sequencing failure. The subsequent processing of processor P3 will in general be incorrect and P3 will act as a faulty processor. Thus node $N_3$ will not be able to tolerate any additional failures. However, if the sequencing failure is detected and then VMQ3 is reordered to be identical to the queues of P1 and P2, the capability of tolerating module failures can be reacquired.

If one assumes that in a given system the probability of sequencing failures is quite low, then the third approach becomes quite attractive. (The analogy with the treatment of deadlocks in an operating system is particularly instructive: if deadlocks occur rarely, prevention is often less attractive than detection with recovery). It might even be possible to design a system based on approach (i) or (ii) so as to reduce the probability of sequencing failures and then use approach (iii) for coping with them. Thus treating sequencing failures as exceptions appears to be a good strategy.

Another attractive reason for the introduction of exception handling is to pass valuable information to those parts of the system that are responsible for dynamically reconfiguring the system. For example, suppose a non-faulty voter detects a disagreement, and that it is known that the node whose results are being voted has not suffered a sequencing failure. Then, one can deduce that there is a hardware fault in the node or in some communication link. Thus, detection of a disagreement by a voter as discussed above can be treated as an exception, whose treatment is to record some information for the purposes of reconfiguration.

3. Basic assumptions

We assume that the system receives requests for processing from the environment. Each request consists of a copies of a message encoding the processing requirements. A request message $m_4$ is in the triple $<s(t_1), t_1, r_1>$ where $r_1$ is the request (or data) part of $m_4$ containing the processing requirements. $t_1$ is a unique timestamp (a sequence number) and $s(t_1)$ is a unique signature derived from $t_1$. As $m_4$ passes from node to node, only the data part gets modified, that is, a processor processing a request $m_4$ will produce results which are placed in the data part before the results are sent to the following node. The following basic assumptions will be made regarding the system.

A1: For any timestamp $t_1$, it is not possible for any (faulty or non-faulty) processor in the system to generate a value equal to $s(t_2)$, the signature derived from $t_1$.

A2: A non-faulty voter has an authenticator function, which, given values $s(t_1)$ and $t_1$, returns either true or false. A true value indicates that $s(t_1)$ is the signature of $t_1$.

A3: A processor (or voter) receiving a message can obtain the identity of the sender of the message.

A4: A non-faulty processor transmitting a message includes enough redundancy in the message such that a non-faulty processor (or voter) receiving the message can deduce whether the contents of the received message are identical to the contents of the sent message. As a consequence of A4, a voter can discard those received messages that have got corrupted during transmission. Out of the accepted messages, a voter can further discard those messages whose signatures do not correspond with the timestamps (assumption A1, A2). This permits a voter to discard messages from faulty processors that tamper with signatures or timestamps. Thus, only authenticated and non-corrupted messages are used for voting. Finally, as a consequence of A3, a non-faulty voter is in a position to send a response to the sender of a message.
4. Principles of exception handling

In this section the basic concepts of exception handling are briefly introduced (for more details see [1]). Let us consider a software system structured as a hierarchy of modules. When a module calls a procedure exported by a lower level module, then either the call terminates normally (the specified normal response is obtained) or a specified exceptional return is obtained, indicating that the called module was unable to provide the specified normal service. The caller can provide specific exception handlers for treating exceptional returns.

For a given module, one can define its standard domain (SD) such that the execution of the module begins in a state in SD terminates with normal intended results. If the module is executed with the initial state not in SD then the standard service may not be obtained (an exception will occur during the execution of the module). The set of such states is termed the exceptional domain (ED). It is the responsibility of the programmer designing a module to introduce runtime checks in order to detect whether the module has been invoked in ED, and if so, to signal an exception to the caller.

In the following section the standard and three exceptional domains will be defined for a voter.

5. Exception handling in voting

A voter maintains a pool of authenticated, uncorrupted messages received from other NMR nodes. Whenever a majority of messages from a given NMR node are present in the pool, the voter removes these messages from the pool and constructs an input n-tuple \(<i_{n_1}, \ldots, i_{n_p}>\) for voting (where \(n\) is the degree of replication). In particular, consider that the n-tuple is to be formed for messages received from an NMR node \(N_i\), then if a message is present from processor \(1 \leq j \leq n_i\) then \(i_1\) contains this message, otherwise \(i_1\) is set to null.

Formally, let \(M\) represent the set of all non-null messages; \(M\) consists of the set of triples (see Section 4) of the form:

\(<x(t_x), t_x, r_x> \in M>\)

then the input domain \(D_n\) of a voter is defined by:

\[ D_n = (V_i : 1 \leq i \leq n_1) \in M \cup \{ \text{null} \} \] AND \(I \geq [n/2] \)

where, \(n\) is the degree of replication (\(n \geq 3\)), and \(I\) is the number of non-null messages in \(<i_{n_1}, \ldots, i_{n_p}>\):

\[ I = (| \{i | i \neq \text{null}\}|) \]

The domain \(D_n\) can be divided into a standard and three exceptional domains. In order to formally state these domains of a voter, the following two operations on \(i_{n_1}\) are defined:

\[ i_{n_1} = \begin{cases} x_1 & \text{if } i_{n_1} \neq \text{null} \\ \text{null} & \text{otherwise} \end{cases} \]

\[ i_{n_1} = \begin{cases} x_1 & \text{if } i_{n_1} \neq \text{null} \\ \text{null} & \text{otherwise} \end{cases} \]

In addition, we use a function \(\text{equal}\) that returns the maximum number of non-null equal values in an n-tuple.

The characteristic predicate of the standard domain of the voter is:

\[ SD = D_n \text{ AND } (V_i : 1 \leq i \leq n_1) : i_{n_1} \in M \text{ AND } \text{equal}(i_{n_1}, r_{n_1}, \ldots, i_{n_p}, r_{n_p}) \geq I \]

SD constraints all non-null messages of the input n-tuple to be identical.

The voter is invoked in the input exceptional domain (IED) if the following predicate is true:

\[ IED = D_n \text{ AND } (V_i : 1 \leq i \leq n_1) : i_{n_1} \in M \text{ AND } \text{equal}(i_{n_1}, r_{n_1}, \ldots, i_{n_p}, r_{n_p}) \geq I \]

That is, there is a disagreement among the timestamps of non-null inputs. This can only mean that the NMR node supplying these inputs has suffered a sequencing failure.

A voter is invoked in the hardware exceptional domain (HED) if the n-tuple has the following property:

\[ HED = D_n \text{ AND } (V_i : 1 \leq i \leq n_1) : i_{n_1} \in M \text{ AND } \text{equal}(i_{n_1}, r_{n_1}, \ldots, i_{n_p}, r_{n_p}) < I \]

That is, all non-null messages present for voting have identical timestamps and only a majority of those messages, but not all, are in agreement. This situation can only mean that the NMR node supplying the messages has suffered a hardware failure (i.e., some processors within the node are faulty, producing results that do not agree with the outputs of correct processors).

Finally, a voter is invoked in the fail exceptional domain (FED) if the following predicate is true:

\[ FED = D_n \text{ AND } (V_i : 1 \leq i \leq n_1) : i_{n_1} \in M \text{ AND } \text{equal}(i_{n_1}, r_{n_1}, \ldots, i_{n_p}, r_{n_p}) \leq (n/2) \]

That is, all the messages present for voting have the same timestamp, but no majority voting is possible. This represents a failure in voting and indicates a hardware failure that cannot be masked.

THEOREM: The union of the standard domain and the exceptional domains gives the domain of the voter:

\[ D_n = SD \cup IED \cup HED \cup FED \]

Proof: see the appendix.

The various domains are depicted in Fig. 3. The theorem allows us to state that if a non-faulty voter is capable of detecting the exceptions IED, HED, and FED, it can provide a total service consisting of either normal responses or specified exceptional responses.

It will be assumed that the probability of a majority of the processors of an NMR node failing in such a manner as to produce identical incorrect output values is sufficiently small. This assumption permits us to treat the output of a voter that is invoked in HED to be considered standard output (i.e., a voter can mask the consequences of being
Fig. 3. Partitioning of the voter domain.

invoked in HED).

Let an NMR node $N_j$ send its outputs to an NMR node $N_k$ (Fig. 1). The algorithm for the voters of $N_j$ will then be described. A non-faulty voter $V_k$ of $N_j$ can simultaneously produce a signal and a voted output - the former for the sender node $N_j$ and the latter for VMO$_k$ of $N_j$.

According to the predicates given for the four domains, it is possible to devise run-time checks to determine in which domain the voter has been invoked. Next, a high level description of the voting algorithms will be presented.

Voting algorithm for a voter

**IF** Input 6 SD -> insert the voted message in VMO$_k$;
    send 'successful voting' message to all the processors of node $N_j$;

**IF** Input 6 HED -> insert the voted message in VMO$_k$;
    send 'successful voting' message to all the processors of node $N_j$;
    construct a list of processors whose outputs are not same as the voted message;
    send a 'hardware exception message' with this list to the reconfiguration subsystem;

**IF** Input 6 IED -> send a 'sequence exception' message to all the processors of $N_j$;
    no output for VMO$_k$ is produced;

**IF** Input 6 FED -> send a 'fail' message to the reconfiguration subsystem.

A voter can signal two exceptions: indicating a hardware failure or a sequencing failure. The treatment of a hardware failure exception by the subsystem responsible for reconfiguration is not within the scope of this paper. However, one can state that a reasonable strategy for reconfiguration would be for the subsystem to maintain a database of (voted) hardware exceptions received, and if a particular VMO node is found to be responsible for number of such exceptions, to invoke reconfiguration procedures. In line with the NMR processing strategy, messages from voters will also be voted before processing them (see the next section).

Notice that, when the voter is invoked in IED, no output for VMO is produced (even if a majority can be formed). This is because, a sequencing failure and hardware failures occurring together can cause wrong output to be majority voted. The safest solution is therefore not to produce any outputs.

The treatment of a sequence exception by a node is presented in the next section.

6. Exception handling in an NMR node

In terms of exception handling terminology, when a node $N_j$ (Fig. 1) invokes a voter of node $N_k$, either a normal return is obtained or an exceptional return, indicating a sequence failure is obtained.

First of all, to cope with voter failures, the reply messages from voters themselves must be voted; such a voter will be referred to as an exception voter EV (Fig. 4)

![Fig. 4](image)

A non-faulty processor, such as $P_k$ of $N_j$, takes a copy of the message at the head of VMO$_k$, performs the requisite computation, and sends n copies of the result message to the following node $N_j$. $P_k$ awaits a notification from EV$_k$ (this synchronous processing ensures that all the non-faulty processors of $N_j$ process exceptions in the same order as they are produced by $N_j$). If the voter EV$_k$ receives a majority of 'successful voting' messages from $N_j$, then there is no sequence failure and $P_k$ discards the message at the head of the queue and picks up the next message for processing. If the voter EV$_k$ receives a majority of sequence exception messages, then it is known that some processors of $N_k$ have processed wrong messages. The processor $P_k$ deals with this situation as follows: $P_k$ checks if the message at the head of VMO$_k$ has the smallest timestamp among those present in VMO$_k$. If this is the case then $P_k$ simply renodes n copies of the result message to $N_j$. If this is not the case then $P_k$ performs backward recovery to undo the state changes produced by the computation performed for the most recently processed message of VMO$_k$ and then reorders VMO$_k$ to contain the earliest message at the head. This message is then normally processed. The voters of $N_j$ will be in a position to vote again and respond to $N_k$.

Provided that any hardware failures are within the masking capability of an NMR node and there is no 'malicious' behaviour from failed processors (see below), a sequencing failure will eventually be recovered from. This is because, whenever a message is voted and inputted by a voter to some VMO$_k$, then for some other VMO$_l$ of that node, either that message is already present, or will eventually be queued.

It is possible for exception voters EVs to perform exception detection in a manner similar to other voters. For the sake of simplicity, this possibility is not discussed here.
Lastly, we will discuss the treatment of a particularly "malicious" behaviour by a faulty processor. Suppose that a processor of \( N_4 \) deliberately behaves as follows: it preserves a copy of some message it has processed, and keeps sending this message to \( N_4 \) with some arbitrary value in its data part. If such a message arrives uncorrupted at a processor of \( N_4 \), it will be authenticated and accepted and will eventually cause a detection of a sequence failure. In order to cope with such malicious behaviour of processors, it is necessary for the voter of \( N_4 \) to maintain some state information: if a sequence exception keeps occurring, a voter no longer signals this exception, rather, it performs majority voting using only messages with identical signatures. This certainly complicates the design of voters. We believe however that in most practical systems, the possibility of processors behaving maliciously in this manner can be discounted.

7. Concluding remarks

The presence of concurrency can introduce the possibility of sequencing failures in distributed replicated systems. If sequencing failures are not detected, the ability of NMR nodes to mask hardware failures is degraded. The exceptional domains for majority voting have been identified, and, assuming signed messages, a voter algorithm for detecting sequencing and hardware failures has been presented. A practical method of handling sequencing exceptions has also been presented.

In the absence of any operational experience with replicated distributed systems, it is hard to evaluate the merits of the present approach over preventive approaches. All that can be said is that the exception based approach could be attractive if sequence failures occur rarely. We are currently planning a detailed simulation study to perform a comparative evaluation of the various approaches.

One of the drawbacks of the approach presented is that it requires a processor to be capable of performing backward error recovery. One possible way of avoiding the use of backward error recovery is to detect a sequence failure before the processing of a message: by inserting a voter between a VMQ and its processor. Such a voter receives its inputs from all the VMQs of that node, so only the majority voted messages are processed. We believe that the exception handling techniques developed here can be adopted for such a system; a detailed analysis of such a system however is not within the scope of this paper.

The main advantages of the exception handling techniques presented here are summarised below:

(i) Sequencing failures are a special class of failures caused by factors such as variable transmission delays in communication. Such failures can be treated by ordering of messages in VMQs, and no hardware reconfiguration is needed. The paper has described how a voter can detect such failures.

(ii) It is possible to tolerate \( 2^n/n/2 \) failures simultaneously inside each NMR node: \( n/2 \) hardware failures plus \( n/2 \) sequencing failures.

(iii) As mentioned in Section 2, one can consider the following approach: use preventive measures to reduce the probability of sequencing failures and then employ the technique presented here for coping with them. It is likely that this approach will improve the overall performance of the system, since algorithms based on average transmission and processing delays can be constructed (thus leaving the possibility of sequencing failures). The development and evaluation of such a scheme is left as a topic for further research.

(iv) Even if preventive measures are employed to deal with the possibilities of sequencing failures, the exception handling framework presented here can be utilised for the treatment of hardware failures.

We conclude by drawing attention to related research work in replicated systems currently being undertaken at the University of Newcastle upon Tyne. A formal specification of the correctness requirement for replicated systems is discussed in [7]. In [6], the author discusses the problem posed by non determinacy in application programs, and techniques of coping with it. In [5], the authors discuss scheduling of real time processing requests in replicated systems and present algorithms for "locally fair" and "globally fair" scheduling.

The construction of fault tolerant algorithms require making assumptions about the behaviour of failed components. In [4], the authors present a fault classification that has been developed for specifying faulty behaviour of components with replicated responses. Plans are currently being formulated to construct an experimental test bed for trying out ideas on replicated processing.

8. Acknowledgements

This work has been supported by grants from the U.K. Science and Engineering Research Council and the Royal Signals and Radar Establishment of the Ministry of Defence. We thank Paul Ezhichelvan, Maciej Koutny, and Brian Randell for their technical comments on the ideas presented here.

References

APPENDIX

This appendix contains a proof of the theorem which states that:

\[ D_n = SD + IED + HED + FED \]

PROOF:

**HED + FED =**

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } ([n/2] < equal(in_{1,r}, \ldots , in_{n,r}) < 1)] \]

\[ \text{OR } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } (equal(in_{1,r}, \ldots , in_{n,r}) \leq [n/2])) \]

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } (equal(in_{1,r}, \ldots , in_{n,r}) < 1) \]

**HED + FED + SD =**

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } (equal(in_{1,r}, \ldots , in_{n,r}) < 1) \]

\[ \text{OR } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } (equal(in_{1,r}, \ldots , in_{n,r}) = 1)) \]

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{AND } (equal(in_{1,r}, \ldots , in_{n,r}) \leq 1) \]

Since, \((equal(in_{1,r}, \ldots , in_{n,r}) \leq 1)\), the above equation can be simplified to:

**HED + FED + SD =**

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

Finally,

**HED + FED + SD + IED =**

\[ = D_n \text{ AND } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \Rightarrow in_{i,t} = in_{j,t}) \]

\[ \text{OR } ((\forall i,j : 1 \leq i,j \leq n \text{ s.t. } in_{i,j} \in QM \text{ AND } in_{i,t} \neq in_{j,t})) \]

This can be simplified to:

**HED + FED + SD + IED = D_n**

where we have made use of the following two laws:

1. \((\forall i : Ei) = \text{not} (\exists i : \text{not}(Ei))\)
2. \(\text{not}(P \Rightarrow Q) = P \text{ AND not}(Q)\)

QED