NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents

Canada

A Functional Model for Specification and Analysis of Distributed Real-Time Systems : Formalism and Applications

Geetha Ramanathan

A Thesis

in

The Department

of

Computer Science

Presented in Partial fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montréal, Québec, Canada

November 1988

ISBN 0-315-49117-5

ABSTRACT

A Functional Model for Specification and Analysis of Distributed Real-Time Systems : Formalism and Applications

Geetha Ramanathan, Ph.D.
Concordia University, 1988

A mathematical model of distributed real-time systems behavior for problem specification, analysis and proof is presented. The behavior or external view of a system is the "consequence closure" of the specification in which events and their relationships are defined. The pure behavior model is independent of any abstract machine notion. The model takes into account a metric notion of time, and makes use of an absolute time as perceived by an external user for describing event histories. An algebra of events is defined, and the algebraic structure is characterized and asserted through several theorems. A set of functions and predicates are defined on the algebraic structure of events, and they are used in performing proofs on event histories. Examples from hardware design, robotics, and concurrency control in distributed databases are given to illustrate the versatality of the model.

Acknowledgements

I gratefully acknowledge the invaluable assistance of my thesis advisor Professor V. S. Alagar, in the course of the preparation of this thesis. I have received financial support, technical assistance and encouragement from Dr. Alagar throughout my studies at Concordia University. I thank the Quebec Ministry of Education for a fee remission award, Concordia University for a Graduate Fellowship and the Faculty members of the Department of Computer Science, Concordia University for providing a stimulating environment to conduct this research.

Table of Contents

CHAPTER 1

INTRODUCTION

In this thesis a mathematical model for the specification, analysis and proof of correctness of the functional behavior of distributed and real-time systems is presented. This research originated from a study of robotics and database systems - problems in both areas exhibit a significant degree of concurrency and real-time constraints. Tools for specifying time dependent functions and proofs for time dependent analysis are presented. The expressive power and usefulness of the technique are demonstrated through the specification of four problems - one on low-level arbiter design, two on robotic applications and one on database concurrency. The main contribution of the study, however, is to demonstrate that functional model defined on the algebra of events combines the clarity and precision required for formal specification with the ability to provide proofs for a spectrum of applications.

A formal model is necessary to define precisely and unambiguously the desired characteristics of a system - what is not specified cannot be proved and what is not proved may be incorrect. The need for formal specification was first recognized in program design; recently it has been recognized as the first step in designing

complex process control systems. In all situations, the underlying motivation is the same - namely, to define formally the characteristics of a system to be built and provide the tools for verifying the properties. Because automatic verification may never become a reality (or may be prohibitively expensive), in practice, proof of correctness must be done simultaneously with system design and development. This requires the use of mathematics to achieve preciseness in the description of high level design decisions as they are made. Much of the current research on formal specification has concerned itself with the degree of formalism required to communicate ideas as well as their meanings. The formal model in this thesis uses conventional mathematical notions of relations, functions, partial orders and lattices. We heavily use functions and higher order functions and hence call the formalism, a functional model. This approach is consistent with recent developments such as VDM [Cohen 86] and Z [Hayes 87] methodologies which combine recursion equations and abstract data types. That is, the resulting specifications are very close to being abstract functional programs. However, as will be explained later, these approaches are different from the functional model described in this thesis.

The view taken in this thesis is that most real-time systems (such as process control systems and communication systems) are concurrent (or distributed) systems with explicit timing constraints

over their computations and actions. From the point of view of logical formalism and behavior description, distributed systems are no different from any other kind of concurrent systems; the differences are only quantitative and not qualitative. Thus, it may be thought at first that one way to specify a real-time distributed system is to add time to an existing model of a distributed system. We take a different approach which differs from the usual models of distributed processing (see Chapter 2 for a brief review) by two aspects : 1) The metric notion of time referring to an absolute one, such as perceived by an external observer to the system, is taken into account ; and 2) No abstract notion of machine is taken into account so that the model is a pure behavior model. As remarked in [Caspi 86, Cohen 86], a formal specification based on these two aspects provides the meaning of what is specified in the consequence closure; that is, the consequence closure comprises all the time dependent behavior that could be observed regardless of the internal structuring.

Some remarks are in order to motivate and convince the reader on the relevance of the study reported in this thesis to application areas such as robotics and database. The design and operational issues of a distributed database are well understood; however, only recently [Ould 87, Shin 87] robotic researchers have begun to identify the need and relevance of distributed view in robotic applications. Multirobot systems for manufacturing have numerous advantages over

stand alone independent systems. For the navigation (of autonomous vehicles) and cooperating robots on an assembly line, it is both natural and advantageous to distribute control over several processes. This distributed view requires not only the coordination of multiple robots but also requires synchronization and communication with external physical devices such as sensors. Significant actions characterizing the coordinated workspace are communication between robots, communication between robots and external devices, synchronization with external events, waiting and monitoring for external event occurrences, and other simultaneous motion (collision free) on the floor shop. The distributed nature of such an integrated multirobot system will necessarily require sophisticated control and communication techniques; unless these are defined, specified and analyzed, the software realizing such a system cannot be guaranteed to be error free. Different applications may place different demands on the robots and the computers controlling them; however, in an abstract level the primitive tasks can be captured in our model.

Analogously, distributed concurrency control for distributed database management systems is an important research issue. As remarked earlier, this problem is well understood and many solutions have been proposed. Most of the proposed algorithms are complex and difficult to prove correct. One of the main reasons for the complexity of understanding concurrency control algorithms and the

process of providing proofs is due to the lack of a uniform formal style for expressing the algorithms. Only recently, two correctness proofs are reported : serialization of executions [Bernstein 87] and proof of correctness of the majority consensus algorithm [Ravichandran 86]. In this thesis a formal specification and proof of correctness of Holler's algorithm is given in Chapter 4. Hence our attempt and experience in this research must be considered significant - the analytical model helps in demonstrating the stated behavior from the constructed design.

Chapter 2 of this thesis is short and is a brief review of some commonly used models for specifying internal structuring (such as interprocess specification) of distributed systems. We refer to several published reports to bring out three issues: 1) although the inadequacies of a specific method are well understood, no extended comparison between the models have been undertaken; 2) which model is appropriate for which kind of problems is often raised but left unanswered; and 3) different models are appropriate to express different properties and hence to express a specific property, one must choose an appropriate method from a class of models which represent that property.

Chapter 3 discusses a functional model, which is a generalization and enrichment of the event based model discussed by Caspi and Halbwachs [Caspi 86]. The starting point of this

extended formalism is the notion of *durational events*. The word *event* has a wide range of interesting interpretations - from micro level to macro level. A discussion, comparison and a justification of its meaning and its intended usage in this thesis is warranted here.

In everyday experience one comes across statements such as " the *events* for the month of October are ...", " in the *event* that I don't come, cancel the meeting ", " 8 is an *event* in throwing a pair of dice " and " there are several parallel *events* in Winter Olympics ". Another important definition from Webster's Dictionary is that an *event* is also " the fundamental entity of observed physical reality represented by a point designated by three coordinates of place and one of time in the space-time continuum postulated by the theory of relativity ". This last definition is adopted by Lamport [Lamport 86]. The other definitions convey the meanings that an *event* is a temporal marker, a condition, a subset of possible outcomes of an experiment and a collection of actions. If an event is regarded as a collection of actions that are schedulable and partially ordered, then one is lead to consider it to be a nonatomic and hence a durational entity.

In Computer Science, one approach to defining data types is to use a recursive definition of a discriminated union of primitive data types. A typical example of such an entity is *generalized list*. Similarly an event may be defined using the definitions above. The

advantage in this approach is the obtained generality that permits the applicabilty of the same term uniformly to several levels of abstract descriptions.

Concurrent systems are traditionally described hierarchically, wherein each level implements the level above it. For describing the units of work at various levels, researchers use different terms. To cite an example from database, "transferring $500 from account number 24098 in Branch 102 to the account number 19675 in Branch 201 at 11:10 A.M. on November 11, 1988" is a high level unit of work. In an intermediate level, say in a high-level programming language, this must be translated into a sequence of language constructs, which in turn are translated to next lower level machine language operational sequences. Finally, these machine language instructions must be implemented by the circuits and flip-flops. Lamport [Lamport 86] uses the terms system execution, operation execution and events (space-time view) for describing the units of work at the three levels. In developing axioms for correct interpretation and implementation for lower level descriptions corresponding to high-level descriptions, Lamport observes that even flip-flops have *definite durations* associated with their settling times. That is, system executions, operation executions and events have definite (bounded) durations. This thesis, aimed at capturing and analyzing time dependent properties, takes the view that durations of

units of schedulable work form the dominating uniform property at various levels and must be captured in the external (observational) specification. So, it is decided to use the term *event* (with duration) to refer to unit of work at all levels of description. The advantages are that a user (not necessarily a Computer Scientist) can understand the term as applied in everyday usage and at system implementation level the term applies equally well.

In describing the timing behavior of real-time systems, Jahanian and Mok [Jahanian 86] use an *event-action* model embedded in logic. For them, the notion of event is that of a temporal marker. An action can be either primitive or composite (a partial ordering of actions defined recursively) that consumes a bounded amount of resources. Stop and start events initiate and terminate actions, whereas transition events assert safety properties that remain true over a *period of time.* It is also assumed that there is a *finite delay* between the occurrence of a physical event in the external world (such as pressing a button or reading a sensor) and its recognition by a computer system. That is, the medium interfacing external to internal, with its unknown delay, must become part of event occurrence process. Hence, we are lead to conclude that stop and start are instantaneous events and others are not (although event is defined to be a point in time). This inconsistency, however, is overlooked by Jahanian in [Jahanian 86].

As remarked at the outset, the model of specification attempted here is *purely behavioral* and hence is devoid of states, transitions and operations (actions) with side effects. Hence, the term event, applicable to several levels of descriptions, seems more unifying and consistent with the chosen model.

Finally, the specification and proof of correctness of the arbiter design in [Caspi 86] assumes a time factor due to communication delay; that is, without this assumption, their design cannot be shown to meet the initial specification based on instantaneous events. In the extended formalism discussed in this thesis, the proof of correctness (for the same arbiter design problem) is shown to follow with no major change in the original proof structure; no assumption outside the specification is necessary. Hence the model built on instantaneous events [Caspi 86] is inadequate for handling such low-level problems and as we show in Chapter 4, the extended model is necessary for describing a spectrum of other problems.

Chapter 4 contains the other major new results of this thesis, the specification and proof of correctness of four problems: 1) the design of a distributed arbiter derived from its initial specification; 2) Holler's algorithm for multiple copy update problem in distributed database; 3) robotic assembly (asynchronous algorithm) with multiple arms and 4) the design of a robotic navigation controller

(asynchronous) for the coordinated motion of several robots on a floor shop.

Chapter 5 is an attempt to unify the loose ends and identify some of the related work started and left unfinished by us; these are topics for future work.

CHAPTER 2

SPECIFICATION MODELS - A BRIEF REVIEW

This chapter presents a brief survey of the theory and experience in the specification of distributed systems. The discussion is not exhaustive and is limited to the most notable methods: algebraic approach [Milner 83], net theoretic [Petri 62, Thiagarajan 83, Ould 87], temporal logic [Burstal 74, Pnueli 77, Lamport 80a, Koymans 83] and axiomatic [Hoare 75].

Two key issues are to be addressed in a specification method: acquiring information and expressing the acquired information. In our discussion, we address only the second issue - the utility and expressive power of the specification method.

2.1 Algebra for Concurrency

Building an algebraic model to express concurrency is due to Milner [Milner 83]. The basic elements of an algebra are *objects*, *operators* and *rules* for combining objects into complex objects. This model, apparently, arose out of dataflow models due to Dennis [Dennis 74] which was later put on algebraic footing by Kahn [Kahn 74]. That is, the abstract machine model, more or less, is derived

from finite state automata. The semantics assigned to the nodes and arcs changes the algebra and the meaning of algebraic expressions. The objects, also called agents, interact among themselves. Such an interaction signifies that something noticeable has occurred in the system. Each agent (realized by one or more processors), has several ports on its periphery at which events occur. Some events are atomic in nature; that is, when atomic events occur it is the case that something has occurred instantaneously. It is conventional to use a Greek letter to name a port as well as an event occurring at that port.

There are four types of operators: atomic action ($\alpha.$), summation (+). product ($\&_\beta$ or $|_{\text{beta}}$) and encapsulation ($/_\beta$ or $\backslash_\beta$). An agent P which alternates (possibly nonterminatingly) between two events α and β (in that order) is defined by the equation $P = \alpha.\beta.P = (\alpha.\beta)^+$. An agent Q which alternatingly performs either α_1 or α_2, then β, is given by the equation $Q = \alpha_1.\beta.Q + \alpha_2.\beta.Q$; which may be abbreviated as $Q = (\alpha_1 + \alpha_2).\beta.Q$. The regular expression $((\alpha_1 + \alpha_2).\beta)^+$ is a solution to this equation. Notice that the '+' between agent expressions denotes "disjunction" and the superscript '+' denotes 'one or more times'. A hierarchical decomposition of an agent $P = \alpha_1.P_1 + \alpha_2.P_2 + \ldots + \alpha_n.P_n + \ldots$ indicates the possible next actions of P. For example the algebraic expression $\Sigma\ \alpha_i.\beta_i.P$ denotes an agent P (interpret as a buffer) with two ports α and β where

$\alpha_i(\beta_i)$ denotes the port α (β) for the i-th value produced (delivered).

Product operators combine several agents to a single agent. If $P = \alpha.\beta.P$ and $Q = \beta.\gamma.Q$ (which are buffer like agents) are pictured as in Figure 2.1 (a) and 2.1 (b), the agent $R = P \&_\beta Q$ – called β - product or β synchronization can be pictured as in Figure 2.1 (c). The actions α and γ in this product agent may occur independently but β occurs (possibly an infinite number of times) only when both P and Q are capable of a handshake. Since R performs α first, then repeatedly performs β followed by either α or γ, R can be pictured with three ports as in Figure 2.1 (d).

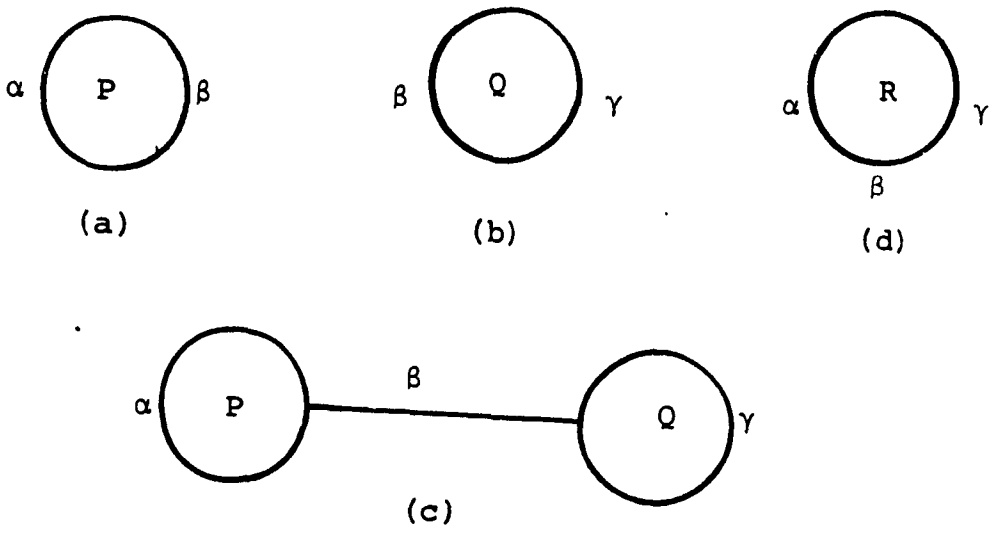

Figure 2.1

Moreover, an expression for R is $R = \alpha.S$, $S = \beta.(\alpha.\gamma.S + \gamma.\alpha.S)$. In general the β - product of $P = \Sigma\gamma_j..Q_j$ is R where

$P = \Sigma\alpha_i.P_i$, $Q = \Sigma\gamma_j.Q_j$ and

$$R = \sum_{\alpha_i \neq \beta} \alpha_i.\ (P_i \ \&_\beta\ Q) + \sum_{\gamma_j \neq \beta} \gamma_j.(P\ \&_\beta\ Q_j) + \sum_{\alpha_i = \gamma_j = \beta} \beta.(P_i\ \&_\beta\ Q_j).$$

It is easy to see that β - synchronization permits the modeling of multi-way hand shakes; infact $\&_\beta$ is both commutative and associative.

The operator $/_\beta$, called β - hiding, releases β from further synchronization. Thus $R' = R/_\beta$, $R = \alpha.\beta.P\ \&_\beta\ \beta.\gamma.Q$ is the system of equations $R' = \alpha.S'$, $S' = \alpha.\gamma.S' + \gamma.\alpha.S'$. Refer [Milner 83] for the semantics of other operators and examples illustrating the algebra for specification. The chief advantage of the algebraic approach is the ability to define and use general combinators and provide a way for formal (mechanical) proof-checking. Since behavior is defined as an equivalence class upon the set of machines, the proof of a system reduces to the proof of the equivalence between the abstract machines representing the specification and the implementation of the system. Based on this algebra for communicating agents a formal specification and proof of the firing squad problem [Minsky, 1972, p.28] is also given in [Milner 83].

> A line of n FSMs (Finite State Machines) are given. At one end is a 'general', at the other end is a 'sergeant' and in between are (n−2) identical 'soldiers'. It is required to design the machines so that all soldiers fire at the same time after the General has given the order. The machines operate in step, with the state of a machine at time t + 1 depends on the states of the neighboring machines and its state at time t.

As remarked by in [Milner 80], we can prove equations between agent expressions but there are interesting properties of processes which cannot be expressed algebraically.

2.2 Net Theory

On the surface, there seems to be some relationship between net theory and the algebraic approach. However, the emphasis of these models are different. Net theory emphasizes causal independence whereas algebra emphasizes (only) communication among agents.

As the name suggests, Petri nets were first proposed by Petri [Petri 62] for modeling concurrent systems. A net may be viewed as a directed bipartite graph with two kinds of nodes: nodes denoting atomic states (places) and nodes denoting local transitions. The places, represented by circles, define locations in the graph where *tokens* reside. These are linked by arcs to transition nodes, represented by boxes. More formally, a net N is a triple N = (S, T, F) where $S \cup T \neq \emptyset$, $S \cap T = \emptyset$, $F \subseteq (S \times T) \cup (T \times S)$, domain (F) $\cup$ range (F) = $S \cap T$.

Here S is the set of state nodes, T is the transition nodes and F is the set of arcs (flow relations). See Figure 2.2.

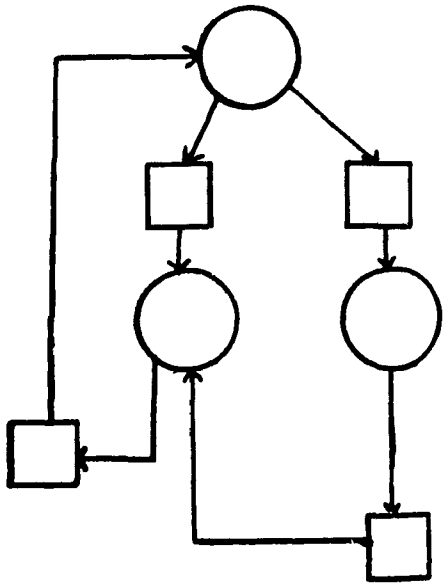

Figure 2.2

The chief advantage of Petri nets is that they provide a simple and abstract setting for investigating distributed systems. The basic notions of sequence (causality), selection (conflict, choice, nondeterminism), synchronization (concurrency) and failure of synchronization (conflict + concurrency ≡ confusion) can be clearly stated as in Figures 2.3(a) - 2.3(d).

The situation in Figure 2.3 (a) is causality where e_2 follows e_1.

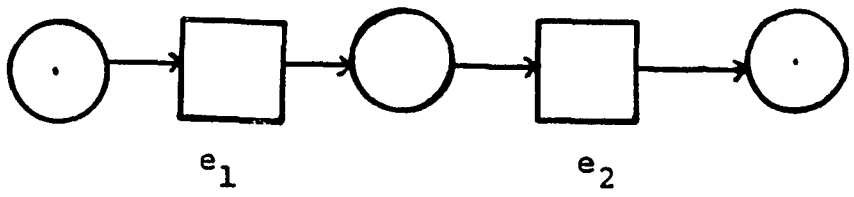

Figure 2.3 (a)

Due to the shared condition b in Figure 2.3 (b), the transitions e_1 or e_2 but not both can occur. This is a conflict and portrays nondeterminism in a limited fashion.

Nondeterminism in its full generality enters when b is true so that both e_1 and e_2 are possible. There is no conflict here in a simultaneous transition but net theory cannot capture nondeterminism in its generality.

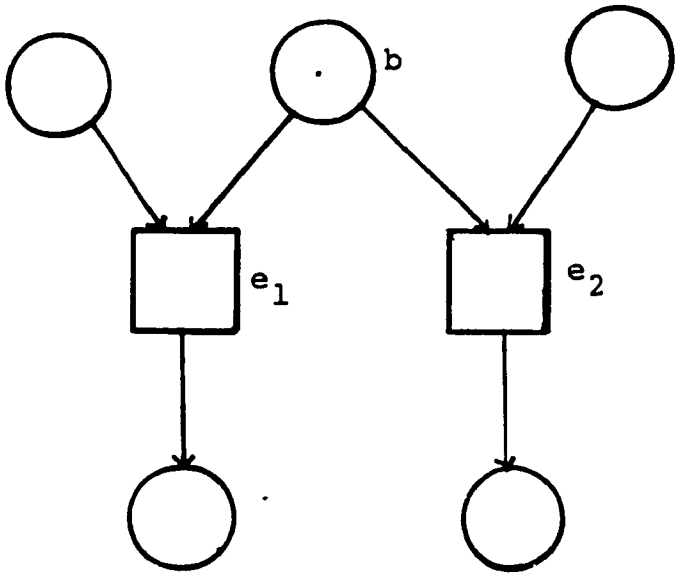

Figure 2.3 (b)

Concurrency is illustrated in Figure 2.3 (c). The transitions, in general, can be partially ordered but the ordering cannot be inferred from the specification.

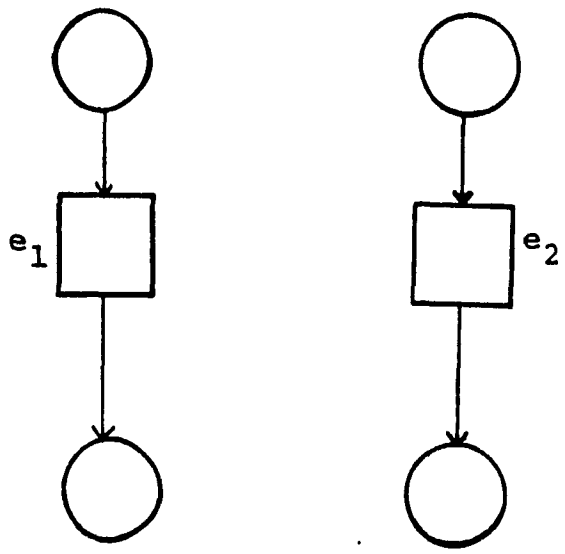

Figure 2.3 (c)

Confusion arises whenever there is an overlap of concurrency and conflict. A symmetric form of confusion is shown in Figure 2.3 (d): there may be disagreement among observers as to how the transition occurred from $\{b_1, b_2\}$ to $\{b_3, b_4, b_5\}$. To resolve the dispute, more assertions on the properties of states and data variables must be assumed

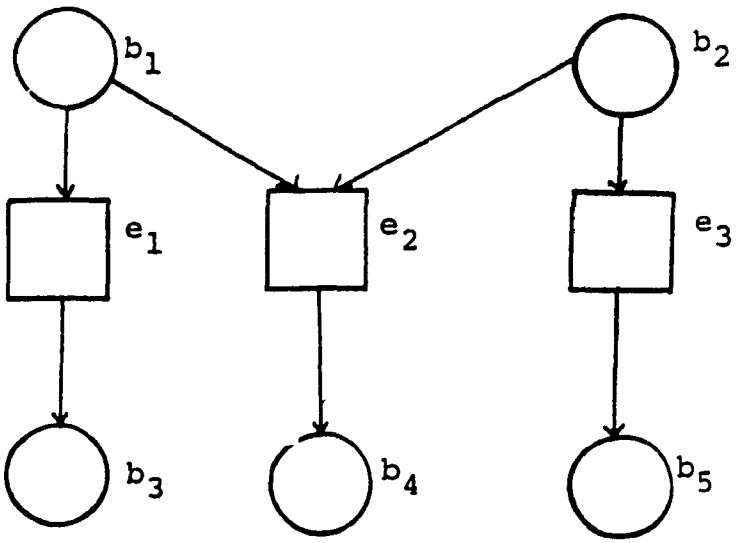

Figure 2.3 (d)

Petri net formalism has been successfully applied to model numerous distributed computing applications. We briefly mention two such applications of interest to us - database and robotics.

The use of Petri nets to model concurrency control algorithms is not new. Ellis [Ellis 77] and Genrich and Lautenbach [Genrich 79] are the first to formalize concurrency control algorithms using several variants of Petri net models. However no analysis or performance evaluation was attempted. It was in [Ramamoorthy 80] that a formal performance analysis based on safe Petri nets was reported. Recently Ozsu [Ozsu 85] has generalized the Petri net model to Extended Place Transition Nets supporting (1) the concept of transition time; (2) precondition

specification to enable the modeling of situations where a place p participates in the activation of only a subset of whose set of input places include p; and (3) the movement of tokens carrying performance related data.

As an application of net theory for specification in robotics we mention the recent work of [Ould 87] - the behavior specification of a flexible assembly cell. The assembly cell consists of two robots, conveyor, cameras, a gripper and an assembly table. The functional requirements are: 1) parts are placed on a conveyor and as soon as the first camera detects the presence of an object, the conveyor stops; the conveyor is restarted only after the object is identified; 2) robot R_1 picks up the identified object and leaves it on a table and 3) robot R_2 does the assembly operations. The petri net for this problem describes the places, tools and functions (by transitions). A formal analysis and simulation based on this net specification is given in [Ould 87]. It is worthwhile comparing this net specification with our specification on a similar robotic assembly problem discussed in Chapter 4.

In spite of the many advantages and its widespread usage, the chief disadvantage of the Petri net approach is that it produces complex and unstructured descriptions of real life systems. To remedy this disadvantage research is directed towards first-order marked nets and hierarchy of sub-nets. This work draws Petri nets closer to logic.

2.3 Temporal Logic

Temporal logic belongs to the class of modal logics, the origin of which can be traced to the ancient Greeks. Necessity (bound to be true), impossibility (bound to be false), contingency (neither necessary nor impossible) and possibility are the four modal notions. These notions gave rise to the three basic modal operators $\Box$ (henceforth), $\Diamond$ (eventually) and $\odot$ (next). Several other derived operators are given in [Moszkowski 86]; for our exposition we stick to the simplest form of temporal logic with a few simple operators.

Temporal logic has been used for specification of both hardware and software systems ever since Burstal [Burstal 74] suggested it for reasoning about programs and Pnueli [Pnueli 77] showed its application to concurrent programs. Bochman [Bochman 82] was probably the first to use temporal logic for hardware specification. In his work he introduced " reachabilty analysis" as a method for analyzing self-timed arbiter. The recent monograph [Moszkowski 86] describes a new language, called " TEMPURA" based on temporal logic.

Some of the notable work in the application of temporal logic to concurrent systems are due to Hailpern [Hailpern 82], Lamport [Lamport 83b], Moszkowski [Moszkowski 83]. Recently, temporal logic based correctness proof for the Thomas majority consensus algorithm for concurrency control of fully replicated databases is given by Ravichandran [Ravichandran 86].

Temporal logic assertions are well-formed formulas constructed according to the rule:

1) an atomic predicate is a wff.
2) if f is a wff, then so are $\sim$ (f), $\Box$ (f), $\Diamond$(f) and $\odot$ (f).
3) if f and g are wffs, then so are (f $\vee$ g), (f $\wedge$ g), (f $\rightarrow$ g) and (f $\equiv$ g).

The operators $\Box$ and $\Diamond$ are duals:

$$\Box \equiv \sim \Diamond \sim P$$

$$\Diamond P \equiv \sim \Box \sim P$$

So, in temporal logic the propositional connectives retain their original meanings; the universal and existential quantifiers of predicate logic give rise to (quantified over time domain) $\Box$ and $\Diamond$ respectively.

The derived operator $\sim>$ is a binary operator: P $\sim>$ Q means that if P ever becomes true then Q must be true at that instant or later. It is easy to see that

$$P \sim> Q \equiv \Box (P \rightarrow Q)$$

and $\sim>$ is transitive.

As an example, consider the simple problem of communication between two parties. Abstractly, let {a,b} be the set of endpoints of a channel. For any element e $\in$ {a,b}, $\bar{e}$ denotes the other element.

That is if e = a (b), $\bar{e}$ denotes the other endpoint b (a). Let m $\in$ M denote an arbitrary message transmitted over the channel. The two atomic formulas that we consider are *accept*(m, e) and *deliver*(m, e), m $\in$ M, e $\in$ {a, b}. To assert that the channel cannot accept two different messages at the same endpoint at any given instant we shall have to prove [*accept*(m, e) $\wedge$ *accept*(m′, e)] $\rightarrow$ m = m′

We can state that both endpoints are accepting messages all the time by the formula.

$$\hat{\Diamond} \Box [accept(m, a) \wedge accept(m',b)].$$

To prove that the channel cannot copy messages, we must establish that a message delivered at time t cannot be redelivered (unless it was accepted again) at time t' > t; that is, we must show

$$[deliver(m, e) \wedge \sim \Diamond (accept(m, e)] \rightarrow \odot \Box deliver(m, e)$$

Liveness and safety are two important properties to be satisfied by concurrent systems. Temporal logic can be used to express these properties. A liveness axiom is of the form B ~> A which means that when B holds (B is a condition) then A (an action) must happen at some time in the future. To illustrate the liveness specification for the example under consideration, let us use the symbolism e $\Rightarrow$ $\bar{e}$ to denote the set of all functions from one endpoint to the other; that is, e $\Rightarrow$ $\bar{e}$ denotes the infinite queue of messages. Let *front* (e $\Rightarrow$ $\bar{e}$) denote the

message at the front of the queue. The liveness property of the action *deliver*(m, e) is that the message at the front of the queue must eventually be delivered. This is specified as

$$[\mathit{front}(e \Rightarrow \bar{e}) = m] \rightsquigarrow \mathit{deliver}(m, \bar{e})$$

Safety property asserts that some property P, if it becomes true, remains true forever. To state that P is such a property (also called inductive assertion) we write

$$\Box\ (P \rightarrow \Box\ P).$$

Another important class of safety properties have the form

$$I \rightarrow \Box\ (A \rightarrow B \mathbin{\underline{\triangle}} C)$$

where $B \mathbin{\underline{\triangle}} C$ means that C holds at least as long as B does. For example, first-come-first-served property of a queue of processes can be expressed in this form. We conclude our discussion on temporal logic with an elaborate formal specification of the channel described below :

> The channel between endpoints a and b can pass messages $m \in M$ in both directions simultaneously, until it receives a 'disconnect' message from one end, after which it neither delivers nor accepts messages at that end. It continues to deliver and accept messages at the other end until the 'disconnect' message arrives, after which it can do nothing. The order of messages sent in a given direction is preserved.

In addition to the specifications given above, we have the following additional axioms which use the operator '<' (before)

An accepted message at e will be delivered at $\bar{e}$ unless $\bar{e}$ has issued a disconnect message:

$$accept(m, e) \rightarrow \Diamond (deliver\ (m, \bar{e}) \vee < accept(\text{DISCONNECT}, \bar{e})).$$

This axiom asserts that there can be no loss of messages in an active channel and messages in that case are eventually delivered.

The order of messages sent in a given direction are preserved:

$$[accept(m, e) \wedge \odot \Diamond accept(m', e) \wedge m' \neq \text{DISCONNECT}]$$

$$\rightarrow \Box\ [delivered(m, \bar{e}) \rightarrow \sim < deliver(m', \bar{e})].$$

It is assumed that m is not a disconnect message and the messages can be distinguished. The above two axioms taken together imply that both m and m' are delivered in the order of their acceptance.

Finally, the effect of a disconnect message at an end e is specified by the following axiom:

$$[accept(\text{DISCONNECT}, e) \vee deliver(\text{DISCONNECT}, e) \rightarrow$$

$$[(m \neq \text{DISCONNECT}) \wedge \odot \Box\ (\sim accept(m, e) \wedge \sim deliver(m,e)]$$

If the channel specification is modified to include real-time requirements

then new constructs as in [Koymans 83] must be introduced in temporal logic.

2.4 Assertional Methods

In an assertional method the states (and not events) are specified and reasoned about. The chief advantage of this method is that the proof is reduced to a series of precise steps that can be machine verified; however the number of states may increase exponentially.

Typically a specification method enables expressing the properties of a system and providing proofs; but a programming language is required to translate the specification and consequently two separate notations are employed simultaneously. For example, temporal logic can be thought of as a tool for specifying and proving properties of programs written in, say, Hoare's CSP [Hoare 85]. This dichotomy exists in almost all formalisms - especially when one is interested in program verification. Assertional proof methods such as Hoare's logic is closer to a programming logic formalism in which every state s is viewed as a relation between two predicates: a precondition and a postcondition. Thus an execution E consisting of a sequence of states is expressed by {P} E {Q},which is correct (or valid) if whenever E is started in a state s_0 that satisfies P and E terminates in a state s_t, then s_t satisfies Q.

The first assertional method for proving properties of concurrent programs was outlined in [Ashcroft 71]. Hoare [Hoare 72] gave proof

rules extending the partial correctness notion of sequential programs to concurrent programs. But there were severe restrictions in the application of these rules. For example, these rules are inadequate for proving programs in which processes communicate because assertions appearing in the proof of one process are not allowed to mention variables local to another and only designated shared variables can appear in an invariant. The work reported in [Owicki 76] extended Hoare's proof rules to handle the proofs of concurrent programs that synchronize and communicate using shared variables. At the same time, Lamport [Lamport 77] independently developed monotone assertions as part of a more general method for both safety and liveness properties of concurrent programs. Soon after, Generalized Hoare Logic(GHL) was used by Lamport [Lamport 80b, Lamport 84b] to develop techniques for proving safety properties of concurrent programs written in Hoare style programming logic.. The basic idea here is to derive the invariance properties of a concurrent program from the invariant properties of individual processes.

An important feature of GHL is that proofs can be given when the atomic actions are not known but the invariants are known. Hoare's programming logic was further augmented by the programming notation CSP proposed by Hoare [Hoare 1985]. This notation enables synchronous message passing facility, a distinguishing feature of distributed programs. In this notation, an input command

inp: A? var

in process B matches an output command

out: B! expr

in process A if the type of expr matches the type of var. Input and output commands are executed synchronously in matching pairs thus implementing distributed assignments statements.

To illustrate the expressive power of this notation, we reproduce below Hoare's solution to the firing squad problem:

Notation Neighboring soldiers are combined by the operator >>. A >> B means that the right channel of A is connected to the left channel of B; all communications along this line are synchronized. The event 'fire' requires simultaneous participation of both A and B; otherwise (A >> B) will deadlock. The firing line is

$$\text{LINE (n)} = \text{GENERAL} >> (>_i>_{<n} \text{SOLDIER}) >> \text{SERGANT}$$

The line is started by a 'go' signal in general's left channel and it terminates with a valley of simultaneous shots from all the soldiers:

$$\text{LINE } (n) = (\text{left.go} \rightarrow \text{fire} \rightarrow \text{stop}),\ n > o.$$

$$\text{SOLDIER} = \text{SOLDIER}_0$$

where

$$\text{SOLDIER}_i = \text{if b(i) then fire else}$$
$$\text{left?j} \rightarrow \text{left!i} \rightarrow$$
$$\text{right!i} \rightarrow \text{?right k} \rightarrow$$
$$\text{SOLDIER}_{f(i,j,k)}$$

where f is a function with range (0, ..., n). Each soldier communicates his state with his neighbors and if they simultaneously reach a state i for which b(i) is true then they all fire.

This brief review is intended to show the similarities and interdependence of the various specification styles rather than to bring out the inadequacies of each method. Algebraic expressions come out of nets - although their emphasis may be different. Net models with the properties of first order predicate logic have been studied in [Thiagarajan 83] and this shows that some mappings between net theory and temporal logic can be set up. Linear time temporal logic is close to assertional specification style. Manna and Wolper [Manna 84] have investigated techniques for automatically synthesizing CSP synchronization from temporal logic specifications. For example, consider

a synchronizer S that regulates the activity of two processes P_1 and P_2; in particular S ensures that they do not simultaneously operate in their critical regions. Assume the construct $S!begin_i$ to represent a request by P_i to enter its critical region. Similarly $S!end_i$ represents the construct for P_i to leave the critical region. The temporal logic specification

$S!begin_i \wedge [\Box (S!begin_i \rightarrow \odot S!end_i)] \wedge [\Box (S!end_i \rightarrow \odot S!begin_i)]$

means that P_i makes a request to enter its critical region. Further, whenever it enters (exits) the region, it subsequently exits (enters). The specification of S is $S_1 \wedge S_2$ where

S_1: $\Box[P_1?begin_1 \rightarrow ([\sim P_2?begin_2] \text{ until } [P_1?end_1])]$

S_2: $\Box(P_2?begin_2 \rightarrow ([\sim P_1?begin_1] \text{ until } [P_2?end_2])]$

and the operator 'until' is to resolve conflicts. From this initial specification (which has borrowed ? and ! from CSP) Manna shows how CSP programs can be derived automatically. Browne et al. [Browne 85] use a version of temporal logic, called CTL, to automatically verify synchronous circuits. Accepting the behavioral specification given in CTL, their technique is to generate a state-transition graph (a form of net) with associated predicates against which various temporal properties can be verified. This state-transition graph is a first level implementation of the CTL specification of the circuit. The functional model discussed in the next chapter has some characteristics in common

with temporal logic; however, the algebraic structure of events, the quantitative aspects of time dependent functions are outside of temporal logic formalism.

CHAPTER 3

FUNCTIONAL MODEL : FORMALISM

A formal mathematical system has three important components. These are 1) formation rules dealing with the definition of objects and rules for combining them; 2) a basic set of axioms, also called assertions, regarding the subject matter to be discussed in the system; and 3) effective procedures to determine the truth of statements that can be made within the system. It is also agreed that a proof in a formal mathematical system is a method to derive the truth of an assertion from hypotheses according to given rules of formation. Thus, when reasoning about the behavior of a computing system, the first task is to embed the system in a formal model so that the consequence closure can be looked at for all behavioral properties.

The classical approach to proving the correctness of sequential systems is to model them as finite state transformers and embed the set of input-output assertions in predicate logic wherein correctness or contradictions can be established. In the context of distributed systems, timing constraints induced by synchronization and communication do arise. Several approaches for parallel programming systems [Kahn 74, Hoare 85] extend the classical proof method by

inventing tools to model the behavior of parallel programs that are independent of actual execution times. In such models, two systems exhibiting the same output behavior on identical input streams will be declared identical, although the nature and durations of intermittent or interleaving computations may differ. Hence this modeling approach may not be acceptable for real time systems.

As we remarked in Chapter 1, most of the real-time systems are concurrent (or distributed) systems with explicit timing constraints over their computations and actions. In such systems, the metric notion of time is used not only to compare the performance of several implementations, but also to decide of the adequacy of the system to its specifications. The functional model discussed here fully explores the time dependencies between internal and external events of a system. Once again, there are two admissible notions of time. Researchers [Lamport 78a, Hoare 85] whose primary objective is an investigation of the structural (internal) description of distributed systems have expressed the view that global clock is quite expensive and even unnecessary.Hence they resort to relative times measured by subsystem clocks. Under this assumption the model of analysis is a partial ordering relation on the set of internal actions of the system. The other view is the notion of time referring to an absolute one, such as perceived by an external observer to the system. So, in the logical level of description, the problem of the

relative times as measured by "local clocks" do not arise. This latter view is quite consistent with the traditional algebraic specification methodology [Gehani 86], where internal operations are masked from axioms so that the consequential closure can be derived only from externally visible operations. This view is taken in behavior descriptions; see Caspi [Caspi 86]. So, we follow this tradition and refer to an absolute externally visible time in this thesis.

Caspi and Halbwachs [Caspi 86] have introduced a formalism in which time sequences (which indicate instances of event occurrences) are monotonic increasing functions from integers to time (real numbers). These functions admit pseudo inverses [Sachis 77] that are interpreted as occurrence counters. Using conventional tools from elementary function theory, they were able to provide a rich set of tools for the specification and proof of low level real time systems arising in hardware systems.

We consider a generalized formalism in which events have duration (finite or infinite) and illustrate with four examples the reasoning power of this extended model. We believe that only a limited set of primitive actions at the hardware level can be instantaneous. As an example, the event send (receive) can be considered to be instantaneous at the hardware level specification but the actual transmission time is a function of message length. In

high level specification of real time or distributed systems, the events send, receive and assign are not instantaneous events.

Thus it seems natural to associate events with intervals (in addition to the semantics of operations) so that the end points and the length of the interval denote the starting time, the completion time and duration of the event. This generalized formalism is sufficient to define and describe the behavior of events in low level hardware systems and is necessary for a high level modeling and description of the behavior of real time systems, distributed systems and network architecture. In summary, the proposed formalism provides a single framework for a spectrum of abstract specifications.

3.1 Description of the Formal Model

The formal model is intended to support distributed and real time behavior in the entire spectrum of systems extending between hardware simulation and industrial process control such as robotics. The primitive objects that we wish to characterize are formalized as *events*. An event history is characterized by a sequence of intervals denoting the occurrences of that event. For example, assigning a value to a variable, sending and receiving messages are events in the system. The activities or the occurrence of events are subject to time constraints due to two important factors: there is an interaction with a physical process, such as sensor, that must be

read periodically; or, time is imposed by mutual exclusion, synchronization and ordering of the events. Thus 'what is the effect of an event and when the event causing the effect occurs' are important to be considered together.

The occurrences, executions and the effects of events can be continuous and need not be instantaneous. Although the entire process itself may be nonterminating, the individual events making up the process may be ideally viewed as happening continuously over piecewise continuous intervals. Hence we identify time Π with $\mathbb{R}$, the real line. To deal effectively with extreme cases, we let $\overline{\Pi} = \mathbb{R} \cup \{\infty\} \cup \{-\infty\}$ and $\overline{\mathbb{N}} = \mathbb{N} \cup \{0\} \cup \{+\infty\}$.

3.1.1 Basic concepts of event and time

The basic assumptions on time and events are:

[A1] An event can occur any number of times within a system; however within a finite period of time, there can be only a finite number of occurrences.

[A2] An event may occur continuously between its start time and completion time. So, each occurrence associates an interval with it and the event history is given by the associated sequence of intervals.

The following definition is based on these assumptions:

Let INC denote the set of all non-decreasing functions from $\overline{N}$ to $\overline{\Pi}$. Define the higher order functions $TIME_1$ and $TIME_2$,

$$TIME_j : E \rightarrow (\overline{N} \rightarrow \overline{\Pi}),\ j = 1,2$$

where E is the set of all events and $TIME_2(e) \geq TIME_1(e)$. Thus the set of all events is embedded into the set

$$L = \{(f_1, f_2) \mid f_1, f_2 \in INC,\ f_2 \geq f_1\}.$$

The function comparison is done pointwise. Notice that (INC, $\leq$) is a poset and every interval of this poset is a member of L. Although every event in E is mapped onto an interval in L (and hence a sequence of intervals on the real line), corresponding to an arbitrary interval in L there may not be an event in the system. By admitting empty (vacuous) events that do not have any effect in the system and letting them correspond to such intervals, we can overcome this and thus identify E with L.

Example 1

Let $a_i(c_i)$ and $b_i(d_i)$ denote the start and finish times of the i-th occurrence of the event $e(f)$. Then,

$$TIME_1(e)(i) = a_i\ ,$$

$$TIME_2(e)(i) = b_i\ ,$$

$$TIME_1(f)(i) = c_i\ ,$$

$$TIME_2(f)(i) = d_i\ ,$$

$a_i \leq b_i$, $c_i \leq d_i$. Figure 1 shows a sample of event occurrences.

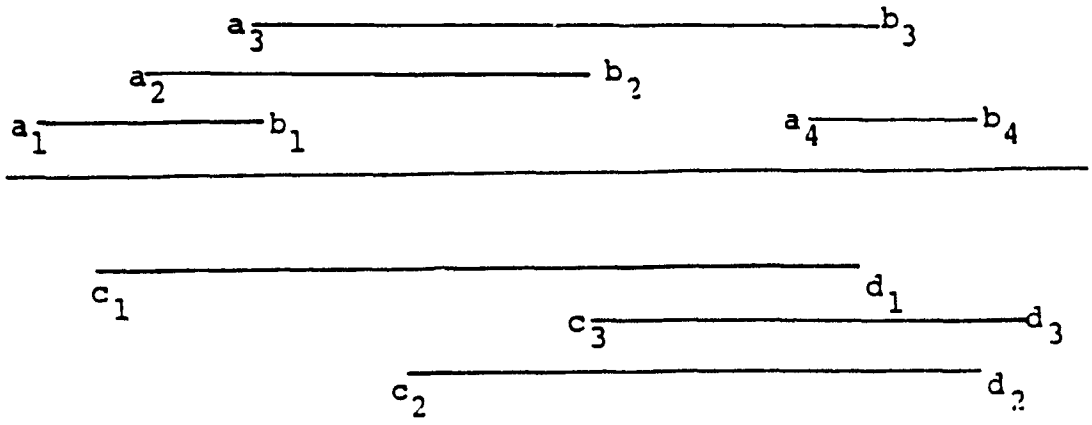

Figure 1

The following properties of $TIME_j$, j = 1, 2 are assumed:

[t_1] $TIME_j(e)$, j = 1, 2 are monotonic non-decreasing functions.

[t_2] $TIME_2(e)(n) = +\infty$, $n \neq \infty$ means that the n-th occurrence of *e* is not yet complete.

[t_3] $TIME_j(e)(0) = -\infty$

[t_4] $TIME_j(e)(+\infty) = +\infty$

Due to [t_3] and [t_4] it is sufficient to consider $TIME_1(e)(n)$ and $TIME_2(e)(n)$, $1 \leq n < \infty$.

The history of a variable in a system is captured by the sequence of values and the times (with duration) of assignment of such values. If $v \in V$ is a variable, then ASSIGN(v) is an event in E and $TIME_1(ASSIGN(v))(k)$ and $TIME_2(ASSIGN(v))(k)$ denote the start and completion times of k-th assignment to v, a value from its

domain. Thus we have the functions,

ASSIGN : $V \rightarrow E$

VALUE : $V \rightarrow (\mathbf{N} \rightarrow DOM)$,

where DOM is the set of values necessary for describing the system.

Although the time sequences associated with $TIME_1(e)$ and $TIME_2(e)$ are monotonic non-decreasing, $TIME_1(e)(n)$ and $TIME_2(e)(n)$ need not correspond to the start time and finish time of the n-th occurrence of e. That is, the durations of different occurrences of an event need not be the same; however for every n, there is a $k \in \overline{\mathbf{N}}$ such that $[TIME_1(e)(n), TIME_2(e)(k)]$ is the interval corresponding to the n-th occurrence of *e*. Thus, there exists a bijective function $P_e : \overline{\mathbf{N}} \rightarrow \overline{\mathbf{N}}$ with $P_e(0) = 0$, $P_e(\infty) = \infty$ such that the composite function $CTIME_2(e) = TIME_2(e) \circ P_e$ is *not* in general monotonic, but $\forall n \in \mathbf{N}$, $[TIME_1(e)(n), CTIME_2(e)(n)]$ define the intervals corresponding to the occurrences of *e*. Since P_e is bijective, it is easy to see that $\forall n$, the intervals $\{[CTIME_1(e)(n), TIME_2(e)(n)]\}$, where $CTIME_1(e) = TIME_1(e) \circ P_e^{-1}$, define the same set of intervals as $\{[TIME_1(e)(n), CTIME_2(e)(n)]\}$.

An event *e* is called single occurrent if $TIME_1(e)(n) \geq TIME_2(e)(n-1)$, $n \geq 1$.

Example 2

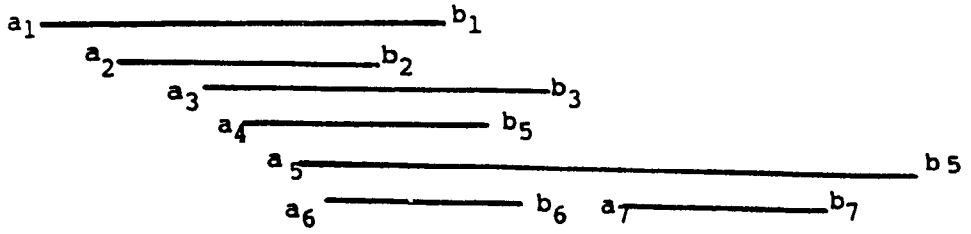

Figure 2

The occurrences of e are the intervals $[a_n, b_n]$, $n \geq 1$; Figure 2 shows the first seven occurrences. The sequences and the bijective map are:

$$\text{TIME}_1(e) \leftrightarrow \{a_1, a_2, a_3, a_4, a_5, a_6, a_7 \ldots\}$$

$$\text{TIME}_2(e) \leftrightarrow \{b_2, b_1, b_4, b_3, b_6, b_7, b_5, \ldots\}$$

$P_e : \mathbb{N} \rightarrow \mathbb{N}$,

$$P_e(1) = 2,\ P_e(2) = 1,\ P_e(3) = 4$$

$$P_e(4) = 3,\ P_e(5) = 7,\ P_e(6) = 5,$$

$$P_e(7) = 6,\ \ldots\ .$$

The next two examples show the usefulness and the expressive power of the formalism in the specification of timing constraints.

Example 3

In robotics certain global sensor variables must be bound to particular sensors. The value of the sensor variable at any instant is the current measurement of the sensor. Such values may have to be monitored continuously or once measured may remain constant over a period of time. Thus, if we want to state that sensor variables are piecewise constants (step functions) with periodicity δ, we can express as:

$$\forall n \in \mathbb{N},\ \mathrm{TIME}_1(\mathrm{ASSIGN}(s))(n+1) = \mathrm{CTIME}_2(\mathrm{ASSIGN}(s))(n) + \delta.$$

Hence, $\forall t \in [\mathrm{CTIME}_2(\mathrm{ASSIGN}(s))(n),\ \mathrm{TIME}_1(\mathrm{ASSIGN}(s))(n+1)]$ the value of s, namely VALUE(s)(n) is a constant.

Example 4

Effector variables denote the parameters of a robot's end effectors and a change in the value of this variable causes the robot to move. Since the end effector should move continuously, the function $y = f(x)$ describing the motion should be computed in real time; that is, $y_n = f(x_n)$ where y_n, x_n are n-th values computed for y and x respectively. However, a response time δ between the receipt of the value x and the sending of the corresponding value y (to the end effector) is frequently assumed. Three different interpretations are possible:

a) The n-th value of y must be computed from the n-th value of x:

$\forall n \in \mathbb{N}$, $VALUE(y)(n) = f(VALUE(x)(n))$ and

$CTIME_2(ASSIGN(x))(n) \leq TIME_1(ASSIGN(y))(n) \leq CTIME_2(ASSIGN(x))(n) + \delta$

b) The value of y may be obtained at a lower frequency than the value of x, but a value of y cannot be computed from a value of x issued at time exceeding δ :

$\forall n \in \mathbb{N}, \exists m \in \mathbb{N},$

$VALUE(y)(n) = f(VALUE(x)(m)),$

$CTIME_2(ASSIGN(x))(m) \leq TIME_1(ASSIGN(y))(n) \leq CTIME_2(ASSIGN(x))(m) + \delta.$

c) The value of y must be computed more frequently than x.

$\forall m \in \mathbb{N}, \exists n \in \mathbb{N},$

$VALUE(y)(n) = f(VALUE(x)(m)),$

$CTIME_2(ASSIGN(x))(m) \leq TIME_2(ASSIGN(y))(n) \leq CTIME_2(ASSIGN(x))(m) + \delta.$

3.1.2 An Algebraic Structure for E

An algebraic system is a set together with certain relations and operations defined on the set. An ordered set as well as the set of complex numbers are simple examples of algebraic systems. In this

section we shall first make E an algebraic system and then characterize its properties.

Two events $e, f \in E$ are equal, written $e = f$, if $TIME_1(e) = TIME_1(f)$ and $CTIME_2(e) = CTIME_2(f)$. Note that two equal events may have different effects in the system and hence are not identical. For example, if for two variables x, y in the system, $TIME_1(ASSIGN(x))(n) = TIME_1(ASSIGN(y))(n)$ and $CTIME_2(ASSIGN(x))(n) = CTIME_2(ASSIGN(y))(n)$ then the two events of assigning values to x and y are equal but not identical events. In short, equal events simultaneously occur all the time.

For two events e and f in E, denote $e \leq f$, if $\forall n \in \mathbb{N}$, the start and completion times of the n-th occurrence of e do not happen earlier than the start and completion times of the n-th occurrence of f. Notice that the start time of e may happen before the completion time of f. Hence

$$e \leq f \quad \text{if } [TIME_1(e) \geq TIME_1(f)] \wedge [CTIME_2(e) \geq CTIME_2(f)]$$

At first, this notation '$\leq$' on E may look counter intuitive. However, interpret "$e \leq f$" as a directed edge from e to f (with e above f in Hasse diagram) and "greater than or equal to" on the real line drawn vertically.

Based on the definition of "$\leq$" and our discussion on time functions,

we state the next theorem.

Theorem 1

i) $(E, \leq)$ is a poset

ii) Let INC = $\{f | f : \overline{\mathbb{N}} \rightarrow \overline{\Pi}$, f is monotonic non-decreasing$\}$

and $L = \{(f_1, f_2, P) \mid f_1, f_2 \in INC, P : \overline{\mathbb{N}} \rightarrow \overline{\mathbb{N}}$, P is bijective,

$f_2 \geq f_1, f_2 \circ P \geq f_1\}$.

The poset $(E, \leq)$ is isomorphic to the algebraic system L.

Two events e and f in E are not related by $\leq$ if and only if there is an interval in the sequence characterizing e (f) which covers the corresponding interval characterizing f (e). That is, $\exists$ n such that

either $TIME_1(e)(n) < TIME_1(f)(n) < CTIME_2(f)(n) < CTIME_2(e)(n)$

or $TIME_1(f)(n) < TIME_1(e)(n) < CTIME_2(e)(n) < CTIME_2(f)(n)$.

In order to compare every pair of elements in $(E,\leq)$, we define glb, the greatest lower bound and lub, the least upper bound for events in E. For two events $e, f \in E$, define $g \in E$, g = glb(e, f):

$TIME_1(g) = \max\{TIME_1(e), TIME_1(f)\}$

$CTIME_2(g) = \max\{CTIME_2(e), CTIME_2(f)\}$.

Both $TIME_2(g)$ and P_g are obtained from $CTIME_2(g)$:

$TIME_2(g)$ is the sorted sequence of $CTIME_2(g)$ and P_g is the bijection forcing the sorted sequence.

Similarly, we define h = lub(e, f) :

$\text{TIME}_1(h) = \min\{\text{TIME}_1(e), \text{TIME}_1(f)\}$

$\text{CTIME}_2(h) = \min\{\text{CTIME}_2(e), \text{CTIME}_2(f)\}$.

The functions $\text{TIME}_2(h)$ and P_h are obtained from the sorted sequence of $\text{CTIME}_2(h)(n)$, $n \geq 1$.

It is easy to see that (E, $\leq$, glb, lub) is a lattice having a greatest element. The lattice is not complete because it does not have a least element.

Since event occurrences need not overlap, we now study the structure of E under a strict precedence relation '<<' (follows). We say that an event e 'follows' an event f and write $e << f$, if $\forall n \in \mathbb{N}$, the start time of the n-th occurrence of e is greater than or equal to the completion time of the n-th occurrence of f :

$e << f$ if $\text{CTIME}_2(f) \leq \text{TIME}_1(e)$

The next theorem gives two properties of <<.

Theorem 2

(i) $e << f \rightarrow e \leq f$

(ii) (E, <<) is an irreflexive poset.

Proof

(i) $e << f \Rightarrow \text{TIME}_1(e) \geq \text{CTIME}_2(f)$

$\Rightarrow [\text{TIME}_1(e) \geq \text{TIME}_1(f)] \wedge [\text{CTIME}_2(e) \geq \text{CTIME}_2(f)]$

$\Rightarrow e \leq f$.

(ii) The relation $<<$ is irreflexive. Let d_e and d_f denote the sequences of duration of events e and f respectively. We have,

$$(e << f) \wedge (f << e) \Rightarrow [TIME_1(e) \geq TIME_1(f) + d_f] \wedge [TIME_1(f) \geq TIME_1(e) + d_e]$$

$$\Rightarrow TIME_1(e) \geq TIME_1(e) + d_e + d_f$$

$$\Rightarrow d_e = d_f = 0.$$

$$\Rightarrow TIME_1(e) = TIME_1(f) = TIME_2(e) = TIME_2(f)$$

$$\Rightarrow e = f.$$

Hence $<<$ is antisymmetric. It is easy to prove that $<<$ is transitive. This proves that (E, $<<$) is an irreflexive poset.

In general, any two events e, f in E need not have a glb in (E,$<<$); however, in later section, we give a criteria to be satisfied by e, f for glb(e,f) (under the relation $<<$) to be a member of E.

3.1.3 Operators, Relations, Counters and Properties

In this section we give several operators and time dependent functions and then prove new theorems characterizing events in our generalized formalism.

3.1.3.1 Delay Relation and Shift Function

In contrast to the original model of Caspi [Caspi 86], the generalized model admits a relation as well as a function to describe delayed events. For events $e \in E$ such that $P_e = I$ and for $\delta \geq 0$, we define the delay relation $\Delta_\delta(e)$ as follows :

$(e,f) \in \Delta_\delta(e)$ if $P_f = I$ and $TIME_1(f) = TIME_2(e) + \delta$.

It is clear that $f << e$ if $(e,f) \in \Delta_\delta(e)$.

The shift function S_δ, $\delta \geq 0$ is defined on E by

$S_\delta(e) = f$ if $TIME_1(f) = TIME_1(e) + \delta$ and $TIME_2(f) = TIME_2(e) + \delta$.

It is clear that S_δ shifts every occurrence of e by a constant amount δ and hence $S_\delta(e) \leq e$. If δ exceeds the maximum duration of all occurrences of e, $S_\delta(e) << e$.

3.1.3.2 Subevents and Subsequences Relations

An event f is called a subevent of e, $f \subseteq e$, if there exists an increasing function $r \in INC$ such that

$TIME_1(f) = TIME_1(e) \circ r$ and

$CTIME_2(f) = CTIME_2(e) \circ r$.

Example 5

In Figure 3, the occurrences of an event e and a subevent f are shown.

$TIME_1(e) \leftrightarrow (a_1, a_2, a_3, a_4, a_5, a_6)$

$TIME_2(e) \leftrightarrow (b_1, b_3, b_2, b_5, b_4, b_6)$

$P_e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 2 & 5 & 4 & 6 \end{pmatrix}$

$TIME_1(f) = (a_2, a_4, a_5)$

$TIME_2(f) = (b_2, b_5, b_4)$

$r : r(1) = 2, r(2) = 4, r(3) = 5$

It is easy to verify that

$\text{TIME}_1(f) = \text{TIME}_1(e) \circ r$

$\text{CTIME}_2(f) = \text{CTIME}_2(e) \circ r.$

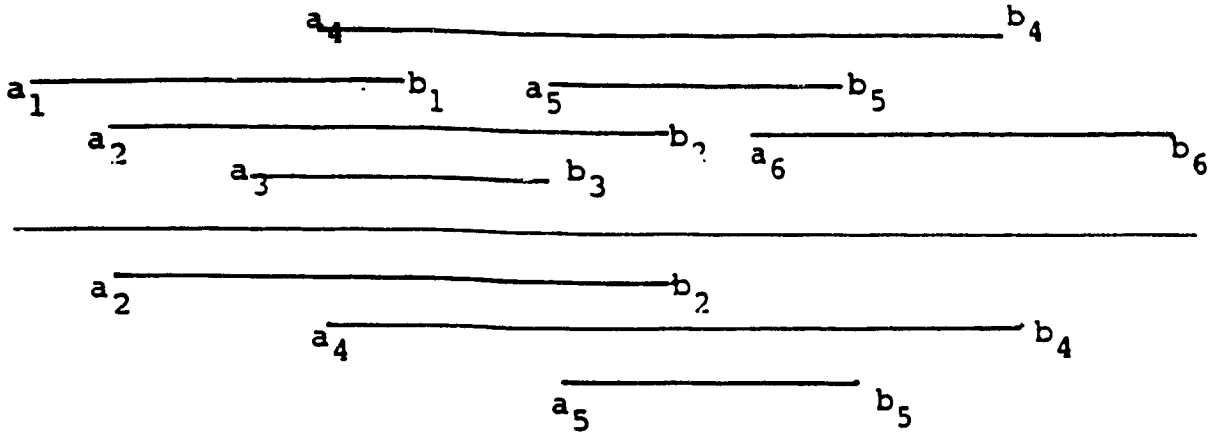

Figure 3

Theorem 3

$f \subseteq e \Rightarrow f \leq e$, iff $P_e = I$.

Proof

Let $P_e = I$. Then,

$f \subseteq e \Rightarrow [\text{TIME}_1(f)(n) = \text{TIME}_1(e)(m)] \wedge$

$[\text{CTIME}_2(f)(n) = \text{CTIME}_2(f)(m)],\ m \geq n$

$\Rightarrow [\text{TIME}_1(f)(n) \geq \text{TIME}_1(e)(n)] \wedge$

$[\text{CTIME}_2(f)(n) \geq \text{CTIME}_2(f)(n)]$

$\Rightarrow e \leq f.$

Conversely, assume that $f \subseteq e \Rightarrow f \leq e$, $e, f \in E$. That is, every subevent f of e is related to e by the relation $\leq$. If $P_e = I$ is not true, there exists a smallest k (> 0) such that

$$CTIME_2(e)(k + 1) < CTIME_2(e)(k).$$

Define f such that, for j, $1 \leq j \leq k - 1$,

$$TIME_1(f)(j) = TIME_1(e)(j)$$

$$CTIME_2(f)(j) = CTIME_2(e)(j)$$

$$TIME_1(f)(k) = TIME_1(e)(k + 1)$$

$$CTIME_2(f)(k) = CTIME_2(e)(k + 1).$$

Now, $f \subseteq e$ holds but $f \leq e$ is false. Hence $P_e = I$ must hold.

3.1.3.3 Counters

Loose and strict counters are inverse functions to the time functions. Using the Galois representation results [Sanchis 77], we know that $TIME_1(e)$ and $TIME_2(e)$ are infinitely supremum and infinitely infimum distributive functions. That is, for any subset E_1 of E,

$$\sup\{TIME_1(e) \mid e \in E_1\} = TIME_1\{\sup(e) | e \in E_1\}$$

and

$$\inf\{TIME_2(e) \mid e \in E_1\} = TIME_2\{\inf(e) | e \in E_1\}$$

It is known that such functions admit inverses; that is, there are two functions $TIME_1(e)^-$, $^-TIME_1(e)$ which represent *loose* and *strict* counters:

$TIME_1(e)^- \equiv LCOUNT_1(e) : \overline{\Pi} \rightarrow \overline{\mathbb{N}}$ where

$LCOUNT_1(e)(t)$ is the number of initiations of e up to and

including the time t.

$^{-}\text{TIME}_1(e) \equiv \text{COUNT}_1(e) : \overline{\Pi} \rightarrow \overline{\mathbb{N}}$, where

$\text{COUNT}_1(e)(t)$ is the number of initiations of e strictly before t.

In the same fashion, we can define two inverse functions for $\text{TIME}_2(e)$:

$\text{TIME}_2(e)^{-} \equiv \text{LCOUNT}_2(e) : \overline{\Pi} \rightarrow \overline{\mathbb{N}}$, where

$\text{LCOUNT}_2(e)(t)$ is the number of completed occurrences of e up to and including t.

$^{-}\text{TIME}_2(e) \equiv \text{COUNT}_2(e) : \overline{\Pi} \rightarrow \overline{\mathbb{N}}$, where

$\text{COUNT}_2(e)(t)$ is the number of completed occurrences of e strictly before t.

The next theorem summarizes the precedence relations on counters induced by the precedence relations on the events E.

Theorem 4

(1) For $e \in E$,

$\text{COUNT}_1(e) \leq \text{LCOUNT}_1(e)$ and $\text{COUNT}_2(e) \leq \text{LCOUNT}_2(e)$

(2) If $e, f \in E$ and $f \leq e$ then the precedence among the counters is shown in Fig. 4, where x ... $\rightarrow$ y means $x \leq y$.

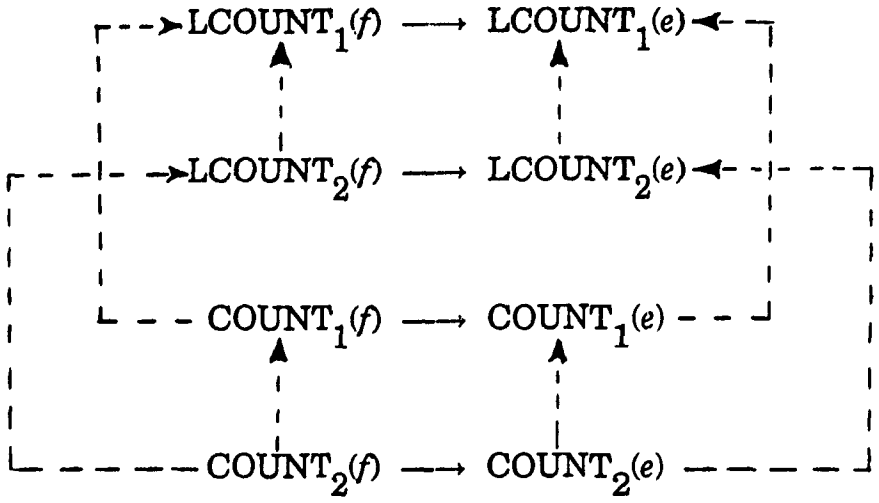

Figure 4

Proof

(1) follows from the definition.

Proof of (2): We will prove

$$COUNT_2(f) \leq COUNT_1(f) \leq COUNT_1(e) \leq LCOUNT_1(e)$$

and

$$COUNT_2(f) \leq LCOUNT_2(f) \leq LCOUNT_2(e) \leq LCOUNT_1(e)$$

and leave the rest to the reader. It is sufficient to prove that

$$COUNT_1(f) \leq COUNT_1(e),\ LCOUNT_1(f) \leq LCOUNT_1(e)$$

$$COUNT_2(f) \leq COUNT_2(e),\ LCOUNT_2(f) \leq LCOUNT_1(e)$$

Let $COUNT_1(f)(t) = k$, $t > 0$; that is, there are k initiations of f strictly before time t. Hence the k-th initiation must have occurred at time $t' < t$; that is,

$$\text{TIME}_1(f)(k) = t', \; t' < t.$$

But $f \leq e \Rightarrow \text{TIME}_1(f)(k) \geq \text{TIME}_1(e)(k), \; \forall k \in \mathbb{N}$

$\Rightarrow \text{TIME}_1(e)(k) \leq t' < t$

$\Rightarrow \text{COUNT}_1(e)(t) \geq k = \text{COUNT}_1(f)(t).$

Next let us prove $\text{COUNT}_2(f) \leq \text{COUNT}_2(e)$. If $\text{COUNT}_2(f)(t) = k$, $t > 0$, then it means that the event f has terminated k times prior to time t. Hence

$$\text{CTIME}_2(f)(k) = t', \; t' < t$$

But $f \leq e \Rightarrow \text{CTIME}_2(f)(k) \geq \text{CTIME}_2(e)(k), \; \forall k \in \mathbb{N}$

$\Rightarrow \text{CTIME}_2(e)(k) \leq t' < t$

$\Rightarrow \text{COUNT}_2(e)(t) \geq k = \text{COUNT}_2(f)(t).$

Similar proof applies to other cases.

In general, $[\text{COUNT}_1(e) \leq \text{COUNT}_1(f)]$ and $[\text{COUNT}_2(e) \leq \text{COUNT}_2(f)]$ does not imply $e \leq f$; however, if $P_e = I$ and $P_f = I$ then the result is true.

There are several precedence relations on the counters induced by the strict precedence operator << on E. For example, e << f implies:

$$COUNT_2(e) \leq COUNT_1(e) \leq COUNT_2(f) \leq COUNT_1(f)$$

and

$$LCOUNT_2(e) \leq LCOUNT_1(e) \leq LCOUNT_2(f) \leq LCOUNT_1(f).$$

However, $[COUNT_1(e) \leq COUNT_2(f)] \rightarrow e \ll f$ if $P_f = I$.

We state without proof the next theorem expressing delayed and shifted events in terms of original event counters.

Theorem 5

For $e \in E$, $\delta > 0$, $f = S_\delta e$

$$COUNT_1(f) = COUNT_1(e) \circ (I - \delta)$$

$$COUNT_2(f) = COUNT_2(e) \circ (I - \delta)$$

$$LCOUNT_1(f) = LCOUNT_1(e) \circ (I - \delta)$$

$$LCOUNT_2(f) = LCOUNT_2(e) \circ (I - \delta),$$

where I is the identity function. If $(e,f) \in \Delta_\delta(e)$, then

$$COUNT_1(f) = COUNT_2(e) \circ (I - \delta)$$

$$LCOUNT_1(f) = LCOUNT_2(e) \circ (I - \delta).$$

3.2 Sum of Events

If an event g is initiated whenever e is initiated or f is initiated and the initiated occurrence of g terminates when the corresponding occurrence of e or f terminates, then g is called the sum of events e and f. Hence it follows that the $TIME_j$ sequence of g is the merge sequence of the $TIME_j$ sequences of e and f; a more

formal definition follows Example 6. Based on the merge sequence, a definition of the sum in terms of the counters can also be given: g is the sum $e + f$ if

$$\mathrm{LCOUNT}_1(g) = \mathrm{LCOUNT}_1(e) + \mathrm{LCOUNT}_1(f)$$

$$\mathrm{LCOUNT}_2(g) = \mathrm{LCOUNT}_2(e) + \mathrm{LCOUNT}_2(f)$$

or equivalently

$$\mathrm{COUNT}_1(g) = \mathrm{COUNT}_1(e) + \mathrm{COUNT}_1(f)$$

$$\mathrm{COUNT}_2(g) = \mathrm{COUNT}_2(e) + \mathrm{COUNT}_2(f).$$

The next example illustrates the computation of $e + f$ from e and f.

Example 6

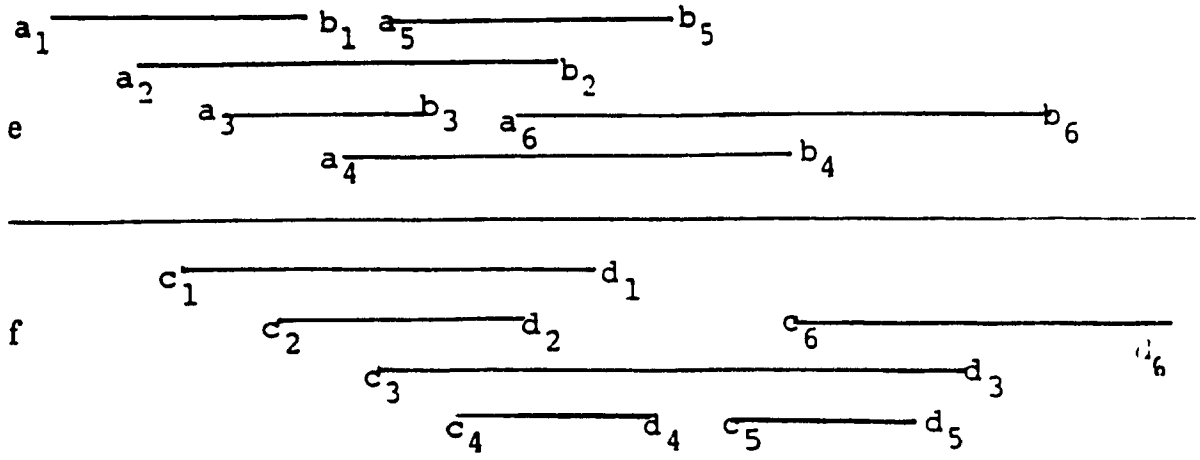

Figure 5

From the occurrences of events e and f shown in Fig. 5, we have

$$\mathrm{TIME}_1(e) \leftrightarrow \{a_1, a_2, a_3, a_4, a_5, a_6\}$$

$TIME_1(f) \leftrightarrow \{c_1, c_2, c_3, c_4, c_5, c_6\}$

$TIME_2(e) \leftrightarrow \{b_1, b_3, b_2, b_5, b_4, b_6\}$

$TIME_2(f) \leftrightarrow \{d_2, d_1, d_4, d_5, d_3, d_6\}$,

$$P_e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 2 & 5 & 4 & 6 \end{pmatrix}$$

$$P_f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 5 & 3 & 4 & 6 \end{pmatrix}$$

Let $g = e + f$.

$TIME_1(g) = \{a_1, a_2, c_1, a_3, c_2, a_4, c_3, a_5, c_4, a_6, c_5, c_6\}$

$TIME_2(g) = \{b_1, b_3, d_2, b_2, d_1, d_4, b_5, b_4, d_5, d_3, b_6, d_6\}$

$$P_g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 4 & 5 & 2 & 3 & 8 & 10 & 7 & 6 & 11 & 9 & 12 \end{pmatrix}$$

In general, we compute $g = e + f$ as described below:

1) $TIME_1(g) = \text{merge}\{TIME_1(e), TIME_1(f)\}$

Among the occurrences having the same starting time, the order is defined depending on their corresponding completion time.

$TIME_2(g) = \text{merge}\{TIME_2(e), TIME_2(f)\}$

2) Define the functions (M_1 is total, M_2 is partial)

$$M_1 : \mathbb{N} \rightarrow \{1, 2\} \times \mathbb{N}$$

$$M_2 : \{1, 2\} \times \mathbb{N} \rightarrow \mathbb{N}$$

such that

$M_1(n)$ = (1, i), if the n-th element in $TIME_1(g)$ sequence is the i-th element from $TIME_1(e)$ sequence.

= (2, j), if the n-th element in $TIME_1(g)$ sequence is the j-th element from $TIME_1(f)$ sequence.

$M_2(1, i) = n$, if the n-th element in $TIME_2(g)$ sequence is the i-th element from $TIME_2(e)$;

$M_2(2, j) = n$, if the n-th element in $TIME_2(g)$ sequence is the j-th element of the sequence $TIME_2(f)$.

The bijection P_g is defined by,

$P_g(n) = M_2(1, P_e(i))$, if $M_1(n) = (1, i)$

$= M_2(2, P_f(j))$, if $M_1(n) = (2, j)$.

It is clear from the definition and Example 6, that $e \subseteq e + f$ and $f \subseteq e + f$.

Further for $e, f \in E$, if $P_{e+f} = I$ then glb(e,f) and lub(e,f) can be defined in the poset (E,<<) as follows :

g = glb(e,f) where

$TIME_1(g) = \max \{TIME_2(e), TIME_2(f)\}$,

$TIME_2(g) = TIME_1(g)$,

and $P_g = I$.

h = lub(e,f), where

$TIME_1(h) = \min \{TIME_1(e), TIME_1(f)\}$,

$TIME_2(h) = TIME_1(h)$,

and $P_h = I$.

Then, $g, h \in$ (E, <<).

A complete characterization of the sum structure is given by five theorems proved in the next section. Now, we shall show that the extended formalism admits several other relations and functional

dependencies.

For every j $\in$ **N**, we define an integer event j as

$$\mathrm{TIME}_1(j)(n) = \begin{cases} -\infty, & n \leq j \\ +\infty & n > j \end{cases}$$

$$\mathrm{TIME}_2(j)(n) = \mathrm{TIME}_1(j)(n)$$

and

P_j = I, the identity function.

In particular the integer event 1 is characterized by ($\mathrm{TIME}_1(1)$, $\mathrm{TIME}_1(1)$, I), where

$$\mathrm{TIME}_1(1) \leftrightarrow (-\infty, +\infty, +\infty, \ldots)$$

$$\mathrm{TIME}_2(1) \leftrightarrow (-\infty, +\infty, +\infty, \ldots)$$

Thus for any event $e \in E$ with

$$\mathrm{TIME}_1(e) \leftrightarrow (a_1, a_2, a_3, \ldots)$$

$$\mathrm{TIME}_2(e) \leftrightarrow (b_1, b_2, b_3, \ldots)$$

we have $f = e + 1$ defined by the functions

$$\mathrm{TIME}_1(f) = (-\infty, a_1, a_2, \ldots)$$

$$\mathrm{TIME}_2(f) = (-\infty, b_1, b_2, \ldots)$$

$$P_f(n) = \begin{cases} 1, & n = 1 \\ P_e(n - 1) + 1, & n > 1 \end{cases}$$

It is easy to show that

$$e \leq e + 1 \text{ iff } P_e = I,$$

and $\quad e \ll e + 1$ iff e is single-occurrent.

Several useful functions arise from functional compositions of counter functions with those already defined. Three of them are described next.

[F1] A variable in the system has a sequence of values. The value of x at any instant should be dependent on the event ASSIGN(x), especially its completion time.

Thus there are two functions

$CURRENT(x), LCURRENT(x) : \overline{\Pi} \rightarrow DOM(x)$

defined by

$CURRENT(x) = VALUE(x) \circ COUNT_2(ASSIGN(x))$

$LCURRENT(x) = VALUE(x) \circ LCOUNT_2(ASSIGN(x))$.

Hence if $e = ASSIGN(x)$ and $COUNT_2(e)(t) \neq LCOUNT_2(e)(t)$, then LCURRENT(x)(t) is a more recent value of x than CURRENT(x)(t). Moreover if y = f(x), then CURRENT(y) and LCURRENT(y) will have three interpretations depending on the renewal period as explained in Example 4.

[F2] For every event $e \in E$, we define $LAST_1(e)$, $LAST_2(e)$ and $NEXT_1(e)$, $NEXT_2(e)$ giving the last and next occurrences of e:

$LAST_1, LAST_2, NEXT_1, NEXT_2 : E \rightarrow (\overline{\Pi} \rightarrow \overline{\Pi})$

$LAST_1(e)(t) = t_1$, if the last initiated time of the event e strictly before t was t_1.

$LAST_2(e)(t) = t_2$, if the last completed time of the event e

strictly before t was t_2.

$NEXT_1(e)(t) = t_3$, if t_3 is the first initiation time of e strictly after time t.

$NEXT_2(e)(t) = t_4$, if t_4 is the first completion time of e strictly after time t.

It is easy to see that

$LAST_1(e) = TIME_1(e) \circ COUNT_1(e)$

$LAST_2(e) = TIME_2(e) \circ COUNT_2(e)$

$NEXT_1(e) = TIME_1(e) \circ LCOUNT_1(e + 1)$

$NEXT_2(e) = TIME_2(e) \circ LCOUNT_2(e + 1)$.

Some applications require the starting time of the last completed occurrence of e strictly before t or/and the starting time of the first completed occurrence of e strictly after t. These are given by

$$SLAST_2(e) = TIME_1(e) \circ P_e^{-1} \circ COUNT_2(e)$$

and

$$SNEXT_2(e) = TIME_1(e) \circ P_e^{-1} \circ LCOUNT_2(e + 1).$$

By relaxing strictly before (or strictly after) to include the time of observation, we get the following six functions:

$LLAST_1(e) = TIME_1(e) \circ LCOUNT_1(e)$

$LLAST_2(e) = TIME_2(e) \circ LCOUNT_2(e)$

$$SLLAST_2(e) = TIME_1(e) \circ P_e^{-1} \circ LCOUNT_2(e)$$

$LNEXT_1(e) = TIME_1(e) \circ COUNT_1(e + 1)$

$LNEXT_2(e) = TIME_2(e) \circ COUNT_2(e + 1)$

$SLNEXT_2(e) = TIME_1(e) \circ P_e^{-1} \circ COUNT_2(e + 1)$

These functions show the importance of integer event and event sum in their definition.

[F3] A condition C is a function from $\overline{\Pi}$ to {true, false} with $C(-\infty)$ = true. If x is a variable with DOM(x) = {true, false}, then CURRENT(x) and LCURRENT(x) are conditions. For e, f $\in$ E,

$(LAST_1(e) = LAST_2(f))$ and $(COUNT_2(e) = COUNT_2(f))$ are conditions.

An event f which is initiated whenever C is true at the completion of e, where P_e = I, is denoted by e | C. Thus a formal definition is the following:

Let

$X = \{f \mid f \in E, \exists\ r : \mathbb{N} \rightarrow \mathbb{N}$ such that

$[r \circ COUNT_2(e) = COUNT_1(f)] \wedge [C \circ LAST_1(f) = true] \wedge$

$[[LAST_2(e) \neq LAST_1(f)] \rightarrow [C \circ LAST_2(e) = false]]\}$.

Now, every member of X is an event e | C; that is, X denotes a class of events which begin whenever e finishes and C is true.

It is clear that if $f = e$ | C and $g = e$ | $\sim$C then

1. $COUNT_2(e) = COUNT_1(f) + COUNT_1(g)$
2. $C \circ LAST_1(f) = \sim C \circ LAST_1(g) = true$.

Let $e \mid_s C$ denote an event f which starts whenever e starts and the condition C is true. We define

$$Y = \{f \mid f \in E, \exists\ r : \mathbb{N} \rightarrow \mathbb{N} \text{ such that}$$
$$[r \circ COUNT_1(e) = COUNT_1(f)] \wedge [C \circ LAST_1(f) = true] \wedge$$
$$[[LAST_1(e) \neq LAST_1(f)] \rightarrow [C \circ LAST_1(e) = false]]\}.$$

Any member of Y is an event $e \mid_s C$. If $f = e \mid_s C$ and $g = e \mid_s \sim C$ then

$$COUNT_1(e) = COUNT_1(f) + COUNT_1(g).$$

Based on these functions we can give a rigorous interpretation to expression evaluations in real-time systems. Let $g = g(x_1, x_2, ..., x_n)$ be a function of n variables. Denoting $LCURRENT(x_i)(t)$ by y_i, we can extend LCURRENT to g as

$$LCURRENT(g)(t) = g(y_1, y_2, ..., y_n),\ t \in \overline{\Pi}.$$

When time explicitly occurs as a parameter in g, $g = g(x_1, x_2, ..., x_n, T)$, then for a $t \in \overline{\Pi}$, $\bar{g}(T) = g(y_1, y_2, ..., y_n, T)$ is a function of time T. As t varies over $\overline{\Pi}$, $\{\bar{g}(T)\}$ gives a collection of time dependent functions exhibiting different behavior over $T \in [T_1, T_2]$. For example, if $g = aT + bT^2$, $a_i = LCURRENT(a)(t_i)$, $b_i = LCURRENT(b)(t_i)$, then $g_1 = a_1T + b_1T^2$ and $g_2 = a_2T + b_2T^2$ are two functions of T which when evaluated over $[T_1, T_2]$ may produce different effects.

Continuous monitoring of significant events give rise to exception handlings in real time systems. Thus, if S_1 must be activated when a condition C becomes true, S_1 must remain active when C remains true and S_1 must be aborted and S_2 simultaneously activated when C becomes false, the specification can be given as follows :

Let *clock* be the event of ticking of the system clock. Let s_1 and s_2 be the events denoting the invocation of S_1 and S_2 respectively.

$$s_1 = clock \mid C \wedge [COUNT_2(s_1) = COUNT_1(s_1)]$$

$$s_2 = clock \mid \sim C \wedge [COUNT_2(s_2) = COUNT_1(s_2)]$$

$$[[COUNT_1(s_1) \circ LAST_1(s_2) \neq COUNT_2(s_1) \circ LAST_1(s_2)]$$

$$\rightarrow [TIME_2(s_1) \circ COUNT_1(s_1) \circ LAST_1(s_2) = LAST_1(s_2)]]$$

During an occurrence of the event s_1, if C becomes false then we define the completion time of that occurrence to indicate that s_1 is aborted.

In a recent paper Clark [Clark 88] has given a similar formal specification of watchdog timer based upon CSP and the *me too* [Bennett 88] method.

3.3 The Structure of Sum of Events

For our discussion in this section we let $e_i = TIME_1(e)(i)$, $e_i' = CTIME_2(e)(i)$, $i \geq 1$, $e \in E$. Moreover we identify an event e with these two time sequences and let $dur(e_i)$ denote the duration of i-th

occurrence of e; that is, $dur(e_i) = CTIME_2(e)(i) - TIME_1(e)(i)$, $i \geq 1$.

For instantaneous events, the sum is order preserving; but this is not so in our model. The following theorems give a set of sufficient conditions for preserving order in the sum of events.

Theorem 6

For any three events $e, f, g \in E$, $e << f \Rightarrow (e + g) << (f + g)$ if for every i, $f_i' \leq g_i$, $g_i' \leq e_i$, and $P_f = P_g = I$, $P_{f+g} = I$.

Proof

Let $h = e + g$, $k = f + g$

Assume that for every i, $f_i' \leq g_i$, $g_i' \leq e_i$ holds and $P_f = P_g = I$, $P_{f+g} = I$. We have to prove that $e << f \Rightarrow (e + g) << (f + g)$.

Recall that the time sequences for sum of events is obtained by merging the individual time sequences. For any fixed i, if we let $X = \{h_1, h_2, ..., h_i\}$, $Y = \{k_1, k_2, ..., k_i\}$, depending upon whether $h_i(k_i)$ is a member of the sequence for $TIME_1(e)$ (or $TIME_1(f)$) or $TIME_1(g)$, we must consider several situations and prove $h_i \geq k_i'$, $i \geq 1$.

Case 1

Let $h_i = g_s$, $k_i = f_t$ for some s, t ≥ 1. Clearly, s is the number of g's in X and t is the number of f's in Y and $s \geq \lceil \frac{i}{2} \rceil$, $t \geq \lceil \frac{i}{2} \rceil$. If s = t, there is nothing to prove. If s > t, then $f_s \notin Y$

due to the fact that the sequences are monotonic non-decreasing. Hence $f'_t \leq f'_s \leq g_s = h_i$. This proves $k'_i \leq h_i$. If $s < t$ then $s' > t'$, where $s' = i - s$, $t' = i - t$. Since Y has $(i - t)$ g's, $g_{i-t+1} \notin Y$. This implies that $f_t \leq g_{i-t+1}$. Since $P_k = I$, it follows that $f'_t \leq g'_{i-t+1}$. Moreover $(i - t) < (i - s)$ implies $(i - t + 1) \leq (i - s)$; the fact that X has $(i - s)$ e's implies $e_{i-t+1} \in X$. Hence $e_{i-t+1} \leq g_s$; that is,

$$k'_i = f'_t \leq g'_{i-t+1} \leq e_{i-t+1} \text{ (by hypothesis)}$$
$$\leq g_s = h_i.$$

Case 2

Let $h_i = e_s$ and $k_i = g_t$. If $s = t$, $e_s \geq g'_t$ by the hypothesis. If $s > t$, $g_s \notin Y$. Since $P_k = I$, $g'_t \leq g'_s$ and $g'_s \leq e_s$ by hypothesis. So it follows that

$$k'_i = g'_t \leq e_s = h_i.$$

If $s < t$, because of the fact that there are only $(i - t)$ f's in Y, $f_{i-t+1} \notin Y$. Since $P_k = I$, $g'_t \leq f'_{i-t+1}$. The fact that there are $(i - s)$ g's in X and $(i - t + 1) \leq (i - s)$ show that $g_{i-t+1} \in X$; that is, $g_{i-t+1} \leq e_s$. Combining these results, we get

$$k'_i = g'_t \leq f'_{i-t+1} \leq g_{i-t+1} \leq e_s = h_i$$

Case 3

Let $h_i = e_s$ and $k_i = f_t$. If $s = t$, then since $e \ll f$, $f'_s \leq e_s$. If $s > t$, then $e_s \geq e_t \geq f_t$. If $s < t$ and $g_s \notin Y$, then $f'_t \leq g'_s \leq$

e_s. If $s < t$ and $g_s \in Y$ then we claim that $g_t \in X$; under this claim, $e_s \geq g_t \geq f'_t$ and hence $h_i \geq k'_i$ follows. To prove the claim, notice that $k_i = f_t$ and $g_s \in Y$ implies that $i \geq s + t$. Since the number of e's in X (under this situation) is only s, the number of g's in X must be at least t; that is, $g_t \in X$.

Case 4

Let $h_i = g_s$ and $k_i = g_t$. We consider two subcases depending on whether or not e_t belongs to X.

If $e_t \in X$, then $e_t \leq g_s$. By using the hypothesis $g'_t \leq e_t$, it follows that

$$k'_i = g'_t \leq e_t \leq g_s = h_i.$$

If $e_t \notin X$, then we claim that $f_s \notin Y$; under this claim, $g'_t \leq f'_s \leq g_s$. That is, $k'_i \leq h_i$. Now to prove the claim, observe that $h_i = g_s$ and $e_t \notin X$ implies that $i < s + t$. Since $k_i = g_t$, the number of f's in Y is strictly less than s. This proves the claim that $f_s \notin Y$.

This completes the proof.

The event occurrences shown in Fig. 6 suggest that $e \leq f$ does not imply $e + g \leq f + g$; the next theorem provides a set of sufficient conditions for preserving the sum of events under the partial order $\leq$.

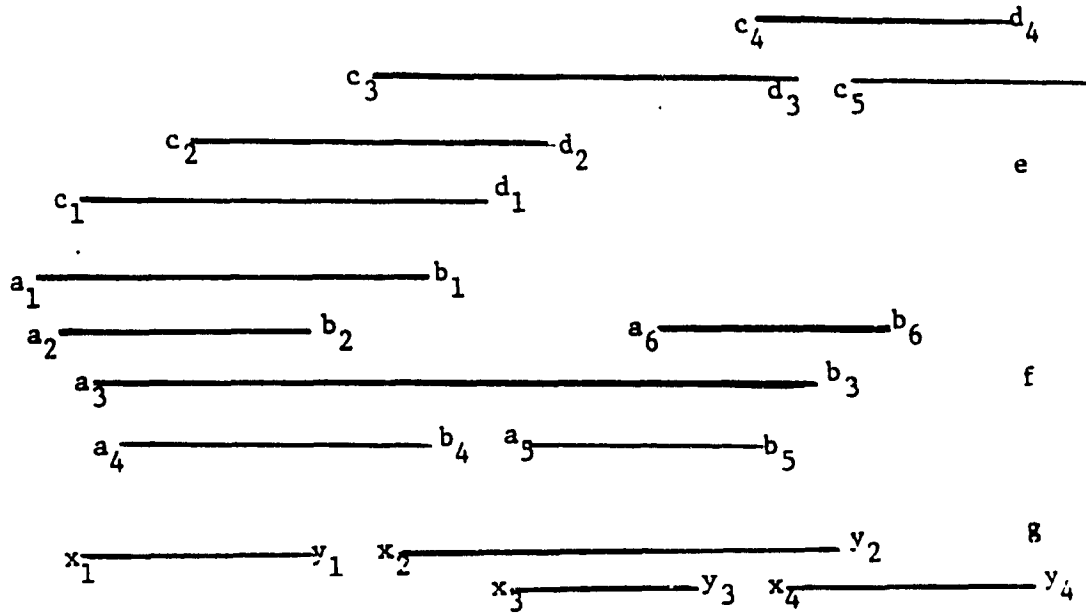

Figure 6

Theorem 7

Let MAX = max{dur(f_i)} and MIN = min{dur(e_i)}.

For $e, f, g \in E$, $e \leq f \Rightarrow e + g \leq f + g$ if MAX $\leq$ dur(g_i) $\leq$ MIN, $\forall i$ and P_g = I.

Proof

Let MAX $\leq$ dur(g_i) $\leq$ MIN, $i \geq 1$ and P_g = I, for some $g \in E$. Denote the sums $e + g$ and $f + g$ by h and k respectively. We will prove that $e \leq f \Rightarrow h \leq k$. It is sufficient to prove that,

$\forall i$, $h_i \geq k_i$ and $h_i' \geq k_i'$.

Let X = {h_1, h_2, ..., h_i} and Y = {k_1, k_2, ..., k_i}. Assume that X has r elements from $TIME_1(e)(n)$ and Y has s elements from $TIME_1(f)(n)$, $n \geq 1$. Since $e \leq f$, it follows that $r \leq s$; thus there are only two cases.

Case 1 $r < s$.

There are $(i - r)$ elements from $TIME_1(g)(n)$ in X and $(i - s)$ elements from $TIME_1(g)(n)$ in Y, $(i - r) > (i - s)$.

1.1 $k_i = f_s$, $h_i = e_r$.

Clearly, $k_i \leq g_{i-s+1} \leq g_{i-r} \leq h_i$.

$k_i' = k_i + dur(k_i)$

$= f_s + dur(f_s)$

Since MAX $\leq$ MIN,

$dur(f_s) \leq dur(e_r)$

Hence,

$f_s + dur(f_s) \leq e_r + dur(e_r) \leq h_i'$

1.2 $k_i = f_s$, $h_i = g_{i-r}$

Since $g_{i-s+1} \notin X$,

$k_i \leq g_{i-s+1} \leq g_{i-r} = h_i$

$k_i' = k_i + dur(f_s) < h_i + dur(g_{i-r})$,

Since $dur(g_{i-r}) \geq$ MAX

$\leq h_i + dur(h_i) = h_i'$.

1.3 $k_i = g_{i-s}$, $h_i = e_r$

$k_i = g_{i-s} < g_{i-r}$,

$\leq e_r = h_i$

$k_i' = k_i + dur(k_i)$

$= k_i + dur(g_{i-s})$

$\leq e_r +$ MIN $< e_r + dur(e_r)$

$\leq h_i'$

1.4 $k_i = g_{i-s}$, $h_i = g_{i-r}$

Since $P_g = I$ and $(i - r) > (i - s)$,

$g_{i-s}' \leq g_{i-r}'$

i.e., $k_i' \leq h_i'$

Case 2 $r = s$.

Once again there are four subcases.

2.1 $h_i = k_i = g_{i-r} = g_{i-s}$. It is easy to see that $h_i' = k_i'$.

2.2 $h_i = e_s$, $k_i = f_s$.

Since $e \leq f$, $e_s \geq f_s$ and $e_s' \geq f_s'$, $s \geq 1$

So the result follows.

2.3 $h_i = g_{i-s}$, $k_i = f_s$

$h_i = g_{i-s} \geq e_s \geq f_s = k_i$

Since $dur(g_i) \geq MAX$, $dur(g_{i-s}) \geq dur(f_s)$;

that is, $h_i' = h_i + dur(h_i)$

$\geq k_i + dur(g_{i-s})$

$\geq k_i + dur(f_s) = k_i + dur(k_i) = k_i'$

2.4 $h_i = e_s$, $k_i = g_{i-s}$

$h_i = e_s \geq g_{i-s} = k_i$

Since $dur(g_{i-s}) \leq MIN$, $dur(g_{i-s}) \leq dur(e_s)$

and hence $h_i' = h_i + dur(h_i)$

$= e_s + dur(e_s)$

$\geq g_{i-s} + dur(g_{i-s}) = k_i'$.

The next theorem gives sufficient conditions for preserving the sum of events under the partial order $<<$.

Theorem 8

Let $e << f$, $g << h$. If $P_{f+h} = I$, then $e + g << f + h$.

Proof

Let $p = e + g$ and $q = f + h$. It is sufficient to prove $\forall i \geq 1$, $p_i \geq q_i'$. For $i \geq 1$, let

$$X = \{p_1 , p_2 , ..., p_i\}$$

$$Y = \{q_1 , q_2 , ..., q_i\}.$$

Case 1

Let $p_i = e_s$, $q_i = f_t$.

If $s \geq t$, $p_i = e_s \geq e_t \geq f_t' = q_i'$, since $e << f$. If $s < t$, X has $(i - s)$ g's, Y has $(i - t)$ h's and $(i - t) < (i - s)$. Hence $(i - t + 1) \leq (i - s)$ and $h_{i-t+1} \notin Y$. Thus, $p_i = e_s \geq g_{i-s} \geq h_{i-s}' \geq h_{i-t+1}' \geq f_t' = q_i'$, since $P_{f+h} = I$.

Case 2

Let $p_i = e_s$ and $q_i = h_t$. If $f_s \notin Y$, due to $e << f$ we have $e_s \geq f_s'$ and due to $P_{f+h} = I$, $f_s' \geq h_t'$. Hence $e_s \geq h_t'$ is proved. If $f_s \in Y$, then X has $(i - s)$ g's, $(i - s) \geq t$. Hence $g_t \in X$ and $g << h$ implies $e_s \geq g_t \geq h_t'$.

Case 3

Let $p_i = g_s$, $q_i = f_t$. If $h_s \notin Y$, then due to $g \ll h$ and $P_{f+h} = I$, $g_s \geq h'_s \geq f'_t$. If $h_s \in Y$, $(i - s) \geq t$ implies $e_t \in X$ and $e \ll f$ implies $g_s \geq e_t \geq f'_t$.

Case 4

Let $p_i = g_s$, $q_i = h_t$. If $s \geq t$, due to $P_{f+h} = I$, it follows that $p_i = g_s \geq h'_s \geq h'_t = q'_i$. If $s < t$, then X has $(i - s)$ e's and Y has $(i - t)$ f's, $(i - s) > (i - t)$. It is clear that $(i - t + 1) \leq (i - s)$ and $f_{i-t+1} \notin Y$. Since $e \ll f$ and $P_{f+h} = I$, we have $p_i = g_s \geq e_{i-s} \geq f'_{i-s} \geq f'_{i-t+1} \geq h'_t = q'_i$.

Since i is arbitrary, $\forall i$ we have proved $p_i \geq q'_i$; that is, $e + g \ll f + h$.

If two events follow the event g, the following theorem gives sufficient conditions under which their sum would follow g.

Theorem 9

$\forall e, f, g \in E$, $e \ll g$ and $f \ll g \Rightarrow e + f \ll g$

if $TIME_2(g)(n) \leq TIME_1(e)(k)$, and $TIME_2(g)(n) \leq TIME_1(f)(k)$, n, k ≥ 1.

The proof is simple and is left to the reader.

The next and final theorem of this section proves the strict precedence additivity of the last theorem under subadditive property

of strict counters.

Theorem 10

For events $e, f, g \in E$, let

$COUNT_1(e) + COUNT_1(f) \leq COUNT_2(g)$.

Then $e << g$ and $f << g \Rightarrow e + f << g$ if $P_g = I$.

(The theorem asserts that for each occurrence of g there is utmost one occurrence of e or one occurrence of f but not both.)

Proof

Let $e << g$, $f << g$, $P_g = I$ and $a = e + f$. Since $P_g = I$,

$COUNT_2(g)(g_i') \leq (i - 1)$.

$COUNT_1(e)(t) + COUNT_1(f)(t) \leq COUNT_2(g)(t)$, $\forall t \in \overline{\Pi}$

So, for $t = g_i'$ we have

$COUNT_1(e)(g_i') + COUNT_1(f)(g_i') \leq i - 1$

Hence $a_i \geq g_i'$, $i \geq 1$. This proves that $a << g$.

CHAPTER 4

Specification and Proof of Correctness : Examples

The tools of the formalism introduced in Chapter 3 are applied here in formulating the specification and deriving the proof of correctness of four examples:

1) a distributed asynchronous bus arbiter design
2) a multiple copy update problem in distributed databases
3) assembling parts with two arms of a robot
4) the design of a navigation controller for multiple robot workplace

These problems are chosen to illustrate the different levels of applicability of the formalism we have proposed. Time dependent actions that arise in database examples are causal, and those arising in arbiter and robotics are due to (physical) real time constraints.

4.1 The Design of a Distributed Bus Arbiter

An asynchronous distributed arbiter is to be provided for n units U_1, U_2, ..., U_n, so that the bus connecting the units can be allocated exclusively to a unit according to a priority rule. Each unit U_i is associated with an arbitration element. Since there are no other centralized resources (such as clocks or shared memory), the units must cooperate among themselves. This is achieved through

communication among the arbitration elements and the knowledge of status of the bus and the name of the requested unit.

The communication among the arbitration elements and the final allocation of bus with emission of acknowledgement are done according to the four rules:

1) (Exclusivity) At each time, at most one unit is allowed to use the bus.

2) (Reactivity) The bus is allocated to a unit only if it requested the bus through its arbitration element.

3) (Priority) When the bus is free and is requested by more than unit, the unit with highest priority (say, the one having smallest index) is granted access to the bus.

4) (Promptness) At no time the bus can be both idle and requested (by a unit).

These requirements are formalized next; the summation sign Σ means $\sum_{i=1}^{n}$.

Problem Specification

Events

req_i : the event of U_i requesting the usage of bus.

ack_i : the event of emitting an acknowledgement to U_i by its arbitration element.

rel_i : the event of releasing the bus by U_i

acc_i : the event of U_i accessing the bus

dec : the event of decision making process of the arbitration elements.

We assume the following two conditions about the underlying communication system.

[A1] For $1 \leq i \leq n$, $rel_i \ll acc_i \ll ack_i$

[A2] Messages are received in the order in which they are sent.

Exclusivity

At any instant $t \geq 0$, the number of units owning the bus is at most 1:

$$0 \leq [\Sigma\ \mathrm{COUNT}_1(acc_i)(t) - \Sigma\ \mathrm{COUNT}_2(rel_i)(t)] \leq 1$$

That is,

$$\Sigma\ \mathrm{COUNT}_2(rel_i)(t) \leq \Sigma\ \mathrm{COUNT}_1(acc_i)(t) \leq \Sigma\ \mathrm{COUNT}_2(rel_i)(t) + 1.$$

Reactivity

Bus access is granted to a unit only after the unit requests for the usage of the bus:

$$ack_i \ll req_i, \text{ for } i,\ 1 \leq i \leq n$$

Priority

Access to the bus should be granted to U_i only if at the starting time of the corresponding decision making event, the bus

was free and no unit having higher priority than U_i requested for the usage of the bus:

For $1 \leq j < i \leq n$,

$COUNT_1(ack_j) \circ LAST_1(dec) \circ LAST_1(ack_i) = COUNT_2(req_j) \circ LAST_1(dec) \circ LAST_1(ack_i)$

Thus, for some $t > 0$, $t_1 = LAST_1(ack_i)(t)$ gives the instant of initiation of the last acknowledgement by the arbitration element of U_i. The instant $t_2 = LAST_1(dec)(t_1)$ is such that $t_2 < t_1$ and the decision making process was last initiated at that time. Now, $COUNT_1(ack_j)(t_2) = COUNT_2(req_j)(t_2)$, $j < i$, $1 \leq j \leq n$, asserts that no request from a unit of higher priority is outstanding at this instant. Hence the specification is a correct description of the priority requirement.

Promptness

At any time t, the bus cannot be both idle and requested.

i) The bus is busy at time t iff

$$\Sigma\ COUNT_1(acc_i)(t) = \Sigma\ COUNT_2(acc_i)(t) + 1$$
$$= \Sigma\ COUNT_2(rel_i)(t) + 1.$$

But, in order to have a feasible specification, we rewrite this as,

$$COUNT_1(dec)(t) = \Sigma\ COUNT_2(rel_i)(t) + 1.$$

ii) The bus is requested at time t iff

$$\Sigma\ \mathrm{COUNT}_2(req_i)(t) > \mathrm{COUNT}_1(dec)(t)$$

iii) The bus is not requested at time t iff the number of completed requests in the system equals the number of initiated decisions:

$$\Sigma\ \mathrm{COUNT}_2(req_i)(t) = \mathrm{COUNT}_1(dec)(t)$$

For the promptness condition, either (i) or (iii) must be true:

$$\forall t,\ \min\{\Sigma\ \mathrm{COUNT}_2(rel_i)(t) + 1 - \mathrm{COUNT}_1(dec)(t),$$
$$\Sigma\ \mathrm{COUNT}_2(req_i)(t) - \mathrm{COUNT}_1(dec)(t)\} = 0$$

$$\forall t,\ \min\{\Sigma\ \mathrm{COUNT}_2(rel_i)(t) + 1,\ \Sigma\ \mathrm{COUNT}_2(req_i)(t)\} = \mathrm{COUNT}_1(dec)(t)$$

Traveling Token Algorithm — A Formal Specification

Traveling token algorithm refers to the following simple sequential solution for solving the arbitration problem: if there is an outstanding request for the usage of the bus, as soon as the bus becomes free, a token is sent through U_1 to U_2, U_2 to U_3, ... etc., until the arbiter of the smallest indexed unit U_i which requested the bus picks up the token. Assuming that a unit U_0, called the scheduler initially emits the token to U_1 and finally gets it back from a U_i, a formal specification of the algorithm is the following:

Events:

$send_i$: event that i-th arbiter sends a token to the (i + 1)st arbiter, $1 \leq i < n$

$next_i$: event that the token travels from i-th to (i + 1)st arbiter, $1 \leq i < n$

get_i : event that the token is received by i-th arbiter, $1 \leq i \leq n$

$dens_i$: event of sending back the token to the scheduler by the i-th arbiter, $1 \leq i \leq n$.

$send_0$: event that the scheduler sends the token to the first arbiter.

$next_0$: event that the token travels from the scheduler to the first arbiter

$retn_i$: event that the token travels from i-th arbiter to the scheduler

$back_i$: event of the scheduler getting back the token from the i-th arbiter

Relations Among the Events

$\forall i,\ 1 \leq i \leq n$

[R1] $dens_i = get_i \mid [\mathrm{COUNT}_2(req_i) \geq \mathrm{COUNT}_1(ack_i + 1)]$,

[R2] $retn_i << dens_i$

[R3] $back_i << retn_i$

[R4] $ack_i << back_i$

[R5] $send_i = get_i \mid [\mathrm{COUNT}_2(req_i) \leq \mathrm{COUNT}_1(ack_i)]$

[R6] $get_{i+1} << next_i << send_i$

[R7] $\mathrm{TIME}_1(send_0) = \max\{\mathrm{TIME}_2(\Sigma\ req_i),\ \mathrm{TIME}_2(\Sigma\ rel_i + 1)\}$

[R8] $get_1 << next_0 << send_0$

[R9] $\mathrm{TIME}_1(dec) = \mathrm{TIME}_1(send_0)$

[R10] $\mathrm{TIME}_2(dec) = \mathrm{TIME}_2(back_i)$

Proof of Correctness

The traveling token algorithm will be shown to satisfy the four properties stated in the arbiter specification.

i) Exclusivity

We first prove that

$$\Sigma\ ack_i << \Sigma\ rel_i + 1$$

Proof

From [R1] to [R4]

$$ack_i << back_i << retn_i << dens_i << get_i, \forall i,\ 1 \leq i \leq n$$

From [R6] to [R8],

$$get_1 << next_0 << send_0 << \Sigma\ rel_i + 1$$

It is sufficient to prove $\Sigma\ ack_i << get_1$

Since $get_i << get_1$, $\forall i \neq 1$, and $ack_i << get_i$, we get

$$ack_i << get_1\ ,\ 1 \leq i \leq n.$$

Next we prove

$$\Sigma\ \mathrm{COUNT}_1(ack_i)(t) \leq \mathrm{COUNT}_2(get_1)(t).$$

[R1] and [R5] gives, $\forall i$

$$\mathrm{COUNT}_2(get_i) = \mathrm{COUNT}_1(dens_i) + \mathrm{COUNT}_1(send_i).$$

From [R2], [R3] and [R4] we have, $\forall i$

$$ack_i << dens_i$$

and this implies

$$\mathrm{COUNT}_1(ack_i) \leq \mathrm{COUNT}_2(dens_i) \leq \mathrm{COUNT}_1(dens_i).$$

Hence

$$\begin{aligned}\mathrm{COUNT}_2(get_i) &= \mathrm{COUNT}_1(dens_i) + \mathrm{COUNT}_1(send_i) \\ &\geq \mathrm{COUNT}_1(ack_i) + \mathrm{COUNT}_1(send_i)\end{aligned}$$

and this implies

$$\sum_{i=1}^{n-1} \mathrm{COUNT}_2(get_i) \geq \sum_{i=1}^{n-1} \mathrm{COUNT}_1(ack_i) + \sum_{i=1}^{n-1} \mathrm{COUNT}_1(send_i). \quad (1)$$

From [R6], we have

$$get_{i+1} << send_i$$

and hence

$$\mathrm{COUNT}_2(get_{i+1}) \leq \mathrm{COUNT}_1(get_{i+1}) \leq \mathrm{COUNT}_2(send_i)$$
$$\leq \mathrm{COUNT}_1(send_i). \quad (2)$$

From (1) and (2) we get

$$\sum_{i=1}^{n-1} \mathrm{COUNT}_2(get_i) \geq \sum_{i=1}^{n-1} \mathrm{COUNT}_1(ack_i) + \sum_{i=1}^{n-1} \mathrm{COUNT}_2(get_{i+1}).$$

That is,

$$\mathrm{COUNT}_2(get_1) \geq \sum_{i=1}^{n-1} \mathrm{COUNT}_1(ack_i) + \mathrm{COUNT}_2(get_n).$$

But $ack_n << get_n$ gives

$$\mathrm{COUNT}_1(ack_n) \leq \mathrm{COUNT}_2(get_n).$$

So, $\mathrm{COUNT}_2(get_1) \geq \sum_{i=1}^{n} \mathrm{COUNT}_1(ack_i)$ and $P_{get_1} = I$ (because of the assumption [A2]). From Theorem 10 and earlier results it follows that

$$\Sigma\ ack_i << get_1 << \Sigma\ rel_i + 1$$

For our further discussion the next result is necessary.

Lemma 1 i) $\Sigma\ rel_i$ is single occurrent

ii) get_1 is single occurrent

and iii) $\Sigma\ ack_i$ is single occurrent.

Proof

i) $rel_i << ack_i$ implies

$\text{COUNT}_1(rel_i) \leq \text{COUNT}_2(ack_i)$,

$\Sigma\ \text{COUNT}_1(rel_i) \leq \Sigma\ \text{COUNT}_2(ack_i) \leq \Sigma\ \text{COUNT}_1(ack_i)$

We have proved that

$$\Sigma\ \text{COUNT}_1(ack_i) \leq \text{COUNT}_2(get_1).$$

Hence it follows,

$$\Sigma\ \text{COUNT}_1(rel_i) \leq \text{COUNT}_2(get_1).$$

From Theorem 10 and $P_{get_1} = I$, we get

$$\Sigma\ rel_i << get_1 << \Sigma rel_i + 1.$$

Hence $\Sigma\ rel_i$ and rel_i are single occurent events.

ii) Since $\Sigma\ rel_i << get_1 << \Sigma\ rel_i + 1$ and $\Sigma\ rel_i$ is single occurrent, get_1 is single occurrent.

iii) $\forall i$, $rel_i << ack_i$ implies $\Sigma\ \text{COUNT}_1(rel_i) \leq \Sigma\ \text{COUNT}_2(ack_i)$.

Now, $\Sigma\ ack_i << \Sigma\ rel_i + 1$ gives

$\Sigma\ \text{COUNT}_1(ack_i) \leq \Sigma\ \text{COUNT}_2(rel_i) + 1 \leq \Sigma\ \text{COUNT}_1(rel_i) + 1$

Hence

$\Sigma\ \text{COUNT}_1(ack_i) \leq \Sigma\ \text{COUNT}_1(rel_i) + 1 \leq \Sigma\ \text{COUNT}_2(ack_i) + 1$

and this proves

$$\Sigma\ ack_i \ll \Sigma\ ack_i + 1.$$

That is, $\Sigma\ ack_i$ and ack_i are both single occurrent.

Similarly it can be shown that $\Sigma\ acc_i$ and *dec* are also single occurrent events. Moreover,

$$\Sigma\ rel_i \ll \Sigma\ acc_i \ll \Sigma\ ack_i \ll dec \ll \Sigma\ rel_i + 1.$$

That is

$$\Sigma rel_i \ll \Sigma acc_i \ll \Sigma rel_i + 1.$$

This gives,

$$\Sigma\ \mathrm{COUNT}_2(rel_i) \leq \Sigma\ \mathrm{COUNT}_1(rel_i) \leq \Sigma\ \mathrm{COUNT}_2(acc_i)$$
$$\leq \Sigma\ \mathrm{COUNT}_1(acc_i) \leq \Sigma\ \mathrm{COUNT}_2(rel_i) + 1.$$

ii) Reactivity

From [R1] to [R4] it follows that

$$ack_i \ll dens_i = get_i \mid [\mathrm{COUNT}_2(req_i) \geq \mathrm{COUNT}_1(ack_i + 1)]$$

Hence

$$\mathrm{COUNT}_2(req_i) \circ \mathrm{LAST}_1(ack_i) \geq \mathrm{COUNT}_1(ack_i + 1) \circ \mathrm{LAST}_1(ack_i) \geq \mathrm{COUNT}_1(ack_i),$$

since the occurrences of ack_i do not overlap and we assume that at least one of the events among acc_i, *dec* has non-zero duration. This assumption is fair, since the usage of the bus cannot be instantaneous.

That is, $COUNT_2(req_i) \geq COUNT_1(ack_i)$.

Due to [A2], $P_{req_i} = I$, we get $ack_i << req_i$.

iii) Priority

We have to prove, $\forall\ 1 \leq k < i \leq n$,

$COUNT_1(ack_k) \circ LAST_1(dec) \circ LAST_1(ack_i) =$

$COUNT_2(req_k) \circ LAST_1(dec) \circ LAST_1(ack_i)$

The proof is similar to the one given in [1] and requires three lemmas:

Lemma 2

U_k never asks for the bus at the last time the event *dec* was initiated preceding $send_k$.

Lemma 3

For $k < i$, there is an occurrence of $send_k$ between get_i and the last time the event *dec* was initiated preceding it.

Lemma 4

No acknowledgement is issued between get_i and the last time the event *dec* was initiated preceding it.

The proof is now given in three parts:

(i) proves that Lemma 2 and Lemma 3 imply priority condition;

(ii) proves that Lemma 4 implies Lemma 2 and finally

(iii) proves Lemmas 3 and 4.

(i) Lemma 2 and Lemma 3 imply priority condition.

Proof

Lemma 2 $\Rightarrow$ $\text{COUNT}_1(ack_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k)$

$= \text{COUNT}_2(req_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k)$

Lemma 3 $\Rightarrow$ $\forall\ 1 \leq k < i < n$, $\text{LAST}_1(dec) \circ \text{LAST}_1(get_i)$

$= \text{LAST}_1(dec) \circ \text{LAST}_1(send_k) \circ \text{LAST}_1(get_i)$

Since $ack_i \ll get_i$, and for every occurrence of get_i if ack_i follows then between these events only $retn_i$ and $back_i$ happens. Using Lemma 3 now we get

$\text{COUNT}_1(ack_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(ack_i)$

$= \text{COUNT}_1(ack_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k) \circ \text{LAST}_1(ack_i)$,

$= \text{COUNT}_2(req_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k) \circ \text{LAST}_1(ack_i)$,

from Lemma 2

$= \text{COUNT}_2(req_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(ack_i)$, from Lemma 3.

This proves the priority condition.

(ii) Lemma 4 $\Rightarrow$ Lemma 2

Lemma 4 states that $\forall i \geq 1$, $\forall k \geq 1$,

$\text{COUNT}_1(ack_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(get_i) = \text{COUNT}_1(ack_k) \circ \text{LAST}_1(get_i)$.

Since $send_k$ can happen only if get_k happens but does not have to happen for every occurrence of get_k , it follows that

$\text{COUNT}_2(req_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k) \geq$

$\text{COUNT}_1(ack_k) \circ \text{LAST}_1(dec) \circ \text{LAST}_1(send_k)$

$= \text{COUNT}_1(ack_k) \circ \text{LAST}_1(send_k)$, by Lemma 4

$\geq$ $COUNT_2(req_k)$ $\circ$ $LAST_1(send_k)$, from [R5]

$\geq$ $COUNT_2(req_k)$ $\circ$ $LAST_1(dec)$ $\circ$ $LAST_1(send_k)$.

Hence

$COUNT_2(req_k)$ $\circ$ $LAST_1(dec)$ $\circ$ $LAST_1(send_k)$ =

$COUNT_1(ack_k)$ $\circ$ $LAST_1(dec)$ $\circ$ $LAST_1(send_k)$.

(iii) Proof of Lemma 3

To prove, for $1 \leq k < i \leq n$,

$LAST_1(dec)$ $\circ$ $LAST_1(get_i)$ = $LAST_1(dec)$ $\circ$ $LAST_1(send_k)$ $\circ$ $LAST_1(get_i)$

Since

$LAST_1(dec)$ $\circ$ $LAST_1(send_k)$ $\circ$ $LAST_1(get_i)$ $\leq$ $LAST_1(dec)$ $\circ$ $LAST_1(get_i)$,

it is sufficient to prove,

$LAST_1(dec)$ $\circ$ $LAST_1(get_i)$ $\leq$ $LAST_1(dec)$ $\circ$ $LAST_1(send_k)$ $\circ$ $LAST_1(get_i)$

Since the event *dec* is single occurrent, this is equivalent to proving

$COUNT_1(dec)$ $\circ$ $LAST_1(get_i)$ $\leq$ $COUNT_1(dec)$ $\circ$ $LAST_1(send_k)$ $\circ$ $LAST_1(get_i)$

From $get_i \ll send_{i-1}$ and

$COUNT_1(send_{i-1})$ + $COUNT_1(ack_{i-1})$ $\leq$ $COUNT_2(get_{i-1})$ $\leq$ $COUNT_1(get_{i-1})$

it follows readily that

$$\begin{aligned} \mathrm{COUNT}_1(get_i) &\leq \mathrm{COUNT}_1(send_{i-1}) \circ \mathrm{LAST}_1(get_i) \\ &\leq [\mathrm{COUNT}_1(get_{i-1}) \circ \mathrm{LAST}_1(get_i) - \\ &\quad \mathrm{COUNT}_1(ack_{i-1}) \circ \mathrm{LAST}_1(get_i)] \end{aligned} \tag{3}$$

and

$$\begin{aligned} \mathrm{COUNT}_1(get_i) &\leq \mathrm{COUNT}_1(send_{i-1}) \\ &\leq \mathrm{COUNT}_1(get_{i-1}) - \mathrm{COUNT}_1(ack_{i-1}) \end{aligned} \tag{4}$$

Iterating on i, we get for k < i,

$$\begin{aligned} \mathrm{COUNT}_1(get_{i-1}) &\leq \mathrm{COUNT}_1(get_{k+1}) - \sum_{m=k+1}^{i-2} \mathrm{COUNT}_1(ack_m) \\ &\leq \mathrm{COUNT}_1(send_k) - \sum_{m=k+1}^{i-2} \mathrm{COUNT}_1(ack_m) \end{aligned}$$

Hence,

$$\mathrm{COUNT}_1(get_{i-1}) \circ \mathrm{LAST}_1(get_i) + \sum_{m=k+1}^{i-2} \mathrm{COUNT}_1(ack_m) \circ \mathrm{LAST}_1(get_i)$$

$$\leq \mathrm{COUNT}_1(send_k) \circ \mathrm{LAST}_1(get_i) \tag{5}$$

From (3) and (5) it follows,

$$\begin{aligned} &\mathrm{COUNT}_1(get_i) + \sum_{m=k+1}^{i-1} \mathrm{COUNT}_1(ack_m) \circ \mathrm{LAST}_1(get_i) \\ &\quad \leq \mathrm{COUNT}_1(send_k) \circ \mathrm{LAST}_1(get_i) \end{aligned} \tag{6}$$

Similarly it can be shown that

$$\begin{aligned} &\mathrm{COUNT}_1(send_k) + \sum_{m=1}^{k} \mathrm{COUNT}_1(ack_k) \circ \mathrm{LAST}_1(send_k) \\ &\quad \leq \mathrm{COUNT}_1(send_0) \circ \mathrm{LAST}_1(send_k) \\ &\quad = \mathrm{COUNT}_1(dec) \circ \mathrm{LAST}_1(send_k) \end{aligned} \tag{7}$$

Similar to the proof for $\Sigma ack_i << get_1$, it can be proven that

$$\sum_{m=i}^{n} ack_m << get_i.$$

Since $dec << \Sigma ack_i + 1 = \sum_{m=1}^{i-1} ack_m + 1 + \sum_{m=i}^{n} ack_m$,

we get $dec << get_i + \sum_{m=1}^{i-1} ack_m + 1.$

$\Rightarrow$ $\text{COUNT}_1(dec) \circ \text{LAST}_1(get_i) \leq \text{COUNT}_1(get_i) + \sum_{m=1}^{i-1} \text{COUNT}_1(ack_m) \circ \text{LAST}_1(get_i)$

$\leq \text{COUNT}_1(dec) \circ \text{LAST}_1(send_k) \circ \text{LAST}_1(get_i)$

and this completes the proof.

Finally, it remains to prove Lemma 4.

Since $\Sigma\ ack_i << dec << \Sigma\ ack_i + 1$,

it follows that

$$\text{COUNT}_1(\Sigma ack_i) \circ \text{LAST}_1(dec)$$

$$= \text{COUNT}_1(dec) \circ \text{LAST}_1(\Sigma ack_i) \circ \text{LAST}_1(dec)$$

$$= \text{COUNT}_1(dec) \circ \text{LAST}_1(dec) = (I - 1) \circ \text{COUNT}_1(dec) \qquad (8)$$

Since $get_i << next_{i-1}$,

$\text{COUNT}_1(get_i) \circ \text{LAST}_1(get_i) \leq (I - 1) \circ \text{COUNT}_1(next_{i-1}) \circ \text{LAST}_1(get_i)$.

This implies

$$\Sigma\text{COUNT}_1(get_i) \circ \text{LAST}_1(get_i) \leq (I - 1) \circ \Sigma\text{COUNT}_1(next_{i-1}) \circ \text{LAST}_1(get_i) \qquad (9)$$

From [R1] and [R5] it follows that

$$\Sigma\text{COUNT}_1(get_i) \geq \Sigma\text{COUNT}_1(dens_i) + \Sigma\text{COUNT}_1(send_i)$$

$$\geq \Sigma COUNT_1(ack_i) + \Sigma COUNT_1(send_i)$$

This implies

$$\Sigma COUNT_1(ack_i) \circ LAST(get_i) \leq \Sigma COUNT_1(get_i) \circ LAST_1(get_i) - \Sigma COUNT_1(send_i) \circ LAST_1(get_i)$$

$$\leq (I - 1) \circ \Sigma COUNT_1(next_{i-1}) \circ LAST_1(get_i) - \Sigma COUNT_1(send_i) \circ LAST_1(get_i), \text{ from (9)}$$

$$\leq (I - 1)\ \Sigma COUNT_1(next_{i-1}) \circ LAST_1(get_i) - \Sigma COUNT_1(next_i) \circ LAST_1(get_i),$$

$$\leq (I - 1) \circ COUNT_1(next_0) \circ LAST_1(get_i)$$

$$= (I - 1) \circ COUNT_1(dec) \circ LAST_1(get_i)$$

$$= COUNT_1(\Sigma ack_i) \circ LAST_1(dec) \circ LAST_1(get_i)$$

$$\leq COUNT_1(\Sigma ack_i) \circ LAST_1(get_i) \qquad (10)$$

Hence

$$COUNT_1(\Sigma ack_i) \circ LAST_1(dec) \circ LAST_1(get_i) = COUNT_1(\Sigma ack_i) \circ LAST_1(get_i),$$

and in turn follows the result that $\forall k$,

$$COUNT_1(ack_k) \circ LAST_1(dec) \circ LAST_1(get_i) = COUNT_1(ack_k) \circ LAST_1(get_i).$$

(iv) Promptness

The proof follows immediately from the specification of the algorithm.

Whereas the previous example illustrated the formalism in the case of hardware level problem, the next problem is a high level

concurrency control method for multiple copy update problem in a distributed database.

4.2 Holler's Algorithm for Multiple Copy Update Problem in a Distributed Database

For us the notion of a database is a set of data items without regard to the granularity of data items. At any instant an item v_i can take a value VALUE(v_i) from a domain DOM(v_i). A transaction is a sequence of read/write operations enforced by admissible queries in the system. In a fully redundant distributed database system, each site has a complete copy of the database and VALUE(v_i^s) denotes the value of i-th data item at the s-th site. At any instant, it may be insisted that an access to the database should see the same information regardless of which copy is accessed. This is a strong condition requiring complex and costly coordination mechanisms. A somewhat weaker condition is the notion of database consistency incorporating two aspects: mutual consistency of redundant copies and internal consistency of each copy.

Mutual consistency requires that at any time all database copies are identical. In practice this is difficult to achieve. Hence a weaker version of mutual consistency is to require that when all transactions are completed, multiple copies must converge to the same final state.

Internal consistency brings out two issues: semantic integrity and serializability. The task of enforcing semantic integrity is not specific to distributed databases and so is not considered as an issue here. A concurrent transaction set preserves internal consistencies only if there is a sequential order for the atomic components of the transaction preserving the required integrity constraints; in this case, the transactions are said to be serializable. For example, assume that the relation $x + y + z = 3$ should be preserved by the data items x, y and z, when they all have the initial value 1. Each of the updates

$$T_1 : x \leftarrow 3,\ y \leftarrow -1$$

and

$$T_2 : y \leftarrow 3,\ z \leftarrow -1$$

preserves the condition $x + y + z = 3$; but neither the sequence (T_1, T_2) nor (T_2, T_1) satisfies the condition $x + y + z = 3$. So, the transactions are not serializable and in a concurrent execution, one of the transactions should be rejected.

Hence the multiple copy update problem in distributed databases requires mechanism guaranteeing mutual consistency and serializability. Towards a formal specification of this problem the events are defined next.

Assumptions

1. There are n sites and each site has a database copy and a controller for obtaining permission to schedule the processing of a transaction. The database copies have m variables v_1, v_2, ... ,v_m and the variable v_i at the i-th site is denoted v_i^s.
2. Each site has a storage processor and is responsible for manipulating data in that site.
3. Only those data items bound to a transaction can be changed by an execution of that transaction.

$submit_i$: event that a transaction is submitted to the access controller C_i at the i-th site.

$execute_i$: event that a transaction is being executed at the i-th site.

$update_j^i$: event that a transaction submitted to the i-th site is taken up for updating the local copy of the database at the j-th site, $j \neq i$.

Specification of Mutual Consistency

Let $c_k^s : \overline{\Pi} \rightarrow \overline{\mathbb{N}}$ such that $c_k^s(t) = \text{COUNT}_2(\text{ASSIGN}(v_k^s))(t)$, $t \geq 0$, $k = 1, ..., m$ and $s = 1, ..., n$. (The number of times a variable is updated need not be the same in all sites).

At any time t, $1 \leq i \neq j \leq n$

$\text{COUNT}_2(\Sigma submit_i)(t) = \text{COUNT}_1(\Sigma execute_i)(t)$

$$= \text{COUNT}_2(\Sigma execute_i)(t) = \text{COUNT}_1(\sum_{i \neq j} update_j^i + execute_j)(t)$$

$$= \text{COUNT}_2(\sum_{i \neq j} update_j^i + execute_j)(t)$$

$$\Rightarrow \text{VALUE}(v_k^1)(c_k^1) = \text{VALUE}(v_k^2)(c_k^2) = \ldots = \text{VALUE}(v_k^3)(c_k^3)$$

$$\ldots = \text{VALUE}(v_k^n)(c_k^n)$$

$$k = 1, 2, \ldots, m.$$

Specification of Serializability

The serializability condition can be rephrased as follows:

The distributed execution of transactions is serializable if there is a sequential schedule (consistent with integrity constraints) for the execution of transactions such that the sequence of values taken by a data item at any site during the original execution of the transactions is a subsequence of the values taken by the same data item in the serial execution of the transactions. Hence

$\text{VALUE}(v_i^k) = \text{VALUE}(v_i) \circ q_i^k$, where

$q_i^k : \overline{N} \rightarrow \overline{N}$

is a nondecreasing function, $1 \leq i \leq m$, and $1 \leq k \leq n$. Here, if *e* denotes the serialized executions then v_i's are assumed to be modified due to *e*.

Specification for a Starvation-Free System

A transaction should be committed within a finite amount of time. That is, $[\text{TIME}_2(submit_i)(k) < \infty] \rightarrow$

$[[\text{TIME}_1(update_j^i)(k) < \infty, \forall\ j \neq i]$ and

$[TIME_1(execute_i)(k) < \infty]]$, $1 \leq i \leq n$, $k \geq 1$.

Next, we discuss a synchronous algorithm, originally due to Holler [4], for solving this problem and provide a formal proof of its correctness. It is assumed that for a given set of transactions the ordering can be deduced from a tag attached to each transaction. The algorithm enforces a voting procedure based on message exchanging between controllers so that agreements on global sequence of transactions can be reached.

An Informal Description of a Synchronous Voting Algorithm

At any time, the set of transactions in the system are assumed to be totally ordered. The ordering can be based on their time of arrival. That is, if a transaction T_1 is submitted earlier than T_2 then T_1 has higher priority than T_2. If T_1 and T_2 are submitted at two different sites C_i and C_j at the same time then T_1 has higher priority than T_2 if $i < j$ and viceversa. When a request T is submitted to a site, the controller broadcasts a vote for this transaction, if not having initiated a voting for a previous transaction. The receiving site compares the priority of T with the priority of each incomplete transaction at that site. If T has higher priority, then a vote is broadcast to all sites; otherwise T is queued. Whenever every site receives $(n - 1)$ votes on a transaction, the transaction is executed at that site. After completing the execution,

end-signals are broadcast. After receiving (n – 1) end-signals, a site processes the transaction of next highest priority.

A Formal Specification of the Algorithm

Definition of Events

$send_{ij}$: event that C_i sends request for voting to C_j, $i \neq j$.

$next_{ij}$: event that the request for voting goes from C_i to C_j, $i \neq j$.

get_{ij} : event that C_j receives the request for voting from C_i , $i \neq j$.

ack^i_{jk} : event that C_j sends an acknowledgement to C_k for a transaction submitted to C_i , $j \neq k$, $i \neq j$.

$back^i_{jk}$: event that the acknowledgement goes from C_j to C_k for a transaction submitted to C_i , $i \neq j$, $j \neq k$.

$dens^i_{jk}$: event that C_k receives the acknowledgement sent by C_j for a transaction submitted to C_i , $i \neq j$, $j \neq k$.

$send\text{-}end^i_{jk}$: event that C_j sends an end-signal to C_k for a transaction submitted to C_i , $j \neq k$.

$next\text{-}end^i_{jk}$: event that the end signal goes from C_j to C_k for a transaction submitted to C_i , $j \neq k$.

$get\text{-}end^i_{jk}$: event that C_k gets the end-signal from C_j for a transaction submitted to C_i , $j \neq k$.

Informal and Formal Description of the Steps

1. The controller C_i at the i-th site sends a request to every other controller for voting on a new transaction submitted to it if

   a) it has completed issuing end-signals to all other controllers for the previous transaction submitted at C_i .

   b) it has received the end-signals from all other controllers for the last transaction submitted at C_i .

   c) there are outstanding transactions at C_i for execution.

$$\begin{aligned} \text{TIME}_1(send_{ij}) = \max\{&\text{TIME}_2(submit_i), \\ &\max_{k\neq i}\{\text{TIME}_2(get\text{-}end^i_{ki} + 1), \\ &\text{TIME}_2(send\text{-}end^i_{ik} + 1)\}\} \end{aligned} \tag{11}$$

2. The controller C_j at the j-th site issues an acknowledgement (vote) to C_k for a transaction T submitted to C_i if:

   a) among the requests submitted to j-th site, C_j does not have any outstanding (pending) transaction whose priority is greater than the priority of T.

   b) among the incomplete transactions for which requests were received for voting by C_j from other sites, there is no transaction with priority greater than the priority of T.

Let f_j be the function which at any time t determines the number of completed transactions in the system from among those submitted to the j-th site:

$$f_j : \overline{\Pi} \rightarrow \overline{\mathbf{N}},$$

$$f_j = \min_{s \neq j} \{\min(\text{COUNT}_2(\textit{send-end}^j_{js})), \min_{s \neq j}(\text{COUNT}_2(\textit{get-end}^j_{sj}))\}.$$

and $(f_j + 1)(t) = f_j(t) + 1.$

Similarly, let g^r_j be the function which at any time t determines the number of transactions completed in the system from among those submitted to r-th site, $r \neq j$.

$$g^r_j : \overline{\Pi} \rightarrow \overline{\mathbf{N}},$$

$$g^r_j = \min_{s \neq j} \{\min(\text{COUNT}_2(\textit{send-end}^r_{js})), \min_{s \neq j}(\text{COUNT}_2(\textit{get-end}^r_{sj}))\},$$

and $(g^r_j + 1)(t) = g^r_j(t) + 1.$

Now we can formally state the conditions:

$$\begin{aligned}
\text{COND}_1 = {} & [\text{COUNT}_2(\textit{submit}_j) = f_j] \vee \\
& [(\text{COUNT}_2(\textit{submit}_j) > f_j) \wedge \\
& ((\text{TIME}_2(\textit{submit}_j) \circ (f_j + 1) < \\
& \text{TIME}_2(\textit{submit}_i) \circ \text{LCOUNT}_2(\textit{get}_{ij})) \vee \\
& ((\text{TIME}_2(\textit{submit}_j) \circ (f_j + 1) = \\
& \text{TIME}_2(\textit{submit}_i) \circ \text{LCOUNT}_2(\textit{get}_{ij})) \\
& \wedge (i < j))))] \qquad (12)
\end{aligned}$$

$$\begin{aligned}
\text{COND}_2 = \bigwedge_{r \neq j} {} & [(\text{COUNT}_2(\textit{get}_{rj}) = g^r_j) \vee \\
& ((\text{COUNT}_2(\textit{get}_{rj}) > g^r_j) \wedge \\
& ((\text{TIME}_2(\textit{submit}_r) \circ (g^r_j + 1) < \\
& \text{TIME}_2(\textit{submit}_i) \circ \text{LCOUNT}_2(\textit{get}_{ij}))
\end{aligned}$$

$$\vee\ ((\mathrm{TIME}_2(submit_r) \circ (g_j^r + 1) =$$

$$\mathrm{TIME}_2(submit_r) \circ \mathrm{LCOUNT}_2(get_{ij}))$$

$$\wedge\ (i < r)))))] \qquad (13)$$

Define events e_j and h_j^r, $r \neq j$, $r = 1, \ldots, n$, as follows:

$$\mathrm{TIME}_1(e_j)(n) = \mathrm{TIME}_2(e_j)(n) = \min\{t \mid f_j(t) = n\}$$

$$\mathrm{TIME}_1(h_j^r)(n) = \mathrm{TIME}_2(h_j^r)(n) = \min\{t \mid g_j^r(t) = n\}$$

Then

$$ack_{jk}^i = [get_{ij} \mid [\mathrm{COND}_1 \wedge \mathrm{COND}_2]] +$$

$$(e_j + \sum_{r \neq j} h_j^r) \mid [(\mathrm{COUNT}_2(get_{ij}) > \mathrm{COUNT}_1(ack_{ji}^i)) \wedge$$

$$\mathrm{COND}_1 \wedge \mathrm{COND}_2],\ k \neq j,\ i \neq j \qquad (14)$$

3. The storage processor at site i executes a transaction when C_i has received the acknowledgement from all other controllers.

$$\mathrm{TIME}_1(execute_i) = \max_{j \neq i} \{\mathrm{TIME}_2(dens_{ji}^i)\} \qquad (15)$$

4. Storage processor at site j updates a transaction submitted to C_i when it has received the acknowledgements from the rest of the controllers and it has acknowledged the transaction to other controllers.

$$\mathrm{TIME}_1(update_j^i) = \max\{\max_{k \neq i,j}\{\mathrm{TIME}_2(dens_{kj}^i)\},\ \max_{k \neq j}\{\mathrm{TIME}_2(ack_{jk}^i)\}\} \qquad (16)$$

Relations Among the Events

[H1] $\quad get_{ij} \leq next_{ij} \leq send_{ij}$, $\forall\ j \neq i$

[H2] $\quad dens_{jk}^i \leq back_{jk}^i \leq ack_{jk}^i$, $\forall\ k \neq j,\ \forall\ j \neq i$

[H3] $\text{TIME}_1(\textit{send-end}^i_{ij}) = \text{TIME}_2(\textit{execute}_i),\ \forall\ i,\ i \neq j$

[H4] $\textit{get-end}^i_{ij} \leq \textit{next-end}^i_{ij} \leq \textit{send-end}^i_{ij}\ ,\ \forall\ i,\ i \neq j$

[H5] $\textit{get-end}^i_{jk} \leq \textit{next-end}^i_{jk} \leq \textit{send-end}^i_{jk},\ \forall\ i,\ \forall\ j \neq i,\ \forall\ k \neq j$

[H6] $\text{TIME}_1(\textit{send-end}^i_{jk}) = \text{TIME}_2(\textit{update}^i_j),\ \forall\ k \neq j,\ \forall\ j \neq i,\ \forall\ i$

Proof of Correctness

We prove two lemmas and then show that the proof of correctness follows from these lemmas.

Lemma 5 proves that the execution sequence corresponds to the total order imposed by the priority. Lemma 6 proves that the updates and execution of a transaction are completed before the update or execution of another transaction starts.

Let $T_1, T_2, \ldots, T_N$ be the sequence of transactions defined by the total ordering. Consider the event u_i defined as follows :

$$\text{TIME}_1(u_i) = \min_{j \neq i} \{\text{TIME}_1(\textit{execute}_i), \text{TIME}_1(\textit{update}^i_j)\}$$

$$\text{TIME}_2(u_i) = \max_{j \neq i} \{\text{TIME}_2(\textit{execute}_i), \text{TIME}_2(\textit{update}^i_j)\}$$

Lemma 5

Let the k-th occurrence of Σu_i be the k_1-th occurrence of u_j for some j and the (k − 1)-st occurrence of Σu_i be the k_2th occurrence

of u_r for some r. Then

$\text{TIME}_2(submit_r)(k_2) < \text{TIME}_2(submit_j)(k_1)$

or

$[(\text{TIME}_2(submit_r)(k_2) = \text{TIME}_2(submit_j)(k_1)) \wedge (r < j)]$. That is, the transactions are executed in the decreasing order of their priority.

Proof

If the result is not true, then

$$\text{TIME}_1(ack^r_{js})(k_2) \geq \max\{\max_{s\neq j}\{\text{TIME}_2(get\text{-}end^j_{sj})(k_1)\}, \max_{s\neq j}\{\text{TIME}_2(send\text{-}end^j_{js})(k_1)\}\}$$

$$\Rightarrow [(\text{TIME}_1(ack^r_{js})(k_2) \geq \text{TIME}_2(execute_j)(k_1)) \wedge (\text{TIME}_1(ack^r_{js})(k_2) \geq \text{TIME}_2(update^j_s)(k_1))]$$

$$\Rightarrow \text{TIME}_1(ack^r_{js})(k_2) \geq \text{TIME}_2(u_j)(k_1),$$

$$\text{TIME}_1(execute_r)(k_2) \geq \text{TIME}_1(ack^r_{js})(k_2)$$

$$\Rightarrow \text{TIME}_1(execute_r)(k_2) \geq \text{TIME}_2(u_j)(k_1), \tag{17}$$

But

$$\text{TIME}_1(update^r_s)(k_2) \geq \text{TIME}_2(ack^r_{js})(k_2),\ s \neq r,\ s \neq j$$

and $$\text{TIME}_1(update^r_j)(k_2) \geq \text{TIME}_2(ack^r_{js})(k_2).$$

This gives

$$\text{TIME}_1(u_r)(k_2) \geq \text{TIME}_2(ack^r_{js})(k_2) \geq \text{TIME}_2(u_j)(k_1).$$

$$\Rightarrow \text{TIME}_1(\Sigma u_i)(k-1) \geq \text{TIME}_2(\Sigma u_i)(k)$$

This contradicts the property that TIME_1 is an increasing function.

Hence the theorem is proved.

Lemma 6

The storage processor for site i processes T_j iff processing of T_1, T_2, ..., T_{j-1} are completed. That is, both u_i and Σu_i are single-occurrent events.

Proof

From [H3], $\forall\ j \neq i$, $send\text{-}end^i_{ij} << execute_i$

$$\Rightarrow \text{TIME}_1(send\text{-}end^i_{ij}) \geq \text{TIME}_2(execute_i)$$

$$\Rightarrow \text{TIME}_1(send\text{-}end^i_{ij} + 1) \geq \text{TIME}_2(execute_i + 1)$$

$$\Rightarrow \max_{j\neq i}\{\text{TIME}_1(send\text{-}end^i_{ij} + 1)\} \geq \text{TIME}_2(execute_i + 1).$$

From [H5] and [H6]

$$get\text{-}end^i_{jk} << update^i_j\ ,\ \forall k \neq j,\ \forall j \neq i.$$

$$\Rightarrow \text{TIME}_1(get\text{-}end^i_{jk} + 1) \geq \text{TIME}_2(update^i_j + 1)$$

$$\Rightarrow \text{TIME}_1(get\text{-}end^i_{ji} + 1) \geq \text{TIME}_2(update^i_j + 1),\ j \neq i$$

$$\Rightarrow \max_{j\neq i}\{\text{TIME}_1(get\text{-}end^i_{ji} + 1)\} \geq \max_{j\neq i}\{\text{TIME}_2(update^i_j + 1)\}.$$

From (11), $\forall j \neq i$,

$$\text{TIME}_1(send_{ij}) \geq \max_{j\neq i}\{\text{TIME}_2(get\text{-}end^i_{ji} + 1),\ \text{TIME}_2(send\text{-}end^i_{ij} + 1)\}$$

$$\geq \max\{\text{TIME}_2(execute_i + 1), \max_{j \neq i}\{\text{TIME}_2(update_j^i + 1)\}\}$$

$$\geq \text{TIME}_2(u_i + 1)$$

$\Rightarrow send_{ij} << u_i + 1$

But $execute_i << send_{ij}$ and $update_j^i << send_{ij}$, $j \neq i$

imply that $u_i << send_{ij}$. Hence, $u_i << u_i + 1$ which proves that u_i is single-occurrent.

Next, we prove that Σu_i is a single-occurrent event; that is,

$\text{TIME}_1(\Sigma u_i)(k) \geq \text{TIME}_2(\Sigma u_i)(k-1)$, $k \geq 1$.

Let the k-th occurrence of Σu_i be k_1-th occurrence of u_j for some j and (k − 1)-st occurrence of Σu_i be the k_2-th occurrence of u_r for some r. It is known from Lemma 5 that the transactions are executed in decreasing order of their priority. So, from (14)

$\forall$ s, $\text{TIME}_1(ack_{rs}^j)(k_1) \geq$

$$\max\{\max_{s \neq r}\{\text{TIME}_2(get\text{-}end_{sr}^r)(k_2)\}, \max_{s \neq r}\{\text{TIME}_2(send\text{-}end_{rs}^r)(k_2)\}\}$$

$\Rightarrow \text{TIME}_1(ack_{rs}^j)(k_1) \geq \text{TIME}_2(execute_r)(k_2)$

and $\text{TIME}_1(ack_{rs}^j)(k_1) \geq \text{TIME}_2(update_s^r)(k_2)$, $r \neq s$

$\Rightarrow \text{TIME}_1(ack_{rs}^j)(k_1) \geq \text{TIME}_2(u_r)(k_2)$

Since $\text{TIME}_1(update_s^j)(k_1) \geq \text{TIME}_2(ack_{rs}^j)(k_1)$,

$\text{TIME}_1(execute_j)(k_1) \geq \text{TIME}_2(ack_{rj}^j)(k_1)$

and from (16),

$\text{TIME}_1(update_r^j)(k_1) \geq \text{TIME}_2(ack_{rs}^j)(k_1)$,

it follows that

$\min\{\text{TIME}_1(execute_j)(k_1), \text{TIME}_1(update_r^j)(k_1)\}$

$\geq \max\{\text{TIME}_2(ack_{rj}^j)(k_1), \text{TIME}_2(ack_{rs}^j)(k_1)\}$.

Hence $\text{TIME}_1(u_j)(k_1) \geq \text{TIME}_2(u_r)(k_2)$

i.e., Σu_i is single occurrent.

Finally, we prove that the algorithm specification is correct; that is, it meets the problem specification.

Proof of Mutual Consistency

Lemma 5 and Lemma 6 imply that transactions are executed in the order of their priority. Moreover they prove that the execution and update corresponding to one transaction is done before the execution and update of another one starts. So, if at time $t > 0$ we have

$$\text{COUNT}_1(\Sigma execute_i)(t) = \text{COUNT}_2(\Sigma execute_i)(t)$$

$$= \text{COUNT}_1(\sum_{i \neq j} update_j^i + execute_j)(t)$$

$$= \text{COUNT}_2(\sum_{i \neq j} update_j^i + execute_j)(t) = k$$

then $\text{COUNT}_1(\Sigma u_i)(t) = \text{COUNT}_2(\Sigma u_i)(t) = k$.

Hence by Lemma 6

$\text{COUNT}_1(\Sigma u_i) \circ \text{LAST}_1(\text{ASSIGN}(v_j^1)) =$

$\text{COUNT}_1(\Sigma u_i) \circ \text{LAST}_1(\text{ASSIGN}(v_j^2)) =$

$\ldots = \text{COUNT}_1(\Sigma u_i) \circ \text{LAST}_1(\text{ASSIGN}(v_j^n))$

$= k', k' \leq k.$

Hence

$\text{COUNT}_2(\text{ASSIGN}(v_j^s))(t) = k', \forall s.$

$\Rightarrow \text{VALUE}(v_j^s) \circ \text{COUNT}_2(\text{ASSIGN}(v_j^s))(t) = \text{VALUE}(v_j^s)(k'), \forall s.$

That is, $T_{k'}$ is the last transaction that changed the value of v_j at all sites. This proves the claim that for every variable v_j, $\text{VALUE}(v_j^1)(k') = \text{VALUE}(v_j^2)(k') = \ldots = \text{VALUE}(v_j^n)(k')$.

Proof of Serializability

Lemma 6 proves this.

Proof of Starvation Freeness

We prove that $\forall$ i, k, $\text{TIME}_2(submit_i)(k) < \infty \Rightarrow \text{TIME}_1(u_i)(k) < \infty$

Let $\text{TIME}_2(submit_i)(k) = t$.

$\Rightarrow \text{LCOUNT}_2(\Sigma submit_i)(t) = s < \infty,$

since only finite number of occurrences happen in a finite interval time.

$\Rightarrow$ $\text{TIME}_1(\Sigma u_i)(s') = \text{TIME}_1(u_i)(k)$, $s' \leq s$.

Since $\forall\ r > 0$ $\text{dur}(\Sigma u_i)(r)$ is finite,

$$\text{TIME}_1(\Sigma u_i)(s') \leq \text{TIME}_2(submit_i)(k) + \sum_{r=1}^{s-1} \text{dur}(\Sigma u_i)(r) + st',$$

where $t' = 2$ (maximum communication delay between sites). Hence $\text{TIME}_1(u_i)(k) < \infty$.

The next two examples are taken from robotics. Notice that the algorithms are asynchronous and timing constraints enter in the definition of events. Because of simplicity of assumptions, proofs are not long. However if we are to describe the internal structure of the system based on event occurrences, then issues on object specifications and details of vision system must be addressed. As remarked earlier, for external behavior description, the internal details are assumed to exist in their total correctness.

4.3 The Design of a Robotic Assembly

A recent work [Ould 87] describes the behavioral specification of a flexible assembly cell. This cell is used as a test bed in Laboratoire d' Automatique et d'Analyse des Systemes du C.N.R.S., France. As we briefly mentioned in Chapter 2, this formalism is based on Petri nets; however a CSP based language called LCS(Language for Communicating Systems) is used for specifying the

interaction between agents(such as conveyor, vision, gripper and robots). Notice that two different styles of notations are used here. Proving correctness in such situations is very difficult and so it is not entirely surprising that no correctness proof is given in [Ould 87].

We consider a robot with two arms engaged in assembly operations. For simplicity assume that a robot having two arms and equipped with a vision system can recognize two types of objects: c (cup) and d (dish). The objects, n of each kind, are placed in a circular conveyor belt moving at a constant speed and the robot may pick up an item from the conveyor belt at a prespecified location. The robot is expected to assemble the objects in the order <c, d> so that n assemblies are output for any arbitrary placement of the 2n objects on the conveyor belt. Assuming the correctness of computations involved in object recognition and coordinate transformation, a simple specification of the problem, an algorithm for its solution and a proof of its correctness are given here.

Problem Specification

The various events characterizing the problem are as follows:

cup : event that a cup is recognized on the conveyor belt by the robot

dish : event that a dish is recognized on the conveyor belt by the robot

$pick_{\ell c}$: event that the left arm of the robot picks up a cup.

$pick_{\ell d}$: event that the right arm of the robot picks up a dish.

The events $pick_{\ell c}$ and $pick_{rd}$ are defined similarly.

assemble : event that an assembly <c, d> is made by the robot.

place : event that the robot places the assembled item on a tray.

Assuming that the robot's arm is synchronized to pick up the item as soon as it is recognized and an arm can hold at most one item at any time, the events satisfy the following properties:

[S1] $pick_{\ell c} + pick_{\ell d}$ is single occurrent.

[S2] $pick_{rc} + pick_{rd}$ is single occurrent.

[S3] $place << assemble$

We have to prove

[S4] $\text{TIME}_1(place)(n) = \text{TIME}_2(pick_{\ell c} + pick_{\ell d} + pick_{rc} + pick_{rd})(2n) + \delta$

for some constant $\delta > 0$.

An Informal Description of Robotic Assembly Algorithm

Initially both arms are empty and the stack (used for storing objects) is empty. Whenever both arms are empty and the stack is empty, the left arm picks up an item from the conveyor belt. When the left arm holds an item, the right arm picks up the next item from the conveyor belt. If the item on the right hand is the same

as the item on the left hand, the item from the right hand is pushed onto the stack; otherwise the items are assembled and placed on a tray. If both arms are empty but the stack is not empty, the left arm picks up (pops) an item from the stack. When the conveyor belt is empty, the stack becomes empty and the robot's hands are free, the algorithm ensures that n assemblies are correctly done.

Formal Specification of the Algorithm

The events and their relations defined below describe the algorithm.

$push_c$: event of pushing a cup onto the stack by the right arm

pop_c : event of popping a cup from the stack by the left arm

$push_d$ and pop_d are defined similarly.

Since the left arm holds a cup (dish) whenever a cup (dish) is picked up from the conveyor belt or the stack is popped resulting in a cup (dish), the events $hold_c$ (left arm holding a cup) and $hold_d$ (left arm holding a dish) satisfy the relations

$\text{TIME}_1(hold_c) = \text{TIME}_1(pick_{\ell c} + pop_c)$,

$\text{TIME}_1(hold_d) = \text{TIME}_1(pick_{\ell d} + pop_d)$

$hold = hold_c + hold_d$.

LEMPTY and REMPTY are conditionals that at any time indicate whether or not the left and right arms are respectively empty. Thus

LEMPTY = [$\text{COUNT}_1(pick_{\ell c} + pick_{\ell d} + pop_c + pop_d) = \text{COUNT}_2(place)$]

REMPTY = [$\text{COUNT}_2(push_c + push_d) - \text{COUNT}_2(pop_c + pop_d) + \text{COUNT}_2(place)] \leq \text{COUNT}_1(pick_{rc} + pick_{rd})$

The events *push* and *pop* can be written as the sum of events:

$push = push_c + push_d$

$pop = pop_c + pop_d$

The condition SEMPTY denoting whether or not the stack is empty can be written using *push* and *pop*:

SEMPTY = $\text{COUNT}_1(push) = \text{COUNT}_1(pop) = \text{COUNT}_2(push) = \text{COUNT}_2(pop)$

We define the function TOP which at any time $t \in \overline{\Pi}$ gives $k \in \overline{\mathbb{N}}$ which is the occurrence number of the push corresponding to the current top element in the stack. Hence

TOP(t) = if [$\text{LCOUNT}_1(push) \circ \text{LAST}_2(push)(t) - \text{COUNT}_1(push) \circ \text{LAST}_2(pop) \circ \text{LAST}_2(push)(t)$]

> S(t) then [$\text{LCOUNT}_1(push) \circ \text{LAST}_2(push)(t) - S(t)$]

else TOP($\text{LAST}_2(pop) \circ \text{LAST}_2(push)(t)$),

where

$$S : \overline{\Pi} \rightarrow \overline{\mathbb{N}},$$

$S = \text{COUNT}_1(pop) - \text{LCOUNT}_1(pop) \circ \text{LAST}_2(pop) \circ \text{LAST}_2(push)$

The events of picking items from the conveyor belt pushing (or

popping) items from the stack and assembling can be expressed as:

$pick_{\ell c}$ = *cup* | [LEMPTY ∧ SEMPTY]

$pick_{\ell d}$ = dish | [LEMPTY ∧ SEMPTY]

$pick_{rc}$ = *cup* | [not (LEMPTY) ∧ REMPTY]

$pick_{rd}$ = *dish* | [not (LEMPTY) ∧ REMPTY]

pop_c = place | [[$COUNT_2(push_c)$ > $COUNT_2(pop_c)$] ∧
[$LCOUNT_1(push_c)$ ∘ $TIME_1(push)$ ∘ TOP] ≠
$COUNT_1(push_c)$ ∘ $TIME_1(push)$ ∘ TOP]]

pop_d = place | [[$COUNT_2(push_d)$ > $COUNT_2(pop_d)$] ∧
[$LCOUNT_1(push_d)$ ∘ $TIME_1(push)$ ∘ TOP ≠
$COUNT_1(push_d)$ ∘ $TIME_1(push)$ ∘ TOP]]

$push_c$ = $pick_{rc}$ | [$COUNT_1(hold_c)$ > $COUNT_2(hold_c)$]

$push_d$ = $pick_{rd}$ | [$COUNT_1(hold_d)$ > $COUNT_2(hold_d)$]

assemble = $pick_{rc}$ | [$COUNT_1(hold_d)$ > $COUNT_2(hold_d)$] +
$pick_{rd}$ | [$COUNT_1(hold_c)$ > $COUNT_2(hold_c)$]

$TIME_2(hold)$ = $TIME_2(place)$

Proof of Correctness

It is sufficient to prove that the algorithm satisfies [S4]; that is, if 2n items (n of each kind) are picked up at some time t, then after a short time delay, say δ, n assemblies are completed by the

robot. We prove first that the stack at any time is either empty or contains items of the same type. Next we apply this result to prove the correctness of the algorithm.

Lemma 7

If for any $n \geq 1$,

$TIME_1(push)(n) = TIME_1(push_c)(k)$ (or $TIME_1(push_d)(k')$) and

$TIME_1(push)(n+1) = TIME_1(push_d)(j)$ (or $TIME_1(push_c)(j')$)

for some k(k'), j (j') then $\exists$ t such that

$TIME_1(push)(n) < t < TIME_1(push)(n+1)$ and SEMPTY(t) = true.

Proof

We prove this lemma by the method of contradiction. Suppose the lemma is false, then $\exists$ n such that

$TIME_1(push)(n) = TIME_1(push_c)(k)$ (or $TIME_1(push_d)(k')$) and

$TIME_1(push)(n+1) = TIME_1(push_d)(j)$ (or $TIME_1(push_c)(j')$)

and $\forall$t such that

$TIME_1(push)(n) < t < TIME_1(push)(n+1)$, SEMPTY(t) = false.

Choose the smallest n satisfying those conditions and without loss of generality, we can assume

$$TIME_1(push)(n) = TIME_1(push_c)(k)$$

and $$TIME_1(push)(n+1) = TIME_1(push_d)(j)$$

$$\Rightarrow COUNT_2(push_c) \circ TIME_1(push)(n) \geq COUNT_2(pop_c) \circ TIME_1(push)(n)$$

and $\quad \mathrm{COUNT}_2(push_d) \circ \mathrm{TIME}_1(push)(n) = \mathrm{COUNT}_2(pop_d) \circ \mathrm{TIME}_1(push)(n)$

Let $t' = \mathrm{TIME}_1(push)(n)$ and $t'' = \mathrm{TIME}_1(push)(n+1)$.

Then

$$\mathrm{COUNT}_1(hold_d)(t') = \mathrm{COUNT}_2(hold_d)(t') \tag{18}$$

$$\mathrm{COUNT}_2(push_d)(t') = \mathrm{COUNT}_2(pop_d)(t')$$

$$\Rightarrow \quad \mathrm{COUNT}_2(push_d)(t''') = \mathrm{COUNT}_2(pop_d)(t'''),\ \forall\ t''' \in [t', t''].$$

$$\Rightarrow \quad \mathrm{COUNT}_2(pop_d)(t') = \mathrm{COUNT}_2(pop_d)(t'') \tag{19}$$

$\mathrm{SEMPTY}(t) = \text{false}\ \forall t \in (t', t'')$

$$\Rightarrow \quad \mathrm{COUNT}_2(pick_{\ell c} + pick_{\ell d})(t') = \mathrm{COUNT}_2(pick_{\ell c} + pick_{\ell d})(t'').$$

$$\Rightarrow \quad \mathrm{COUNT}_2(pick_{\ell d})(t') = \mathrm{COUNT}_2(pick_{\ell d})(t'') \tag{20}$$

From (18), (19) and (20) we get

$\mathrm{COUNT}_1(hold_d)(t'') = \mathrm{COUNT}_2(hold_d)(t'')$

which is a contradiction to the fact that

$t'' = \mathrm{TIME}_1(push)(n+1) = \mathrm{TIME}_1(push_d)(j)$ for some j.

Corollary 1

$$[\text{not (SEMPTY)}] \Rightarrow [[\mathrm{COUNT}_2(push_c) > \mathrm{COUNT}_2(pop_c)] \wedge [\mathrm{COUNT}_2(push_d) = \mathrm{COUNT}_2(pop_d)]] \vee$$

$$[[\mathrm{COUNT}_2(push_d) > \mathrm{COUNT}_2(pop_d)] \wedge [\mathrm{COUNT}_2(push_c) = \mathrm{COUNT}_2(pop_c]].$$

In a similar fashion, we can prove

(i) If for any $n \geq 1$,

$\mathrm{TIME}_1(pop)(n) = \mathrm{TIME}_1(pop_c)(k)$ (or $\mathrm{TIME}_1(pop_d)(k')$)

and

$TIME_1(pop)(n + 1) = TIME_1(pop_d)(j)$ (or $TIME_1(pop_c)(j')$)

for some k(k'), j(j') then $\exists$ t such that

$TIME_1(pop)(n) < t < TIME_1(pop)(n + 1)$ and SEMPTY(t) = true.

(ii) If for any $n \geq 1$,

$TIME_1(hold)(n) = TIME_1(hold_c)(k)$ (or $TIME_1(hold_d)(k')$)

and

$TIME_1(hold)(n + 1) = TIME_1(hold_d)(j)$ (or $TIME_1(hold_c)(j')$)

for some k(k'), j(j') then $\exists$t such that

$TIME_1(hold)(n) < t < TIME_1(hold)(n + 1)$ and SEMPTY(t) = true.

Lemma 8

$SEMPTY(TIME_1(pick)(2n)) =$ true where

$$pick = pick_\ell + pick_r \text{ , } pick_\ell = pick_{\ell c} + pick_{\ell d}$$

and $pick_r = pick_{rc} + pick_{rd}$.

Proof:

Suppose not, then $SEMPTY(TIME_1(pick)(2n)) =$ false.

$\Rightarrow$ $TIME_1(pick)(2n) = TIME_1(pick_r)(k)$ for some k.

$\Rightarrow$ $LEMPTY(TIME_1(pick_r)(k)) =$ false.

Since $SEMPTY(TIME_1(pick)(2n)) =$ false, from Corollary 1 without loss of generality, we assume

$$[COUNT_2(push_c)(t) > COUNT_2(pop_c)(t)] \wedge [COUNT_2(push_d)(t) = COUNT_2(pop_d)(t)] \quad (21)$$

where $t = TIME_1(pick)(2n)$.

LEMPTY(t) = false $\Rightarrow$ $COUNT_1(hold)(t) > COUNT_2(hold)(t)$.

$COUNT_2(push_c)(t) > COUNT_2(pop_c)(t)$ and $COUNT_1(hold)(t) > COUNT_2(hold)(t)$

imply $COUNT_1(hold_c)(t) > COUNT_2(hold_c)(t)$ (22)

(21) and (22) contradict the fact that there are n objects of each kind.

Corollary 2

$COUNT_2(assemble) \circ TIME_1(pick)(2n) = n - 1$.

Proof of [S4] :

Since SEMPTY(t) = true where

$t = TIME_1(pick)(2n)$ and we have 2n objects with n of each kind,

LEMPTY(t) = false.

$\Rightarrow TIME_1(pick)(2n) = TIME_1(pick_r)(k)$ for some k.

Further, LEMPTY(t) = false implies

$COUNT_1(hold)(t) > COUNT_2(hold)(t)$.

If $COUNT_1(hold_c)(t) > COUNT_2(hold_c)(t)$ then

$TIME_1(pick)(2n) = TIME_1(pick_{rd})(k')$ for some k'

and vice versa.

This is due to the fact that there are 2n objects, n of each kind.

If we set δ to

$dur(pick_{2n}) + dur(assemble_n) + dur(place_n)$,

the proof follows.

We remark that cup and dish are synonyms for two objects and thus the above specification can be viewed as a parametric functional specification. To validate an assembly line operation involving any number of robots, the same specification can be instantiated for each robot with two parameters - the assembled object thus far and the new part to be assembled at this site.

4.4 The Design of a Robotic Navigation Controller

We consider the coordinated motion of several robots in a common workspace. Some of the significant actions characterizing this coordinated workspace are communication between robots, communication between robots and external devices, synchronization with external events, waiting and monitoring for some event occurrences, and dealing with concurrent activities. For simplicity, we assume that cartesian robots move in a large rectangular-shaped workspace and the only requirement is that collisions must be avoided. Let us also assume that the workspace has sufficient width so that three robots, each in a lane of its own, can move simultaneously. Under this assumption we give the design specification and the proof of correctness of the design of a distributed navigation controller that is to be installed at the intersections.

This problem can be further simplified and abstracted to the design of a traffic controller that is to be installed at the intersection of two two-way highways (east-west and north-south) so that cars coming at the intersection can pass the intersection in a finite amount of time with no collision. Notice that both in the original problem (with robots) and in the abstracted problem (with cars), the speed and the space (width of lane) for robot motion is not required in the specification of the controller.

We assume the following : 1) at a predetermined proximity to the intersection there are three lanes on each highway; 2) a car in the right lane must turn right; a car in the middle lane goes straight and a car in the left lane must turn left; 3) the traffic controller at the intersection should allow the car that arrives at the intersection first to cross the intersection first (without collision); in case of a tie, traffic on east-west highway has priority over the traffic on north-south highway; 4) to enhance the flow of traffic, cars that can cross the intersection in parallel should be allowed to do so; and 5) cars, when allowed to cross the intersection, do so in a finite amount time (they do not break down at the intersection).

Events

east : event that the sensor at the intersection goes high indicating the arrival of a car to go east.

west : event that the sensor at the intersection goes high

indicating the arrival of a car to go west.

north : event that the sensor at the intersection goes high indicating the arrival of a car to go north.

south : event that the sensor at the intersection goes high indicating the arrival of a car to go south.

en : event that the sensor at the intersection goes high indicating the arrival of a car from west to go north (to make a left turn at the intersection)

es : event that the sensor at the intersection goes high indicating the arrival of a car from west to go south (to make a right turn at the intersection)

wn : event that the sensor at the intersection goes high indicating the arrival of a car from east to go north (to make a right turn at the intersection)

ws : event that the sensor at the intersection goes high indicating the arrival of a car from east to go south (to make a left turn at the intersection)

ne : event that the sensor at the intersection goes high indicating the arrival of a car from south to go east (to make a right turn at the intersection)

nw : event that the sensor at the intersection goes high indicating the arrival of a car from south to go west (to make a left turn at the intersection)

se : event that the sensor at the intersection goes high indicating the arrival of a car from north to go east (to make a left turn at the intersection)

sw : event that the sensor at the intersection goes high indicating the arrival of a car from north to go west (to make a right turn at the intersection)

e-high : event that the sensor at the intersection indicating the arrival of a car to go east, remains high.

w-high : event that the sensor at the intersection indicating the arrival of a car to go west, remains high.

n-high : event that the sensor at the intersection indicating the arrival of a car to go north, remains high.

s-high : event that the sensor at the intersection indicating the arrival of a car to go south, remains high.

en-high : event that the sensor at the intersection indicating the arrival of a car from west to go north, remains high.

es-high : event that the sensor at the intersection indicating the arrival of a car from west to go south, remains high.

wn-high :event that the sensor at the intersection indicating the arrival of a car from east to go north, remains high.

ws-high : event that the sensor at the intersection indicating the arrival of a car from east to go south, remains high.

ne-high : event that the sensor at the intersection indicating the

arrival of a car from south to go east, remains high.

nw-high :event that the sensor at the intersection indicating the arrival of a car from south to go west, remains high.

se-high : event that the sensor at the intersection indicating the arrival of a car from north to go east, remains high.

sw-high : event that the sensor at the intersection indicating the arrival of a car from north to go west, remains high.

e-go : event that the car at the intersection bound to go east is allowed to cross the intersection.

w-go : event that the car at the intersection bound to go west is allowed to cross the intersection.

n-go : event that the car at the intersection bound to go north is allowed to cross the intersection.

s-go : event that the car at the intersection bound to go south is allowed to cross the intersection.

en-go : event that the car at the intersection from west bound to go north is allowed to cross the intersection.

es-go : event that the car at the intersection from west bound to go south is allowed to cross the intersection.

wn-go : event that the car at the intersection from east bound to go north is allowed to cross the intersection.

ws-go : event that the car at the intersection from east bound to go south is allowed to cross the intersection.

ne-go : event that the car at the intersection from south bound to go east is allowed to cross the intersection.

nw-go : event that the car at the intersection from south bound to go west is allowed to cross the intersection.

se-go : event that the car at the intersection from north bound to go east is allowed to cross the intersection.

sw-go : event that the car at the intersection from north bound to go west is allowed to cross the intersection.

e-cross : event that the car at the intersection bound to go east crosses the intersection.

w-cross : event that the car at the intersection bound to go west crosses the intersection.

n-cross : event that the car at the intersection bound to go north crosses the intersection.

s-cross : event that the car at the intersection bound to go south crosses the intersection.

In a similar fashion, we define events *ne-cross*, *nw-cross*, *se-cross*, *sw-cross*, *en-cross*, *es-cross*, *wn-cross* and *ws-cross*.

e-low : event that the sensor which senses the arrival of the east-bound car going low.

Similarly, we define events *w-low*, *n-low*, *s-low* ...

Formal Specification of the problem

Any design of the traffic controller should satisfy the following two

conditions.

1) At any time t, there should not be any collision between the cars at the intersection.

2) Cars that go to the intersection should be allowed to cross the intersection within a finite amount of time. That is, the system should be starvation-free. This ensures that there is no deadlock in the system.

Next, we define the conditions using the above events that in turn lead to the predicate for a collision free motion of the cars.

$$\text{INTER-E} = \text{COUNT}_1(e\text{-}go) > \text{COUNT}_2(e\text{-}cross)$$

$$\text{INTER-W} = \text{COUNT}_1(w\text{-}go) > \text{COUNT}_2(w\text{-}cross)$$

$$\text{INTER-N} = \text{COUNT}_1(n\text{-}go) > \text{COUNT}_2(n\text{-}cross)$$

$$\text{INTER-S} = \text{COUNT}_1(s\text{-}go) > \text{COUNT}_2(s\text{-}cross)$$

$$\text{INTER-EN} = \text{COUNT}_1(en\text{-}go) > \text{COUNT}_2(en\text{-}cross)$$

$$\text{INTER-ES} = \text{COUNT}_1(es\text{-}go) > \text{COUNT}_2(es\text{-}cross)$$

$$\text{INTER-WN} = \text{COUNT}_1(wn\text{-}go) > \text{COUNT}_2(wn\text{-}cross)$$

$$\text{INTER-WS} = \text{COUNT}_1(ws\text{-}go) > \text{COUNT}_2(ws\text{-}cross)$$

$$\text{INTER-NE} = \text{COUNT}_1(ne\text{-}go) > \text{COUNT}_2(ne\text{-}cross)$$

$$\text{INTER-NW} = \text{COUNT}_1(nw\text{-}go) > \text{COUNT}_2(nw\text{-}cross)$$

$$\text{INTER-SE} = \text{COUNT}_1(se\text{-}go) > \text{COUNT}_2(se\text{-}cross)$$

$$\text{INTER-SW} = \text{COUNT}_1(sw\text{-}go) > \text{COUNT}_2(sw\text{-}cross)$$

The meaning of these conditions should be clear; for example, the condition INTER-E is true if there is an east-bound car that is

allowed to cross the intersection but has not yet crossed the intersection.

E = INTER-S ∨ INTER-N ∨ INTER-WS ∨ INTER-NW

EL = INTER-S ∨ INTER-WS ∨ INTER-NW ∨ INTER-W ∨ INTER-SE

W = INTER-S ∨ INTER-N ∨ INTER-EN ∨ INTER-SE

WL = INTER-N ∨ INTER-EN ∨ INTER-E ∨ INTER-NW ∨ INTER-SE

S = INTER-E ∨ INTER-W ∨ INTER-NW ∨ INTER-EN

SL = INTER-W ∨ INTER-NW ∨ INTER-N ∨ INTER-EN ∨ INTER-WS

N = INTER-E ∨ INTER-W ∨ INTER-SE ∨ INTER-WS

NL = INTER-E ∨ INTER-SE ∨ INTER-S ∨ INTER-EN ∨ INTER-WS

NO-COLLISION = [INTER-E → ~E] ∧ [INTER-EN → ~EL] ∧ [INTER-W → ~W] ∧ [INTER-WS → ~WL] ∧ [INTER-S → ~S] ∧ [INTER-SE → ~SL] ∧ [INTER-N → ~N] ∧ [INTER-NW → ~NL].

Liveness condition (Starvation-free system) can be stated as: The duration of each occurrence of the events *e-high*, *w-high* ... is finite. That is,

dur_i(*e-high*) = finite, ∀ i.

dur_i(*w-high*) = finite, ∀ i and so on.

Specification of the design of the traffic controller

Some of the relations governing the events are the following:

e-high << *east*

w-high << *west*

n-high << *north*

s-high << *south*

en-high << *en*

es-high << *es*

wn-high << *wn*

ws-high << *ws*

ne-high << *ne*

nw-high << *nw*

se-high << *se*

sw-high << *sw*

If an east-bound car arrives at the intersection, it should be allowed to cross the intersection, provided the intersection is clear. The following conditions are defined for specifying this requirement:

Let,

$$\text{E-BEFORE-N} = [\text{COUNT}_2(n\text{-}go) = \text{COUNT}_1(n\text{-}high)] \vee [[\text{COUNT}_2(n\text{-}go) < \text{COUNT}_1(n\text{-}high)] \wedge [\text{TIME}_1(n\text{-}high) \circ (\text{COUNT}_2(n\text{-}high) + 1) \geq \text{TIME}_1(e\text{-}high) \circ (\text{COUNT}_2(e\text{-}high) + 1)]]$$

$$\text{E-BEFORE-S} = [\text{COUNT}_2(s\text{-}go) = \text{COUNT}_1(s\text{-}high)] \vee [[\text{COUNT}_2(s\text{-}go) < \text{COUNT}_1(s\text{-}high)] \wedge [\text{TIME}_1(s\text{-}high) \circ (\text{COUNT}_2(s\text{-}high) + 1) \geq \text{TIME}_1(e\text{-}high) \circ (\text{COUNT}_2(e\text{-}high) + 1)]]$$

$$\text{E-BEFORE-WS} = [\text{COUNT}_2(ws\text{-}go) = \text{COUNT}_1(ws\text{-}high)] \vee [[\text{COUNT}_2(ws\text{-}go) < \text{COUNT}_1(ws\text{-}high)] \wedge [\text{TIME}_1(ws\text{-}high) \circ (\text{COUNT}_2(ws\text{-}high) + 1) \geq \text{TIME}_1(e\text{-}high) \circ (\text{COUNT}_2(e\text{-}high) + 1)]]$$

$$\text{E-BEFORE-NW} = [\text{COUNT}_2(nw\text{-}go) = \text{COUNT}_1(nw\text{-}high)] \vee [[\text{COUNT}_2(nw\text{-}go) < \text{COUNT}_1(nw\text{-}high)] \wedge [\text{TIME}_1(nw\text{-}high) \circ (\text{COUNT}_2(nw\text{-}high) + 1) \geq \text{TIME}_1(e\text{-}high) \circ (\text{COUNT}_2(e\text{-}high) + 1)]]$$

$$\text{E-BEFORE-SE} = [\text{COUNT}_2(se\text{-}go) = \text{COUNT}_1(se\text{-}high)] \vee [[\text{COUNT}_2(se\text{-}go) < \text{COUNT}_1(se\text{-}high)] \wedge [\text{TIME}_1(se\text{-}high) \circ (\text{COUNT}_2(se\text{-}high) + 1) \geq \text{TIME}_1(e\text{-}high) \circ (\text{COUNT}_2(e\text{-}high) + 1)]]$$

In a similar fashion, we define functions W-BEFORE-N, W-BEFORE-S, W-BEFORE-EN, W-BEFORE-NW, W-BEFORE-SE, N-BEFORE-NE, N-BEFORE-EN, N-BEFORE-WS, S-BEFORE-NW, S-BEFORE-EN, N-BEFORE-WS, EN-BEFORE-WS, SE-BEFORE-NW, SE-BEFORE-EN, SE-BEFORE-WS, NW-BEFORE-EN, NW-BEFORE-WS.

Due to symmetry we have,

$$\text{N-BEFORE-E} = [\text{COUNT}_2(\textit{e-go}) = \text{COUNT}_1(\textit{e-high})] \vee [[\text{COUNT}_2(\textit{e-go}) < \text{COUNT}_1(\textit{e-high})] \wedge [\text{TIME}_1(\textit{e-high}) \circ (\text{COUNT}_2(\textit{e-high}) + 1) > \text{TIME}_1(\textit{n-high}) \circ (\text{COUNT}_2(\textit{n-high}) + 1)]]$$

Similarly, we define functions N-BEFORE-W, S-BEFORE-E, S-BEFORE-W, EN-BEFORE-S, EN-BEFORE-N, EN-BEFORE-W, WS-BEFORE-S, WS-BEFORE-N, WS-BEFORE-E, SE-BEFORE-E, SE-BEFORE-W, SE-BEFORE-N, NW-BEFORE-E, NW-BEFORE-W, NW-BEFORE-S, EN-BEFORE-NW, EN-BEFORE-SE, WS-BEFORE-NW, WS-BEFORE-SE, WS-BEFORE-EN, NW-BEFORE-NE.

All events are not independent; for example we have:

$$\textit{e-go} = (\textit{e-high} + \textit{n-high} + \textit{s-high} + \textit{ws-high} + \textit{nw-high} + 1) \mid [\text{COUNT}_2(\textit{e-go}) < \text{COUNT}_1(\textit{e-high})] \wedge \text{E-BEFORE-N} \wedge \text{E-BEFORE-S} \wedge \text{E-BEFORE-WS} \wedge \text{E-BEFORE-NW}.$$

$$\textit{w-go} = (\textit{w-high} + \textit{n-high} + \textit{s-high} + \textit{en-high} + \textit{se-high} + 1) \mid [\text{COUNT}_2(\textit{w-go}) < \text{COUNT}_1(\textit{w-high})] \wedge \text{W-BEFORE-N} \wedge \text{W-BEFORE-S} \wedge \text{W-BEFORE-EN} \wedge \text{W-BEFORE-SE}.$$

$$\textit{en-go} = (\textit{en-high} + \textit{s-high} + \textit{ws-high} + \textit{nw-high} + \textit{se-high} + \textit{w-high} + 1) \mid [\text{COUNT}_2(\textit{en-go}) < \text{COUNT}_1(\textit{en-high})] \wedge$$

EN-BEFORE-S ∧ EN-BEFORE-WS ∧ EN-BEFORE-NW

∧ EN-BEFORE-W ∧ EN-BEFORE-SE.

ws-go = (*ws-high* + *n-high* + *en-high* + *nw-high* + *se-high* + *e-high* + 1) | [COUNT_2(*ws-go*) < COUNT_1(*ws-high*)] ∧

WS-BEFORE-N ∧ WS-BEFORE-EN ∧ WS-BEFORE-NW

∧ WS-BEFORE-E ∧ WS-BEFORE-SE.

In a similar fashion, we give the specification of the events *s-go*, *se-go*, *n-go* and *nw-go*.

ne-go = (*ne-high* + 1) | [COUNT_1(*ne-high*) > COUNT_2(*ne-go*)]

sw-go = (*sw-high* + 1) | [COUNT_1(*sw-high*) > COUNT_2(*sw-go*)]

es-go = (*es-high* + 1) | [COUNT_1(*es-high*) > COUNT_2(*es-go*)]

wn-go = (*wn-high* + 1) | [COUNT_1(*wn-high*) > COUNT_2(*wn-go*)]

Some of the partial orderings and the completion time sequences of these new events are:

e-cross << *e-go*

w-cross << *w-go*

n-cross << *n-go*

s-cross << *s-go*

en-cross << *en-go*

es-cross << *es-go*

wn-cross << *wn-go*

ws-cross << *ws-go*

ne-cross << *ne-go*

nw-cross << *nw-go*

se-cross << *se-go*

sw-cross << *sw-go*

e-low << *e-cross*

w-low << *w-cross*

n-low << *n-cross*

s-low << *s-cross*

$\text{TIME}_2(e\text{-}high) = \text{TIME}_2(e\text{-}low)$

$\text{TIME}_2(w\text{-}high) = \text{TIME}_2(w\text{-}low)$

$\text{TIME}_2(n\text{-}high) = \text{TIME}_2(n\text{-}low)$

$\text{TIME}_2(s\text{-}high) = \text{TIME}_2(s\text{-}low)$

In a similar fashion, we define events *en-low*, *ws-low* ... and the completion time sequence of the events *en-high*, *ws-high* and so on.

Proof of correctness of the traffic controller design

We prove that the design of the traffic controller meets the two criteria, namely, 1) there is no collision and 2) the system is starvation-free.

Proof of 1) :

From the definition of the conditions E, EL, W, WL, N, NL, S and SL, it is clear that the condition for collision free system is equivalent to the following condition:

[[INTER-E → ~E] ∧ [INTER-EN → ~EL]] ∨

[[INTER-E → ~E] ∧ [INTER-W → ~W]] ∨

[[INTER-E → ~E] ∧ [INTER-SE → ~SL]] ∨

[[INTER-EN → ~EL] ∧ [INTER-N → ~N]] ∨

[[INTER-W → ~W] ∧ [INTER-WS → ~WL]] ∨

[[INTER-W → ~W] ∧ [INTER-NW → ~NL]] ∨

[[INTER-WS → ~WL] ∧ [INTER-S → ~S]] ∨

[[INTER-N → ~N] ∧ [INTER-NW → ~NL]] ∨

[[INTER-S → ~S] ∧ [INTER-SE → ~SL]].

We prove that [[INTER-E → ~E] ∧ [INTER-EN → ~EL]]; a similar proof can be given for other clauses. Proving INTER-E → ~E is equivalent to proving the events

i) *e-go* + *s-go*, ii) *e-go* + *n-go*, iii) *e-go* + *ws-go* and iv) *e-go* + *nw-go* are single-occurrent events.

From the definition of conditions E-BEFORE-S and S-BEFORE-E, they cannot be true simultaneously. At any time t, at most one of them can be true. For the event *e-go* to start occurring, E-BEFORE-S should be true at that time. For the event *s-go* to start occur, S-BEFORE-E should be true at that time. Suppose E-BEFORE-S is true at time t and the event *e-go* starts at that time, then S-BEFORE-E is false at least until time t′ at which the corresponding occurrence of the event *e-high* finishes. Hence *s-go* cannot start up until that time. Similarly, if an occurrence of *s-go*

starts, until the corresponding occurrence of *s-high* finishes, an occurrence of *e-go* cannot start. This implies, *e-go* + *s-go* is single-occurrent. Similarly we can prove (ii), (iii) and (iv). Similar argument can be carried out to prove the other condition [INTER-EN $\rightarrow$ $\sim$EL].

Proof of 2) :

We prove that $dur_i(ws\text{-}high)$ is finite $\forall$ i $\in$ $\mathbb{N}$. When an occurrence of the event *ws-high* starts happening (a west-bound car that is about to make a left turn at the intersection is sensed), there might be a maximum of six cars that have to cross the intersection before this car is allowed to cross the intersection. According to our assumption, cars do not break down at the intersection and they take only finite amount of time to cross the intersection. This implies, $dur_i(ws\text{-}high)$ is finite $\forall$ i.

CHAPTER 5

CONCLUSION

Most real-time systems are either concurrent or distributed with timing constraints on the actions and computations. There are two points of view in describing such systems : internal (structure) and external (behavior). Present day specification methodologies emphasize and try to decouple these points of views. From a user's point of view only the behavior description is important. It is important to assure the user that stringent timing constraints are met and imperfect execution environments do not arise. Hence any safety assertion stated by the user must be related to stated timing constraints or those derivable from initial specifications. This thesis has made contributions to time dependent specification, behavior description and formal reasoning of real-time systems; this is achieved without any regard to any formal (internal)representation and computation model. The model due to Caspi [Caspi 86] has been generalized and enriched. The expressive power of this extended model is illustrated through the specification of a spectrum of problems chosen from hardware arbiter design, database and robotics.

After briefly reviewing a selected set of specification models in Chapter 2, we described the generalized functional formalism in Chapter 3. Relaxing the constraints that events are instantaneous we showed an algebraic structure for events and provided tools for problem specifications. The new results of this thesis are contained in Chapters 3 and 4. The specifications and proofs given in Chapter 4 are significant and non-trivial contributions. They demonstrate the adequacy of time-dependent functions defined in Chapter 3 for specification of distributed real-time systems behavior such as robotics. We can claim with some degree of confidence that the formalism presented in this thesis provides a unifying framework for several other problems in this field.

Although the class of problems that we have chosen to illustrate the formalism are from different areas and levels of complexity, a great deal of investigation is yet to be done on this approach. We mention some of these next.

From a designer's point of view, the structure (internal view) of a real-time distributed system is significant. Both control-related and data-related properties, actions and computations must be specified. Since resource-limitations, in addition to timing constraints, are central to the design and implementation of real-time programs, they must also be specified in the event-based model. Only then, correct semantics for real-time programs can be given. Moreover, the

primary objective of the internal view specification is to provide a document which the design and implementation team can use. The choice of the functional model necessarily forces the structure of specification to be functional and thus the design and implementation descriptions are close to being functional programs. The natural next step is the validation of this detailed specification (implementation level specification). Because it is close to being a functional program, one can make it a functional program that is executable. This step, once again, requires semantics for real-time functional constructs introduced in the model that are used in the actual functional program. We believe that the definitions of time delay and shift operators, conditions and counters discussed in Chapter 3 provide sound semantics for similar constructs when used in functional programs. In Alagar et al [Alagar 86], a functional language with real-time constructs was proposed for robotic programming. It is important to investigate the adequacy of the model for internal specification of a system with limited resources and derive the semantics of time dependent constructs defined in Chapter 3 within this context. This investigation pertains to the notion of applicative real-time programming and is an open problem.

Finally, the suitability of this approach for specification of network systems can be investigated. The several layers spanning from low level protocols to application level algorithms in a network

may be specified under one framework. If specification at one level is viewed as a transformation of the specification at the next higher level, a conventional method would require proving the correctness of transformation at each level. This does not seem necessary in this formalism, since the same functional model can support all levels.

References

[Alagar 86] Alagar, V.S., Grogono, P.D., Ramanathan, G., "A Functional Language for Robotic Programming", Proc. Japan-USA Symposium on Flexible Automation, Osaka, Japan, pp.739-747, July 1986.

[Alagar 87] Alagar, V.S., Grogono, P.D., Ramanathan, G., "On the Design of a Programmable Robotic System", IFAC Proc. on Theory of Robots, Vienna, pp.315-318, 1987.

[Alagar 88a] Alagar, V. S., Ramanathan, G., "Formal Environment and Tools Description for the Analysis of Real-Time Concurrent Systems", Workshop on the Specification and Verification of Concurrent Systems, Stirling, Scotland, July 1988.

[Alagar 88b] Alagar, V. S., Ramanathan, G., "A Functional Model for Specification and Analysis of Distributed Real-Time Systems: Formalism and Applications", submitted for publication, October 1988.

[Ashcroft 71] Ashcroft, E., Manna, Z., "Formalization of Properties of Parallel Programs", *Machine Intelligence* 6 (1971), Edinburgh University Press, pp. 17-41.

[Bennett88] Bennett, S. M., Minkowitz, C., Rowles, J. S., *"me too* reference manual", STC Technology Ltd and Stirling University, 1988.

[Bernstein 80] Bernstein, P. A., Shipman, D. W., "The Correctness of Concurrency Control Mechanisms in a System for Distributed Databases (SDD-1)", *ACM TRANS. Database Systems*, Vol. 5, No. 1 (1980), pp. 52-68.

[Bernstein 81] Bernstein, P. A., Goodman, N., "Concurrency Control in Distributed Database Systems", *Computing Surveys*, Vol. 13, No. 2 (1981) pp. 185-221.

[Bernstein 87] Bernstein, P. A., Goodman, N., "A Proof Technique for Concurrency Control and Recovery Algorithms for Replicated Databases", *Distributed Computing*, Vol. 2 (1987), pp. 32-44.

[Bochman 82] Bochman, G. V., "Hardware Specification with Temporal Logic: An Example", *IEEE Trans. on Computers*, Vol. C-31, No. 3 (1982), pp.

[Browne 85] Browne, M., Clarke, E., Dill, D., Mishra, B., "Automatic Verification of Sequential Circuits using Temporal Logic", in *Computer Hardware Description Languages and their Applications*, C-J. Koomen and T. Moto-Oka (eds.), Elsevier Science Publishers B. V. (1985).

[Burstal 74] Burstal, R. M., "Program Proving as Hand Simulation with a Little Induction", Information Processing 74, North Holland (1974), pp. 308-312.

[Caspi 86] Caspi, P., Halbwachs, N., "Functional Model for Describing and Reasoning about Time Behaviour of Computing Systems", *Acta Informatica*, Vol. 22 (1986), pp. 595-627.

[Clark 88] Clark, R. G., "The Design and Development of Ada Real-Time Embedded Systems", Proceedings of BCS-FACS Workshop on Specification and Verification of Concurrent Systems, Stirling, Scotland, July 1988.

[Cohen 86] Cohen, B., Harwood, W.T., Jackson, M. I., *The Specification of Complex Systems*, Addison-Wesley Publishing company, 1986.

[Dasarathy 85] Dasarathy, B., "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them", *IEEE Trans. Software Eng.*, Vol. SE-11 (1985), pp. 80-86.

[Dennis 74] Dennis, J.B., "First Version of a Data Flow Procedural Language", Proc. Colloque Sur Programmation, LNCS 19, 1974.

[Dijkstra 81] Dijkstra, E. W., "An Assertional Proof of a Protocol by G. L. Peterson", Tech. Report EWD779, Burroughs Corp., (1981).

[Ellis 77] Ellis, C. A., "Consistency and Correctness of Duplicate Database Systems", Proc. 6th Symp. Operat. Syst. Principles, (1977), pp. 67-84.

[Gehani 82] Gehani, N., H., "Specifications : Formal and informal - A Case Study", *Software-Practice and Experience*, 12, pp.433-444.

[Grenrich 79] Grenrich, H. J., Lautenbach, R., "The Analysis of Distributed Systems by Means of Predicate/Transition Nets", *Semantics of Concurrent Computation*, G. Kahn, Ed. New York: Springer-Verlag (1979), pp. 123-146.

[Hailpern 82] Hailpern, B. T., *Verifying Concurrent Processes Using Temporal Logic*, Lecture Notes in Computer Science, 129, Springer-Verlag, (1982).

[Hayes 87] Hayes, I., *Specification Case Studies*, Prentice-Hall International, 1987.

[Henderson 86] Henderson, P., "Functional Programming, Formal Specification and Rapid Prototyping", *IEEE Transactions on Software Engineering*, vol.SE-12, No.2, 1986, pp.241-250.

[Hoare 72] Hoare, C. A. R., "Towards a Theory of Parallel Programming", in *Operating Systems Techniques*, C. A. R. Hoare and R. Perrot (eds.), Academic Press, New York (1972).

[Hoare 75] Hoare, C. A. R., "Parallel Progamming: An Axiomatic Approach", *Computer Languages*, Pergamon Press (1975), pp. 151-160.

[Hoare 85] Hoare, C. A. R., *Communicating Sequential Processes*, Prentice Hall, International, New Jersey (1985).

[Holler 74] Holler, E., "Multiple Copy Files in Computer Networks", Kernforschungszentrum, Karlsruhe (1974).

[Jahanian 86] Jahanian, F. and Mok, A. K., "Safety Analysis of Timing Properties in Real-time Systems", IEEE Transactions on Software Engineering, Vol. SE-12, No. 9, (1986), pp. 890-904.

[Kahn 74] Kahn, G., "A Simple Theory of Parallel Programs", IFIP Congress Proceedings (1974).

[Koymans 83] Koymans, R., de Roever, W. P., "Examples of Real-Time Temporal Logic Specification", Workshop on The Analysis of Concurrent Systems, LNCS 207 (1983), pp. 231-251.

[Lamport 76] Lamport, L., "Towards a Theory of Correctness for Multi-User Data Base Systems", Report CA-7610-7612, Massachusetts Computer Associates, Wakefield, MA., October 1976.

[Lamport 78a] Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System", *CACM* 21, 7 (July 1978), pp. 558-565.

[Lamport 78b] Lamport, L., "The Implementation of Reliable Distributed Multiprocess Systems", *Computer Networks* 2 (1978), pp. 95-114.

[Lamport 80a] Lamport, L., "'Sometime' is Sometimes 'Not Never': A Tutorial on the Temporal Logic of Programs", Proceedings of the Seventh Annual Symposium on Principles of Programming Languages, ACM SIGACT-SIGPLAN (January 1980).

[Lamport 80b] Lamport, L., "The 'Hoare Logic' of Concurrent Programs", *Acta Informatica* 14 (1980).

[Lamport 82a] Lamport, L., Milliar-Smith, P. M., "Synchronizing Clocks in the Presence of Faults", Op. 60, Computer Science Laboratory, SRI International, Menlo Park, CA (March 1982).

[Lamport 82b] Lamport, L., "An Assertional Correctness Proof of a Distributed Algorithm", *Science of Computer Programming* 2, 3 (December 1982), pp. 175-206.

[Lamport 83] Lamport, L., "What Good is Temporal Logic?" Information Processing 83, R. E. A. Mason, ed., (1983) North Holland, Amsterdam.

[Lamport 84] Lamport, L., Schneider, F. B., "The 'Hoare Logic' of CSP and All That", *ACM TOPLAS* 6, 2 (April 1984).

[Lamport 85] Lamport, L., Schneider, F. B., "Formal Foundation for Specification and Verification", in *Distributed Systems - Methods and Tools for Specification*, LNCS 190 (1985), pp. 203-285.

[Lamport 86] Lamport, L., "The Mutual Exclusion Problem : Part I - A Theory of Interprocess Communication", JACM, Vol. 33, No. 2, (1986), pp. 313-326.

[Lamport 86] Lamport, L., "The Mutual Exclusion Problem : Part II - Statement and Solutions", JACM, Vol. 33, No. 2, (1986), pp. 327-348.

[Lauer 79] Lauer, P. E., Torrigiani, P. R. , Shields, M. W., "COSY — A System Specification Language Based on Paths and Processes", *Acta Informatica*, Vol. 12 (1979)

[Manna 84] Manna, Z., Wolper, P. L.,"Synthesis of Computing Processes from Temporal Logic Specification", *ACM TOPLAS*, Vol. 6, No. 1 (1984), pp. 68-93.

[Milne 85] Milne, G., "CIRCAL and the Representation of Communication, Concurrency and Time", *ACM TOPLAS*, Vol. 7, No. 2 (1985), pp. 270-298.

[Milner 80] Milner, R., "A Calculus of Communicating Systems", LNCS, Vol. 92 (1980).

[Milner 83] Milner, R., "Using Algebra for Concurrency: Some Approaches", Workshop on The Analysis of Concurrent Systems, LNCS 201 (1983), pp. 7-25.

[Minsky 72] Minsky, M., *Computation: Finite and Infinte Machines*, Prentice-Hall, 1972.

[Moszkowski 83] Moszkowski, B., "A Temporal Analysis of Some Concurrent Systems", Workshop on The Analysis of Concurrent Systems, LNCS 207 (1983), pp. 359-364.

[Moszkowski 85] Moszkowski, B., "Temporal Logic for Multilevel Reasoning about Hardware", *IEEE Computer*, Vol. 18, No. 2 (1985), pp. 10-19.

[Moszkowski 86] Moszkowski, B., *Executing Temporal Logic Programs*, Cambridge University Press (1986).

[Ould 87] Ould Kaddour, N., Courvoisier, M., "Issues for Concurrent Programming Real-Time Systems", IEEE Int'l. Conference on Robotics and Automation (1987), pp. 1469-1474.

[Owicki 76] Owicki, S., Gries, D., "An Axiomatic Proof Technique for Parallel Programs I", *Acta Informatica*, No. 6 (1976), pp. 319-340.

[Ozsu 85] Ozsu, M. T., "Modeling and Analysis of Distributed Database Concurrency Control Algorithms Using an Extended Petri Net Formalism", *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 10 (1985), pp. 1225-1239, 1985.

[Petri 62] Petri, C. A., "Kommunikation mit Automaten", Schriften des IIM 2, Institute für Instrumentelle Mathematik, Bonn, West Germany (1962).

[Pnueli 77] Pnueli, A., "The Temporal Logic of Programs", Proc. of the 18th Symposium on the Foundations of Computer Science, IEEE (1977).

[Ramamoorthy 80] Ramamoorthy, C. V., Ho, G. S., "Performance Evaluation of Asynchronous Concurrent Systems using Petri Nets", *IEEE Trans. Software Eng.*, Vol. SE-6 (1980), pp. 440-449.

[Ravichandran 86] Ravichandran, A., Shyamasundar, R. K., "Correctness Proof of Majority Consensus Algorithm", *Information Sciences* 38 (1986), pp. 213-227.

[Sanchis 77] Sanchis, L. E., "Data Types as Lattices: Retractions, Closures and Projection", *RAIRO Theoretical Computer Science*, Vol. 11 (1977), pp. 329-344.

[Schwartz 82] Schwartz, R. L., Melliar-Smith, P. M., "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", *IEEE Trans. on Communications*, Vol. CAM-30 (1982), pp. 2486-2496.

[Shin 87] Shin, K. G., Epstein, M. E., "Intertask Communications in an Integrated Multirobot System", *IEEE J. of Robotics and Automation*, Vol. RA-3, No. 2 (1987), pp. 90-100.

[Smoliar 79] Smoliar, S. W., "Using Applicative Technique to Design Distributed Systems", in Proceedings Specifications of Reliable Software Conference, Cambridge, MA, 1979, pp. 150-161.

[Smoliar 81] Smoliar, S. W., "Applicative and Functional Programming", in *Software Engineering Handbook* edited by C. V. Ramamoorthy and C. R. Vick, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Thiagarayam 83] Thiagarayam, P. S., "Some Aspects of Net Theory", Workshop on The Analysis of Concurrent Systems, LNCS 207 (1983), pp. 26-54.

[Thomas 79] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases", *ACM Trans. on Database Systems*, Vol. 4 (1979), pp. 180-209.

[Zave 82] Zave, P.,"An Operational Approach to Requirements Specification for Embedded Systems", *IEEE Trans. Software Eng.*, Vol. SE-8 (1982), pp. 250-269.