Automating the Analysis and Design

of

Discrete Communicating Processes

Kevin Rea

Department of Electrical Engineering
McGill University
Montreal, Quebec
August 1984

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fullfillment of the requirements for
the degree of Masters of Electrical Engineering

Abstract

Large systems, such as operating systems and computer communication networks, can generally be thought of as networks of interacting components or processes. As the size of systems increase the complexity of the component interactions becomes incomprehensible. This, in turn, leads to costly errors when informal design methods are used. An objective methodology for the analysis of communicating processes is needed.

Here we have evaluated the modeling and analytical capabilities of an algebraic theory of communicating processes by applying it to a variety of problems. To accomplish this, the analysis procedures were implemented in LISP, yielding a potentially powerful tool for the design of such systems. The theory was found to have a broad applicability and is particularly well suited to the analysis of communication protocols.

In addition, the equational nature of the theory makes it possible to 'solve for' or 'synthesize' processes to satisfy equationally expressed constraints. The synthesis procedures were also automated introducing the novel area of automatic process design.

Sommaire

Les grands systèmes, tels que les systèmes d'exploitation des ordinateurs et les réseaux de communications informatiques, peuvent, de façon générale, être considérés comme des réseaux de composants ou de processus interactifs. Plus le système est grand, plus les interactions entre ses composants sont complexes. L'usage de méthodes informelles de conception peut résulter en erreurs coûteuses; une méthodologie objective pour l'analyse des processus en communication est requise

Ici, nous avons evalué une théorie algébrique des processus en communication. La pertinence de cette théorie pour l'analyse et le développement de modèles de ces processus a été mesurée en l'appliquant à la solution de différents types de systèmes. Les procédures d'analyses ont été produites en LISP, fournissant ainsi un outil puissant très utile pour la conception de tels systèmes. La théorie s'est révélée très souple et est applicable, entre autres, à l'analyse des protocoles de communication.

La nature mathématique de la théorie permet de 'résoudre' ou 'synthétiser' des processus pour satisfaire des contraintes exprimées sous forme d'equations. Les procédures de synthèse ont été automatisées également, introduisant le domaine nouveau de la conception automatique de processus.

Table of Contents

List of Figures

Acknowledgements

I would like to thank my thesis supervisor Dr. R. de B. Johnston for many helpful and illuminating discussions as well as for his guidance both in the theoretical aspects of this work and the preparation of this document.

Chapter 1 Introduction

Consider the notion of a process. Operating systems, digital circuits, communication protocols, concurrent computations and distributed systems in general are commonly referred to as processes or as being composed of processes. What common aspect of these things is captured by the process abstraction? Each involves interactions between several dynamic components. In this light, a process may be viewed as an object which evolves in time through interaction with its external environment. This interaction implies, in fact, a flow or exchange of information.

This thesis deals with discrete communicating processes (DCP's) which comprise a subset of the set of processes. This class consists of processes that interact through the exchange of information in the form of discrete messages or signals. These messages may assume any form whatever: symbols, electrical impulses, binary words, etc. A wide range of physical systems can be modeled as DCP'S.

Specifically this thesis is concerned with the evaluation of the formal theory of discrete communicating processes as developed by R. Johnston [1]. This theory is in turn based on the algebraic ideas of Smyth [2] and Plotkin [3] and on the

process modeling notions of Milner [4] and Milne [5]. As we shall soon see, this formulation has firm mathematical footings and provides a possible basis for the objective analysis of communicating process behaviour. Since process behaviour is not a physical quantity, validation of the theory cannot be accomplished by empirical measurements. Assessment of the modeling and analytical capabilities can only be achieved through application of the model to a variety of specific problems and comparison of the results obtained with intuitive expectations

From a practical stand point, the long and tedious computations required by the operators ingrained in the theory needed to be automated. Thus, after formally introducing the DCP model, this document proceeds to describe the automation of the process operators. During this part of the work, it became apparent that the equational nature of the theory meant that an unknown process variable could be 'solved for' or in some sense synthesized This led to the development of algorithms permitting the automatic generation of a process that satisfied equationally expressed constraints.

In summary, the results were (1) translation of mathematically defined operators into working programs, (2) evaluation of the underlying theory through its application to a variety of problems, and (3), development of working synthesis procedures. The results were encouraging in that the answers computed were in agreement with intuitive expectations

1.1 Motivation

In the past systems have been designed, with relative success, by informal meth-

ods. Today, however, systems tend to be larger and hence involve more complex interactions Increasingly numerous and complex communication protocols are necessary to implement computer networks and distributed systems [6]. Current speed limitations of computers with sequential Von Neumann type processers suggest that the next, so-called 'fifth generation', computers will be highly parallel or concurrent machines using many rather than a few processers [7]. In these contexts the inherent explosion of detail often results in designs that exhibit unexpected errors and undesirable behaviour which may have adverse economic implications.

This points to a need to rationalize the design process and to formalize our understanding of interacting systems The potential advantages of this type of approach are clear A suitable theory would provide conceptual clarity for designers as well as the framework for analytical verifications of designs This latter aspect could lead to financial savings as the exercise of physical implementation and failure could be short circuited

The complexity of system interactions suggests that computer aided design tools will not only be appropriate, but required for any practical design methodology. The development of such tools is a major part of the work reported here

1.2 Issues

Deep at the heart of any attempt to formalize or rationalize a discipline one finds the concept of abstraction or modeling. The purpose of abstraction is to distill from the myriad of details the facts considered important or germane to a particular problem. Through this means we are able to free our minds from

the burdensome task of considering all the irrelevant details and hence are able to accomplish more. A model of a phenomenon is a representation. Through manipulation of this representation we hope to gain knowledge without the cost, danger, or inconvenience of dealing directly with physical reality.

Modeling has been used to great advantage in almost all disciplines of science. In astronomy, where the amounts of energy and the time scale are prohibitive, models are used to study the birth and evolution of stars. In sociology, where the manipulation of groups of real human beings might cause ethical problems, models of human behaviour are used. Mathematical models enable the study of certain biological systems where otherwise the quantities of time, space, and food would be impossible to manage.

A good model must encapsulate all the details that are considered relevant in the thing that we wish to represent. In this case we wish to represent systems, so our model should retain the fundamental idea that systems are composed of separate interacting components. The process abstraction must also portray the dynamic evolving nature of systems, and do so without particular concern for the specific instances of implementation details. Our primary desire is to be able to reason and draw conclusions about process behaviour in a given environment.

There exists, in the literature, a wealth of process models too numerous to be considered in detail here. Most of these, however, can be loosely grouped into two or three descriptive categories.

Finite state machine (FSM) models were one of the first types to be proposed for, among other things, the specification of communication protocols [8] A finite state machine consists of a finite set of states and possible transitions between them Typically, transitions may occur spontaneously or be coupled with the transitions of another machine This construct allows certain modeled operations to be synchronized One of the problems with FSM s is that they are, by definition, finite and cannot be used to represent systems having a possibly infinite number of states

Petri nets and related models have a broader modeling ability than FSM's This follows from the fact that any FSM can be represented as a Petri net, but the converse does not hold The Petri net model can handle the possibility of an infinite number of states by allowing the number of tokens within a net to grow without bound [9] The main drawback to Petri nets (as well as FSM's) is that there is a 'state' explosion as the modeled system becomes more complex Several attempts at circumventing this problem through enhancements to the basic model have been proposed [9,10] Though these enhancements do result in a broader modeling scope and a more compact representation, their generality makes any sort of formal analysis difficult

High level programming languages have also been used as process models In this case the modeled system is represented as a set of statements in a formal language As these languages are universal, their modeling capabilities are broader than either FSM s or Petri nets [8] Although formal languages are convenient for representing numbers variables and data they do not do so well in representing control structures [8] On the other hand, both FSM's and Petri nets bring control structure to the foreground As a result there have been several efforts to combine

these approaches in an attempt to retain the best of both [11,12].

Ultimately, of course, we do not wish merely to model the systems we are interested in studying, but would like to be able to perform some sort of analysis. We would like to be able to determine whether or not a particular combination of communicating processes exhibits certain properties or satisfies a given specification. Manipulation of our models in a manner which corresponds to the real world is a prerequisite to this sort of analytical ability. In addition comparison of processes against service specifications and other processes must be possible. Here a calculus or equational theory of processes would provide a appropriate basis for analysis.

Given a model we must be aware of its limitations, that is we must know what details of the physical reality that it faithfully retains. In the physical sciences a model of theory can be evaluated by comparing the predicted values of physical quantities with empirical measurements. In the case of process behaviour this approach is inappropriate and evaluation becomes, in some sense, a subjective term. The best we can do is ensure that the results that we obtain are reasonable in that they correspond with our intuitive expectations.

1.3 Related Work

There have been two main approaches to process analysis and verification. The most commonly encountered technique is that of reachability analysis. This approach involves exhaustively examining the possible interactions of processes which form a system. It is equivalent to generating the portion of the combined or global statespace which is reachable. This method is generally associated with finite state

machine type models. Assertion proving, borrowed from program verification methods, has been used in the analysis of systems modeled in high level languages

In terms of automated analysis, the reachability analysis approach is the easiest to deal with. Indeed there are several existing computer aided systems that rely on variations of this approach [13, 14, 15]. In fact the work presented here is similarly based. The main problem encountered is state space explosion which occurs as the complexity of the modeled processes increases.

Assertion proving techniques have proved more cumbersome to mechanize, and there exists at present no truly automatic systems that employ this approach. The AFFIRM system [16, 17] allows assertions and theorems to be proved in an interactive manner. It is the user that develops the proofs with the system carrying out its mechanics.

There is very little in the literature on the subject of automatic process synthesis. An interactive design method for communication protocols has been proposed by Zafiropulo et al [18]. The method developed by P. Merlin and G. Bochmann [19] seems to be similar in capability to the one used here, but is based on a set theoretical approach and has not, as yet been automated.

1.4 Document Outline

The next Chapter introduces the discrete communication process model and provides an intuitive justification for it. This is followed, in Chapter 3, by an

introduction to the algebraic structure and process operators of R. Johnston. For a more complete treatment of this subject the reader is referred to [1].

Chapter 4 deals with the computational tools developed to automate the analytical procedures. The theoretical basis for the process synthesis procedure is established in Chapter 5. Several problems and unanswered questions are also discussed here. Chapter 6 describes some of the qualification and restrictions of the current implementation of the synthesis procedures.

Chapter 7 is the meat of our attempt to validate the DCP model and contains a variety of problems as well as the results obtained with the aid of the computational tools. Included are two examples of problems involving process synthesis and the results obtained from the automatic synthesis programs.

Chapter 2 Informal Overview of the DCP Model

Returning to our intuitive concept of process, we note that from our external perspective the only way we have at our disposal for distinguishing between processes is through observation of their behaviour. If two processes exhibit the same external behaviour we have no reason to believe that they are different. This behaviour consists of a pattern of information exchange between the process and its environment. It would seem reasonable therefore to retain the exchange of discrete messages as a primitive concept within our model. As a process evolves the set of messages or signals that it is willing to exchange changes allowing us to gauge its progress.

2.1 Primitive Concepts

Conceptually, then, a process offers at a given point in time to exchange any one of a fixed set of messages with its environment. When a message is exchanged, the process is thought to make a transition to a new state where a different set of communication possibilities may be offered. The new 'state' is considered to be a distinct process as it may exhibit a different pattern of behaviour than its

predecessor.

The events of interest,thus, are the exchanges of discrete messages which are assumed to occur instantaneously. These communication events are further assumed to be synchronized, that is, both the source and the sink of a message wait until a communication occurs, and then progress simultaneously. This view is in accordance with current trends in computer language design [20] as well as with existing mathematical formulations of the semantics of multi-process networks [21]. It contrasts, however, with the global shared memory view of inter-process communication.

With this perspective in mind, we can represent a process p as a set of communication event, subsequent process pairs for example,

$$p = \{< e_1, p_1 > \quad , < e_t, p_t >\}$$

Note that this is an inherently recursive definition, processes are defined in terms of processes. Note also that there is little or no distinction between a state and a process. We may represent a process graphically as shown in Figure 2.1, with a node and directed arcs representing the communication events, leading to other nodes.

For a process to be finite state or finite dimensional, the set of processes reachable from it through any possible sequence of communications must be finite. In our graphical representation this corresponds to a graph with a finite number of nodes.

At any given time a process may offer to emit as well as to absorb messages. We

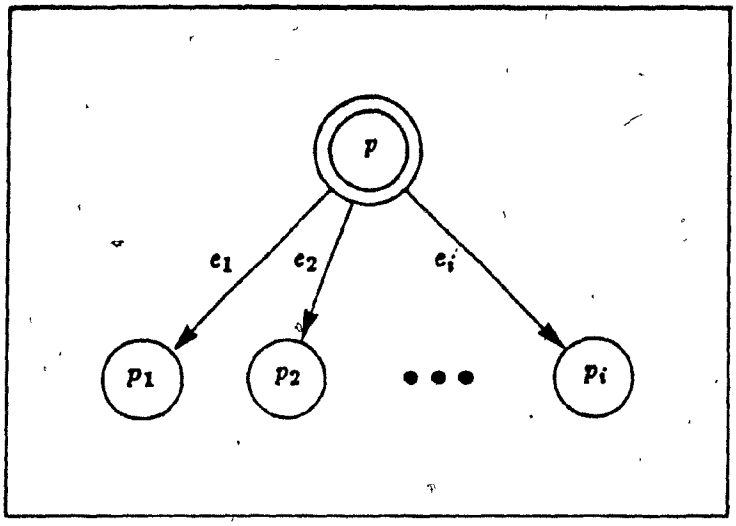

Fig. 2.1 Graphical representation of a process

indicate the difference by appending an exclamation mark to an emitted message and a question mark to one that is absorbed. For instance the process

$$p = \{< q?, p_1 >, < r!, p_2 >\}$$

offers to absorb the message q and resume as process p_1, and to emit message r and resume as process p_2.

Figure 2.2 is a graphical representation of a simple buffer or reliable communications channel. The process initially offers to absorb a message m and then offers to emit a copy of it, thereafter returning to its original state. This process can also be represented as:

$$\begin{array}{l} (\quad p = \{< m?, p_1 >\}, \\ \quad p_1 = \{< \hat{m}!, p >\} \quad) \end{array}$$

Note the recursive nature of this representation when viewed as equations in p and p_1.

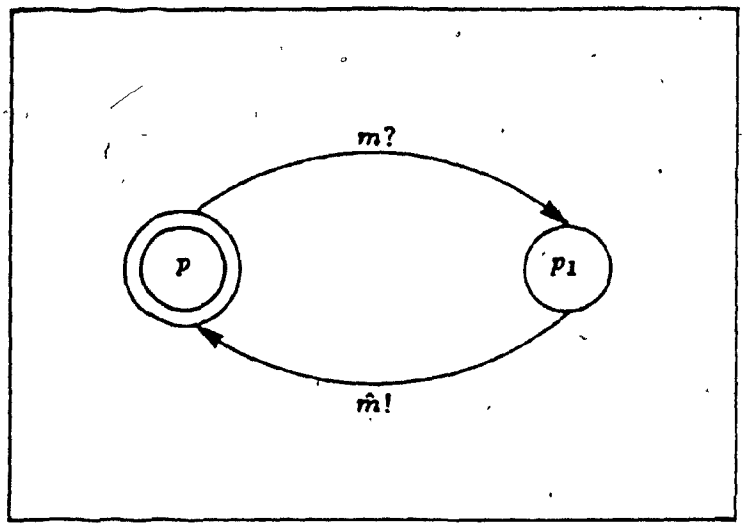

Fig. 2.2 Reliable communications channel

2.2 Non-determinism

A process offers its communication possibilities to its environment: our understanding is that the environment selects which message is exchanged. Now, within the terms of the model it is possible to have more than one arc labeled with the same event emanating from a particular node. The effect of the environment selecting such a message is interpreted as the process non-deterministically selecting a successor amongst the processes upon which the similarly labeled arcs terminate. This construct models non-deterministic behaviour.

Non-determinism is useful when we come to model our ignorance about such things as customers with random requests, or unreliable communication channels. In addition, when two processes are connected or composed, internal exchanges which occur between them are not externally visible and may lead to different behaviours. We can model this by non-determinism as well. In general, deterministic

behaviour is considered a desirable property in system design, and is representative of the types of properties that we may want to automatically verify.

An example of a non-deterministic process is depicted in Figure 2.3. This example is a variation of the simple communication channel of Figure 2.2 and represents an unreliable channel which occasionally loses a message altogether. The corresponding process expression is:

$$\begin{array}{l} (\quad p = \{< m?, p_1 >, < m?, p >\}, \\ \quad\; p_1 = \{< \hat{m}!, p >\} \quad) \end{array}$$

Note the non-deterministic transition that may occur upon reception of the message m.

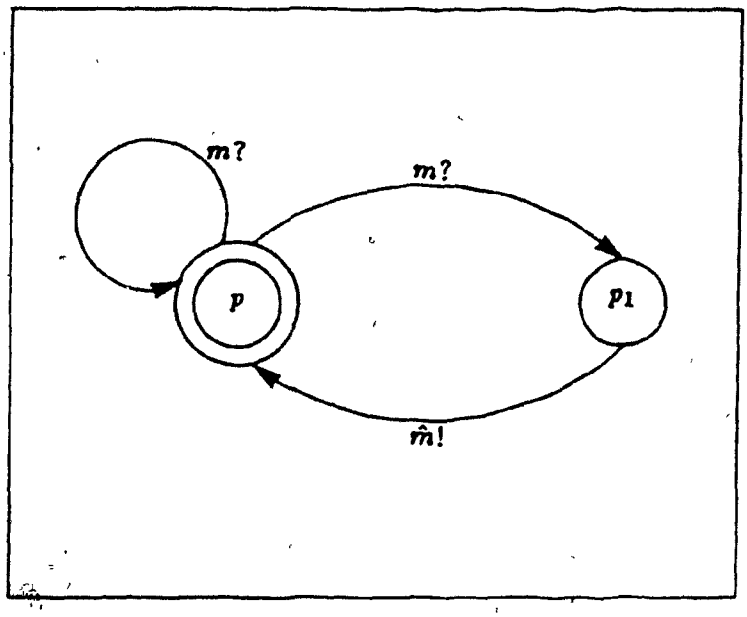

Fig. 2.3 Unreliable communications channel

2.3 The Process Ordering Relation

The notion of relatedness of processes is a fundamental one. It is important

to be able to determine if two processes are equal or in some way equivalent. For instance, we want to know when a complex system can be implemented as, or is equal to, the composition of several simpler processes. In a more general context it is of interest to know if a process is consistent with, although not strictly equal to, another. When this is the case, one process is said to approximate the other. The concept of behavioural approximation, or consistency, is a natural one and is analogous to that of coverability encountered in the study of finite state machines [22]. It is also closely associated with the idea of simulation, where a dynamic system simulates another if it can mimic all of its actions. These natural and intuitive notions lead to mathematical relations that are reflexive and transitive.

In the context of our model, a process p approximates a process q if, when represented graphically, q can be 'overlaid' on p with arcs labeled with the same communication events lying on top of one another. Similarly we can represent processes as infinite tree structures with the root node representing the initial state and labeled branches indicating communication events leading to successor states. Here p approximates q if q is a subtree of p. In either case we say $p \leq q$ or p contains q.

The thrust of this formulation is clear; a process p approximates another process q if it offers a all of the behavioural possibilities of q, and perhaps more. In other words every possible sequence of communication events that can be offered by q can also be offered by p. Clearly this relation is reflexive and transitive and as a result will define an equivalence relation amongst processes.

$$p \leq q \bigwedge q \leq p \Rightarrow p \equiv q$$

This equivalence, indicated by $\equiv$, will partition the set of processes into equivalence classes. Thus two processes that have different representations will be considered equivalent if they belong to the same equivalence class. The equivalence relation is intuitively pleasing; if process p offers all the behavioural possibilities of process q, and the converse is also true, we would certainly want to consider p as being equivalent to q.

Patently the largest process in this ordering is the one that offers to exchange no messages with the environment

$$p' = \{\} = \emptyset$$

This special process will sometimes be referred to by the symbol $\bullet$.

As an example, let us consider the two simple communication channels modeled previously:

$$\begin{array}{l} (\ p = \{< m?, p_1 >\}, \\ \ \ p_1 = \{< \hat{m}!, p >\}\ ) \end{array}$$

$$\begin{array}{l} (\ q = \{< m?, q_1 >, < m^?, q >\}, \\ \ \ q_1 = \{< \hat{m}!, q >\}\ ) \end{array}$$

All of the behavioural possibilities of p are clearly contained in those of q so we have $q \leq p$. On the other hand q can offer to continuously accept messages without emitting copies, a behaviour that p cannot emulate. Hence q is not equivalent to p.

2.4 Process Operators

So far we have dealt exclusively with an abstraction that allows us to model dynamic system components in terms of their communication behaviour. In general,

however, components are physically interconnected to implement a specific system. From a modeling point of view, there should be no real qualitative difference between such a system and its components. That is, we should expect to be able to model a composite system as a DCP if the same is true of its components. In this light it is possible to envision a process operator, or function, whose application to two DCP's will yield another DCP that is a faithful abstraction of the system that would result from the physical interconnection of the modeled entities. In the next chapter such operators are defined after we have placed the model on sound mathematical footings.

Chapter 3 Mathematical Formulation

In a strict mathematical sense we cannot treat processes merely as sets of communication event; subsequent process pairs. To see why this is so consider the set P_E of all processes that exchange messages from a finite set E. If any $p \in P_E$ is a set of ordered event-process pairs, we have

$$p = \{< c_1, p_1 >, \ldots, < c_n, p_n >\}$$

and clearly p is a subset of the Cartesian product $E \times P_E$. So,

$$p \in \mathcal{P}(E \times P_E)$$

where $\mathcal{P}$ is the power set operator. Since this is so for any $p \in P_E$ we have

$$P_E \subseteq \mathcal{P}(E \times P_E)$$

Conversely any element of $\mathcal{P}(E \times P_E)$ is a set of event-process pairs and, intuitively, should be a process; that is,

$$\mathcal{P}(E \times P_E) \subseteq P_E$$

From which we would deduce, in defiance of Cantor's Theorem, that

$$P_E = \mathcal{P}(E \times P_E)$$

This is an equation that cannot be satisfied for reasons of cardinality, by any P_E.

What follows is a brief outline of the construction of an appropriate mathematical space whose elements correspond to the intuitive notion of process we have already discussed. This is a construction that was proposed by R Johnston. Readers interested in a detailed account are referred to [1].

3.1 The Process Space P_E

An approach to resolving the problem of cardinality raised above involves reducing the number of elements of $\mathcal{P}(E \times P_E)$ by redefining the powerset operator to not include all of the possible subsets. A mathematically succinct way of accomplishing this is to partition the set $\mathcal{P}(E \times P_E)$ into equivalence classes thus obtaining a quotient set. Such equivalence classes can be induced by a partial ordering. The reader is referred to Manna [23] for the definitions arising in the theory of partial orders Plotkin [3], Smyth [2], as well as Johnston [1] employ this approach, although the ordering used by Plotkin seems unnatural.

The elements of the process space P_E can be thought of as tree structures The space contains both finite length trees and infinite length trees that are more or less analogous to rational and irrational numbers in the set of real numbers. The construction of P_E involves a sequence of approximating spaces whose limit, in some sense, is the desired space. Each of the finite intermediate spaces $P_E^{(i)}$ is a partially ordered set whose elements can be thought of as trees of at most length i The limiting space P_E is also a partial order that contains a bottom element $\perp$, and has a top element $\bullet$, representing the null process. It is complete in the sense that

any directed set D of elements belonging to P_L possesses a least upper bound also contained within P_E. Here a directed set means a set in which any two elements have an upper bound in the same set. The least upper bound of a directed set D is indicated by

$$\bigsqcup D$$

The bottom element $\perp$, represents the completely undefined or unconstrained process. Conversely, the null process $\bullet$, can be interpreted as the completely defined or constrained process with no degrees of freedom.

There are two fundamental operations or functions that are defined on P_E. The first is, in fact, the class of functions $e_- : P_E \rightarrow P_E$ where $e \in E$, which prefixes a process with the event e. In set notation this corresponds to

$$e; p = \{< e, p >\}$$

These functions can be shown to be monotonic and continuous, that is they preserve ordering and lower upper bounds. If D is a directed set of processes belonging to P_E then

$$e; \bigsqcup D = \bigsqcup e; D$$

where $e; D = \{e; d \mid d \in D\}$. The second operation $+ : P_E \times P_E \rightarrow P_E$ is a binary function corresponding to the aggregation of behaviours. It can be shown to be monotonic and continuous with respect to both arguments. In the set notation it corresponds to set union,

$$p + q = p \bigcup q$$

returning a process that has all the communication possibilities of both p and q.

There are several basic properties involving these operators that arise from the partial order structure of P_L. These properties will be stated without proof. Readers wishing to pursue proofs are again referred to [1]. The following properties hold in P_L:

(i) $e_1 . q_1 \leq e_2 : q_2 \Leftrightarrow ((e_1 = e_2) \wedge (q_1 \leq q_2))$

(ii) $x_1 + \ldots + x_N \leq e : q \Leftrightarrow x_i \leq e : q$ for some i

(iii) The greater lower bound (glb) of any set of processes exists. In particular for $\{p, q\}$, where $p, q \in P_L$, it is given by the process $p + q$.

(iv) $\forall p \in P_L , p = \sum_i e_i : q_i$ where the sum is taken over all $e, q \geq p$

A process p belonging to P_L is called finitely branching if it can be expressed as a finite sum of elements of the form $e . q \in P_L$. That is

$$p = \sum_{i=1}^{N} e_i . q_i$$

for some finite N. Suppose that $e_i . q_i \leq e_j . q_j$ for some i and j, then by definition $e_i : q_i$ is the glb of $e_i : q_i$ and $e_j : q_j$. So from property (iii), $e_i . q_i = e_i . q_i + e_j . q_j$ and we can replace $e_i . q_i + e_j . q_j$ in the sum simply with $e_i . q_i$. As a result any finitely branching process can be expressed as above with the stipulation that

$$e_i . q_i \not\leq e_j . q_j \quad \forall i \neq j$$

It follows from properties (i), (ii), and (iii) that this is, in fact, a unique expression. In other words there is a canonical relationship between finitely branching processes and sets of event-process pairs. So a finitely branching processes has a canonical form or expression associated with it.

3.2 Processes as Fixed Points

Recall that in our intuitive development we represented processes with a finite number of successors as sets of process definitions. These types of processes are called finite dimensional. We can formalize this notion using the mathematical operators ϵ and $+$. Consider for instance the unreliable communications channel modeled previously. Using algebraic notation we have

$$\begin{cases} p_1 = m^0 p_2 + m^{0'} p_1, \\ p_2 = m^1 p_1 \end{cases}$$

Formally this can be interpreted as a set of two equations involving the process variables p_1 and p_2. Rewriting the equations in matrix format we would have

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} m^{0'} & m^0 \\ m^1 & \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

or,

$$P = F_1 P$$

where F is a two dimensional function $F : P_I \times P_I \to P_I \times P_I$. Now F is composed of the primitive operations ϵ and $+$ which are monotonic and continuous. As a result F itself is a monotonic and continuous function on P_I^2. The complete partial ordering of P_I is extended to P_I^n in the following manner.

$$(p_1, \ldots, p_n) \leq (q_1, \ldots, q_n) \Leftrightarrow p_i \leq q_i \forall i = 1, n$$

Consider the sequence of processes

$$\{\Omega, F[\Omega], F^2[\Omega], \ldots\}$$

where $F^i[\Omega] = F[F^{i-1}[\Omega]]$, and $\Omega = (\perp, \perp)$. By definition $\Omega \leq F[\Omega]$ and since F is monotonic we can conclude that $F^i[\Omega] \leq F^{i-1}[\Omega]$. Hence the sequence $\{F^i[\Omega]\}$ is a

chain and constitutes a directed set in P_L^2. As P_L^2 is a complete partial order there exists a least upper bound $f = \bigsqcup\{F^i[\Omega]\} \in P_L^2$. The element f is a fixed point of F since by continuity

$$F[f] = F[\bigsqcup\{F^i[\Omega]\}] = \bigsqcup\{F^{i+1}[\Omega]\} = \bigsqcup\{F^i[\Omega]\} = f$$

Invoking fixed point theory [23], f is in fact the least fixed point of the operator F.

Consequently, finite state (or finite dimensional) processes can be uniquely specified as fixed points of continuous multi-dimensional operators. In practice this means that physical systems representable as finite state machines can be described as finite sets of process equations.

3.3 Algorithmically Determining Process Ordering

A finite process is one that terminates after a finite sequence of communication events and is representable as a tree of finite length. For such processes, relatedness is easy to determine algorithmically as follows.

$$p = \sum_{i=1}^{N} c_i.p_i$$

$$q = \sum_{i=1}^{M} c_i.q_i$$

$$p \leq q \;\equiv\; \forall\, c_i.q_i \in q \;\exists\, c_j.p_j \in p \mid ((c_i = c_j) \bigwedge (p_j \leq q_i))$$

where $c.p \in q$ indicates that $< c.p >$ is a member of the canonical set representation of the process q. This recursive definition of $\leq$ is grounded since p and q are finite processes. In the general case where p and q are defined as the first

component of the least fixed points of some operators F and G

$$P = F[P]$$
$$Q = G[Q]$$

this recursion is not finite in nature

Let $\Psi_i : P_1^n \to P_i$ be a projection function mapping $(p_1, \ldots, p_n)$ onto its i^{th} component p_i. The function Ψ_i is obviously monotonic and continuous. Corresponding to the sequence $\{F^n[\Omega]\}$ in P_1^n is the sequence $\{\Psi_1(F^n[\Omega])\}$, which forms a directed set in P_1. The same is true of the sequence $\{\Psi_1(G^n[\Omega])\}$. Assume that $\Psi_1(F^m[\Omega]) \leq \Psi_1(G^n[\Omega])$ for all n. Then since $\Psi_1(G^m[\Omega]) \leq \bigsqcup\{\Psi_1(G^n[\Omega])\}$ $\Psi_1(\bigsqcup\{G^n[\Omega]\})$ it follows that $\Psi_1(\{F^m[\Omega]\}) \leq \Psi_1(\bigsqcup\{G^n[\Omega]\})$ for all n and hence that $\Psi_1(\bigsqcup\{G^n[\Omega]\})$ is an upper bound of the sequence $\{\Psi_1(F^n[\Omega])\}$. So we may conclude that

$$p = \bigsqcup\{\Psi_1(F^n[\Omega])\} \leq \bigsqcup\{\Psi_1(G^n[\Omega])\} = q$$

Thus in order to determine that $p \leq q$ it is only necessary to show that $\Psi_1(F^n[\Omega]) \leq \Psi_1(G^n[\Omega])$ holds for all n. Proceeding by induction we note that $\Psi_1(F^0[\Omega]) = \Omega \leq \Omega = \Psi_1(G^0[\Omega])$ and attempt to prove the induction step

$$\Psi_1(F^{n+1}[\Omega]) \leq \Psi_1(G^{n+1}[\Omega]) = \Psi_1(F^{n}[\Omega]) \leq \Psi_1(G^{n}[\Omega])$$

An example should elucidate matters. Once again let us consider the two simple communication channels modeled previously. In algebraic notation they can be

represented as

$$\begin{pmatrix} p_1 = m?; p_2, \\ p_2 = \hat{m}!; p_1 \end{pmatrix}$$

$$\begin{pmatrix} q_1 = m?, q_2 + m?; q_1, \\ q_2 = \hat{m}!; q_1 \end{pmatrix}$$

or

$$P = F[P]$$
$$Q = G[Q]$$

Subsequently p_j^i will be used to represent $\Psi_j(F^i[\Omega])$ and q_j^i to represent $\Psi_j(G^i[\Omega])$ The following steps are performed to prove $p_1 \leq q_1$

$$\begin{aligned} q_1^i \leq p_1^i &\Leftarrow \hat{m}?, q_2^{i-1} + \hat{m}?, q_1^{i-1} \leq m?, p_2^{i-1} \\ &\Leftarrow q_2^{i-1} \leq p_2^{i-1} \bigvee q_1^{i-1} \leq p_2^{i-1} \end{aligned}$$

$$\begin{aligned} q_2^i \leq p_2^i &\Leftarrow \hat{m}! \, p_1^{i-1} \leq m! \, q_1^{i-1} \\ &\Leftarrow q_1^{i-1} \leq p_1^{i-1} \end{aligned}$$

So we have

$$q_1^i \leq p_1^i \bigwedge q_2^i \leq p_2^i \Leftarrow q_2^{i-1} \leq p_2^{i-1} \bigwedge q_1^{i-1} \leq p_1^{i-1}$$

Clearly for $i = 0$ the right hand side is trivially true, and so $q_1^i \leq p_1^i$ is true for all i and hence $q_1 \leq p_1$

The statement $p \leq q$ is representative of the class of admissible statements A statement $\pi(p)$ is said to be admissible if and only if $\pi(\Gamma^m[\Omega])$ being true for all m logically implies that $\pi(\bigsqcup \Gamma^m[\Omega])$ is also true It can be shown with relative ease that every statement of the form

$$\pi(p) \equiv \bigwedge_{i=1}^{N} (\alpha_i[p] \leq \beta_i[q])$$

where α_i and β_i are continuous functions, is admissible. The validity of an admissible statement $\pi(p)$ is logically implied from the validity of the statement for all finite approximations to p

3.4 Process Operators

Given the mathematical space P_L it is possible to define and analyze process operators of functions that correspond to intuitive notions of process interaction The functions will be required to be monotonic and continuous in order to guarantee the existence of fixed points In the case of finite-dimensional processes, these operators can be interpreted in two fundamental, yet equivalent ways One interpretation is simply that of a n-dimensional function mapping processes into processes A second and more succinct for our purposes involves viewing these operators as functionals that operate on the functions of which processes are the least fixed points This latter view is more tractable due to the finite nature of the functions defining processes, as contrasted with the infinite nature of the processes themselves

Perhaps the most intuitive process operator is process interconnection When two processes are interconnected or composed, it is possible for them to communicate with each other through the exchange of messages The goal of a mathematical composition operator is to predict the externally observable behaviour of composite processes The interconnection operator, denoted $\times$, is defined in two steps. The first is the definition of the total composition operator $\square$ This is a binary function that returns a process retaining the external communication capabilities of each of the arguments but with internal exchanges flagged with the trace event $-$.

Before proceeding with a formal definition of the total composition operator, it will be advantageous to clarify some of the notational conventions used Suppose the process $p \in P_{E_1}$ and $p \in P_{E_2}$ are to be composed. In order to perform the composition the events which p and q can exchange must be known. The sets E_1 and E_2 may contain input events and output events, specified by ? and ! respectively, as well as possibly the trace event $-$. Let E_n' be the set of events E_n with the trace event removed

$$E_n' = E_n - \{-\}$$

The events that a process $p \in P_{E_1}$ can possibly exchange with $q \in P_E$ are

$$(E_1' - E_2') = E_1' \bigcap E_2'$$

where $\bar{E} = \{\bar{e} \mid e \in E\}$ and the function

$$\bar{e} = \begin{cases} e' & \text{for } e = e? \\ e? & \text{for } e = e' \\ - & \text{for } e = - \end{cases}$$

simply changes an input event into an output event and vice versa. Similarly the events that q can exchange with p are

$$(E_2' - E_1') = E_2' \bigcap E_1' = (E_1' - E_2')$$

The set of all communication events $E_1 \cdot E_2$ that may possibly be exchanged between p and q is given by $(E_1' - E_2') \bigcup (E_2' - E_1')$ The set of communications available to the external environment is $(E_1' \bigcup E_2') - (E_1 \cdot E_2)$ The total event set of the composite is

$$E_1 \otimes E_2 = ((E_1' \bigcup E_2') - (E_1 \cdot E_2)) \quad \bigcup \quad \{-\}$$

Now the processes $p \in P_{E_1}$ and $q \in P_{E_2}$ can be expressed as

$$p = \sum_i e_i ; p_i + \sum_j a_j ; p_j$$

$$q = \sum_m c_m ; q_m + \sum_n a_n ; q_n$$

for some $e_j, c_k \in E_1 \otimes E_2$, and $a_j, b_n \in E_1 \oplus E_2$. The total composition operator is defined recursively as

$$\begin{aligned} p \Box q = & \sum_i e_i ; (p_i \Box q) \\ & + \sum_m c_m ; (p \Box q_m) \\ & + \sum_{\substack{j,n \\ a_j \; a_n}} - ; (p_j \Box q_n) \end{aligned}$$

Here the terms $e_i ; (p_i \Box q)$ represent the possibility of the event e_i being exchanged between p and the external environment with the composite resuming as $p_i \Box q$. The terms $c_m ; (p \Box q_m)$ denote the similar case for q. The terms $- ; p_j \Box q_n$ indicate that an internal exchange has transpired between p and q with the composite resuming as $p_j \Box q_n$. The total composite corresponds to an exhaustive generation of the part of the combined state space of p and q in which the composite can exist.

Due to the infinite nature of p and q, the recursive definition of $\Box$ appears to be ill-defined. Such definitions are, in fact, well defined if we consider a recursively defined function as being the least fixed point of a functional mapping functions into functions. This approach is much the same as the one we used for getting a handle on finite dimensional processes, and is a powerful method for analysing such functions as static mathematical objects. An alternative approach, is to regard the recursive definition as an algorithm for the generation of a set of process equations from those of the arguments. The net effect is that of a function that maps the process operators, which define p and q as their least fixed points, onto a similar operator that has the process $p \Box q$ as its least fixed point. Since the process equations are finite in their extent, the algorithm is well defined.

The total composite operator can be shown to be commutative and associative

under the assumption that processes cannot exchange the same message with more than one other process. These are properties that, intuitively, we would expect to hold for composition of processes.

As an example consider the case of a librarian who works at an unassuming library that has two books to lend to its two members. The books constitute a two volume set, and to be fair the library has instituted a rule to the effect that a member may borrow at most one book at a time. The members are capricious and on any given day it is impossible to tell which volume they might like to read, but initially they both want volume one. Member i is modeled as the non-deterministic process.

$$
\begin{aligned}
m1_i &= v1_i?.\hat{m}1_i \\
m2_i &= v2_i?.\hat{m}2_i \\
\hat{m}1_i &= r1_i!.m1_i + r1_i!.m2_i \\
\hat{m}2_i &= r2_i!.m1_i + r2_i!.m2_i
\end{aligned}
$$

where $v1_i?, v2_i?, r1_i!, r2_i!$ represent requests for volume one and two, and the releases of volume one and two respectively. Figure 4 depicts the finite state diagram for member i. Note that member i non-deterministically requests volume one or two upon returning either.

The librarian initially offers to lend either volume to either member, but after having lent one volume, offers only the other volume to the other member. The

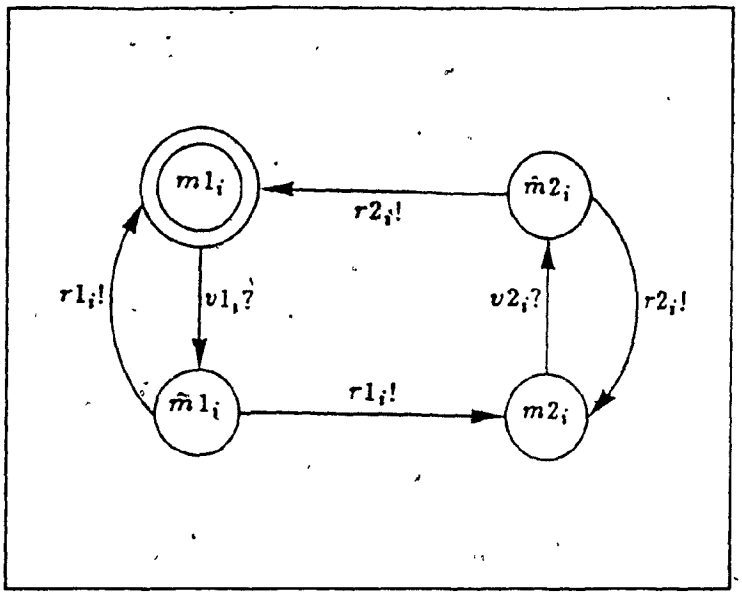

Fig. 3.4 Transition diagram for member i

process expression for the librarian is

$$
\begin{aligned}
L_{00} &= v1_1!; L_{10} + v2_1!; L_{20} + v1_2!; L_{01} + v2_2!; L_{02} \\
L_{01} &= v2_1!; L_{21} + r1_2?; L_{00} \\
L_{02} &= v1_1!; L_{12} + r2_2?; L_{00} \\
L_{10} &= v2_2!; L_{12} + r1_1?; L_{00} \\
L_{12} &= r2_2?; L_{10} + r1_1?; L_{02} \\
L_{20} &= v1_2!; L_{21} + r2_1?; L_{00} \\
L_{21} &= r2_1?; L_{01} + r1_2?; L_{20}
\end{aligned}
$$

where $v1_i!, v2_i!$ represent the lending of volumes one and two to member i and $r1_i?, r2_i?$ represent the return of volumes one and two from member i.

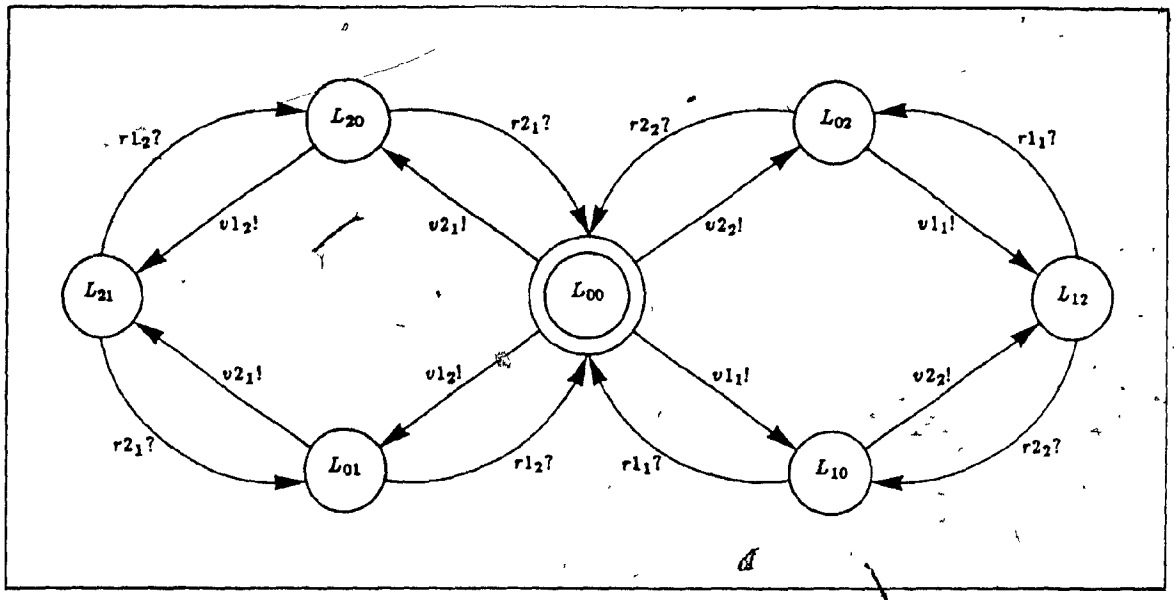

Fig. 3.5 Transition diagram for librarian

The total composite of member 1 with the librarian is:

$$
\begin{aligned}
m1_1 \,\square\, L_{00} &= -; \hat{m}1_1 \,\square\, L_{10} + v1_2!; m1_1 \,\square\, L_{01} + v2_2!; m1_1 \,\square\, L_{02} \\
\hat{m}1_1 \,\square\, L_{10} &= -; m1_1 \,\square\, L_{00} + -; m2_1 \,\square\, L_{00} + v2_2!; \hat{m}1_1 \,\square\, L_{12} \\
m1_1 \,\square\, L_{01} &= r1_2?; m1_1 \,\square\, L_{00} \\
m1_1 \,\square\, L_{02} &= -; \hat{m}1_1 \,\square\, L_{12} + r2_2?; m1_1 \,\square\, L_{00} \\
m2_1 \,\square\, L_{00} &= -; \hat{m}2_1 \,\square\, L_{20} + v1_2!; m2_1 \,\square\, L_{01} + v2_2!; m2_1 \,\square\, L_{02} \\
\hat{m}1_1 \,\square\, L_{12} &= -; m1_1 \,\square\, L_{02} + -; m2_1 \,\square\, L_{02} + r2_2?; \hat{m}1_1 \,\square\, L_{10} \\
\hat{m}2_1 \,\square\, L_{20} &= -; m1_1 \,\square\, L_{00} + -; m2_1 \,\square\, L_{00} + v1_2!; \hat{m}2_1 \,\square\, L_{21} \\
m2_1 \,\square\, L_{01} &= -; \hat{m}2_1 \,\square\, L_{21} + r1_2?; m2_1 \,\square\, L_{00} \\
m2_1 \,\square\, L_{02} &= r2_2?; m2_1 \,\square\, L_{00} \\
\hat{m}2_1 \,\square\, L_{21} &= -; m1_1 \,\square\, L_{01} + -; m2_1 \,\square\, L_{01} + r1_2?; \hat{m}2_1 \,\square\, L_{20}
\end{aligned}
$$

The total composite, $p \,\square\, q$ retains the vestiges of internal communication events, and hence does not describe the composite purely in terms of its external behaviour. Internal transitions that lead to processes with equivalent external behaviour possibilities should be suppressed in such a description. On the other hand, trace events that lead to inequivalent behaviours are detectable externally and should be

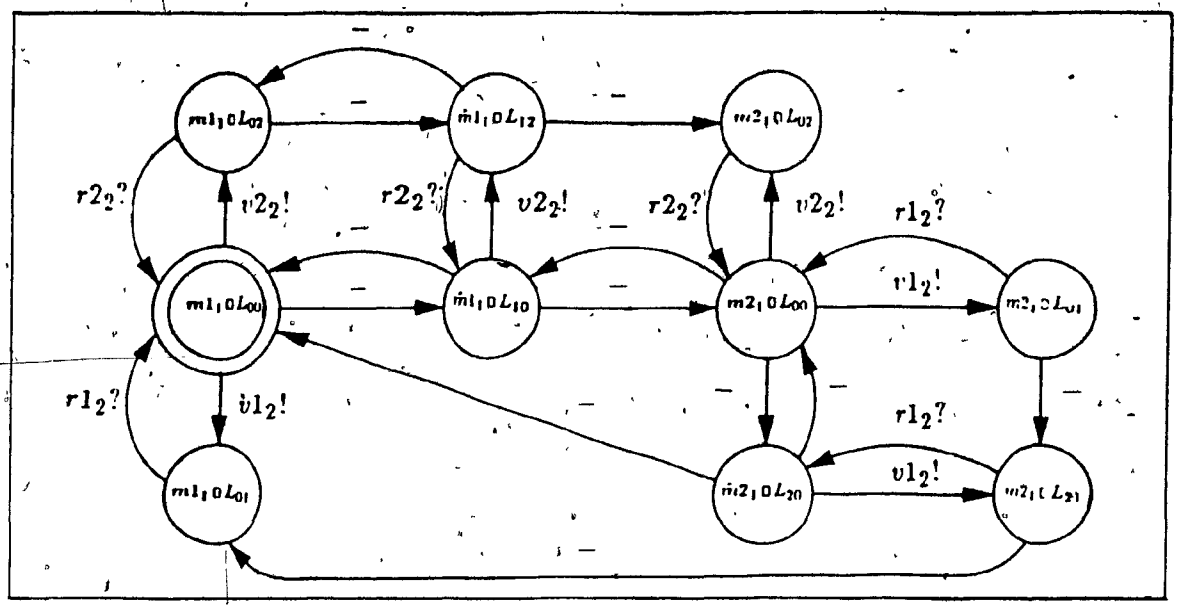

Fig. 3.3 Transition diagram for member-librarian composite

retained

The reachability function $R : P_L \rightarrow P_{L-\{-\}}$ returns a process which is the sum of the externally visible behaviours of its argument process. The trace transitions are merely replaced with all the externally visible transitions that are reachable through them. Formally,

$$R(p) = \sum_{e.q \in .p} \tau(e.q)$$

where

$$\tau(e.q) = \begin{cases} e.R(q) & \text{for } e \neq - \\ R(q) & \text{for } e = - \end{cases}$$

Recall that by $e.q \in .p$ we mean that $< e.q >$ is a member of the canonical set representation of p.

The reachability operation applied to the total composite of the librarian and

member 1 is, after simplification.

$$
\begin{aligned}
R(m1_1 \,\square\, L_{00}) &= v1_2!; R(m2_1 \,\square\, L_{21}) + v2_2!; R(m2_1 \,\square\, L_{02}) \\
R(m2_1 \,\square\, L_{21}) &= r1_2?; R(m1_1 \,\square\, L_{00}) \\
R(m2_1 \,\square\, L_{02}) &= r2_2?; R(m1_1 \,\square\, L_{00})
\end{aligned}
$$

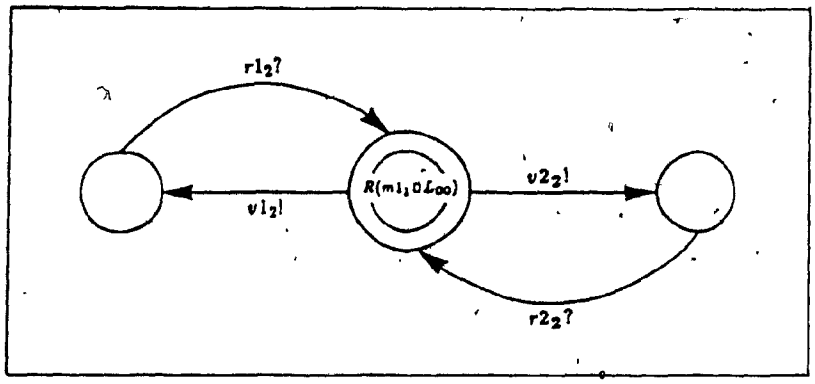

Fig. 3.7 Reachable behaviour of the member-librarian composite

Externally, an internal trace event is detectable if it changes the composite's potential interactions. Formally, for process p containing a trace transition to process p_i, this corresponds to $R(p) \not\equiv R(p_i)$. The interconnection operator $*$ is similar to the R operator, except that it flags variations in the potential external interactions by retaining the trace transition. It is defined as follows:

$$p * q = \mathcal{V}(p \,\square\, q)$$

where

$$\mathcal{V}(p) = \sum_{e;q.\in.p} \gamma(e.q)$$

and

$$\gamma(e,q) = \begin{cases} e; \mathcal{V}(q) & \text{for } e \neq - \\ \mathcal{V}(q) & \text{for } e = - \wedge R(q) \equiv R(p) \\ -; \mathcal{V}(q) & \text{otherwise} \end{cases}$$

The interconnection operator $*$ can be shown to be commutative and associative under the same assumptions as for the total composite. A composite $p * q$ that does not contain any trace events is said to be stable. It is easily seen that if a composite is stable then $p * q \equiv R(p \Box q)$ Stability is an important concept, and a desirable property for a composite system to exhibit. A composite process that is stable in effect possesses a static description of its visible behaviour In such an instance internal transitions are not detectable and hence are irrelevant to the external description. Thus the internal communications of a stable system that is composed of the interconnection of several component processes are transparent An inspection of the interconnection of the librarian and member 1

$$
\begin{aligned}
m1_1 * L_{00} &= v1_2!.\, \hat{m}2_1 * L_{21} + v2_2!.\, m2_1 * L_{02} \\
\hat{m}2_1 * L_{21} &= r1_2?.\, m1_1^{\sim} * L_{00} \\
m2_1 * L_{02} &= r2_2?;\, m1_1 * L_{00}
\end{aligned}
$$

reveals that $m1_1 * L_{00} \equiv R(m1_1 \Box L_{00})$ and hence the composite is stable.

Another important process operator is complementation $co \quad P_l \; - \; P_L$ This function simply returns its argument with all input events switched to output events and vice versa.

$$co(p) = \sum_{e.q.\in.p} e^{\sim}; co(q)$$

The co function is notable in that, for any process p, $co(p)$ is a process that when interconnected with p results in the null process; $p * co(p) \equiv \bullet$.

A process p is non-deterministic if transitions to unrelated processes are possible upon the exchange of the same message. A process is non-deterministic if it can be expressed in the form

$$p = e.\, p_1 + e; p_2 + \ldots$$

and p_1 is not related to p_2. Although non-determinism is essential for modeling ignorance about possible behavioral sequences it is, for some types of analysis, advantageous to replace non-deterministic transitions with deterministic ones while retaining all of the behavioral possibilities. The *det* function accomplishes this. For a process $p \in P_L$, the *det* function is defined as

$$det(p) = \sum_{e \in L} \mu(e)$$

where

$$\mu(e) = \begin{cases} e : det(\sum_{e \cdot q_i \in p} q_i) & \text{for } p \leq e. \bullet \\ \bullet & \text{otherwise} \end{cases}$$

The *det* operator applied to the unreliable communications channel introduced before would yield

$$det(q_1) = m'' : det(q_1 + q_2)$$
$$det(q_1 + q_2) = m'' : det(q_1 + q_2) + m' : det(q_1)$$

It can be shown that $det(p)$ is the largest deterministic process that contains p.

As an example of the application of the process operators we once again turn to our erstwhile librarian and the problem of evaluating the service provided by the library. From each of the members point of view the library should be able to satisfy all of his requests for books (perhaps with some delay) despite the fact that the other member is also using the library. Thus for any pattern of requests and surrenders that member 2 might exhibit the interconnection of the librarian and member 1 should offer the complementary behaviour. This is captured by the statement

$$m_1 1_1 * L_{00} \leq co(det(m 1_2))$$

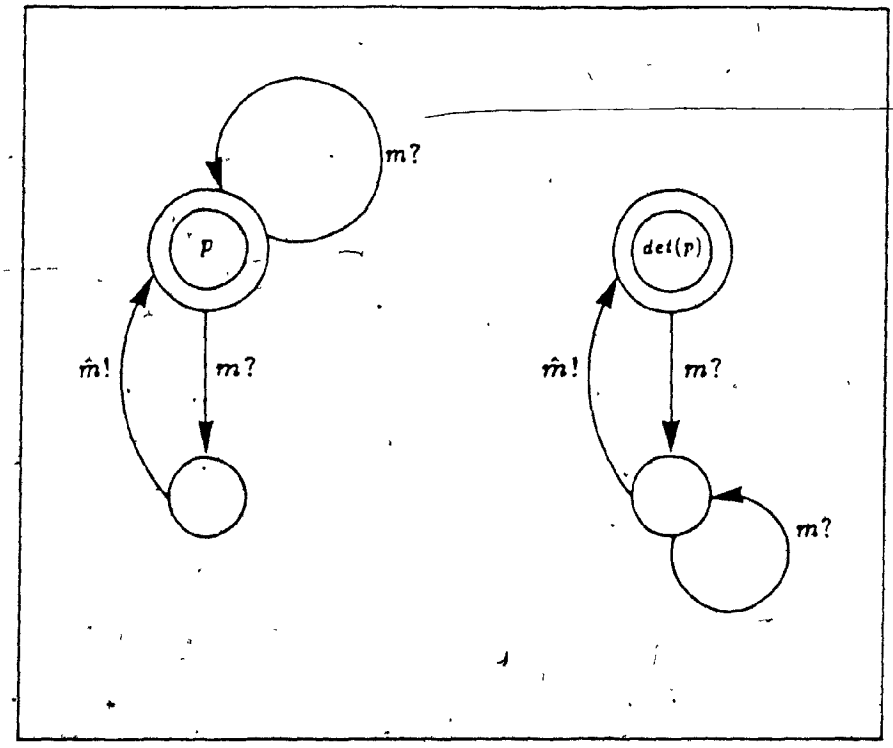

Fig. 3.8 A non-deterministic communications channel p and $det(p)$

which we can verify. First we calculate $x = co(det(m1_2))$ which gives

$$
\begin{aligned}
co(det(m1_2)) &= v1_2!.co(det(\hat{m}1_2)) \\
co(det(\hat{m}1_2)) &= r1_2?.co(det(m1_2 + m2_2)) \\
co(det(m1_2 + m2_2)) &= v1_2!.co(det(\hat{m}1_2)) + v2_2!.co(det(\hat{m}2_2)) \\
co(det(\hat{m}1_2)) &= r2_2?.co(det(m1_2 + m2_2))
\end{aligned}
$$

Next we perform the following steps to show that $m1_1 * L00 \leq co(det(m1_2))$

$$m1_1 * L00 \leq co(det(\hat{m}1_2)) \Leftarrow r1_2!\, \hat{m}2_1 * L_{21} + r2_2!\, m2_1 * L_{02} \leq r1_2!\; co(det(\hat{m}1_2))$$
$$\Leftarrow \hat{m}2_1 * L_{21} \leq co(det(\hat{m}1_2))$$

$$m2_1 * L_{21} \leq co(det(m1_2)) \Leftarrow r2_2'', m1_1 * L_{00} \leq r2_2''\; co(det(m1_2 + m2_2))$$
$$\Leftarrow m1_1 * L_{00} \leq co(det(m1_2 + m2_2))$$

$$m1_1 * L_{00} \leq co(det(m1_2 + m2_2)) \Leftarrow r1_2!, \hat{m}2_1 * L_{21} + r2_2!, m2_1 * L_{02} \leq r1_2!\; co(det(\hat{m}1_2))$$
$$+\; r2_2!; co(det(\hat{m}2_2))$$
$$\Leftarrow m2_1 * L_{21} \leq co(det(m1_2)) \bigwedge m2_1 * L_{02} \leq co(det(\hat{m}2_2))$$

$$m2_1 * L_{21} \leq co(det(\hat{m}1_2)) \Leftarrow r1_2''\; m1_1 * L_{00} \leq r1_2''\; co(det(m1_2 + m2_2))$$
$$\Leftarrow m1_1 * L_{00} \leq co(det(m1_2 + m2_2))$$

$$m2_1 * L_{02} \leq co(det(\hat{m}2_2)) \Leftarrow r2_2'', m1_1 * L_{00} * r2_2''\; co(det(m1_2 + m2_2))$$
$$\Leftarrow m1_1 * L_{00} \leq co(det(m1_2 + m2_2))$$

This proves the induction step required to show that $(m1_1 * L_{00} \leq co(det(m1_2))$

Chapter 4

Automating the Analytical Procedures

Our approach to the development of computer aided analysis of interacting processes rests on the automation of the mathematical operators of P_f. To accomplish this, an appropriate programming language had to be selected and a suitable representation of the DCP model within the constraints of the language had to be developed. The implementation of procedures that manipulate these representations in a manner consistent with the functions defined on P_f yields a system that allows the practical analysis of processes and their interactions. This system permits the prediction of the behaviour of interconnected networks of processes as well as the verification of that behaviour against a specification.

4.1 Choice of Language

There are several factors that virtually dictate the programming language most apt for this application. The DCP model is essentially symbolic in nature. Processes communicate by exchanging symbols. An appropriate language would be geared towards the manipulation of symbols. Numeric computational ability is not an

issue. The ability to easily create data structures analogous to tree and graph structures would also be advantageous.

All of these considerations point to the family of programming languages typically used for artificial intelligence applications. Of these, LISP is perhaps the most popular and pervasive. For these reasons, LISP was chosen as the language of implementation.

4.2 Process Representation

A finite dimensional process is represented within the system as a LISP atom whose value is that of a symbolic list. The list structure, in turn, represents the process equations that define the process as a least fixed point. For example, the non-deterministic communications channel

$$\begin{cases} q_1 = m_1^2.q_2 + m_2^2.q_1, \\ q_2 = m_1^1.q_1 \end{cases}$$

would be defined to the system by the evaluation of the LISP expression

```
(setq q1 '( (q1 ((m2  q1) (m2  q2)))
            (q2 ((m1  q1)))))
```

The LISP function setq would then create an atom q1 whose value would be the list

```
( (q1 ((m2  q1) (m2  q2)))
  (q2 ((m1  q1))))
```

representing the process equations above.

4.3 Operators and Procedures Implemented

All of the operators discussed in Chapter 3 are implemented as LISP functions. When applied to atoms whose values are valid process definitions, they return a list structure representing the process equations that define the resulting process. For instance the LISP expression

```
(det q1)
```

would return

```
( ({det_q1}((m?  {det_q1_q2})))
  ({det_q1_q2}((m?  {det_q1_q2}), (m-!  {det_q1}))))
```

Note that the process variables in the resulting list are in fact concatenations of the applied function and the existing process variables. In this way it is possible to determine the sequence of functional applications resulting in a particular process.

The mathematical functions $\Box$, R, $\vee$, co and det correspond to the LISP functions x, R, V, co and det respectively. It should be noted that LISP uses prefix notation, that is the functional application of a binary function such as $\Box$ to the arguments p and q corresponds to evaluation of the LISP expression (x p q). The interconnection of processes p and q, $p \cdot q$ can be calculated by evaluating the LISP expression (V (x p q)). The LISP function <= can be used to determine whether or not two processes are related. If $p \cdot q$ then (<= p q) would return t. If, on the other hand $p \not\leq q$ then nil would be returned.

In addition to this core, the LISP functions simplify and canon have been defined. The first eliminates equivalent process states from a process definition while

Summary of LISP Functions		
Lisp Function	ProcessOperator	Description
co	*co*	Returns the complement of a process
det	*det*	Returns the deterministic version of a process
x	$\square$	Performs the composition of two processes
R	*R*	Applies the reachability operator to a process
V	*V*	Applies the visibility operator to a process
<=	$\leq$	Implements the less than or equal to predicate
canon		Returns the canonical representation of a process
simplify		Returns a process with equivalent states removed

Fig. 4.9 Summary of LISP functions implemented

the latter returns the canonical representation of a process.

Chapter 5 The Theory of Process Synthesis

In the last chapter we dealt specifically with analysis, where we were concerned with verifying that a system p satisfied a property or predicate π. If p satisfies π then $\pi = true$. Typically the property π was a process equation involving either an inequality ($\leq$) or an equality ($=$). In this chapter we examine the related problem of synthesizing a system p to satisfy a property π. Here π can be interpreted as a constraint or equation that is to be solved for p. Automatic synthesis of processes is a potentially powerful tool for system designers.

5.1 Approach

In general the set X_π of all processes $\in P_E$ that satisfy the constraint π will be a non-empty subset of P_E. In this sense the constraint π does not necessarily possess a unique solution. It may, however have a unique minimal solution. Recall that any subset X of P_E has a unique greater lower bound (glb) $\sum X$ in P_E. If the glb of the solution set X_π is itself a member of X_π, it can be considered the optimal minimum process satisfying π. The process $\sum X_\pi$ is optimal in the sense that it contains all of the possible behaviours that satisfy the constraint π, and as such is

the broadest or maximally parallel process consistent with π. In some, but not all, cases we will be able to guarantee that $\sum X_\pi \in X_\pi$. In any case our approach will be to calculate $\sum X_\pi$ for a specification statement π and if $\sum X_\pi \in X_\pi$, it will be considered the optimal solution.

5.2 Mathematical Foundations

A continuous process function F is additive if

$$\forall p, q \in P_L, \quad F(p+q) = F(p) + F(q)$$

Let S_π be the solution set for the simple lower constraint $\pi(x) \equiv (l_0 \leq F(x))$, where l_0 is a given process and F is an additive function. Now if $x_1, x_2 \in S_\pi$ then we have

$$\begin{aligned} \pi(x_1) \bigwedge \pi(x_2) \Rightarrow (l_0 \leq F(x_1)) \bigwedge (l_0 \leq F(x_2)) &\Leftrightarrow (l_0 \leq glb\{F(x_1), F(x_2)\}) \\ &\Leftrightarrow (l_0 \leq F(x_1) + F(x_2)) \\ &\Leftrightarrow (l_0 \leq F(x_1 + x_2)) \end{aligned}$$

so $x_1 + x_2$ is also a member of S_π. So S_π is closed under addition in the sense that the sum (or glb) $x_1 + x_2$ of any solutions $x_1, x_2 \in S_\pi$ is also a solution.

A finitely based set of processes is one whose glb is expressible as the sum (or glb) of a finite number of it's elements. That is, a set S_π of processes is finitely based if and only if for some finite N and $x_i \in S_\pi$

$$\sum S_\pi = \sum_{i=1}^{N} x_i$$

Note that this also means that the glb of a finitely based set is a finitely branching process. The fact that the processes we are interested in are finitely branching has

been a tacit assumption so far. Indeed it is not clear what the physical counterpart of an infinitely branching process should be.

The additivity property is one that will guarantee that the glb of a solution set is itself a solution, provided that the solution set is finitely based. To see this, first assume that all solution sets are finitely based, then

$$\sum S_\pi = \sum_{i=1}^{N} x_i$$

for some $x_i \in S_\pi$ and finite N Due to additivity the solution set is closed under addition and so $\sum S_\pi \in S_\pi$ We may therefore conclude that $\sum S_\pi$ is the unique minimal solution. It is a straight forward matter to extend this result to the solution of a finite conjunction of simple lower constraints

The actual procedure for calculating $\sum S_\pi$ rests heavily on the Glb Expansion Theorem of Johnston[1] which states that under certain conditions the glb $\hat{x}_0$ of the solution set of a simple lower constraint can be expressed as the finite sum of event successor process pairs where the successor processes $\hat{x}_i$ are themselves the glb's of the solution sets of other simple lower constraints In particular if F_0 is a continuous function on P_E, and $F_i = F_0(e_i; x)$ and $\pi_i(x) \equiv (l_0 \leq F_i(x)) \equiv (l_0 \leq F_0(e_i; x))$ and assuming that all the S_{π_i} are finitely based with $\hat{x}_i = \sum S_{\pi_i} \in S_{\pi_i}$ then

$$\hat{x}_0 = \sum_{i=0}^{N} e_i; \hat{x}_i$$

The Glb Expansion Theorem suggests a recursive technique for the computation of $\hat{x}_0$ involving successive generation of simple lower constraints for successor processes . There are two main problems with this approach. The first is insuring

that the assumptions of the theorem are valid and the second is the question of termination: Does the recursion stop and if so how?

The assumptions of the theorem can be satisfied, in part, by placing the additional constraint that F_0 be an additive function. In this case, as we have already seen, S_{π_0} will be closed under + and contain $\hat{x}_0$ its glb. In practice for the types of functions that we will be interested in $(co.R.V.\sqcap.*)$ we will find that

$$\pi_i(x) = (l_0 \leq F_0(e_i; x)) \Leftrightarrow (\bigwedge_j^N (\bigvee_k^M (l_{j,k} \leq F_0(x))))$$

where for each j and k either $l_0 = \quad + e, l_{j,k} + \ldots$ or $l_0 = l_{j,k}$. This means that the simple constraints for the successor processes inherit the additivity of F_0 and as a result we are guaranteed that all the S_{π_i} are closed under addition. The assumption of the S_{π_i} being finitely based is more difficult to justify. It is clear, however, that the S_{π_i} must be finitely based for the solution $\hat{x}_0$ to be well behaved in the sense that it is finitely branching.

At this point a concrete example of the successive generation of new simple constraints for successor processes might be helpful. Consider the problem of finding a minimal process satisfying

$$\begin{aligned} L &\leq x \\ (co(detC_1)) &\leq x * C_2 \\ (co(detC_2)) &\leq x * C_1 \end{aligned}$$

where

$$\begin{aligned} L &= r_1?; L_{01} + r_2?; L_{01} \\ C_1 &= r_1!; C_{1+} \\ C_2 &= r_2!; C_{2+} \end{aligned}$$

and $x \in P_E$ with $E = \{r_1?, r_2?, s_1?, s_2?, g_1!, g_2!\}$. Assume that $x \leq r1?, x_1$ then we have

$$[r_1?; L_{01} + r_2?; L_{01} \leq r_1?; x_1] \Leftrightarrow [L_{01} \leq x_1]$$
$$[r_1?; (co(det(c_{1+}))) \leq r_1?; (x_1 * c_2) + r_2!; (x * c_{2+})] \Leftrightarrow [(co(det(c_{1+}))) \leq x_1 * c_2]$$
$$[r_2?; (co(det(c_{2+}))) \leq -.(x_1 * c_{1-}.1)] \Leftrightarrow [(co(det(c_{2+}))) \leq x_1 * c_{1-}.]$$

The successor process x_1 must then satisfy the constraints

$$L_{01} \leq x_1$$
$$(co(det(c_{1-}))) \leq x_1 * c_2$$
$$(co(det(c_{2-}))) \leq x_1 * c_{1-}.$$

which are of the same form as those we started with

The question of the termination of the recursion is also a tricky one. In theory, the recursion terminates when all the new constraints that are generated have already been encountered during the expansion process. In other words, if the constraint $\pi_i(x) \equiv (l_0 \leq F_i(x))$ is generated and then subsequently $\pi_j(x) \equiv (l_0 \leq F_j(x))$ where $e_i = e_j$, due to the uniqueness of the minimal solution it follows that $\hat{x}_i = \hat{x}_j$. We cannot, in the general case, insure that at some point no new constraints will be generated. In practice, when the constraints that we are solving are expressed uniquely in terms of finite dimensional processes and functions that preserve finite dimensionality, termination of the recursion is evident from the finite number of constraints that can be generated

All of the above results are extendable to the solution of the conjunction of a finite number of simple lower constraints, $\pi(x) \equiv \bigwedge_i^N {}_0(l_i \leq F_i(x))$, where the F_i are additive functions.

Several interesting questions remain unanswered. What are the necessary conditions for all the S_{π_1} to be finitely based? What class of constraints yields a finite dimensional solution? Can a methodology be found for solving constraints involving non-additive functions? Is it possible to solve constraints involving more than one occurrence of the process unknown (*i.e.* $I_0 \leq F(F_1(x), F_2(x))$ and if so under what conditions?

Chapter 6

Automating the Synthesis Procedures

The automation of the recursive technique suggested in the previous chapter by the Glb Expansion Theorem results in a system capable of generating a minimal process that satisfies a set of constraints. In practice, as we will see, these constraints are somewhat restricted in form. None-the-less there are several interesting types of synthesis problems that are tractable within this framework.

6.1 Qualifications and Restrictions

The existing implementation allows the automatic generation of the minimal solution satisfying a constraint of the form

$$\pi(x) \equiv \bigwedge_{i=1}^{N} (l_i \leq F_i(x))$$

where the l_i's are finite dimensional and the F_i's are additive functions. This insures that the solution set S_π is closed under addition and therefore that the glb of S_π is also a solution.

In practice the F_i's are made up of a combination of the operators co R and Σ as well as finite dimensional constant processes. The complement and reachability functions can readily be shown to be additive. The total composite is a binary operator and as a result can only appear in the constraints when linked with a constant process ($a \cdot c \square x$). The function $Q(x) = c \cdot x$ is additive if the constant process c is trace free (ie. stable) and communicates only with the unknown process x [1]. It must not offer any externally visible communication events.

In order to solve a set of constraints it is necessary to know what process space the solution lies in. This is equivalent to knowing apriori what communication events the solution process p can exchange. These events form the event set E. Which events belong to E depends on the processes that p is connected to communicate with. In general E must be specified by the system designer before proceeding with the synthesis. It is possible under certain assumptions to automate this operation.

Let l and c be trace free finite-dimensional processes. Then the solution x for the constraint

$$\tau \equiv (l \leq c \cdot x)$$

is closed under addition if c communicates uniquely with the process x. Mathematically this latter condition translates to $c \in P_{E_c}$, $x \in P_{E_x}$, and $E_c \cdot E_c \cdot E_x$. For any x satisfying the constraint, the composite $c \cdot x$ must be trace free since l is trace free. Thus for all $x \in S_\pi$, the process $c \cdot x$ is stable and so $c \cdot x = R(c \square x)$. Now the function $R(c \square x)$ is additive since it is composed of additive functions implying that S_π is closed.

The above observation allows us to solve a class of synthesis problems that we will refer to as Central Server problems. In these problem we will be concerned with finding a central process S that interacts with a finite number of client processes $C_i, i = 1, \ldots, N$. Each client process communicates exclusively with the central server S. The process S will be required to satisfy a virtualization principle with respect to each client that states

$$coldet(C_i) \leq (\times_{j=1}^{N} C_j) \setminus S$$

for $i = 1, \ldots, N$, as well as some resource constraint

$$L(S) \leq \ldots$$

In some cases, the minimal solution may not be an appropriate one from the designers point of view. It may be too general in nature. In addition a solution p to the constraint $L(c \times x) \leq \ldots$ may be such that the composite $c \times p$ contains possible transitions to the null process. In this situation the minimal solution p may be pruned to obtain a more satisfactory process. The pruning operation simply involves the removal of communication events whose occurrences leads to the offending deadlock. Pruning of this nature lends itself quite easily to automation. Mathematically, deadlock can be eliminated by imposing an upper constraint such as $c \times x \leq \mu_0$.

Chapter 7 Results

With the analysis and synthesis aids in place it is possible to undertake a practical assessment of the applicability of the DCP model. Of course, as we cannot verify the results empirically, assessment is a subjective term. By examining specific problems and their formulations within the terms of the DCP model we are able to compare the results with intuitive expectations and thus in some sense evaluate the model.

7.1 Analysis

7.1.1 Simple Resource Allocator

Consider a simple resource allocator that controls two identical resource units and deals with two customers. Each customer non-deterministically requests one unit and returns it or sequentially requests two units before releasing both. Cus-

tomer i is modeled as

$$
\begin{aligned}
C_i &= r_i!; C_{i1} \\
C_{i+} &= g_i?; C_{i1} + g_i?; C_i \\
C_{i-} &= s_i!; C_i \\
C_{i1} &= r_i!; C_{i1+} \\
C_{i1+} &= g_i?; C_{i2} \\
C_{i2} &= s_i!; C_{i-}
\end{aligned}
$$

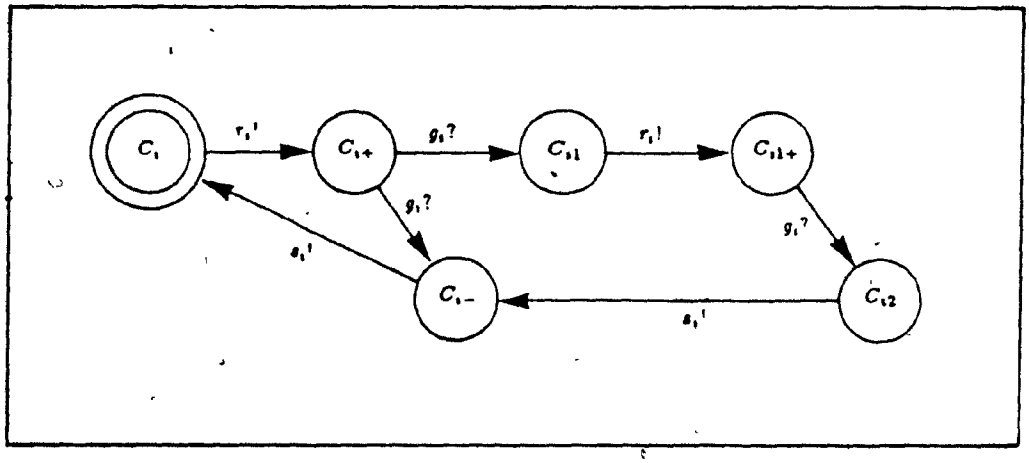

Fig. 7.10 Transition diagram for customer i

where r_i, g_i, s_i correspond to the request, grant, and surrender of one resource unit for customer i

The allocator initially offers both resource units to both customers, but after having granted one to customer i, offers the second unit to the same customer until

both units are returned. The allocator is modeled as

$$
\begin{aligned}
A &= r_1?; A_{\cdot,0} + r_2?; A_{0,\cdot} \\
A_{\cdot,0} &= g_1!; A_{1,0} \\
A_{1,0} &= r_1?; A_{1,\cdot,0} + s_1?; A \\
A_{1,\cdot,0} &= g_1!; A_{2,0} \\
A_{2,0} &= s_1?; A_{\cdot,0} \\
A_{\cdot,0} &= s_1?; A + r_2?; A_{\cdot,\cdot} \\
A_{0,\cdot} &= g_2!; A_{0,1} \\
A_{0,1} &= r_2?; A_{0,1,\cdot} + s_2?; A \\
A_{0,1,\cdot} &= g_2!; A_{0,2} \\
A_{0,2} &= s_2?; A_0 \\
A_{0,\cdot} &= s_2?; A + r_1?; A_{\cdot,\cdot} \\
A_{\cdot,\cdot} &= s_1?; A_{0,\cdot} + g_2!; A_{\cdot,\cdot} \\
A_{\cdot,\cdot} &= s_2?; A_{\cdot,0} + g_1!; A_{\cdot,\cdot} \\
A_{\cdot,\cdot} &= s_1?; A_{0,1} + s_2?; A_{1,0}
\end{aligned}
$$

It would be desirable that the interconnected system $A * C_2$ be able to satisfy the resource demands of C_1. In fact, since C_1 is non-deterministic, $A * C_2$ must be able to contend with the sum of the behavioural possibilities of C_1 as represented by $det(C_1)$. Mathematically this translates to the verification of the equation

$$A * C_2 \leq co(det(C_1))$$

Using the automated analysis system we calculate $A * C_2$:

$$
\begin{aligned}
A * C_2 &= r_1?.(A_{\cdot,0} * C_2) \\
A_{\cdot,0} * C_2 &= g_1!.(A_{1,0} * C_2) \\
A_{1,0} * C_2 &= r_1?.(A_{1,\cdot,0} * C_2) + s_1?.(A * C_2) \\
A_{1,\cdot,0} * C_2 &= g_1!.(A_{2,0} * C_2) \\
A_{2,0} * C_2 &= s_1?.(A_{\cdot,0} * C_2) \\
A_{\cdot,0} * C_2 &= r_1?.(A_{1,\cdot,0} * C_2) + s_1?.(A * C_2) + -:(A_{*,*} * C_{21}) \\
A_{*,*} * C_{21} &= s1?.(A * C_2)
\end{aligned}
$$

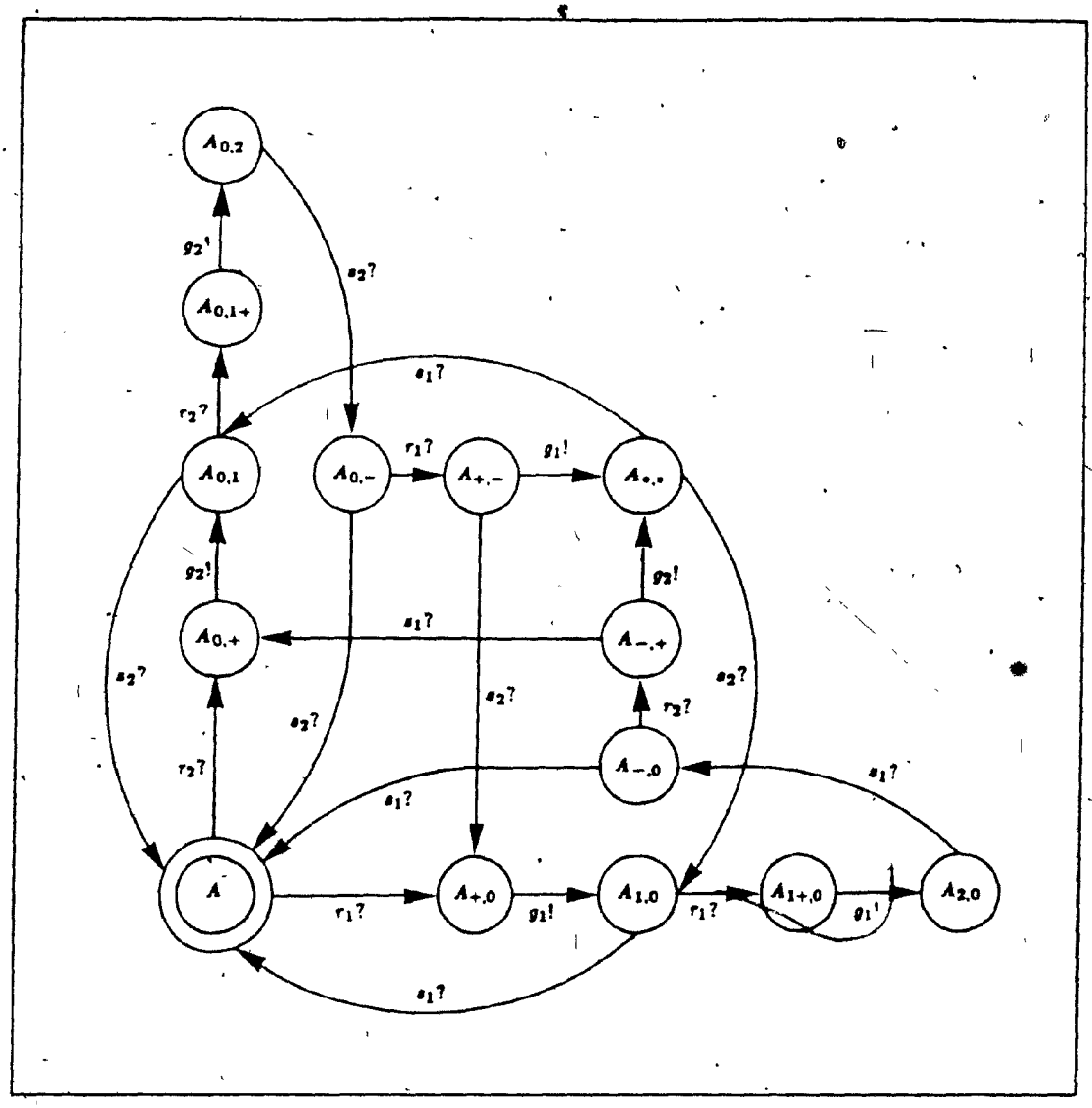

Fig 7 11 Transition diagram for simple resource allocator

Evaluation of the LISP expression (<= (V (x (A C2))) (co (det C1))) returns t, verifying that $A = C_2$ indeed satisfies all of the requirements of C_1

This example is a particular instance of a central server configuration In general, a central server process A will be required to satisfy a finite number N of (possibly non-deterministic) customer processes $C_i, i = 1, N$ The central server

may control shared resources and arbitrate between the customers in a manner so that they do not interfere with each other. An appropriate allocator must satisfy a virtualization principle which states that the interconnection of the allocator and all the customers other than C_i must contain the behaviour of $co(detC_i)$. This condition insures that the customer C_i is not blocked indefinitely.

7.1.2 Alternating Bit Protocol

The main objective of alternating bit protocols is to provide a reliable communication path over a channel that is susceptible to transmission errors and message loss. Variations of this type of protocol have been discussed by Lynch[24], Bartlett et al [25], and Bochmann[26] among others. The essential idea of the protocol is to sequence the transmitted messages by appending a one-bit number to them. The receiver, upon receipt of a message, sends an acknowledgement with the same sequence number attached back to the sender. After a time out period and until the appropriate acknowledgement is received, the sender retransmits the same message (with the sequence number unchanged). When an acknowledgement with the right sequence number is received, a new message with the sequence number complemented is sent (hence the name of the protocol).

The time-out mechanism effectively counters message loss in the channel. Transmission errors are compensated for through the intentional transmission of an incorrect acknowledgement to the sender, triggering retransmission of the message. We would like to verify that the protocol does indeed transform an unreliable channel into a reliable one.

The unreliable communications channel either delivers a message correctly, introduces a detectable error into the message, or loses the message completely. It is modeled by the non-deterministic process

$$
\begin{aligned}
C_m &= \sum_{u \in U} (u?; C_u + u?; C_{u_e} + u?; C_m) + \sum_{v \in V} (v?; C_v + v?; C_{v_e} + v?; C_m) \\
C_u &= \hat{u}!; C_m \\
C_{u_e} &= \hat{u}_e!; C_m \\
C_v &= \hat{v}!; C_m \\
C_{v_e} &= \hat{v}_e!; C_m
\end{aligned}
$$

where $U = \{m_0, m_1\}$ consists of messages from the sender with a 0 or a 1 appended, $V = \{a_0, a_1\}$ consists of the acknowledgement with a 0 or a 1 appended, and the signals $\hat{u}_e$ and $\hat{v}_e$ correspond to erroneous transmissions.

The Sender process is modeled as:

$$
\begin{aligned}
S_0 &= m?.S_{m_0} \\
S_{m_0} &= m_0!; S_{\hat{a}_0} + \hat{a}_e?.S_{m_0} + \hat{a}_1?; S_{m_0} + \hat{a}_0?.S_1 \\
S_{\hat{a}_0} &= \hat{a}_e?.S_{m_0} + \hat{a}_1?.S_{m_0} + t?.S_{m_0} + \hat{a}_0?; S_1 \\
S_1 &= m?.S_{m_1} \\
S_{m_1} &= m_1!; S_{\hat{a}_1} + \hat{a}_e?.S_{m_1} + \hat{a}_0?.S_{m_1} + \hat{a}_1?.S_0 \\
S_{\hat{a}_1} &= \hat{a}_e?.S_{m_1} + \hat{a}_0?.S_{m_1} + t?; S_{m_1} + \hat{a}_1?.S_0
\end{aligned}
$$

where m is the message to be transmitted, a is the acknowledgement, and t is a time-out signal sent by the time-out process T

$$T = t!; T$$

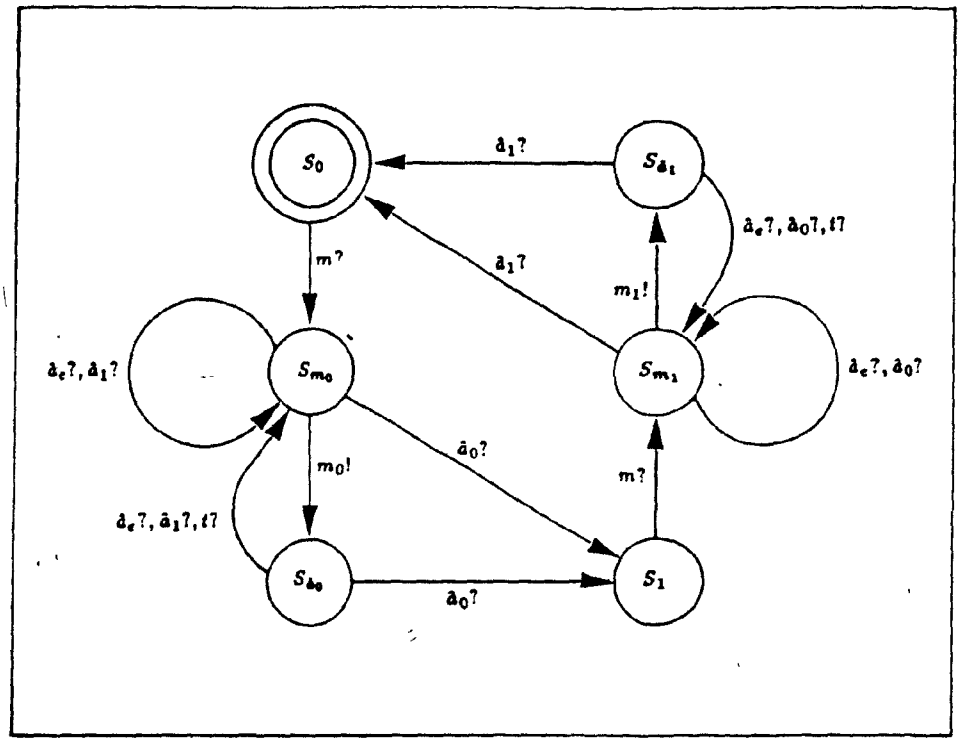

Fig. 7.3 The Sender process

The Receiver process is modeled as

$$
\begin{aligned}
R_0 &= \hat{m}_e?; R_{a_1} + \hat{m}_1?; R_{a_1} + \hat{m}_0?; R_{\hat{m}_0} \\
R_{\hat{m}_0} &= \hat{m}!, R_{a_0} + \hat{m}_e?; R_{\hat{m}_0} + \hat{m}_0?; R_{\hat{m}_0} \\
R_{a_0} &= a_0!; R_1 + \hat{m}_e?, R_{a_0} + \hat{m}_0?; R_{a_0} \\
R_1 &= \hat{m}_e?; R_{a_0} + \hat{m}_0?; R_{a_0} + \hat{m}_1?; R_{\hat{m}_1} \\
R_{\hat{m}_1} &= \hat{m}!; R_{a_1} + \hat{m}_e?, R_{\hat{m}_1} + \hat{m}_1?; R_{\hat{m}_1} \\
R_{a_1} &= a_1!; R_0 + \hat{m}_e?; R_{a_1} + \hat{m}_1?; R_{a_1}
\end{aligned}
$$

If the protocol works as it is supposed to we would expect that the interconnection of S_0, T, C_m, and R_0 would be equivalent to a simple one element queue. Mathematically, this amounts to verification of the equation

$$S_0 * T * C_m * R_0 = Q_1^U$$

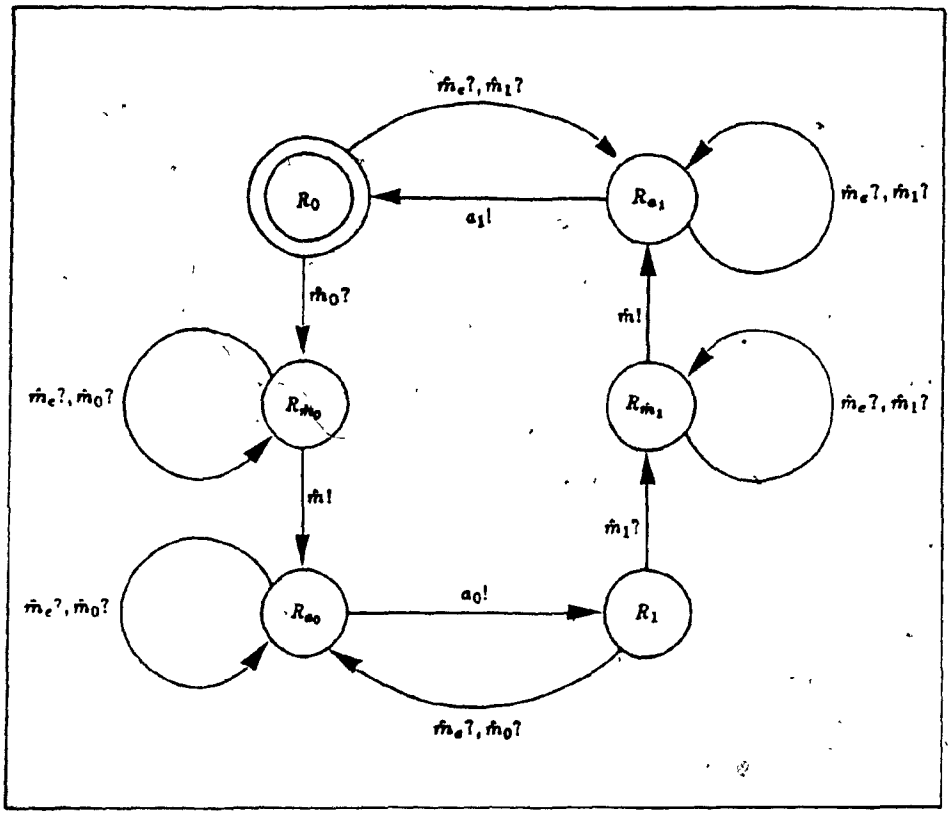

Fig. 7.4 The Receiver process

where by Q_1^λ we mean the single element queue defined as

$$Q_1^X = \sum_{x \in X} x?; Q_1^X(x)$$
$$Q_1^X(x) = \dot{x}!; Q_1^\lambda$$

Calculation of $S_0 * T * C_m * R_0$ with the aid of the automated analysis system indeed yields Q_1^X.

7.1.3 Dekker's Algorithm

Dekker's algorithm is an ingenious solution to what is known in the references as the critical section problem. When several sequential processors may asynchronously modify data in a shared memory space it is necessary to prevent simultaneous access and changes by two or more of them If this protection is not provided there is no guarantee that the data will faithfully reflect the intended modifications. The sections of the programs running on the various processors that access the common data store are called critical sections Thus the problem is to insure that at most one of the programs has entered its critical section at any one time. In order to effect this mutual exclusion, the programs can communicate through common variables kept in the shared memory space We can assume, without loss of generality, that the programs are cyclic Further the following assumptions are made

(i) Writing to and reading from the data store are considered to be indivisible operations.

(ii) Simultaneous access to a common data location results in sequential access of unknown order

(iii) No assumptions can be made about the relative speed of execution of the programs

(iv) A program may halt outside of its critical section and this must not interfere with the others.

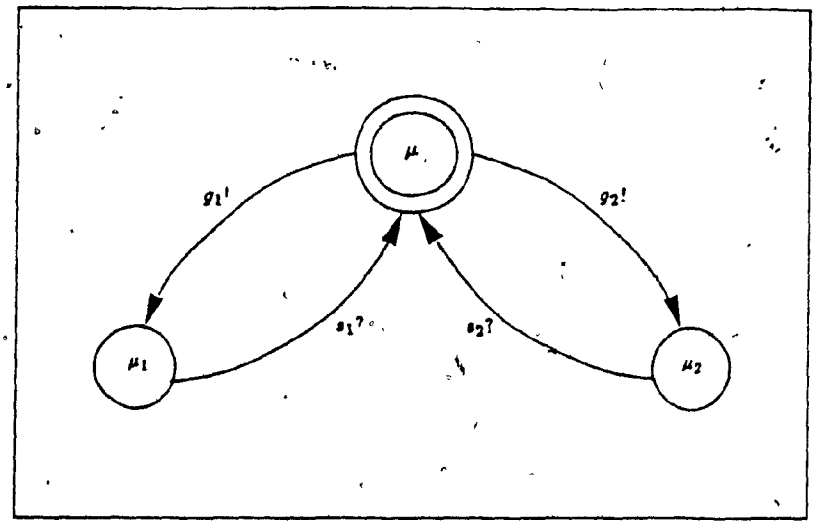

Fig 7.14 Idealized Mutual Exclusion Process μ

The solution to this problem was so elusive that some researchers doubted that it could be solved at all [27] In order to appreciate the subtleties involved consider the trial solution for two sequential processes expressed below in pseudo-ALGOL

```
begin boolean c1, c2.
      c1 := c2 := true
      P1: begin A1  c1 := false
                  L1  if not(c2) then goto L1.
                  critical section 1.
                  c1 := true:
                  remainder of program 1.
                  goto A1
            end;
      P2: begin A2: c2 := false:
                  L2. if not(c1) then goto L2:
                  critical section 2;
                  c2 := true;
                  remainder of program 2:
                  goto A2
            end
end.
```

The idea here is that the variable ci is set to *true* when Pi is well outside of its critical section. Before entering its critical section **P1** checks to make sure $c2$ is *true* before proceeding. If $c2$ is *false*, indicating that **P2** is in, or about to enter, its critical section, **P1** waits until $c2$ is reset.

An appropriate solution must guarantee, among other things, that deadlock does not occur and that mutual exclusion is enforced. Presuming that we can faithfully model the concurrent programs **P1**,**P2** and the variables $c1$,$c2$ as the DCP's P_1, P_2, $c1$, and $c2$ respectively, deadlock can be easily detected by the presence of the null process in the composite $P_1 \| c1 \| c2 \| P_2$ or in $P_1 \Box c1 \Box c2 \Box P_2$. Mutual exclusion is a little more difficult to verify. One way is to examine the total composite $P_1 \Box c1 \Box c2 \Box P_2$ for states where both P_1 and P_2 are in their respective critical sections. Another is to insert into the model externally visible signals to flag entry into (g_i') and exit from (s_i'') the critical section of P_i. Then verifying the statement

$$\mu \leq R(P_1 \| c1 \| c2 \| P_2)$$

where μ is an idealized mutual exclusion process

$$\mu = g_1' \mu_1 + g_2' \mu_2$$
$$\mu_1 = s_1'' \mu$$
$$\mu_2 = s_2'' \mu$$

is tantamount to verifying the mutual exclusion condition. This is so since the equation $\mu \leq R(P_1 \| c1 \| c2 \| P_2)$ being true means that the reachable behaviour of the composite is contained in that of μ thus disallowing any behaviours in which matched pairs of grant (g_i) and surrender (s_i) signals are interleaved. Yet another and perhaps less contrived, approach is to model the concurrent programs as customers vying for a unique resource managed by some allocator. The resource is the permission to enter a critical section and it is unique since at any instant at most one of the programs may have this permission. In this scenario each program would be modeled by a process that issues a request (r_i') when it wants to enter its critical section, accepts a grant (g_i'') from the allocator before proceeding and issues a surrender (s_i') immediately after exiting. The allocator managing the

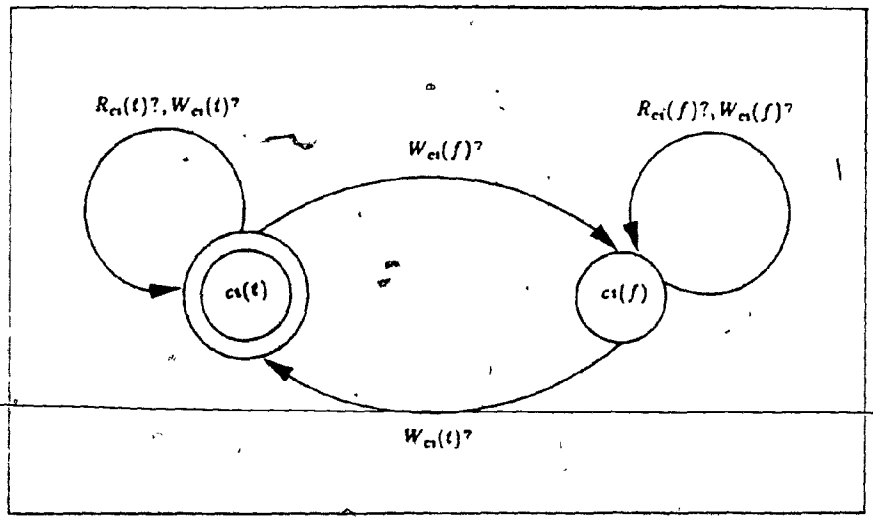

Fig. 7.6 The variable ci

granting of permissions would be in fact the interconnection of the processes (c1, c2) representing the shared variables and two other processes (P_1, P_2) that access these variables accept requests and surrenders and issue the grants Mutual exclusion would be guaranteed if the equation

$$\mu' \leq R(P_1 \square c1 \square c2 \square P_2)$$

where the process μ' is defined as

$$\begin{aligned} \mu' &= g_1!.\mu'_1 + g_2!.\mu'_2 + r_1\mu' + r_2?.\mu' \\ \mu'_1 &= s_1?.\mu' + r_1\mu'_1 + r_2?.\mu'_1 \\ \mu'_2 &= s_2?.\mu' + r_1\mu'_2 + r_2?.\mu'_2 \end{aligned}$$

could be shown to be true Again this condition will hold only if matching grants and surrenders are not interleaved.

These examples demonstrate the flexibility of the DCP model and show how the same problem can be formulated in different ways The appropriate formulation depends on the interpretation that we choose to assign to the abstract models that we use

In order to verify the trial solution we will proceed by the second method presented above that is each of the concurrent programs will be modeled as a process P_i with the external signals g_i' and s_i'' inserted to indicate entry into and exit from the critical section. The variable c_i is modeled as the process

$$c_i(t) = R_{c_i}(t)' \; c_i(t) + W_{c_i}(t)'' \; c_i(t) + W_{c_i}(f)'' \; c_i(f)$$
$$c_i(f) = R_{c_i}(f)' \; c_i(f) + W_{c_i}(f)'' \; c_i(f) + W_{c_i}(t)'' \; c_i(t)$$

where $W_{c_i}(t), W_{c_i}(f)$ are write signals modifying c_i to be *true* or *false* respectively and the signals $R_{c_i}(t), R_{c_i}(f)$ correspond to reading the value of c_i. Note that $c_i(t)$ corresponds to the variable c_i initialized to *true*. The program **P1** is modeled as

$$\begin{aligned}
P1 &= W_{c1}(f)' \; P1_{check} \\
P1_{check} &= R_{c2}(t)'' \; P1_{wait} + R_{c1}(f)'' \; P1_{wait} \\
P1_{wait} &= g1' \; P1_{surrender} \\
P1_{surrender} &= s1'' \; P1_{unflag} \\
P1_{unflag} &= W_{c1}(t)' \; P1 \\
P1_{wait} &= R_{c2}(t)' \; P1_{wait} + R_{c2}(f)'' \; P1_{wait}
\end{aligned}$$

where g_i' and s_i'' are the externally visible signals indicating entry into and exit from the critical section. Program **P2** is modeled similarly by the process $P2$. Note that the assumptions (i) and (ii) are inherent in the models that we have chosen for the variables. The assumptions (iii) and (iv) stipulate conditions that we are not strictly verifying here.

Calculation of $P_1 + c1(t) + c2(t) + P_2$ yields a non-stable process with internal transitions that eventually lead to the null process. In other words the composite deadlocks indicating that the trial solution wrong. Examination of the fixed point equations reveals that the deadlock occurs due to the following chain of events

(i) **P2** sets $c2$ to *false*

(ii) **P1** sets $c1$ to *false*

(iii) **P1** examines $c2$ and finds it to be *false*

(iv) **P2** examines $c1$ and finds it to be *false*

(v) **P1** and **P2** cycle checking variables $c2$ and $c1$ respectively each believing that the other is in it's critical section.

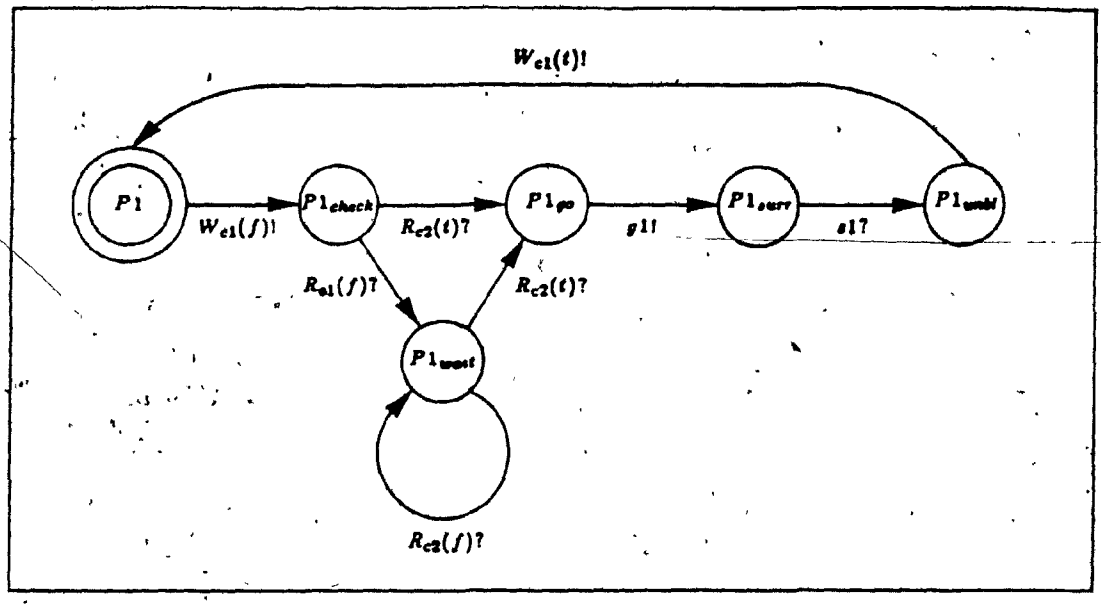

Fig. 7.10 The Process $p1$

Dekker's algorithm can be expressed in pseudo-ALGOL as follows [27]:

```
begin boolean c1, c2;
      integer turn;
      c1 := c2 := true; turn = 1.
      P1: begin A1. c1 := false;
                L1. if not(c2) then
                      begin if turn = 1 then goto L1;
                            c1 := true.
                            B1: if turn = 2 then goto B1;
                            goto A1
                      end.
                    critical section 1.
                    c1 := true. turn := 2;
                    remainder of program 1.
                    goto A1
          end;
      P2: begin A2 c2 := false.
                L2 if not(c1) then
                      begin if turn = 2 then goto L2;
                            c2 := true
                            B2 if turn = 1 then goto B2;
                            goto A2
                      end.
                    critical section 2.
                    c2 := true. turn := 1;
                    remainder of program 2;
                    goto A2
          end.
end.
```

We shall attempt to verify its correctness with respect to mutual exclusion and absence of deadlock. To accomplish this we begin by modeling the variables $c1$ and $c2$ as before. The variable $turn$ is modeled similarly as the process $t(1)$

$$t(1) = R_t^1(1)!.t(1) + R_t^2(1)!.t(1) + W_t(1)''.t(1) + W_t(2)''.t(2)$$
$$t(2) = R_t^1(2)!.t(2) + R_t^2(2)!.t(2) + W_t(2)''.t(2) + W_t(1)''.t(1)$$

where the signals $R_t^i(n)?$ represent process i testing the value of $turn$. The program

P1 is modeled as

$$
\begin{aligned}
P1 &= W_{c1}(f)!.P1_{check} \\
P1_{check} &= R_{c2}(t)?.P1_{go} + R_{c1}(f)?.P1_{turn} \\
P1_{go} &= g1!.P1_{surrender} \\
P1_{surrender} &= s1?.P1_{reset} \\
P1_{reset} &= W_{c1}(t)!.P1_{next\ turn} \\
P1_{next\ turn} &= W_t(2)!.P1 \\
P1_{turn} &= R_t^1(1)?.P1_{check} + R_t^1(2)?.P1_{reset\ c1} \\
P1_{reset\ c1} &= W_{c1}(t)!.P1_{wait} \\
P1_{wait} &= R_t^1(1)?.P1_{wait} + R_t^1(2)?.P1
\end{aligned}
$$

and program **P2** is modeled in a similar manner

Calculation of $P_1 \times c1(t) \times c2(t) \times t(1) \times P_2$ yields a non-stable process that is deadlock-free. The process $X' = R(P1 \square c1(t) \square c2(t) \square t(1) \square P2)$ is described by the fixed point equations

$$
\begin{aligned}
X &= g_1!.X_1 + g_2!.X_2 \\
X_1 &= s_1?.X \\
X_2 &= s_2?.X
\end{aligned}
$$

By inspection we can see that $X = \mu$, indicating that the algorithm insures mutual exclusion.

7.1.4 Conferencing Network

A typical conferencing network consists of a number of identical terminals connected in a star configuration through a communications bridge. The bridge serializes the packets sent from terminals by time of arrival. The packets are then rebroadcast by the bridge to all of the terminals. In this manner each terminal

receives all of the packets emitted by any one of the terminals (including itself). A problem arises, however, when it is necessary to transmit a series of uninterrupted packets. The packets may contain, for instance, graphics primitives that must be processed in an uninterrupted fashion to produce a faithful display at each of the terminals. In order to accommodate this possibility each terminal must be able to gain exclusive control of the bridge thus insuring that the packets that it sends will not be interleaved with others. The problem is to develop a suitable protocol that will enforce mutually exclusive access to the bridge.

For the sake of brevity we will consider here the relatively simple case of two terminals and a maximum of two packets in transit. The strategy used by terminal i is to inform the other terminals of its desire to gain exclusive access of the bridge by emitting a request signal. If no similar request from the other terminals is received in the interval between the time when the request was broadcast and the time the echo of the request is received at the terminal, exclusive access is assumed. At this point the terminal ignores requests from the other terminal and goes about its business. In order to relinquish control of the bridge the terminal again emits the request signal. On the other hand, if a foreign request is received prior to the echo of its own request the terminal assumes that another terminal has control and waits for the bridge to be released before trying again.

In order to verify that the protocol proposed above does indeed guarantee mutual exclusion, we proceed much as we did for Dekker's algorithm. Terminal.1 is

modeled by the process $t1$ as defined by

$$
\begin{aligned}
t1 &= r1!.t1_{listen} + r2_1?.t1_{wait} \\
t1_{listen} &= r1_1?.t1_{go} + r2_1?.t1_{wait+} \\
t1_{go} &= g1!.t1_{use} + r2_1?.t1_{go} \\
t1_{use} &= s1?.t1_{relinquish} + r2_1?.t1_{use} \\
t1_{relinquish} &= r1!.t1_{echo} + r2_1?.t1_{relinquish} \\
t1_{echo} &= r1_1?.t1 + r2_1?.t1_{echo} \\
t1_{wait} &= r2_1?.t1 \\
t1_{wait+} &= r1_1?.t1_{wait} + r2_1?.t1_{listen}
\end{aligned}
$$

Here the signals $rn_i?$ correspond to echoes of requests from the n^{th} terminal sent to the i^{th} terminal by the bridge. The symbols $g1!$ and $s1?$ are the externally visible signals inserted to indicate when the terminal assumes that it has control of the bridge.

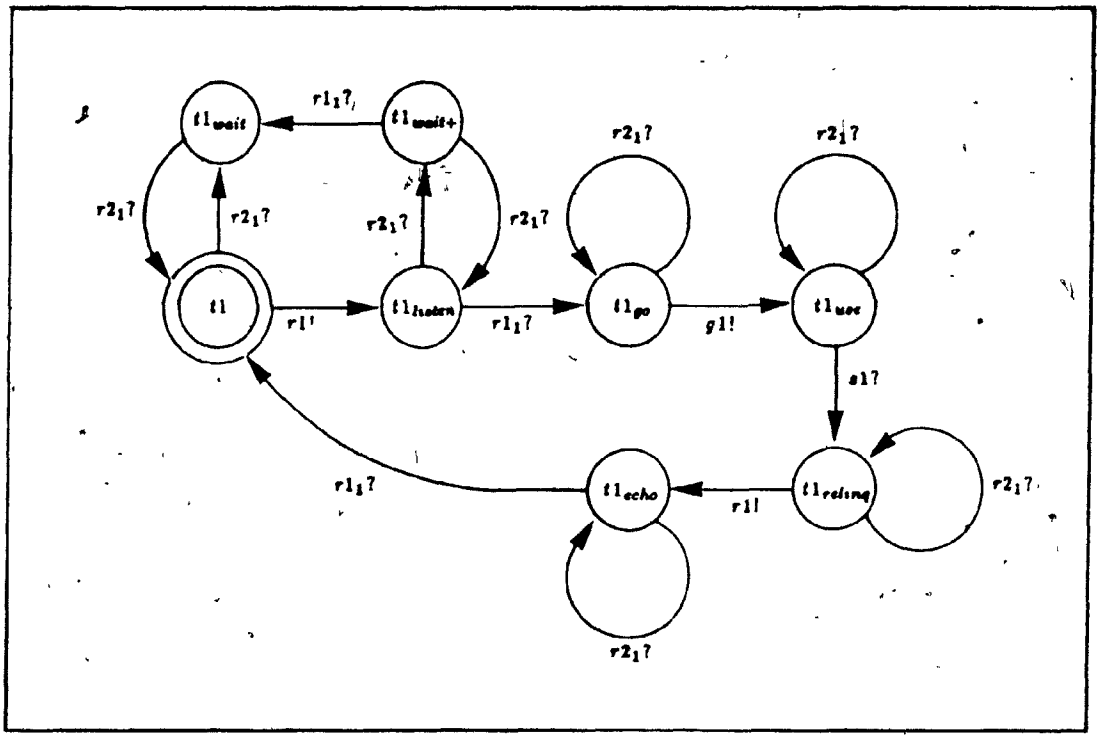

Fig. 7.8 Transition diagram for terminal $t1$

The bridge is modeled as the interconnection of a FIFO queue of length two and a broadcaster process that takes messages from the queue and sends copies of them to both of the terminals. The queue acts as a buffer and accomplishes the task of serializing the messages It is defined by

$$Q_2^X = \sum_{x \in X} x?.Q_2^{\lambda}(x)$$

$$Q_2^{\lambda}(x) = \hat{x}!;Q^X + \sum_{x' \in X} x'?;Q_2^X(x,x')$$

$$Q_2^{\lambda}(x,x') = \hat{x}!;Q_2^X(x')$$

where $X = \{r1, r2\}$.

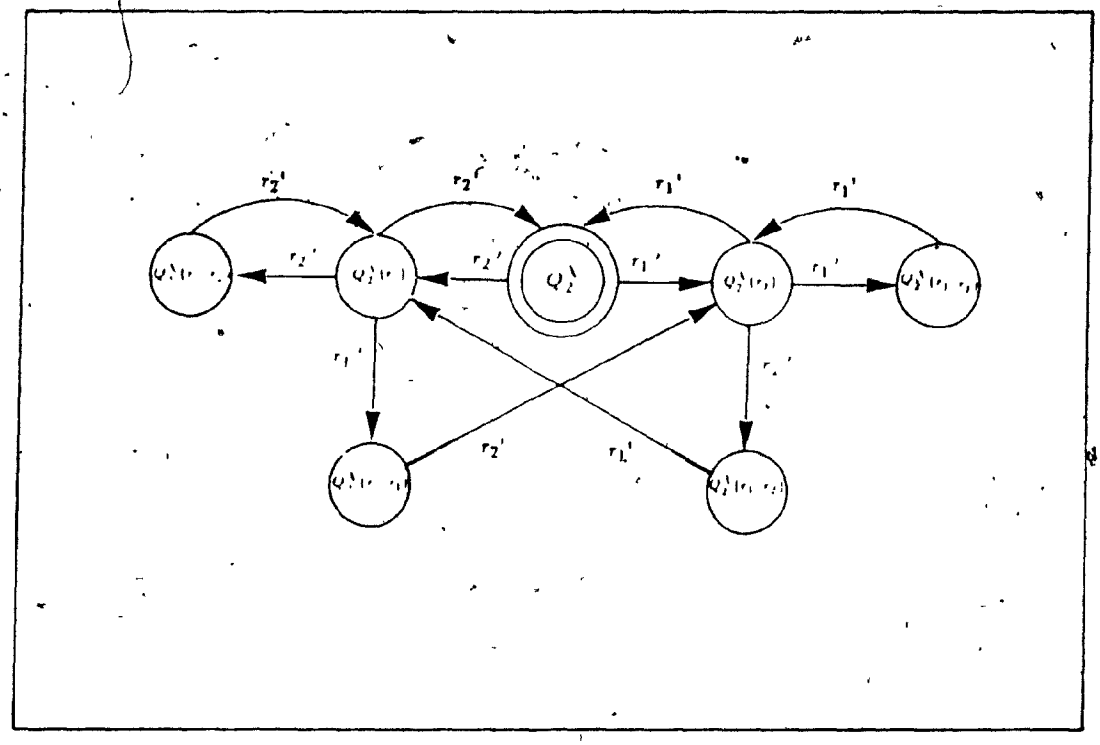

Fig. 7.18 Transition diagram for Q_2^X

The fixed point equations for the broadcaster are

$$
\begin{aligned}
B &= \hat{r}1?; B_1 + \hat{r}2?; B_{r2} \\
B_{r1} &= r1_1!; B_{r1_2} + r1_2!; B_{r1_1} \\
B_{r1_2} &= r1_2!; B \\
B_{r1_1} &= r1_1!; B \\
B_{r2} &= r2_2!; B_{r2_1} + r2_1!; B_{r2_2} \\
B_{r2_1} &= r2_1!; B \\
B_{r2_2} &= r2_2!; B
\end{aligned}
$$

Notice that the broadcaster is flexible in that the duplicate messages are not always sent to the terminals in the same order

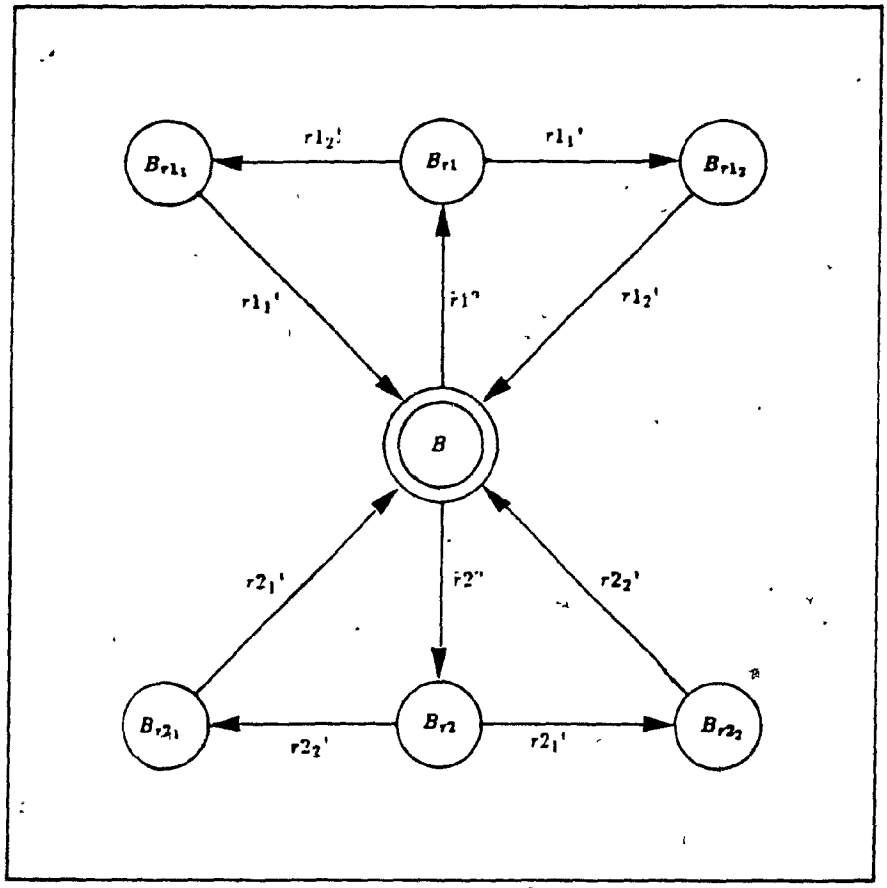

Fig. 7.19 Transition diagram for broadcaster B

Using the automated analysis system to calculate $t1 * Q_2^X * B * t2$ we discover that the interconnected system is deadlock-free. Calculation of the reachable behaviour of the composite yields a process that is equal to the idealized mutual exclusion process μ indicating that mutual exclusion is enforced by the protocol.

7.2 Synthesis

7.2.1 Simple Resource Allocator

In this example we examine the problem of designing a simple central server process that manages two identical resource units between two customers. Each of the customers, as in the analysis example, may request one unit and then non-deterministically decide to request another or surrender the first. Customer i is modeled as the process C_i

$$
\begin{aligned}
C_i &= r_i!; C_{i+} \\
C_{i+} &= g_i?; C_{i1} + g_i?, C_{i-} \\
C_{i-} &= s_i!; C_i \\
C_{i1} &= r_i!; C_{i1+} \\
C_{i1+} &= g_i?; C_{i2} \\
C_{i2} &= s_i!; C_{i-}
\end{aligned}
$$

It is the allocators job to accept requests, issues grants and accept releases. In doing so it must not violate the physical constraints of the situation. In other words it must only issue a grant in response to a request from a customer, it must issue at most one grant per resource unit, and it must keep track of the available resources. These physical constraints are illustrated graphically in Figure 7.20.

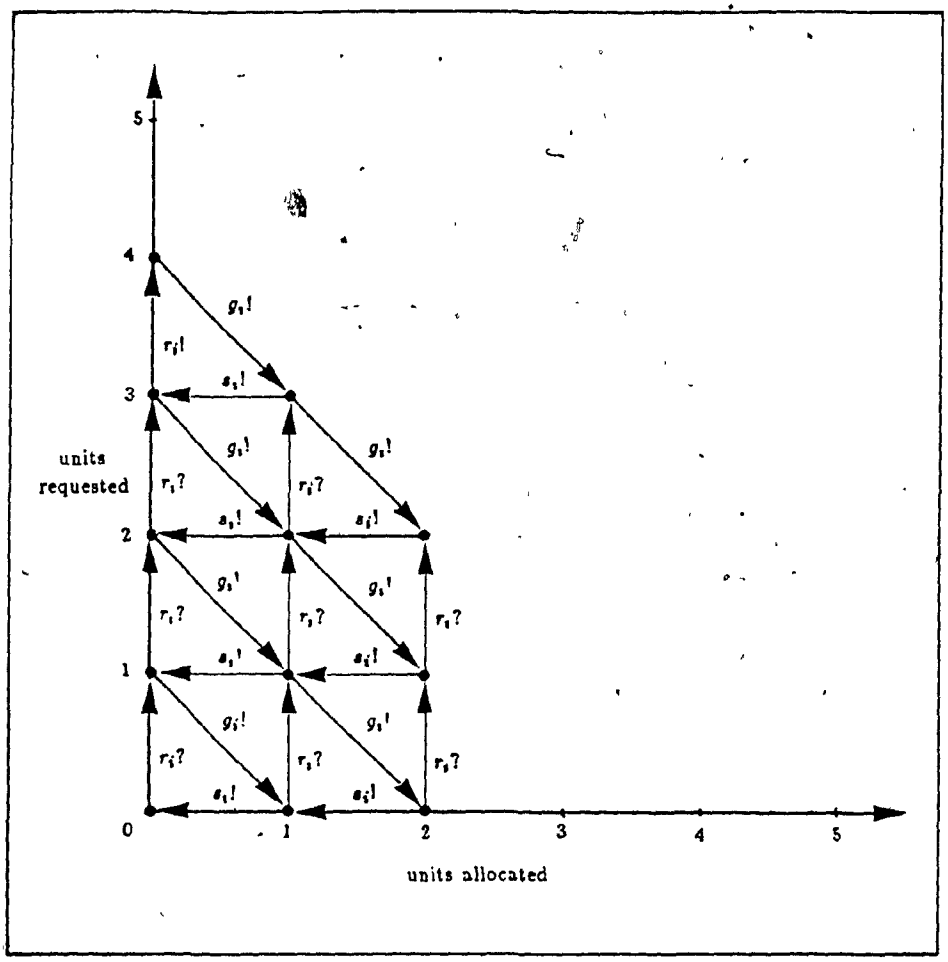

Fig. 7.20 Conservation of Resource Units

The allocator may exist in each of the states indicated on the graph. A request from a customer effectively increments the number of queued requests by one while the surrender of a unit decrements the same queue. A grant simultaneously decrements the request queue and increases the number of allocated units by one. The

process L embodying these relations is defined as

$$
\begin{aligned}
L &= r_1?.L_{01} + r_2?.L_{01} \\
L_{10} &= r_1?\ L_{11} + r_2?.L_{11} + s_1?\ L + s_2?\ L \\
L_{20} &= r_1?.L_{21} + r_2?.L_{21} + s_1?.L_{10} + s_2?\ L_{10} \\
L_{01} &= r_1?\ L_{02} + r_2?\ L_{02} + g_1!\ L_{10} + g_2!\ L_{10} \\
L_{11} &= r_1?.L_{12} + r_2?\ L_{12} + g_1!\ L_{20} + g_2!\ L_{20} + s_1?.L_{01} + s_2?.L_{01} \\
L_{21} &= r_1?.L_{22} + r_2?.L_{22} + s_1?\ L_{11} + s_2?.L_{11} \\
L_{02} &= r_1?.L_{03} + r_2?.L_{03} + g_1!.L_{11} + g_2!\ L_{11} \\
L_{12} &= r_1?\ L_{13} + r_2?\ L_{13} + g_1!\ L_{21} + g_2!.L_{21} + s_1?\ L_{02} + s_2?.L_{02} \\
L_{22} &= s_1?\ L_{12} + s_2?.L_{12} \\
L_{03} &= r_1?.L_{04} + r_2?.L_{04} + g_1!\ L_{12} + g_2!.L_{12} \\
L_{13} &= g_1!.L_{22} + g_2!\ L_{22} + s_1?\ L_{03} + s_2?.L_{03} \\
L_{04} &= g_1!\ L_{13} + g_2!.L_{13}
\end{aligned}
$$

The behaviour of any reasonable allocator must be contained in the behaviour of L. In addition the allocator must satisfy the virtualization principle with respect to each customer. Thus the allocator A' must satisfy the constraint

$$\pi = (L \leq A') \bigwedge (cotdet(c_1)) \leq c_2 \cdot A') \bigwedge (cotdet(c_2)) \leq c_1 \cdot A')$$

The finite state diagram for the solution process A' produced by the automatic synthesis program is shown in Figure 7.21. Using the automated analysis aids we can verify that the constraints are indeed satisfied by this process. It is interesting to compare this allocator to the one derived manually introduced in the analysis example. The first and perhaps most striking difference is the increased branching factor of the automatically generated solution. It contains many more possible behaviour patterns. This is to be expected, however, as the automatically derived allocator should after all be the minimum process satisfying the constraints. Indeed a quick check verifies that $A' \leq A$.

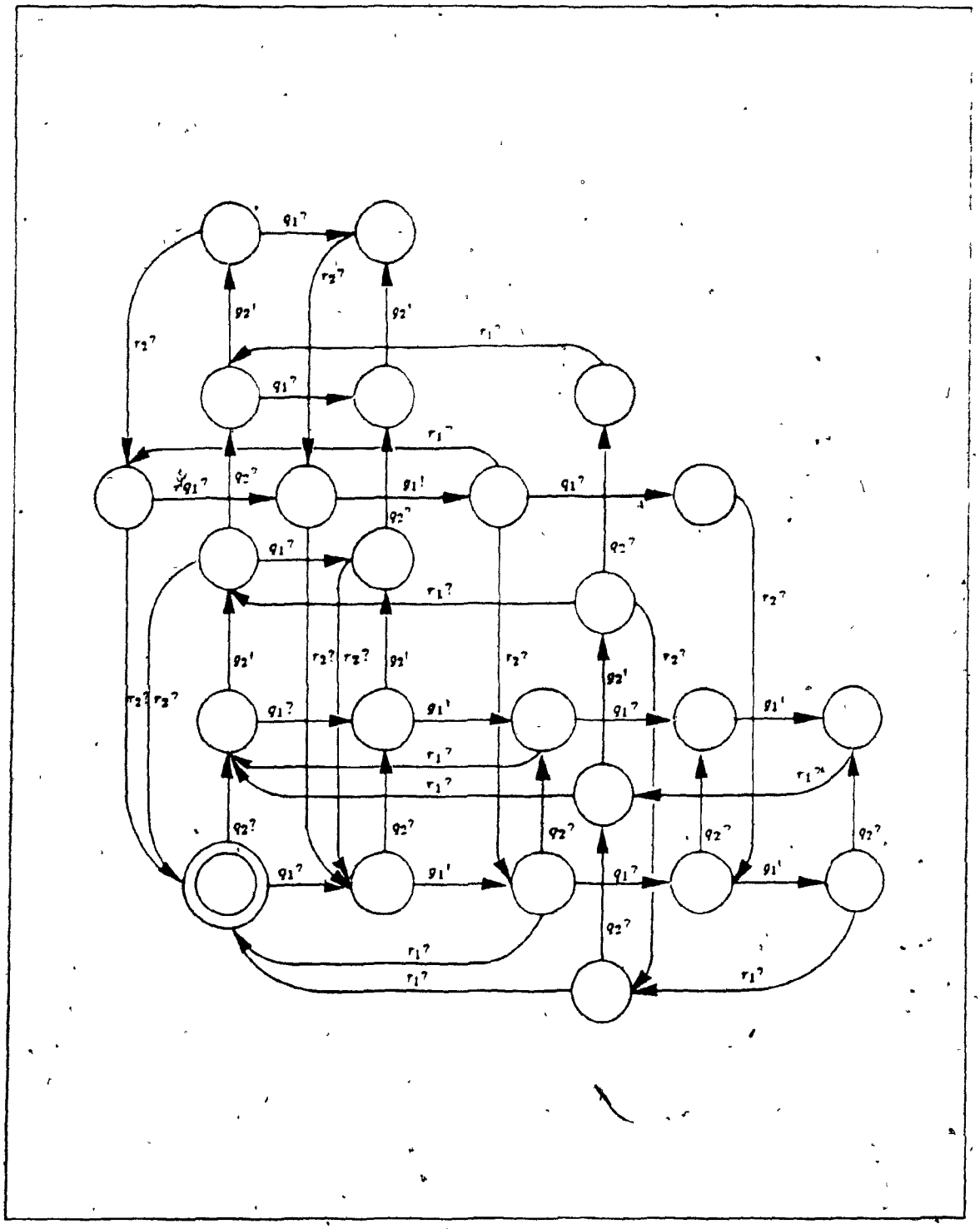

Fig. 7.21 Minimal Solution Process A'

7.2.2 Train Dispatcher

This is another central server type problem with the interesting variation that the client processes are not homomorphic. We are concerned with the design of a train dispatcher that allocates track segments to various trains who need them to complete their journeys. In this case two trains travel in opposite directions on four track segments with two crossover points (see Figure 7.22). The trains are not identical. Train A carries freight while Train B is a passenger train. The passenger train must adhere to a strict schedule of stops on each of the track segments. The freight train on the other hand does not care which particular track segments it traverses as long as it is making progress towards its destination.

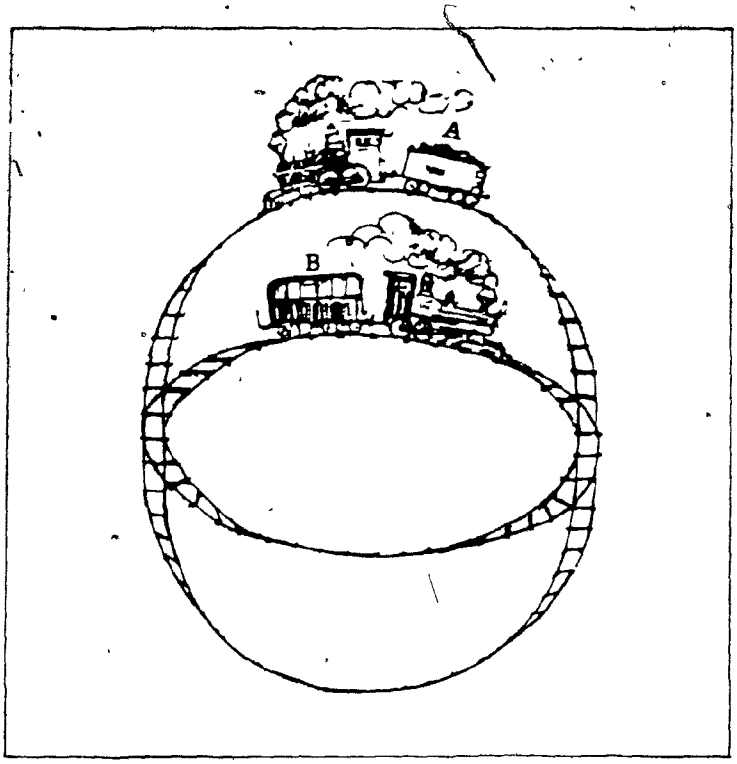

Fig. 7.22 Trains A and B on the tracks

We model Train A as the process a. When the Train A comes to a switching junction it issues a request for a track segment (q_{a_o}'). It is then prepared to accept grants from the dispatcher for the inside track segment (g_{a_i}') or the outside track segment (g_{a_o}').

$$a = q_{a_o}'.a$$
$$a = q_{a_i}'.a_i + g_{a_o}'.a$$

On the other hand when Train B reaches a junction it will require a specific track segment before it can continue. As the dispatcher will never know for sure which segment that will be required we model Train B as the non-deterministic process b

$$b = q_{b_i}''.b_i + q_{b_o}''.b_o$$
$$b_i = q_{b_i}'.b$$
$$b_o = q_{b_o}'.b$$
$$b = g_{b_i}''.b_i + g_{b_o}''.b_o$$

Note that after having received a grant (g_{b_i}? or g_{b_o}?) for a track segment b makes a non-deterministic transition either to a state where it requests an inside track segment (q_{b_i}') or to a state where it requests an outside track segment (q_{b_o}').

The dispatcher D must observe the physical constraint of the finite number of track segments as well as the requirement that the trains cannot simultaneously use the same track segment (not without disastrous consequences). These constraints

are embodied in the process L_{sm}.

$$
\begin{aligned}
L_{sm} &= q_a{}'' \cdot L_{sm+} + q_{bi}{}' L_{dm} + q_{bo}{}' \cdot L_{do} \\
L_{sm} &= q_a{}'' \cdot L_{sc+} + q_{bi}{}' \cdot L_{sm} + q_{bo}{}'' \cdot L_{sm} \\
L_{sc+} &= q_{ao}{}' L_{do} + q_{ai}{}' L_{di} + q_{bi}{}' L_{sc+} + q_{bo}{}'' \cdot L_{sc+} \\
L_{si+} &= q_{ao}{}' L_{di} + q_{ai}{}' L_{do} + q_{bi}{}' L_{di+} + q_{bo}{}' \cdot L_{di+} \\
L_{sc} &= q_a{}'' \cdot L_{si+} + q_{bi}{}'' L_{sc} + q_{bi}{}' L_{sc} \\
L_{si+} &= q_{ai}{}' L_{di} + q_{ao}{}' \cdot L_{do} + q_{bi}{}' L_{si+} + q_{bi}{}'' \cdot L_{si+} \\
L_{sc} &= q_a{}'' \cdot L_{si+} + q_{bo}{}' L_{di} + q_{bi}{}' L_{di} \\
L_{si+} &= q_{ai}{}' L_{do} + q_{ao}{}' \cdot L_{di} + q_{bi}{}' \cdot L_{di+} + q_{bi}{}' L_{di+} \\
L_{do} &= q_a{}' \cdot L_{di+} + q_{bi}{}'' L_{do} + q_{bo}{}'' \cdot L_{do} \\
L_{di+} &= q_{ao}{}' L_{so} + q_{bi}{}'' L_{di+} + q_{bo}{}'' L_{di+} \\
L_{di} &= q_a{}' \cdot L_{di+} + q_{bo}{}' L_{sm} \\
L_{di+} &= q_{ai}{}' L_{sm} + q_{bi}{}' \cdot L_{si+} \\
L_{di} &= q_a{}' \cdot L_{di+} + q_{bo}{}'' L_{di} + q_{bi}{}'' \cdot L_{di} \\
L_{di+} &= q_{ai}{}' L_{sm} + q_{bi}{}'' L_{di+} + q_{bi}{}'' L_{di+} \\
L_{di} &= q_a{}' \cdot L_{di+} \pm q_{bi}{}' \cdot L_{sm} \\
L_{di+} &= q_{ai}{}' L_{sm} + q_{bo}{}' \cdot L_{si+} \\
L_{di} &= q_a{}'' \cdot L_{di+} + q_{bi}{}'' \cdot L_{di} + q_{bo}{}'' \cdot L_{di} \\
L_{di+} &= q_{ao}{}' \cdot L_{sm} + q_{bi}{}'' \cdot L_{di+} + q_{bi}{}'' \cdot L_{di+} \\
L_{di} &= q_a{}'' \cdot L_{di+} + q_{bo}{}' \cdot L_{sm} \\
L_{di+} &= q_{ai}{}' \cdot L_{sm} + q_{bo}{}' L_{si+} \\
L_{do} &= q_a{}'' \cdot L_{di+} + q_{bi}{}'' L_{do} + q_{bo}{}'' \cdot L_{do} \\
L_{do+} &= q_{ai}{}' \cdot L_{sm} + q_{bi}{}'' \cdot L_{do+} + q_{bo}{}'' \cdot L_{do+} \\
L_{do} &= q_a{}'' \cdot L_{di+} + q_{bi}{}' \& L_{sm} \\
L_{do+} &= q_{ai}{}' \cdot L_{sm} + q_{bi}{}' \cdot L_{si+}
\end{aligned}
$$

The three subscripts to each of the state names designates a different possible configuration of the trains. The first subscript is s indicating that the trains are on the same side of the track diagram (adjacent segments) or d when they are on different sides (non-adjacent segments). The second and third subscripts apply to Train A and Train B respectively indicating if they are on the inside track (i) or the outside track (o). A '+' is appended to these if the train has reached a junction and has requested another segment. Thus the process L_{sm} deals with the situation

where the trains are on adjacent track segments. Train A on the outside track, and Train B on the inside track having already requested another segment. This process is the minimum behaviour for a dispatcher consistent with the physical constraints.

In addition we will require the dispatcher to satisfy a variation of the virtualization principle with respect to each of the trains. The dispatcher D must meet the constraint

$$\pi = (L_{sol} \leq D) \bigwedge (co(det(a)) \leq R(b \square D)) \bigwedge (co(det(b)) \leq (a * D))$$

The difference here is that the interconnection of the dispatcher with Train B is not required to be stable. This in effect allows the dispatcher to change its mind as to which track it will offer Train A which is fine since Train A doesn't care which track it travels on.

The finite state diagram describing the dispatcher process produced by the synthesis program is depicted in Figure 7.23. Upon verification this process is seen to satisfy the constraints imposed.

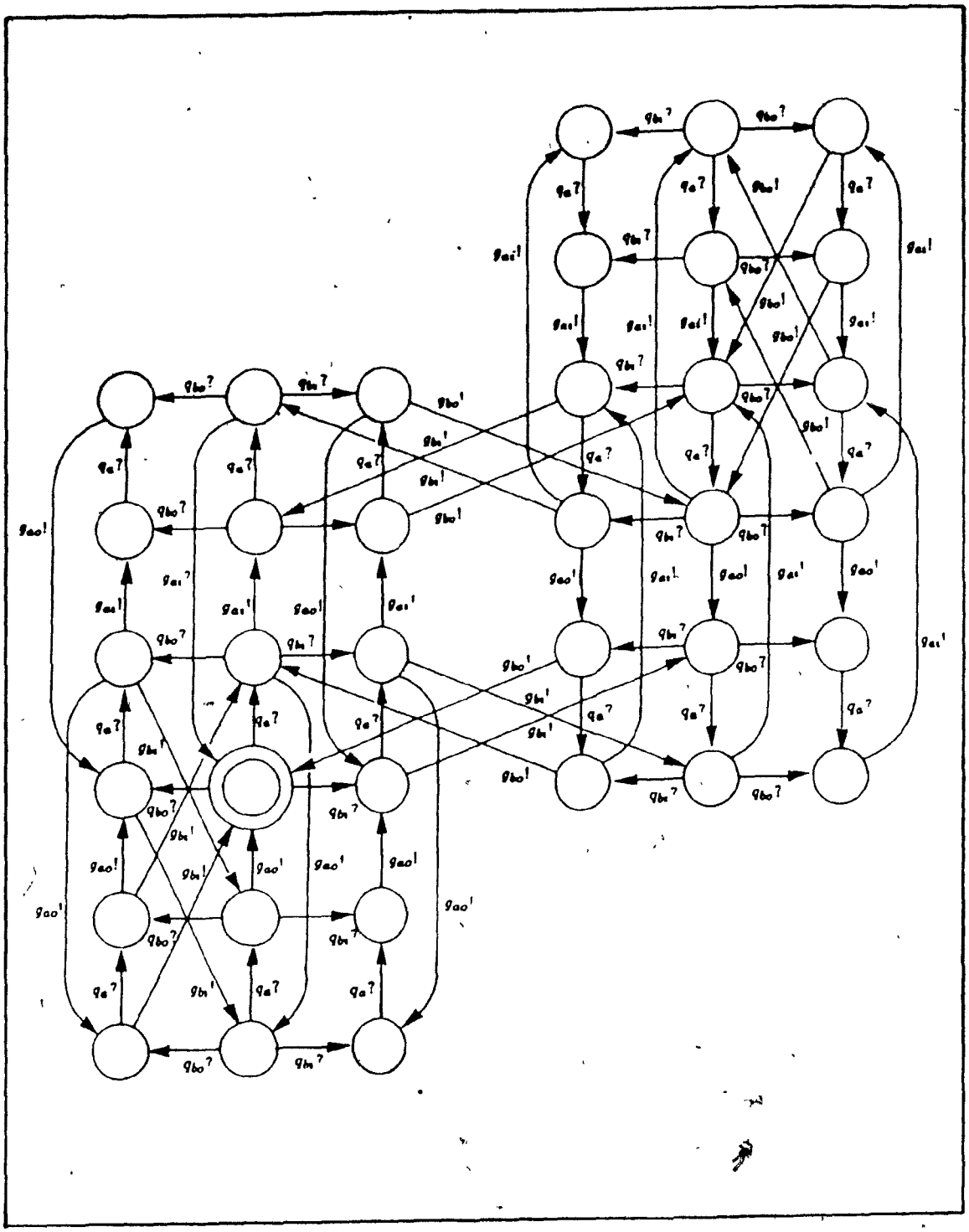

Fig. 7.14 Train dispatcher D as synthesized

Chapter 8 Discussion

The DCP model is a "black-box" model that represents a system by its externally observable behaviour. No assumptions are made about the internal structure of the modeled entities. This external behaviour consists of the exchange of discrete messages or signals with the environment. The exchange of a message is considered to be a primitive event that is instantaneous and indivisible. Processes communicate and evolve via the exchange of messages. There are two main aspects of a model that determine its usefulness. The first is the scope of its modeling ability and the second is ease of analysis. In general there is a trade off here with increased modeling power implying reduced analytical ability and vice versa. Indeed it is for this very reason that we have concentrated on finite-dimensional processes.

The more situations that a model can adequately abstract, the better. As we have seen communications protocols and resource allocation systems can be readily modeled as DCP's. The DCP model seems to be particularly well suited to the modeling of concurrent systems and is perhaps most appropriate for the modeling of computer network protocols.

The real advantage of the DCP model is its rigorous mathematical foundation in which processes are well defined mathematical objects. Suitable operators allow the formal prediction and analysis of system behaviour. Within this framework analysis is formalized as the verification of mathematical properties. This in turn means that the analysis techniques are easily automated resulting in a practical system that allows precise analysis where otherwise complexity would be prohibitive.

It is informative to interpret the actions of the total composite and interconnection operators on finite dimensional processes in terms of state space analysis. As we have already mentioned, the total composite corresponds to the generation of that part of the combined state space in which the composite process can exist. The interconnection, being the visibility operator applied to the total composite, essentially groups states that are externally indistinguishable together. This results in a state space compression that yields a computational advantage when calculating the interconnection of several processes. To see how this is so, consider the interconnection of three finite-dimensional processes a, b and c. In order to arrive at the interconnection we proceed by calculating the total composite a, b and then apply the visibility operator. This corresponds to a complete state space expansion followed by a compression of the type mentioned above. We then interconnect this process with the process c following the same procedures. In doing so we never generate all of the combined state space reachable by the composite. It is, however possible to do this by explicitly calculating $a \Box b \Box c$. To some extent this effect counters the problem of state space explosion normally encountered in this type of analysis.

The model attempts to retain the relevant information about a physical situa-

tion, but this does not mean that is is always successful in doing so. Some situations can not be adequately dealt with in the context of the model. Caution must be exercised in determining what the physical interpretation of an analytical result should be. To illustrate this point we return to another trial solution to the problem successfully solved by Dekker's algorithm. It can be expressed in pseudo-ALGOL as follows.

```
begin boolean c1, c2,
      c1 = c2 = true
      P1: begin L1 c1 = false,
                if not(c2) then
                   begin c1 = true, goto L1 end,
                critical section 1,
                c1 = true
                remainder of program 1
                goto L1
          end;
      P2: begin L2 c2 = false,
                if not(c1) then
                   begin c2 = true goto L2 end;
                critical section 2
                c2 = true,
                remainder of program 2
                goto L2
          end
end
```

This solution attempts to avoid the deadlock that we observed in the first algorithm that we examined by having the programs set and reset their c variable while waiting for the other to exit from its critical section. Thus the condition where both processes are about to enter their critical sections and have set their respective c variables to *false* will not necessarily lead to deadlock. The variables will be reset to *true* thereby creating an opportunity for one of the programs to proceed. If we perform the same type of analysis as before we find that the composite $P_1 * c1(t) * c2(t) * P_2$ is deadlock-free in the sense that it does not contain any transitions to the null process. In addition we can verify that $\mu = R(P_1 \square c1(t) \square c2(t) \square P_2)$

insuring that mutual exclusion is enforced.

It is, however, still possible for the processes to cycle indefinitely, each resetting and setting it's c variable before the other examines it. For the algorithm to be considered correct we are forced to make certain assumptions about the relative speeds of the processes, which in the context of this problem are not justified. It is clear that the type of analysis that we have performed does not bring this anomaly to the foreground. This does not mean that model is necessarily deficient, but rather that we must be careful in our interpretation of the results. In fact this behaviour could have detected through the examination of the total composite $P_1 \Box c1!(t) \Box c2!(t)$, P_2 for possible transition loops consisting entirely of trace events. Such loops indicate the possibility of indefinite cycling and are indeed present in this instance. The interconnection operator as it is presently defined is not designed to detect such occurrences. Perhaps the visibility operator should be modified to retain a trace transition to the null process in this situation.

One must not lose sight of the fact that the semantics arise from the interpretation that we choose to impose between the abstraction and physical reality.

One of the most exciting aspects of the equational theory of discrete communicating process is the formal basis it provides for process synthesis. There are still many avenues to be explored in this context. There is also some question as to what the minimal solution to a set of constraints actually represents. In general such a solution is not the only process to satisfy the constraints. It can be viewed as the most accommodating solution process in that it will never refuse to exchange a message that does not result in a strict violation of the constraints. Conceptually

it is the least blocking process in the sense that it never needlessly blocks a process that it interacts with by withholding a communication event.

A practical system for the automated synthesis of processes would provide the system designer with two important pieces of information. In a typical application the designer would begin by modeling all the relevant components in terms of their discrete behaviour, formulate the problem statement in terms of the appropriate constraints, and then use the system to generate the minimal solution. This will indicate to the designer whether or not any solution exists to his problem. Secondly, if a solution does exist, the system will provide him with a process whose behaviour is the lower bound of any possible solution process. This minimal solution may be of direct use to the designer but if not a satisfactory solution can probably be arrived at through judicious pruning.

Chapter 9 Conclusions

We have attempted to evaluate the algebraic theory of DCP's through its application to a variety of problems. The results are encouraging. The theory seems to have a broad applicability. It is particularly suited to the modeling and analysis of communication protocols. In general it can be used to objectively specify desired behaviour, predict the behaviour of composite systems and verify that such a system exhibits certain properties.

We have shown that the analytical procedures suggested by the theory can be automated resulting in analytical capabilities where otherwise complexity would be prohibitive. These automated tools, in conjunction with the theory provide a possible basis for an objective methodology for the analysis of communicating processes in general and communication protocols in particular.

The novel area of process synthesis has been introduced. Here we have demonstrated the feasibility of automating the synthesis procedures through the implementation of a system that generates the minimal process satisfying a set of constraints. This is a subject that requires more research for a better and more complete un-

derstanding.

At present, rapid state space explosion remains a very real problem Systems that contain queues of realistic length or protocols that use parametrized messages to affect system behaviour are just two examples where direct modeling will lead to an unmanageable state space (even for a computer) Further investigation of process homomorphisms that effectively compress the state spaces may yield a solution to this thorny problem The visiblity operator is an example of such a homomorphism Another possible direction for further research is the optimization of the computational algorithms with a view towards developing a practical CAD system for the analysis and design of communication protocols

REFERENCES

[1] R. de B. Johnston, "Discrete Communication Processes." INRS-Telecommunications Internal Report No. 84-07, 1984.

[2] M. B. Smyth, "Powerdomains " Theory of Computation Report 12, Department of Computing Science, University of Warwick, 1976.

[3] G. D. Plotkin, "A Power Domain Construction." *Siam Journal on Computing*, Vol. 5, pp. 452-487, 1976.

[4] G Milner, "Concurrent Processes and Their Syntax " *JACM*, Vol. 26, No 2, pp. 302-321, April 1979

[5] G. Milne, "A Mathematical Model of Concurrent Computation " Doctoral dissertation, University of Edinburgh, 1977.

[6] G. V. Bochmann and C. Sunshine, "Formal Methods in Communication Protocol Design." *IEEE Transactions on Communications*, Vol Com-28, No 4, pp. 624-631, April 1980.

[7] A L. Davis, "Tomorrow's Computers: Computer Architecture" *Spectrum*, Vol. 20, No. 11, pp.94-99, Nov. 1984.

[8] P. M. Merlin, "Specification and Validation of Protocols." *IEEE Transactions on Communications*, Vol. Com-27, No. 11, pp. 1671-1680, Nov. 1979.

[9] J L. Peterson, *Petri Net Theory and the Modeling of Systems* Prentice Hall Inc., Englewood Cliffs, N. J., U.S.A., 1981.

[10] F. J. W. Symons, "Introduction To Numerical Petri Nets, A General Graphical Model of Concurrent Processing Systems." *Australian Telecommunications Research*, Vol. 14, No. 1, pp. 28-32, 1980.

[11] G. V. Bochmann and J. Gecsei, "A Unified Method for the Specification and Verification of Protocols." *Proceedings of IFIP Congress*, Toronto, Canada, pp. 229-234, August 1977

[12] P. M. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol." *IEEE Transactions on Communications*, Vol Com-27, pp. 1230-1237, 1979

[13] H. Rudin, C. H. West, P. Zafiropulo, "Automated Protocol Validation: One Chain of Development" *Proceedings of Computer Network Protocols Symp.*, Liege, Belgium, Feb 1978

[14] C. H. West and P. Zafiropulo, "Automated Validation of a Communications Protocol: The CCITT X21 Recommendation" *IBM Journal of Research and Development*, Vol 22, No. 1, Jan 1978

[15] P. Zafiropulo, "Protocol Validation by Duologue Matrix Analysis" R2816, IBM Zurich Research Laboratories, 8803 Ruschlikon, Switzerland, 1977.

[16] D. H. Thompson, C. A Sunshine, R. W Erickson, S. L Gerhart, D Schwabe, "Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models" ISI/RR-81-88, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, Mar 1981.

[17] D. Schwabe, "Formal Specification and Verification of a Connection Establishment Protocol." ISI/RR-81-88, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, Mar. 1981.

[18] P. Zafiropulo, "Towards Analysing and Synthesizing Protocols." *IEEE Transactions on Communications*, Com-28, No. 4, pp. 651-661, April 1980.

[19] P. Merlin, G. V. Bochmann, "On the Construction of Submodule Specifications and Communication Protocols." *ACM Transactions on Programming Languages and Systems*, Vol. 5, No. 1, pp. 1-25, Jan. 1983.

[20] C. A. R. Hoare, "Communicating Sequential Processes." *Communications of the ACM*, Vol. 21, No. 8, Aug 1978.

[21] R. Milner, *A Calculus of Communicating Systems*, Springer-Verlag, Berlin, 1980.

[22] G. Birkoff and T. C. Bartee, *Modern Applied Algebra*, McGraw-Hill Book Company, New York, 1970.

[23] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill Book Company, New York, Chapter 5, 1974.

[24] W. C. Lynch, "Reliable Full-Duplex Transmission Over Half-Duplex Telephone Lines" *Communications of the ACM*, Vol. 11, No. 6, pp. 361-372, June 1968.

[25] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Note on Reliable Full-Duplex Transmission Over Half-Duplex Lines." *Communications of the ACM*, Vol. 12, No. 5, pp. 260-261, May 1969.

[26] G. V. Bochmann, "Finite State Description of Communication Protocols." *Computer Networks*, Vol. 12, pp. 361-372, Oct. 1978

[27] E. W. Dijkstra, "Co-operating Sequential Processes." Technological University, Eindhoven, The Netherlands, 1965. (reprinted in *Programming Languages*, F. Genuys, ed., Academic Press, New York, New York, 1968).

Appendix 1

Sample LISP Programs

```
;
;
;
;The following function definitions are designed to implement
;the lessthan or equal to predicate function
(defun <= (p q)
      (prog (stack top)
            (setq top (list (caar p) (caar q)))
            (setq stack (cons top stack))
            (return (=<))))
;
;
;
(defun =< ()
     (prog (result p0 q0)
           (setq top (car stack))
           (setq p0 (car top))
           (setq q0 (cadr top))
           (cond ((not (subset (eset q0 q) (eset p0 p)))
                  (setq stack (cdr stack))
                  (return nil)))
           (setq result (=<$p0 (epset q0 q)))
           (cond ((not result)(setq stack (cdr stack))))
           (return result)))
;
;
;
(defun =<$(p0 epsetq0)
      (cond ((null epsetq0) t)
            ((=<$$(epset p0 p) (car epsetq0))
             (=<$p0 (cdr epsetq0)))))
;
;
;
(defun =<$$(epsetp0 epairq)
      (cond ((null epsetp0) nil)
            ((=<$$$(car epsetp0) epairq) t)
            ((=<$$(cdr epsetp0) epairq))))
;
;
;
(defun =<$$$(epairp epairq)
      (prog ()
            (cond ((not (eq (car epairp) (car epairq)))
                   (return nil)))
            (setq top (list (cadr epairp) (cadr epairq)))
            (cond ((member top stack) (return t))
                  (t (setq stack (cons top stack))
                     (return (=<))))))
```