

Department of Applied Analysis  
and Computer Science

January 1973

CS-73-01

PROVING ASSERTIONS ABOUT

PARALLEL PROGRAMS

by


E. A. Ashcroft

**Faculty of Mathematics**  
**University of Waterloo**  
**Waterloo, Ontario**  
**Canada**



**Department of Applied Analysis**  
**&**  
**Computer Science**

Department of Applied Analysis  
and Computer Science

  
January 1973

CS-73-01

PROVING ASSERTIONS ABOUT

PARALLEL PROGRAMS

by

E. A. Ashcroft

## PROVING ASSERTIONS ABOUT PARALLEL PROGRAMS

E. A. Ashcroft\*  
Computer Science  
University of Waterloo  
Waterloo, Canada

### Abstract

A simple but general parallel programming language is considered. The semantics of programs are defined in a concise and natural way using relations. 'Correctness conditions' derived from the semantic definitions enable Floyd's method of proving correctness to be applied to the parallel programs. Proofs of properties of programs using the correctness conditions are claimed to be more systematic versions of the informal arguments normally used to check parallel programs. A program simulating an elementary airline reservation system is given, and several properties of the program are demonstrated using the technique.

---

\* This research was supported by the National Research Council of Canada.

## INTRODUCTION

The method of Floyd [3, see also 6] for proving assertions about programs has not yet become a widely used technique. Part of the reason for this may be that the method requires a discipline of thought that programmers may find both unfamiliar and unnecessary. For the simple programs on which the method is usually demonstrated, a sceptical programmer could maintain that he finds the proofs harder to understand than the original programs. This argument could be countered by providing him with a well-designed, interactive, proof-checking, program-proving system, but for simple programs this appears to be taking a sledgehammer to crack a walnut.

The situation changes however when we consider programs in languages of greater semantic complexity. (Complex programs in simple languages probably just result in more complex proofs, and the situation remains essentially the same.) Recursive programming for example requires a more 'inductive' type of reasoning than does iterative programming, and producing correctness proofs requires a little clerical effort rather than a mental leap [see 7].

Parallel programming also requires a different type of reasoning from sequential programming; one can rely much less on the 'obvious'. Since such programs include operating systems, airline reservation systems and the like, it is crucial that this reasoning be correct. Parallel programs are difficult to debug, and the 'proof by test cases' approach is even more unreliable than for sequential programs. The programmer's confidence in his program has to come from very careful analysis of possible situations.

Any proof method that ensures that he has considered all eventualities can only be helpful (provided it is not impossibly tedious).

The adaptation of the correctness method that we present here results in proofs that are not unnecessarily tedious, and that follow the sort of reasoning that the programmer would ordinarily make.

Although we shall use the term 'correctness method', it is to be understood that the method simply allows one to prove that a program (system) has particular properties. Whether these properties are sufficient to pronounce the system correct or not is outside the scope of this paper. The whole question of what it means for a system to be correct is considered elsewhere [1].

#### PARALLEL PROGRAMS

We wish to keep our parallel programming model as general as possible. All we require is that the model allows a bounded number of computations to be taking place quasi-simultaneously, i.e. the computations can proceed more or less independently but there is some basic level at which no two operations can occur actually simultaneously (or if they do then the effect must be the same as if one preceded the other). Any sort of synchronisation or roadblocking is allowed, and new parallel computations can be initiated and old ones terminated.

However we specify the syntax of such programs, by means of constructs like 'fork' and 'join' or coroutines or tasking, the specification of the semantics must include non-determinism. An easy way is

in terms of non-deterministic programs, and one is tempted to take non-deterministic programs as the model, and relegate the usual sorts of parallel programs to the status of convenient syntactic descriptions that are possible in special cases. This was the approach taken in Karp and Miller [5] and to some extent in Ashcroft and Manna [2]. (In the latter a conventional parallel programming language was considered that used forks and joins, but properties of such programs had to be proved indirectly by way of the corresponding non-deterministic programs.)

Although quite general, the representation of a parallel program by a non-deterministic program has several drawbacks. In particular, the size of the non-deterministic program is some exponential function of the size of the original parallel program. Besides being cumbersome, this means that proofs of correctness tend to be very long. In fact, they are often longer than they need be by an exponential factor. This is because the various quasi-simultaneous computations are usually designed by the programmer to work despite their interactions, not because of the interactions. In other words, enough limitations are imposed, in the form of critical sections and synchronisations, to ensure that each computation can be considered largely independently of the others. Proving  $n$  parallel computations correct should then be roughly linear in  $n$  rather than exponential, as would be the case with the corresponding non-deterministic program.

In this paper we shall specify the syntax of parallel programs using simple fork and join constructions. Synchronisation and critical section features can be introduced by means of constraints on the

execution of statements in particular situations. We shall define the semantics of each parallel program in a natural way in terms of relations. The specification of these relations is directly linked to the program itself and is just as concise. The formula used for proving properties of the program is directly obtained from these relations.

### Syntax of Parallel Programs

A parallel program  $P$  consists of

- i) a domain  $M_P$  of memory-states. Intuitively, each  $m \in M_P$  contains values for all the variables used by the program.
- ii) a set of basic operations  $G_P = \{g_1, g_2, g_3, \dots\}$  where  $g_i : M_P \rightarrow M_P$ .
- iii) a set of basic tests  $Q_P = \{q_1, q_2, q_3, \dots\}$  where each  $q_i \subseteq M_P$  (a relation on  $M_P$ ).
- iv) a finite set  $L_P = \{L_1, L_2, \dots\}$  of symbols called labels.
- v) a finite set  $S_P = \{s_1, s_2, \dots\}$  of labelled statements, using  $G_P$  and  $Q_P$ , with  $s_1$  being the initial statement.
- vi) a set of constraints stops<sub>P</sub>.

A labelled statement can be of one of the following six forms:

- i)  $L_i : \text{do } g_j \text{ then go to } L_k$  (operation)
- ii)  $L_i : \text{if } q_j \text{ then go to } L_k \text{ else go to } L_h$  (test)
- iii)  $L_i : \text{go to } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}$  (fork)
- iv)  $L_{\beta_1}, L_{\beta_2}, \dots, L_{\beta_i} : \text{go to } L_j$  (join)
- v)  $L_i : \text{go to one of } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}$  (branch)

vi)  $L_i : \text{HALT}$  (halt)

where  $L_i, L_j, L_k, L_h, L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}, L_{\beta_1}, \dots, L_{\beta_i}$  are labels.

Each label in  $L_p$  must occur exactly once labelling some statement in  $S_p$ , (i.e. before the colon). The initial statement  $s_1$  is labelled with  $L_1$ .

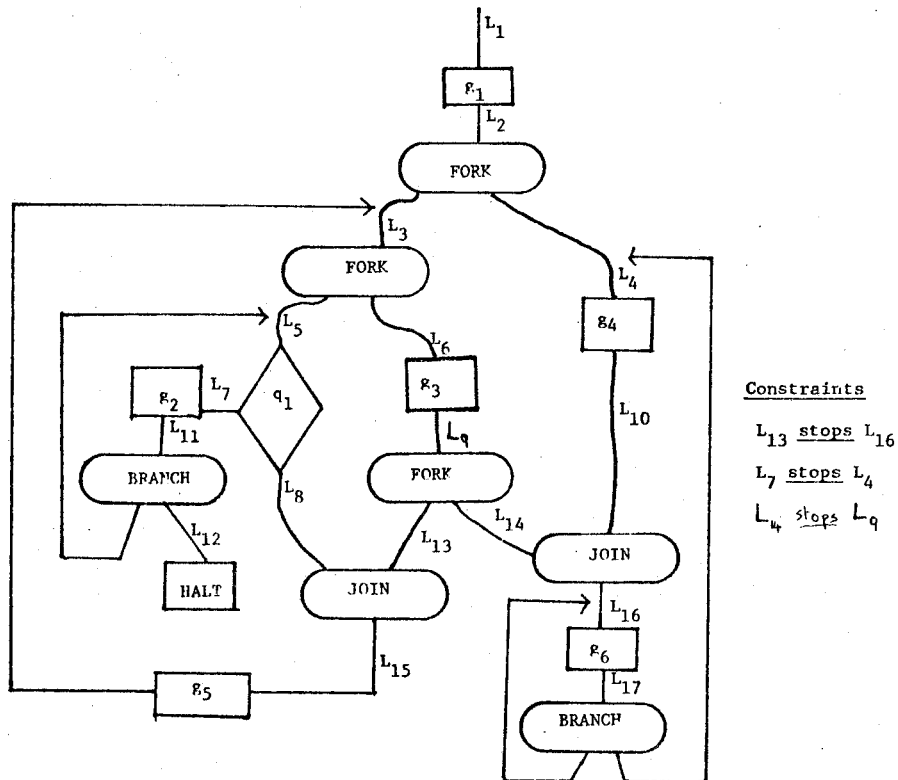
A constraint is an element  $\langle c, \ell \rangle$  of  $2^{L_p} \times L_p$  such that  $\ell \notin c$ .

We consider stops<sub>p</sub> as a relation and say  $c \text{ stops}_p \ell$ . We define

a converse relation allows<sub>p</sub> as follows: for  $c \subseteq L_p, \ell \in L_p$ ,

$c \text{ allows}_p \ell$  if and only if for all  $c' \subseteq c$   $c' \text{ stops}_p \ell$  is false.

Example The following program  $P_1$  is represented diagrammatically, with the labels of statements on the edges leading to those statements.



Program  $P_1$



### Semantics of Parallel Programs

To describe the execution of a program  $P$  we need not only the memory-state at each instant but also the control-state. A control state is a subset of  $L_p$  and represents the statements reached in the program; they are about to be executed at the instant in question. We may describe the execution of  $P$  in the following intuitive way. We think of a control state as a set of markers on the corresponding statements. We start off with a marker on the initial statement  $s_1$ , and are given some initial memory state  $m_0$ . At each step in the process with control state  $c$  and memory state  $m$ , we shall choose some marker  $\ell \in c$  at random such that  $c$  allows <sub>$p$</sub>   $\ell$ . If  $\ell$  marks an operation, then that statement is executed,  $m$  changes to  $g_i(m)$  for some  $g_i$ , and the marker is moved to the next statement, changing  $c$ . If  $\ell$  marks a test, then the marker moves to the appropriate next statement, updating  $c$  but leaving  $m$  unchanged. If  $\ell$  marks a fork, then the marker splits into several markers which are moved to the statements referred to in the fork. If  $\ell$  marks a join, and all the labels of the join are marked, then the markers all fuse into one marker which is moved to the statement referred to in the join. (If all the labels are not marked then  $\ell$  cannot be moved at this step and some other marker must be chosen.) If  $\ell$  marks a branch, then the marker is moved to one of the statements referred to in the branch, chosen at random. If  $\ell$  marks a halt, then the marker is removed from  $c$ . In all cases  $m$  is

unchanged but  $c$  is updated.

The process is repeated either indefinitely or until no marker may move, or no markers are left. Since control-states are defined as subsets of  $L_p$  we will consider a program illegal if it allows any statement ever to get two markers. Syntactic restrictions could be imposed to ensure this, the details of which we shall not be concerned with here.

Example Note that program  $P_1$  is a legal program.

The formal definition of the semantics of  $P$  will be in two stages. Firstly we will define a next-state relation  $\underline{s}$  for every statement  $s$  in  $P$ , and then we give a next-state relation becomes<sub>p</sub> for the whole program. To do this we first introduce the following important notation:

Partitioning notation

for  $A \subseteq L_p$  &  $l \in L_p$

$A(l) = A \cup \{l\}$  if  $A \cap \{l\} = \phi$ , undefined otherwise.

for  $A \subseteq L_p$  &  $c \subseteq L_p$

$A(c) = A \cup c$  if  $A \cap c = \phi$ , undefined otherwise.

We will find the partitioning notation very useful to indicate the replacement of a subset of a set by another set.

The relation  $\underline{s}$

for  $c, c' \subseteq L_p$  &  $m, m' \in M_p$ ,  $(c, m)\underline{s}(c', m')$  is defined as follows

- i) operation:  $(c, m) \underline{L_i} : \underline{\text{do } g_j \text{ then go to } L_k} (c', m')$  if  
and only if  $c \underline{\text{allows}}_p L_i$  &  
 $\exists A \subseteq L_p [c = A(L_i) \ \& \ m' = g_j(m) \ \& \ c' = A(L_k)]$
- ii) test:  $(c, m) \underline{L_i} : \underline{\text{if } q_j \text{ then go to } L_k \text{ else go to } L_h} (c', m')$   
if and only if  $c \underline{\text{allows}}_p L_i$  &  
 $\exists A \subseteq L_p [c = A(L_i) \ \& \ m = m' \ \& \ [\text{IF } q_j(m) \ \text{THEN } c' = A(L_k) \ \text{ELSE } c' = A(L_h)]]^*$
- iii) fork:  $(c, m) \underline{L_i} : \underline{\text{go to } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}} (c', m')$  if  
and only if  $c \underline{\text{allows}}_p L_i$  &  
 $\exists A \subseteq L_p [c = A(L_i) \ \& \ m = m' \ \& \ c' = A(\{L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}\})]$
- iv) join:  $(c, m) \underline{L_{\beta_1}, L_{\beta_2}, \dots, L_{\beta_i}} : \underline{\text{go to } L_j} (c', m')$  if  
and only if  
 $\exists A \subseteq L_p [c = A(\{L_{\beta_1}, L_{\beta_2}, \dots, L_{\beta_i}\}) \ \& \ m = m' \ \& \ c' = A(L_j)]$
- v) branch:  $(c, m) \underline{L_i} : \underline{\text{go to one of } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}} (c', m')$   
if and only if  $c \underline{\text{allows}}_p L_i$  &  
 $\exists A \subseteq L_p [c = A(L_i) \ \& \ m = m' \ \& \ (c' = A(L_{\alpha_1}) \ \vee \ c' = A(L_{\alpha_2}) \ \vee \ \dots \ \vee \ c' = A(L_{\alpha_j}))]$
- vi) halt:  $(c, m) \underline{L_i} : \underline{\text{HALT}}(c', m')$  if and only if  
 $\exists A \subseteq L_p [c = A(L_i) \ \& \ m = m' \ \& \ c' = A]$

---

\* IF A THEN B ELSE C  $\equiv (A \ \& \ B) \ \vee \ (\neg A \ \& \ C) \equiv (A \Rightarrow B) \ \& \ (\neg A \Rightarrow C)$

If  $P$  is a legal program the relation  $\underline{s}$  clearly relates the states (elements of  $2^{L_P} \times M_P$ ) before and after execution of  $s$  in the way expected from the previous intuitive description.

The relation  $\text{becomes}_P$

for  $c, c' \subseteq L_P$  and  $m, m' \in M_P$

$(c, m) \text{ becomes}_P (c', m')$  if and only if  $(c, m) \underline{s} (c', m')$

for some  $s \in S_P$ .

Computations of  $P$

for  $m_0 \in M_P$ , we shall use  $P(m_0)$  to denote program  $P$  with initial memory-state  $m_0$ . A computation of  $P(m_0)$  is then a sequence of states  $\alpha_1, \alpha_2, \alpha_3, \dots$  where  $\alpha_1 = (\{L_1\}, m_0)$ , and  $\alpha_i \text{ becomes}_P \alpha_{i+1}$  for  $i = 1, 2, 3, \dots$ . The computation terminates, if at all, with a state  $\alpha_n$  for which  $\alpha_n \text{ becomes}_P \alpha_j$  is false for all  $\alpha_j \in 2^{L_P} \times M_P$ . If the control-state in  $\alpha_n$  is the empty set then we have normal termination, otherwise we have deadlock.

#### CORRECTNESS METHOD FOR PARALLEL PROGRAMS

A valid predicate for a program  $P$  is a relation  $Q$  on  $(M_P \times M_P) \times 2^{L_P}$  such that  $(m_0, m)Q c$  is true whenever a computation  $P(m_0)$  gets to state  $(c, m)$ . Such predicates can express significant facts about programs. We shall develop a condition  $W_P(Q)$  such that any relation  $Q$  satisfying it is a valid predicate for  $P$ .

The correctness method consists of first finding a relation  $R$  expressing desired properties of a program, and then proving the relation is a valid predicate by showing that  $W_P(R)$  is true.

Definitions Let  $\text{yields}_P$  be the reflexive, transitive closure of  $\text{becomes}_P$ . We define  $\text{at}_P \subseteq (M_P \times M_P) \times 2^{L_P}$  as follows:

for  $m_0, m \in M_P$ ,  $c \subseteq L_P$ ,  $(m_0, m) \text{at}_P c$  if and only if  $(\{L_1\}, m_0) \text{yields}_P (c, m)$ .

Let  $C_P = \{c \subseteq L_P \mid (m_0, m) \text{at}_P c \text{ for some } m_0, m \in M_P\}$ .

This is the set of possible control states for  $P$ .

A valid predicate for  $P$  is a relation  $Q \subseteq (M_P \times M_P) \times C_P$  such that  $\text{at}_P \subseteq Q$ .

The Correctness Condition  $W_P(Q)$

The relation  $\text{at}_P$  has the following properties, which follow immediately from its definition:

- (I) for all  $m \in M_P$ ,  $(m, m) \text{at}_P \{L_1\}$
- (II) for all  $m_0, m, m' \in M_P$ , and  $c, c' \in C_P$ , if  $(m_0, m) \text{at}_P c$  then  $[(c, m) \text{becomes}_P (c', m') \Rightarrow (m_0, m') \text{at}_P c']$ .

We can express the above properties as a condition  $W_P(Q)$  on an arbitrary  $Q \subseteq (M_P \times M_P) \times C_P$

$$W_P(Q) \equiv \forall m_0, m, m' \in M_P, \forall c, c' \in C_P, (m_0, m_0) Q \{L_1\}$$

and if  $(m_0, m) Q c$  then

$$[(c, m) \text{becomes}_P (c', m') \Rightarrow (m_0, m') Q c'] .$$

We have noted that if  $Q \equiv \underline{at}_P$  then  $W_P(Q)$ . In general the converse is not true, but we have

Correctness-condition theorem

For any relation  $Q \subseteq (M_P \times M_P) \times C_P$ , if  $W_P(Q)$  then  $\underline{at}_P \subseteq Q$ , i.e.  $Q$  is a valid predicate for  $P$ .

Proof Assume  $\underline{at}_P \not\subseteq Q$ . Then there exist  $m_0, m \in M_P$  &  $c \in C_P$  such that  $(m_0, m) \underline{at}_P c$  and  $(m_0, m)Q c$  is false. From the definition of  $\underline{at}_P$ ,  $(\{L_1\}, m_0) \underline{yields}_P (c, m)$  and by definition of  $\underline{yields}_P$  there is a finite sequence of states  $\alpha_1, \alpha_2, \dots, \alpha_n$  such that  $\alpha_1 = (\{L_1\}, m_0)$ ,  $\alpha_n = (c, m)$  and  $\alpha_i \underline{becomes}_P \alpha_{i+1}$  for  $i = 1, 2, \dots, n-1$ . Let  $\alpha_j = (c', m')$  be the last state in this sequence for which  $(m_0, m')Q c'$ ; such a state exists since  $(m_0, m_0)Q \{L_1\}$ . Since  $j < n$  let  $\alpha_{j+1}$  be  $(c'', m'')$ . From  $\alpha_j \underline{becomes}_P \alpha_{j+1}$  and  $W_P(Q)$  we get  $(m_0, m'')Q c''$ , a contradiction.  $\square$

$W_P(Q)$  can be stated in a more concise and useful form by first defining a relation  $\underline{Q|S} \subseteq (M_P \times M_P) \times C_P$  for each  $s \in S_P$  and  $Q \subseteq (M_P \times M_P) \times C_P$ .

The Relation  $\underline{Q|S}$

for  $m_0, m \in M_P$  and  $c \in C_P$ ,  $(m_0, m)\underline{Q|S} c$  is defined as follows

i) operation:  $(m_o, m)Q|L_i : \underline{\text{do } g_j \text{ then go to } L_k} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(L_i) \ \& \ c \text{ allows}_p L_i \Rightarrow (m_o, g_j(m))Q A(L_k)]$$

ii) test:  $(m_o, m)Q|L_i : \underline{\text{if } q_j \text{ then go to } L_k \text{ else go to } L_h} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(L_i) \ \& \ c \text{ allows}_p L_i \Rightarrow \text{IF } q_j(m) \text{ THEN } (m_o, m)Q A(L_k) \\ \text{ELSE } (m_o, m)Q A(L_h)]$$

iii) fork:  $(m_o, m)Q|L_i : \underline{\text{go to } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(L_i) \ \& \ c \text{ allows}_p L_i \Rightarrow (m_o, m)Q A(\{L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}\})]$$

iv) join:  $(m_o, m)Q|L_{\beta_1}, L_{\beta_2}, \dots, L_{\beta_i} : \underline{\text{go to } L_j} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(\{L_{\beta_1}, L_{\beta_2}, \dots, L_{\beta_i}\}) \Rightarrow (m_o, m)Q A(L_j)]$$

v) branch:  $(m_o, m)Q|L_i : \underline{\text{go to one of } L_{\alpha_1}, L_{\alpha_2}, \dots, L_{\alpha_j}} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(L_i) \ \& \ c \text{ allows}_p L_i \Rightarrow (m_o, m)Q A(L_{\alpha_1}) \ \& \\ \& (m_o, m)Q A(L_{\alpha_2}) \ \& \dots \\ \& (m_o, m)Q A(L_{\alpha_j})]$$

vi) halt:  $(m_o, m)Q|L_i : \underline{\text{HALT}} \text{ c}$  if and only if

$$\forall A \subseteq L_p [c = A(L_i) \Rightarrow (m_o, m)Q A]$$

It follows immediately from the definitions that for legal program  $P$  and  $m_0, m, m' \in M_P$  and  $c, c' \in C_P$

$$\forall s \in S_P(m_0, m)Q|s c \Leftrightarrow \forall s \in S_P[(c, m) \underline{s} (c', m') \Rightarrow (m_0, m)Q c'] \Leftrightarrow [(c, m) \underline{\text{becomes}}_P (c', m') \Rightarrow (m_0, m)Q c']$$

We can thus state  $W_P(Q)$  as follows

$$W_P(Q) \equiv \forall m_0, m \in M_P, \forall c \in C_P \\ (m_0, m_0)Q \{L_1\} \ \& \\ [(m_0, m)Q c \Rightarrow \forall s \in S_P, (m_0, m)Q|s c]$$

In this form we shall call  $W_P(Q)$  the correctness condition for  $P$ .

#### Using the Correctness Method

To prove that a relation  $R$  is a valid preicate for  $P$  one must check that  $W_P(R)$  is true. In general this involves checking that

$$(m_0, m)R c \Rightarrow (m_0, m)R|s c, \text{ for every } s \in S_P$$

and  $c \in C_P$ . If each of these control-states must be checked individually, we are no better off than if we had defined the semantics of  $P$  using a non-deterministic program and tried to prove it correct; the non-deterministic program would have  $|C_P|$  statements. In practice however, no one could write parallel programs if they had to think of each  $c \in C_P$  individually. One essentially considers a relatively small number of cases by partitioning  $C_P$ , using such reasoning as 'if I am at this statement then such and such is true, no matter where else I am



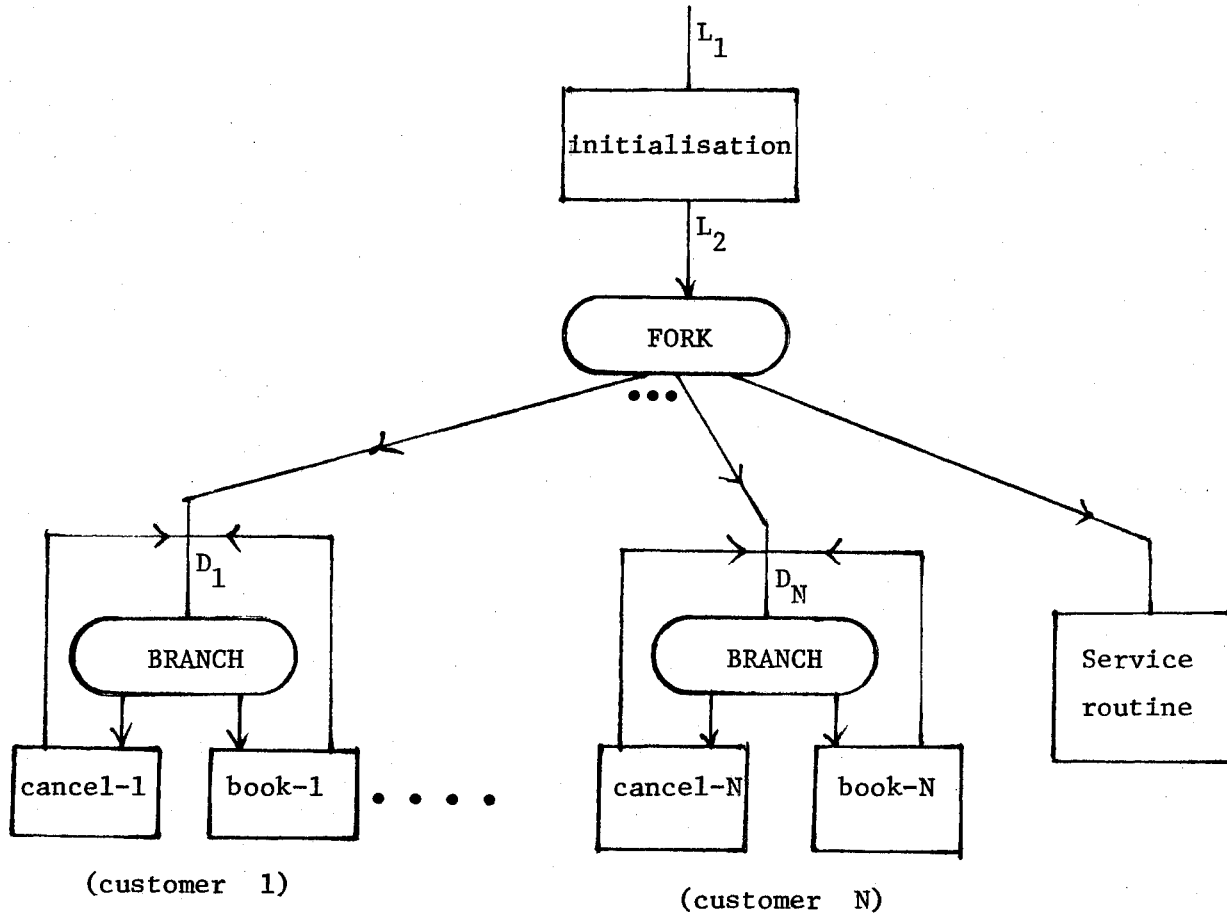
in the program.' Exactly this type of reasoning can be used in checking correctness conditions, which makes the method feasible. The most convincing demonstration of this is probably by an example.

EXAMPLE: AN ELEMENTARY AIRLINE RESERVATION SYSTEM

We shall consider a program P' which simulates a simple airline reservation system for one flight with up to K passengers. Orders to book and unbook customers would normally be received by the system from remote terminals in travel agents' offices. We do not intend to describe this aspect of the system, and instead will simply simulate the 'kernel' of the system where routines for booking and unbooking customers are running in parallel, and are called in random fashion.

There is no subroutine feature in our language, so we will use a different pair of routines for each customer, and postulate a maximum number N of customers. These routines for each customer are simply those for booking and unbooking him, and they will be run repeatedly, in random order, with arbitrary delays at any time (since the language makes no assumptions about the running rates of separate computations). The N customers are run in parallel, together with a service routine to handle the waiting-list, transferring customers from the waiting-list to the flight-list when other customers cancel their bookings.

For convenience we shall number the customers from 1 to N and the outline of the system is then as follows



Program P'

The program maintains two lists of integers, in variables L and W, representing the flight-list and the waiting-list respectively. Before giving details of the subprograms, we will define the basic operations on lists that we shall use.

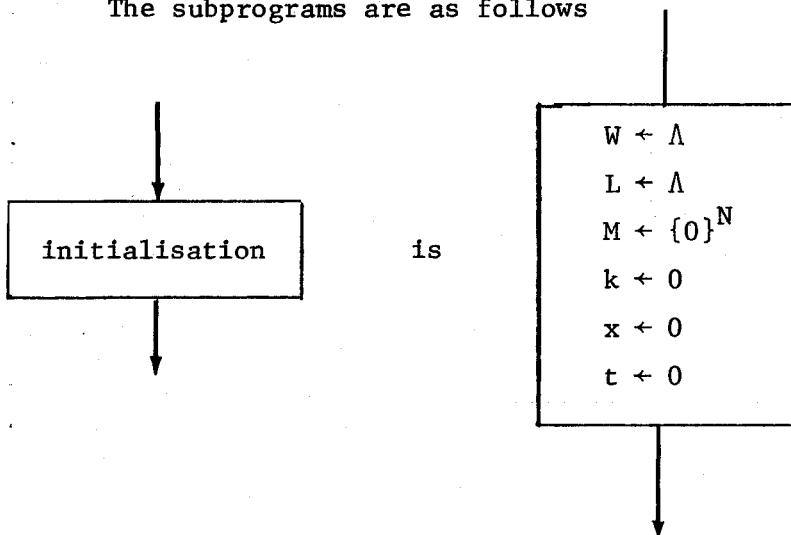
Let J be the set of non-negative integers. For  $x \in J^*$  and  $j \in J$ ,  $\circ$  denotes concatenation and

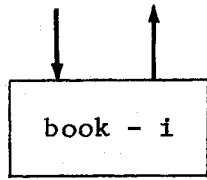
- i)  $j \in x$  if  $x = \alpha \circ j \circ \beta$  for some  $\alpha, \beta \in J^*$
- ii)  $\text{add}(j, x) = x \circ j$
- iii)  $\text{remove}(j, x)$  is defined if  $j \in x$  and  $\text{remove}(j, x) = \alpha \circ \beta$  where  $x = \alpha \circ j \circ \beta$  and  $j \notin \alpha$ .
- iv)  $\text{top}(x)$  is defined if  $x \neq \Lambda$ , and  $\text{top}(x) = j$  where  $x = j \circ \alpha$
- v)  $\text{pop}(x)$  is defined if  $x \neq \Lambda$ , and  $\text{pop}(x) = \alpha$  where  $x = j \circ \alpha$

$\Lambda$  denotes the empty list.

The program also uses a length  $N$  array  $M$  of integers which are to be interpreted as codes for messages sent by the system to the customers. The assignment  $M(j) \leftarrow k$  replaces the  $j$ 'th element of  $M$  by  $k$ , in the usual way.

The subprograms are as follows

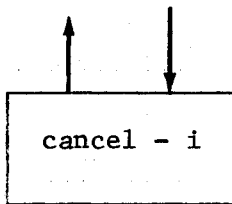
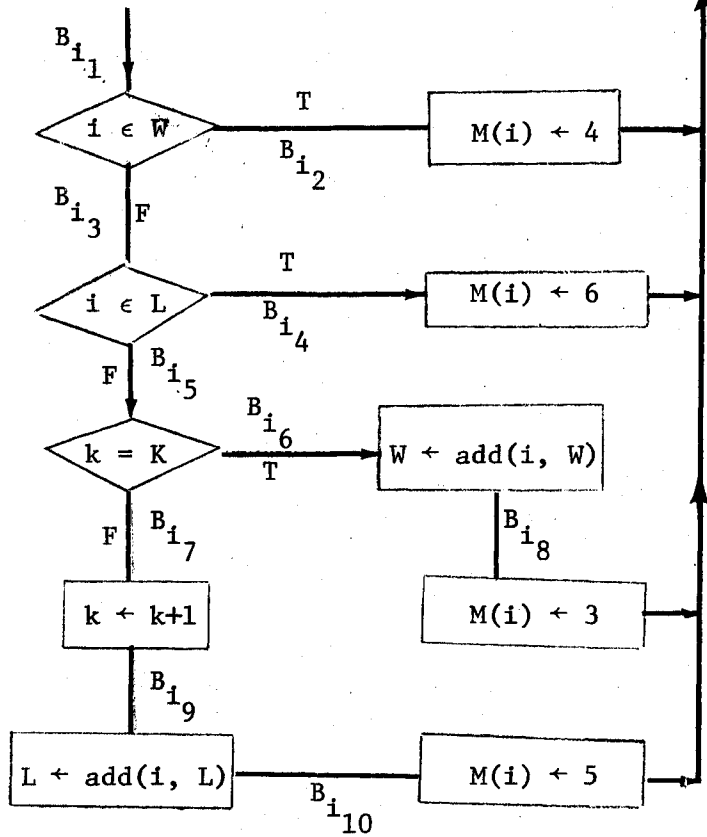




is

Constraints

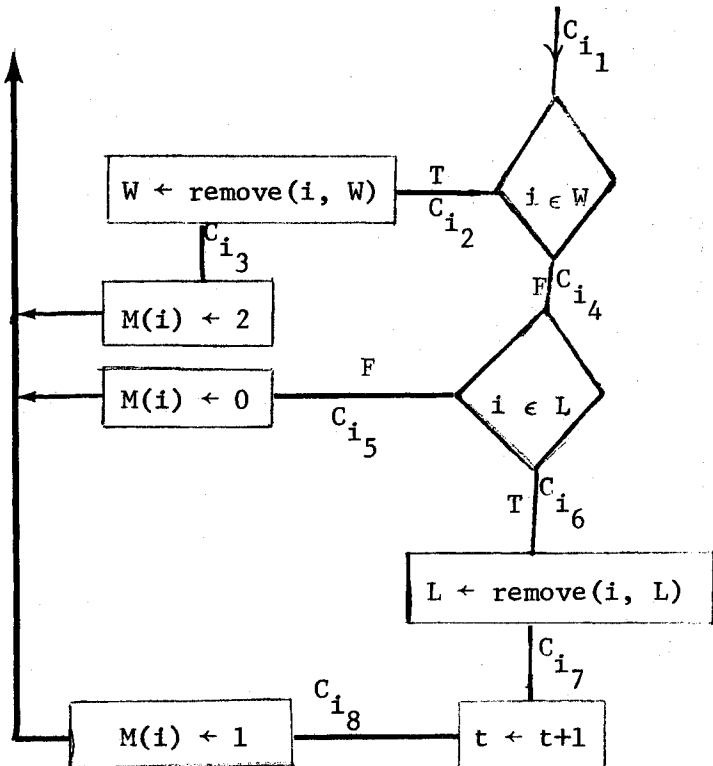
- $B_{i_7}$  stops  $B_{j_5}$   $i \neq j$
- $B_{i_2}$  stops  $T_9$
- $B_{i_6}$  stops  $T_2$
- $B_{i_8}$  stops  $T_9$

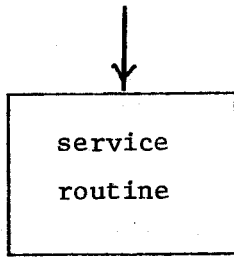


is

Constraints

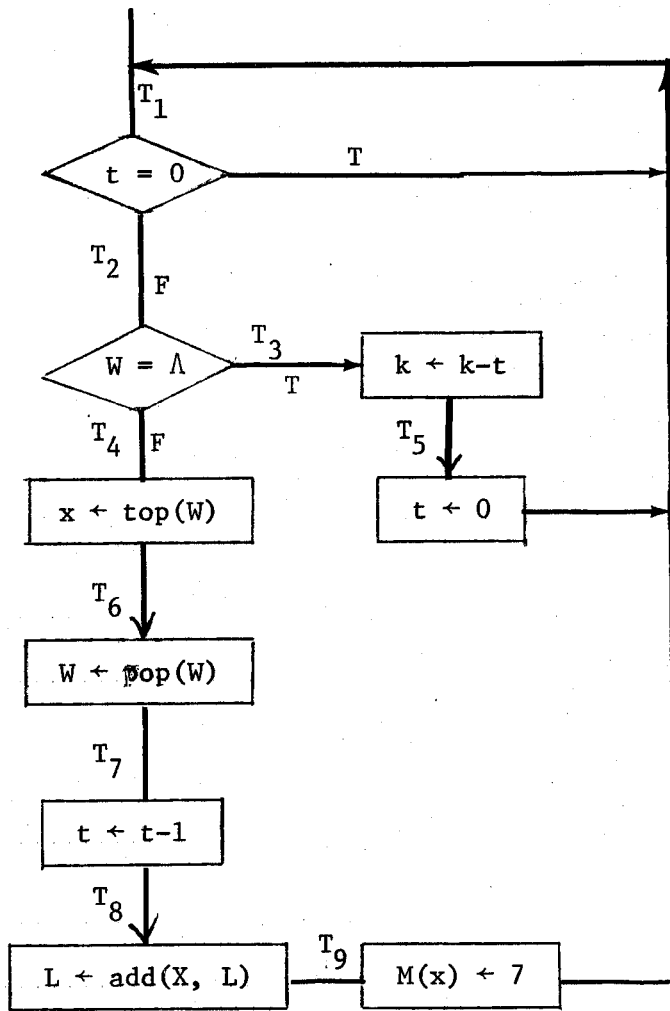
- $C_{i_2}$  stops  $T_2$





Constraints  
for  $1 \leq i \leq N$

- $T_4$  stops  $C_{i_1}$
- $T_6$  stops  $C_{i_1}$
- $T_7$  stops  $B_{i_3}$
- $T_8$  stops  $B_{i_3}$
- $T_7$  stops  $C_{i_4}$
- $T_8$  stops  $C_{i_4}$
- $T_3$  stops  $B_{i_5}$
- $T_5$  stops  $C_{i_7}$
- $T_9$  stops  $C_{i_8}$
- $T_9$  stops  $B_{i_4}$



The message codes should be read as follows

- 0 'not listed'
- 1 'booking cancelled'
- 2 'cancelled from waiting list'
- 3 'wait-listed'
- 4 'already on waiting list'
- 5 'booked'
- 6 'already booked'
- 7 'transferred from waiting to booked'

The routines are straightforward except for use of the variable  $t$ . This is used to indicate to the service routine how many customers should be transferred from the waiting list to the flight list. (The service routine could just keep topping up the flight list when seats became available, but then it would be possible for a customer booking at the appropriate instant to circumvent the waiting list and get booked directly.)

The constraints ensure that the program works correctly; they will all be used in proofs of valid predicates. The constraints  $B_{i_7}$  stops  $B_{j_5}$ ,  $B_{i_6}$  stops  $T_2$  and  $T_3$  stops  $B_{i_5}$  will be used indirectly; they simply ensure that the control states  $A(\{B_{i_7}, B_{j_7}\})$  ( $i \neq j$ ) and  $A(\{B_{i_6}, T_3\})$  are not possible for any  $A \subseteq L_p$ , (i.e. are not members of  $C_p$ ). This is easily seen by considering the possible previous control states. Let  $A(\{B_{i_6}, T_3\})$  be the first control state of this type in a computation. The previous control states can only have been  $A(\{B_{i_5}, T_3\})$  or  $A(\{B_{i_6}, T_2\})$ , and from both of these the transitions are not allowed. Hence  $A(\{B_{i_6}, T_3\})$  does not occur.

#### The Correctness Condition $W_p(Q)$

For readability we will specify  $W_p(Q)$  using at for  $Q$ .

Each  $m \in M_p$  consists of a vector of values for the variables  $L, W, M, k, x$  and  $t$ . The assignment statements in  $P'$  each change only one of these values. For brevity the operation (mapping  $M_p$  into  $M_p$ ) associated with each assignment statement  $a$  we shall denote by  $[a]$ .

This notation is concise and expresses just the relevant information: what has been changed. For example if  $m = \langle \alpha, \beta, \gamma, \delta, \epsilon, \psi \rangle$  (values for  $L, W, M, k, x$  and  $t$  respectively) then

$$[W \leftarrow \text{remove}(i, W)](m) = \langle \alpha, \text{remove}(i, \beta), \gamma, \delta, \epsilon, \psi \rangle .$$

We shall also use this notation for tests, e.g. for the same  $m$  as above  $[k = K](m)$  is true if and only if  $\delta = K$ .

Since all the variables are initialised at the beginning of the program we can drop the initial-memory state  $m_0$  throughout  $W_{P'}(Q)$ . The statements of  $P'$  are naturally partitioned into subprograms: we shall similarly partition  $W_{P'}(Q)$  (and indicate in parentheses which statement we are considering):

$$W_{P'}(Q) \equiv W_{P'_I}(Q) \ \& \ W_{P'_1}(Q) \ \& \ W_{P'_2}(Q) \ \& \ \dots \ \& \ W_{P'_N}(Q) \ \& \ W_{P'_T}(Q)$$

where

(initialisation)

$$\begin{aligned} W_{P'_I}(\underline{\text{at}}) \text{ is } \forall m \in M_{P'}, m \underline{\text{at}} \{L_1\} \quad \text{and} \\ m \underline{\text{at}} \{L_1\} \Rightarrow \langle \Lambda, \Lambda, \{0\}^N, 0, 0, 0 \rangle \underline{\text{at}} \{L_2\} \\ \& \ m \underline{\text{at}} \{L_2\} \Rightarrow m \underline{\text{at}} \{D_1, D_2, \dots, D_N, T_1\} \end{aligned}$$

(book-i and cancel-i)

$W_{P_i}$  (at) is

$$\forall m \in M_{P_i}, \forall c \in C_{P_i}, \forall A \subseteq L_{P_i},$$

$$m \text{ at } c \Rightarrow$$

$$\left[ \begin{aligned} & (D_i) \ c = A(D_i) \Rightarrow m \text{ at } A(B_{i_1}) \ \& \ m \text{ at } A(C_{i_1}) \\ & \&(B_{i_1}) \ c = A(B_{i_1}) \Rightarrow \text{IF } [i \in W](m) \ \text{THEN } m \text{ at } A(B_{i_2}) \ \text{ELSE } m \text{ at } A(B_{i_3}) \\ & \&(B_{i_2}) \ c = A(B_{i_2}) \Rightarrow [M(i) \leftarrow 4](m) \ \text{at } A(D_i) \\ & \&(B_{i_3}) \ c = A(B_{i_3}) \ \& \ T_7 \not\vdash c \ \& \ T_8 \not\vdash c \Rightarrow \text{IF } [i \in L](m) \ \text{THEN } m \text{ at } A(B_{i_4}) \\ & \hspace{20em} \text{ELSE } m \text{ at } A(B_{i_5}) \\ & \&(B_{i_4}) \ c = A(B_{i_4}) \ \& \ T_9 \not\vdash c \Rightarrow [M(i) \leftarrow 6](m) \ \text{at } A(D_i) \\ & \&(B_{i_5}) \ c = A(B_{i_5}) \ \& \ (1 \leq j \leq N \Rightarrow B_{i_7} \not\vdash c) \ \& \ T_3 \not\vdash c \Rightarrow \text{IF } [k = K](m) \\ & \hspace{10em} \text{THEN } m \text{ at } A(B_{i_6}) \ \text{ELSE } m \text{ at } A(B_{i_7}) \\ & \&(B_{i_6}) \ c = A(B_{i_6}) \Rightarrow [W \leftarrow \text{add}(i, W)](m) \ \text{at } A(B_{i_8}) \\ & \&(B_{i_7}) \ c = A(B_{i_7}) \Rightarrow [k \leftarrow k+1](m) \ \text{at } A(B_{i_9}) \\ & \&(B_{i_8}) \ c = A(B_{i_8}) \Rightarrow [M(i) \leftarrow 3](m) \ \text{at } D(D_i) \\ & \&(B_{i_9}) \ c = A(B_{i_9}) \Rightarrow [L \leftarrow \text{add}(i, L)](m) \ \text{at } A(B_{i_{10}}) \\ & \&(B_{i_{10}}) \ c = A(B_{i_{10}}) \Rightarrow [M(i) \leftarrow 5](m) \ \text{at } A(D_i) \\ & \&(C_{i_1}) \ c = A(C_{i_1}) \ \& \ T_4 \not\vdash c \ \& \ T_6 \not\vdash c \Rightarrow \text{IF } [i \in W](m) \ \text{THEN} \\ & \hspace{10em} m \text{ at } A(C_{i_2}) \ \text{ELSE } m \text{ at } A(C_{i_4}) \\ & \&(C_{i_2}) \ c = A(C_{i_2}) \Rightarrow [W \leftarrow \text{remove}(i, W)](m) \ \text{at } A(C_{i_3}) \\ & \&(C_{i_3}) \ c = A(C_{i_3}) \Rightarrow [M(i) \leftarrow 2](m) \ \text{at } A(D_i) \\ & \&(C_{i_4}) \ c = A(C_{i_4}) \ \& \ T_7 \not\vdash c \ \& \ T_8 \not\vdash c \Rightarrow \text{IF } [i \in L](m) \ \text{THEN } m \text{ at } A(C_{i_6}) \\ & \hspace{20em} \text{ELSE } m \text{ at } A(C_{i_5}) \\ & \&(C_{i_5}) \ c = A(C_{i_5}) \Rightarrow [M(i) \leftarrow 0](m) \ \text{at } A(D_i) \\ & \&(C_{i_6}) \ c = A(C_{i_6}) \Rightarrow [L \leftarrow \text{remove}(i, L)](m) \ \text{at } A(C_{i_7}) \\ & \&(C_{i_7}) \ c = A(C_{i_7}) \ \& \ T_5 \not\vdash c \Rightarrow [t \leftarrow t+1](m) \ \text{at } A(C_{i_8}) \\ & \&(C_{i_8}) \ c = A(C_{i_8}) \ \& \ T_9 \not\vdash c \Rightarrow [M(i) \leftarrow 1](m) \ \text{at } A(D_i) \end{aligned} \right]$$



(service routine)

$W_P, (\underline{\text{at}})$  is  $\forall m \in M_P, \forall c \in C_P, \forall A \subseteq L_P,$

$m \underline{\text{at}} c \Rightarrow$

$\left[ (T_1) c = A(T_1) \Rightarrow \text{IF } [t = 0](m) \text{ THEN } m \underline{\text{at}} c \text{ ELSE } m \underline{\text{at}} A(T_2) \right.$

$\&(T_2) c = A(T_2) \& (1 \leq i \leq N \Rightarrow C_{i_2} \not\vdash c \& B_{i_6} \not\vdash c) \Rightarrow \text{IF } [W = \Lambda](m)$

$\text{THEN } m \underline{\text{at}} A(T_3) \text{ ELSE } m \underline{\text{at}} A(T_4)$

$\&(T_3) c = A(T_3) \Rightarrow [k \leftarrow k-t](m) \underline{\text{at}} A(T_5)$

$\&(T_4) c = A(T_4) \Rightarrow [x \leftarrow \text{top}(W)](m) \underline{\text{at}} A(T_6)$

$\&(T_5) c = A(T_5) \Rightarrow [t \leftarrow 0](m) \underline{\text{at}} A(T_1)$

$\&(T_6) c = A(T_6) \Rightarrow [W \leftarrow \text{pop}(W)](m) \underline{\text{at}} A(T_7)$

$\&(T_7) c = A(T_7) \Rightarrow [t \leftarrow t-1](m) \underline{\text{at}} A(T_8)$

$\&(T_8) c = A(T_8) \Rightarrow [L \leftarrow \text{add}(x, L)](m) \underline{\text{at}} A(T_9)$

$\&(T_9) c = A(T_9) \& (1 \leq i \leq N \Rightarrow B_{i_2} \not\vdash c \& B_{i_8} \not\vdash c) \Rightarrow [M(x) \leftarrow 7](m) \underline{\text{at}} A(T_1) \left. \right]$

### Valid Predicates and Their Proofs

We will first show (A) that the flight never gets overbooked and that the flight is fullybooked whenever there is a waiting list.

Then we will show (B) that for every customer, the message he last received always corresponds to his present booking status (except for those situations where his status has just changed and a new message is about to be sent).

#### (A) Numbers of Passengers Booked

It is apparent from the program that  $k$  corresponds to the number of passengers booked, provided the service routine has 'caught up'

with the pending transfers to the flight-list. If we denote the length of  $L$  by  $|L|$ , we should expect that  $|L| + t = k$ . However we must allow for those situations where for example  $k$  has been incremented but the newly booked customer has not yet been put on  $L$ . These situations occur at  $B_{i_9}$ ,  $C_{i_7}$  and  $T_8$ . To account for these situations we must keep a count of them:

$$\text{for } c \in C_P, \text{ let } \text{temp}(c) = |\{\ell \in c \mid \ell = C_{i_7} \vee \ell = B_{i_9} \vee \ell = T_8, \\ 1 \leq i \leq N\}|$$

We can now give the relation  $R$  which we shall show is a valid predicate.

The relation  $R$

For  $m = \langle L, W, M, k, x, t \rangle \in M_{P_1}$ ,  $m R \{L_1\}$  is true and for  $c \in C_P$ ,  $c \notin \{L_1\}$ ,  $\langle L, W, M, k, x, t \rangle R c$  if and only if all the following are true:

- 1)  $(k < K \ \& \ W = \Lambda) \vee k = K$
- 2)  $T_5 \notin c \Rightarrow |L| + t + \text{temp}(c) = k$
- 3)  $T_5 \in c \Rightarrow |L| + \text{temp}(c) = k$
- 4)  $T_3 \in c \Rightarrow W = \Lambda$
- 5)  $B_{i_6} \in c \Rightarrow k = K \quad (1 \leq i \leq N)$
- 6)  $B_{i_7} \in c \Rightarrow k < K \quad (1 \leq i \leq N)$

To show that  $R$  is a valid predicate we shall show that  $W_{P_I}(R)$ ,  $W_{P_i}(R)$  ( $1 \leq i \leq N$ ) and  $W_{P_T}(R)$ . We can easily verify that  $W_{P_I}(R)$  is true. For the rest, notice that we only need check those parts of the correctness condition corresponding to state-transitions which

can make  $R$  false, i.e. the transitions which change  $k$  or  $t$  or  $L$ , add elements to  $W$ , go to labels  $T_3, T_5, C_{i_7}, B_{i_9}, B_{i_6}, B_{i_7}$  or  $T_8$ , or from labels  $T_8, C_{i_7}$  or  $B_{i_9}$ . We check all these transitions informally below.

changing  $k$  :

$$(B_{i_7}) \quad mRc \ \& \ c = A(B_{i_7}) \Rightarrow [k \leftarrow k+1](m) \ R \ A(B_{i_9})$$

$k$  is increased by 1, but temp is also increased by 1.

$\therefore$  2) & 3) remain true. Since  $B_{i_7} \in c$ ,  $k$  was less than  $K$ , and after being incremented still satisfies 1). All the conditions in  $R$  remain true.

$$(T_3) \quad mRc \ \& \ c = A(T_3) \Rightarrow [k \leftarrow k-t](m) \ R \ A(T_5)$$

$k$  is decreased by  $t$ , but we switch from  $c = A(T_3)$ , which does not contain  $T_5$ , to  $A(T_5)$  which does, so 2) & 3) remain true.

$B_{i_6} \notin c$  since  $A'(\{B_{i_6}, T_3\})$  is an impossible control state.  $\therefore$  5) is not contradicted. Since 4) implies  $W = \Lambda$ , 1) remains true.

changing  $t$  :

$$(C_{i_7}) \quad mRc \ \& \ c = A(C_{i_7}) \ \& \ T_5 \notin c \Rightarrow [t \leftarrow t+1](m) \ R \ A(C_{i_8})$$

$t$  is increased by 1, but temp is decreased by 1. The

constraint ensures  $T_5 \notin A(C_{i_7})$  and  $T_5 \notin A(C_{i_8})$ , so 3) is still satisfied.

$$(T_7) \quad mRc \ \& \ c = A(T_7) \Rightarrow [t \leftarrow t+1](m) \ R \ A(T_8)$$

$t$  is decreased by 1, but temp is increased by 1.  $T_5 \notin A(T_7)$

and  $T_5 \notin A(T_8)$  for possible control states, so 3) is still satisfied.

changing L :

$(B_{i_9}) \quad m R c \ \& \ c = A(B_{i_9}) \Rightarrow [L \leftarrow \text{add}(i, L)](m) R A(B_{i_{10}})$

$|L|$  is increased by 1, but temp is decreased by 1.

$(C_{i_6}) \quad m R c \ \& \ c = A(C_{i_6}) \Rightarrow [L \leftarrow \text{remove}(i, L)](m) R A(C_{i_7})$

$|L|$  is decreased by 1, but temp is increased by 1.

$(T_8) \quad m R c \ \& \ c = A(T_8) \Rightarrow [L \leftarrow \text{add}(x, L)](m) R A(T_9)$

$|L|$  is increased by 1, but temp is decreased by 1.

adding to W :

$(B_{i_6}) \quad m R c \ \& \ c = A(B_{i_6}) \Rightarrow [W \leftarrow \text{add}(i, W)](m) R A(B_{i_8})$

$W \neq \Lambda$  after the transition but  $k = K$  since  $B_{i_6} \in c$ .

$\therefore$  1) still satisfied.  $T_3 \notin c$  since  $A'(\{B_{i_6}, T_3\})$  is an impossible control state.

Transitions to  $T_3, B_{i_6}, B_{i_7}, T_5, T_8, B_{i_9}$  and  $C_{i_7}$  :

$(T_2) \quad m R c \ \& \ c = A(T_2) \ \& \ \dots \Rightarrow \text{IF } [W = \Lambda](m) \text{ THEN } m R A(T_3) \ . \ \dots$

$W = \Lambda$  is a necessary condition for the transition.  $\therefore$  2)

is satisfied.

$(B_{i_5}) \quad m R c \ \& \ c = A(B_{i_5}) \ \& \ \dots \Rightarrow \text{IF } [k = K](m) \text{ THEN } m R A(B_{i_6})$   
ELSE  $m R A(B_{i_7})$

$k = K$  is necessary for transition to  $B_{i_6}$ ,  $k < K$  is necessary

for transition to  $B_{i_7}$   $\therefore$  5) & 6) remain true.

- (T<sub>3</sub>) already considered
- (T<sub>7</sub>) already considered
- (B<sub>i<sub>7</sub></sub>) already considered
- (C<sub>i<sub>6</sub></sub>) already considered.

Transitions from T<sub>8</sub>, C<sub>i<sub>7</sub></sub> and B<sub>i<sub>9</sub></sub> :

- (T<sub>8</sub>) already considered
- (C<sub>i<sub>7</sub></sub>) already considered
- (B<sub>i<sub>9</sub></sub>) already considered.

We have thus shown that  $W_p(R)$  is true, and hence  $R$  is a valid predicate for  $P'$ . The 'proof' was informal but could be developed formally.

(B) Correctness of Messages M(i)

We would expect that, at all times,

$$\begin{aligned} M(i) \leq 2 &\Leftrightarrow i \notin L \ \& \ i \notin W \\ 2 < M(i) < 5 &\Leftrightarrow i \in W \\ M(i) \geq 5 &\Leftrightarrow i \in L . \end{aligned}$$

Unfortunately there arise situations when this is not true; namely a) if book-i or cancel-i have altered  $L$  or  $W$ , but not yet sent the corresponding message, and b) if the service routine is in the process of transferring  $i$  from  $W$  to  $L$ . To conveniently test for these critical situations we can define the following conditions:

$$\text{crit}_i(c) \equiv c \cap \{B_{i_8}, B_{i_{10}}, C_{i_3}, C_{i_7}, C_{i_8}\} \neq \phi$$

$$\text{crit}_T(c, x) \equiv c \cap \{T_7, T_8, T_9\} \neq \emptyset \quad \& \quad x = i$$

We can now state the predicate  $Q_i$  which we shall prove to be valid.

The relation  $Q_i$

For  $m = \langle L, W, M, k, x, t \rangle \in M_p$ ,  $m Q_i \{L_1\}$  is true and for  $c \in C_p$ ,  $c \neq \{L_1\}$ ,  $\langle L, W, M, k, x, t \rangle Q_i c$  if and only if

1)  $\neg \text{crit}_i(c) \Rightarrow$

$$\text{IF } \neg \text{crit}_T(c, x) \text{ THEN } M(i) \leq 2 \Leftrightarrow i \notin W \ \& \ i \notin L$$

$$M(i) \geq 5 \Leftrightarrow i \in L$$

$$2 < M(i) < 5 \Leftrightarrow i \in W$$

$$\text{ELSE } 2 < M(i) < 5 \ \& \ \text{IF } T_9 \in c$$

$$\text{THEN } i \in L$$

$$\text{ELSE } i \notin L \ \& \ i \notin W$$

2)  $\text{crit}_i(c) \Rightarrow$

$$\text{IF } \neg \text{crit}_T(c, x) \text{ THEN } B_{i_8} \in c \Leftrightarrow i \in W$$

$$B_{i_{10}} \in c \Leftrightarrow i \in L$$

$$\text{ELSE } (B_{i_8} \in c \ \& \ \text{IF } T_9 \in c \ \text{THEN } i \in L$$

$$\text{ELSE } i \notin L \ \& \ i \notin W)$$

$$\vee ((C_{i_7} \in c \vee C_{i_8} \in c) \ \& \ i \notin L \ \& \ i \notin W \ \& \ T_9 \in c)$$

3) (side conditions when  $\neg \text{crit}_i(c)$ )

a)  $B_{i_2} \in c \Rightarrow 2 < M(i) < 5$

b)  $B_{i_3} \in c \vee C_{i_4} \in c \Rightarrow i \notin W$

c)  $B_{i_4} \in c \vee C_{i_6} \in c \Rightarrow i \in L$

$$d) \{B_{i_5}, B_{i_6}, B_{i_7}, B_{i_9}, C_{i_5}\} \cap c \neq \emptyset \Rightarrow M(i) \leq 2$$

$$e) C_{i_2} \in c \Rightarrow i \in W \ \& \ [T_4 \in c \Rightarrow i \neq \text{top}(W)] \ \& \ [T_6 \in c \Rightarrow x \neq i]$$

$$4) T_6 \in c \Rightarrow x = \text{top}(W)$$

$$\left. \begin{array}{l} 5) \ i \in L \Rightarrow i \notin W \ \& \ i \notin \text{remove}(i, L) \\ \ i \in W \Rightarrow i \notin L \ \& \ i \notin \text{remove}(i, W) \end{array} \right\} \text{ (no double-booking)}$$

To show that  $W_{P_i}(Q_i)$  we note first that  $W_{P_i}(Q_i)$  follows immediately, and that for the rest of  $W_{P_i}(Q_i)$  we need only check the parts of  $W_{P_i}(Q_i)$  corresponding to those state transitions that can make  $Q_i$  false. In other words the transitions that add or remove  $i$  to or from  $L$  or  $W$ , change the message category of  $M(i)$ , change  $\text{crit}_T$  to  $\neg \text{crit}_T$  or vice versa, change  $\text{crit}_i$  to  $\neg \text{crit}_i$  or vice versa, go to any of  $\{B_{i_2}, B_{i_3}, B_{i_4}, B_{i_5}, B_{i_6}, B_{i_7}, B_{i_9}, C_{i_4}, C_{i_5}, C_{i_6}\}$ , or go to  $T_4, T_6$  or  $T_9$ . We can first note that no transitions in book- $j$  or cancel- $j$  can do any of these things ( $j \neq i$ ), so  $W_{P_j}(Q_i)$  is certainly satisfied. On the other hand, practically every transition in book- $i$  and cancel- $i$  will have to be considered, and several in the service routine.

Checking  $W_{P_i}(Q_i)$  :

$$(D_i) \ m \ Q_i \ c \ \& \ c = A(D_i) \Rightarrow \ m \ Q_i \ A(B_{i_1}) \ \& \ m \ Q_i \ A(C_{i_1})$$

No effects.

$$(B_{i_1}) \ m \ Q_i \ c \ \& \ c = A(B_{i_1}) \Rightarrow \text{IF } [i \in W](m) \ \text{THEN } \ m \ Q_i \ A(B_{i_2}) \\ \text{ELSE } \ m \ Q_i \ A(B_{i_3})$$

No effect on 1) & 2). 1) and  $i \in W$  imply  $2 < M(i) < 5 \quad \therefore$

3a) satisfied. 3b) is satisfied since  $i \notin W$  necessary for the transition.

$$(B_{i_2}) \quad m Q_i c \quad \& \quad c = A(B_{i_2}) \Rightarrow [M(i) \leftarrow 4](m) Q_i A(D_i)$$

3a) implies no change of message category. No condition is affected.

$$(B_{i_3}) \quad m Q_i c \quad \& \quad c = A(B_{i_3}) \quad \& \quad T_7 \not\vdash c \quad \& \quad T_8 \not\vdash c \Rightarrow \text{IF } [i \in L](m)$$

$$\text{THEN } m Q_i A(B_{i_4})$$

$$\text{ELSE } m Q_i A(B_{i_5})$$

3c) satisfied since  $i \in L$  necessary for transition.  $i \notin L$  & 3b) &  $T_7 \not\vdash c$  &  $T_8 \not\vdash c$  imply from 1) that  $M(i) \leq 2$ , satisfying 3d).

$$(B_{i_4}) \quad m Q_i c \quad \& \quad c = A(B_{i_4}) \quad \& \quad T_9 \not\vdash c \Rightarrow [M(i) \leftarrow 6](m) Q_i A(D_i)$$

1) and 3c) &  $T_9 \not\vdash c$  imply  $M(i) \geq 5$ . There is therefore no change of message category and no condition is affected.

$$(B_{i_5}) \quad m Q_i c \quad \& \quad c = A(B_{i_5}) \quad \& \quad \dots \Rightarrow \text{IF } [k = K](m) \text{ THEN } m Q_i A(B_{i_6})$$

$$\text{ELSE } m Q_i A(B_{i_7})$$

No effect. 3d) remains true.

$$(B_{i_6}) \quad m Q_i c \quad \& \quad c = A(B_{i_6}) \Rightarrow [W \leftarrow \text{add}(i, W)](m) Q_i A(B_{i_8})$$

We move from  $\neg \text{crit}_i$  to  $\text{crit}_i$ . 3d) and 1) imply  $i \notin L$  &  $i \notin W$  before the transition (so 5) is satisfied) and  $\neg \text{crit}_T$ .  $i \in W$  after the transition so 2) is satisfied.

$$(B_{i_7}) \quad m Q_i c \quad \& \quad c = A(B_{i_7}) \Rightarrow [k \leftarrow k+1](m) Q_i A(B_{i_9})$$

No effect. 3d) remains true.



(B<sub>i<sub>8</sub></sub>)  $m Q_i c \ \& \ c = A(B_{i_8}) \Rightarrow [M(i) \leftarrow 3](m) Q_i A(D_i)$

We move from  $\text{crit}_i$  to  $\neg \text{crit}_i$ . If  $\neg \text{crit}_T$  then 2) implies  $i \in W$  after the transition. If  $\text{crit}_T$  then 2) implies IF  $T_9 \in c$  THEN  $i \in L$  ELSE  $i \notin L \ \& \ i \notin W$  after the transition. Since  $M(i) = 3$  after the transition, 1) is satisfied.

(B<sub>i<sub>9</sub></sub>)  $m Q_i c \ \& \ c = A(B_{i_9}) \Rightarrow [L \leftarrow \text{add}(i, L)](m) Q_i A(B_{i_{10}})$

We move from  $\neg \text{crit}_i$  to  $\text{crit}_i$ . 3d) and 1) imply  $i \notin L \ \& \ i \notin W$  (so 5) is satisfied) and  $\neg \text{crit}_T$ .  $i \in L$  after the transition so 2) is satisfied.

(B<sub>i<sub>10</sub></sub>)  $m Q_i c \ \& \ c = A(B_{i_{10}}) \Rightarrow [M(i) \leftarrow 5](m) Q_i A(D_i)$

We move from  $\text{crit}_i$  to  $\neg \text{crit}_i$ . 2) implies  $\neg \text{crit}_T \ \& \ i \in L$ .  $M(i) = 5$  after the transition so 1) is satisfied.

(C<sub>i<sub>1</sub></sub>)  $m Q_i c \ \& \ c = A(C_{i_1}) \ \& \ T_4 \notin c \ \& \ T_6 \notin c \Rightarrow \text{IF } [i \in W](m)$

THEN  $m Q_i A(C_{i_2})$

ELSE  $m Q_i A(C_{i_4})$

3e) is satisfied since  $i \in W$ ,  $T_4 \notin c$ ,  $T_6 \notin c$  are necessary for the transition. 3b) is satisfied since  $i \notin W$  necessary for the transition.

(C<sub>i<sub>2</sub></sub>)  $m Q_i c \ \& \ c = A(C_{i_2}) \Rightarrow [W \leftarrow \text{remove}(i, W)](m) Q_i A(C_{i_2})$

3e) & 5) imply  $i \notin W$ , &  $i \notin L$  after the transition. 1) and

4) imply  $i \neq \text{top}(W)$  before the transition if  $T_6 \in c$ .  $\therefore$  4)

is satisfied after the transition. 3e) & 1) imply  $\neg \text{crit}_T$ .

We move from  $\neg \text{crit}_i$  to  $\text{crit}_i$ . Since we are not at  $B_{i_8}$  or  $B_{i_{10}}$ , 1) is satisfied.

$(C_{i_3}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_3}) \Rightarrow [M(i) \leftarrow 2](m) \ Q_i \ A(D_i)$

We move from  $\text{crit}_i$  to  $\neg \text{crit}_i$ . 2) implies  $\neg \text{crit}_T$  &  $i \notin L$  &  $i \notin W$ . Since  $M(i) = 2$  after the transition, 1) is satisfied.

$(C_{i_4}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_4}) \ \& \ T_7 \notin c \ \& \ T_8 \notin c \Rightarrow \text{IF } [i \in L](m)$   
 $\text{THEN } m \ Q_i \ A(C_{i_6})$   
 $\text{ELSE } m \ Q_i \ A(C_{i_5})$

3c) is satisfied since  $i \in L$  is necessary for the transition.

3b) implies  $i \notin W$ . If  $i \notin L$  and  $T_7 \notin c$  &  $T_8 \notin c$ , 1) implies  $M(i) \leq 2$ , satisfying 3d).

$(C_{i_5}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_5}) \Rightarrow [M(i) \leftarrow 0](m) \ Q_i \ A(D_i)$

3b) implies  $M(i) \leq 2$  before the transition. There is thus no change in message category.

$(C_{i_6}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_6}) \Rightarrow [L \leftarrow \text{remove}(i, L)](m) \ Q_i \ A(C_{i_7})$

3) & 5) imply  $i \notin L$  &  $i \notin W$  after the transition. 3c) & 1) imply that if  $\text{crit}_T$  then  $T_9 \in c$ . We move from  $\neg \text{crit}_i$  to  $\text{crit}_i$  but 2) is still satisfied, whether  $\text{crit}_T$  or not.

$(C_{i_7}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_7}) \ \& \ T_5 \notin c \Rightarrow [t \leftarrow t-1](m) \ Q_i \ A(C_{i_8})$

We stay in  $\text{crit}_i$  2) remains true.

$(C_{i_8}) \quad m \ Q_i \ c \ \& \ c = A(C_{i_8}) \ \& \ T_9 \notin c \Rightarrow [M(i) \leftarrow 1](m) \ Q_i \ A(D_i)$

We move from  $\text{crit}_i$  to  $\neg \text{crit}_i$ . 2) and  $T_9 \notin c$  imply  $\neg \text{crit}_T$

and  $i \notin L$  &  $i \notin W$ . Since  $M(i) = 1$  after the transition,  
 1) is satisfied.

Checking  $W_P, (Q_i)$  :  
<sub>T</sub>

Only the following transitions need be considered

(T<sub>2</sub>)  $m Q_i c$  &  $c = A(T_2)$  &  $[1 \leq j \leq N \Rightarrow C_{j_2} \notin c \dots] \Rightarrow$

IF  $\dots$  ELSE  $m Q_i A(T_4)$

3e) is satisfied since  $C_{i_2} \notin c$ .

(T<sub>4</sub>)  $m Q_i c$  &  $c = A(T_4) \Rightarrow [x \leftarrow \text{top}(W)](m) Q_i A(T_6)$

4) is satisfied immediately. 3e) is satisfied after the transition since before the transition it implies  $C_{i_2} \in c \Rightarrow i \neq \text{top}(W)$ , and therefore  $x \neq i$  after the transition.

(T<sub>6</sub>)  $m Q_i c$  &  $c = A(T_6) \Rightarrow [W \leftarrow \text{pop}(W)] Q_i A(T_7)$

From 4, we move from  $\neg \text{crit}_T$  to  $\text{crit}_T$  only if  $i = \text{top}(W)$ .

If  $\neg \text{crit}_i$ , then 1) implies that  $i \in W$  only if  $2 < M(i) < 5$ .

If  $i = \text{top}(W)$  before the transition,  $i \notin W$  &  $i \notin L$  after the transition and 1) is satisfied. Similarly, if  $\text{crit}_i$  then 2)

implies that  $i \in W$  only if  $B_{i_8} \in C$ . If  $i = \text{top}(W)$  before

the transition,  $i \notin W$  &  $i \notin L$  after the transition and 2) is

satisfied. 3e) is satisfied after the transition because before

the transition 4 and 3e) imply that  $i \in W$  but  $i \neq \text{top}(W)$ .

(T<sub>7</sub>)  $m Q_i c \ \& \ c = A(T_7) \Rightarrow [t \leftarrow t-1](m) Q_i A(T_8)$

No condition is affected.

(T<sub>8</sub>)  $m Q_i c \ \& \ c = A(T_8) \Rightarrow [L \leftarrow \text{add}(x, L)](m) Q_i A(T_9)$

1) and 2) are affected only if  $x = i$ , i.e. if  $\text{crit}_T$ . If  $\neg \text{crit}_i$  then  $2 < M(i) < 5$  &  $i \notin L$  &  $i \notin W$  before the transition. After the transition we are at  $T_9$  and  $i \in L$ .  $\therefore$  1) is satisfied. If  $\text{crit}_i$  then  $B_{i_8} \in c$  &  $i \notin L$  &  $i \notin W$  before the transition and we are at  $T_9$  and  $i \in L$  after the transition.  $\therefore$  2) is satisfied. In both cases 5) is satisfied.

(T<sub>9</sub>)  $m Q_i c \ \& \ c = A(T_9) \ \& \ [1 \leq j \leq N \Rightarrow B_{j_2} \notin c \ \& \ B_{j_8} \notin c] \Rightarrow$

$[M(x) \leftarrow 7](m) Q_i A(T_i)$

$Q_i$  affected only if  $x = i$ , i.e. if  $\text{crit}_T$  before the transition. We then move from  $\text{crit}_T$  to  $\neg \text{crit}_T$ . If  $\neg \text{crit}_i$  then 1) and  $T_9 \in c$  imply  $i \in L$  before the transition, and we have  $M(i) = 7$  after the transition, so 1) is satisfied. If  $\text{crit}_i$  then  $B_{i_8} \notin c$  and 2) imply that  $i \notin L$  &  $i \notin W$  and  $C_{i_7} \in c \vee C_{i_8} \in c$  before the transition. After the transition we are not at  $B_{i_8}$  or  $B_{i_{10}}$  therefore 2) is satisfied. 3d) is satisfied since  $M(i) \leq 2$  and 1) imply  $\neg \text{crit}_T$  before the transition and 3a) is satisfied since  $B_{i_2} \notin c$ .

We have thus shown informally that  $Q_i$  is a valid predicate

for  $P'$ ; the 'proof' could be made rigorous.

### Observations

1. Both these proofs have been long, yet we consider that they are as short as any convincing proofs could be. They follow quite closely the informal reasoning that must be made; such reasoning will always look tedious when written down in detail. The advantage of the method over simple informal reasoning is clearly that it brings out all the cases. In fact many of the necessary constraints were discovered while trying to find the proofs; even though the program looked correct in fact it wasn't.
2. The proof of (A) is much simpler than (B), because the predicate  $R$  is more uniform than  $Q_1$ . In fact, if a valid predicate is completely uniform then the proofs become very easy. We then get essentially the method of invariants (see for example B. Hansen [4]). We can claim that the invariant method is a special case of the one presented here.
3. Although the proofs were long, note that they were independent of  $N$ . This indicates that our initial requirement that the number of parallel computations in a program be bounded was not necessary. In fact, nothing in the theory requires that  $C_p$  be finite. It is clearly straightforward to extend the method to programs with subroutines, arbitrary numbers of activations of which can be running in parallel. (This is essentially what our example was simulating.)

4. In applying the correctness method to non-parallel programs, considerable effort is saved by the fact that the valid predicate need not describe the memory state for all possible control states. It is sufficient to consider a 'cut-set' of control states, i.e. one control-state per program loop. This saving does not appear to be available for parallel programs. Since arbitrary amounts of computation can take place between one statement and its textual successor, it is necessary to consider every control state in specifying the valid predicate. For example in the previous program it is essential that  $i \notin L$  at  $B_{i_9}$ . This would not necessarily be the case without the constraint  $T_8$  stops  $B_{i_3}$ . Just taking 'cut-sets' of control states would not reveal the need for this constraint.

Nevertheless, it may be possible to simplify the valid predicates to some extent; further study is needed here.

#### Acknowledgements

In matters of notation I have clearly been influenced by the works of Hoare and of Burstall. The choice of an airline reservation system as an example was suggested by John McCarthy.

REFERENCES

- [1] Ashcroft E.A. "On Systems Correctness". (In preparation)
- [2] Ashcroft E.A. & Manna Z. "Formalization of Properties of Parallel Programs", Machine Intelligence 6, Edinburgh University Press (1970).
- [3] Floyd R.W. "Assigning Meaning to Programs", Proc. Symposia in Appl. Math. 19, Amer. Math. Soc. (1967).
- [4] Hansen P.B. "A Comparison of Two Synchronizing Concepts", Acta Informatica 1, No. 3 (190-199) (1972).
- [5] Karp R. & Miller R. "Parallel Program Schemata", J. Comput. System Sci. 3, 147-195.
- [6] Manna Z. "The Correctness of Programs", J. Comput. System Sci. 3, No. 2 (May 1969).
- [7] Manna Z. & Pnueli A. "Formalization of Properties of Functional Programs", J. Assoc, Comput. Mach. 17, No. 3 (July 1970).