計算機アーキテクチャ 59−17
オペレーティング・システム 29−17
(1985・10・25)

リアルタイム処理システムの形式化について

On a Formalization of Real-Time Processing Systems

ワグナー・キエパ・クニア　山下雅史　阿江忠

Wagner Chiepa Cunha*, Masafumi Yamashita and Tadashi Ae

広島大学工学部

Faculty of Engineering, Hiroshima University

(* On leave from Aeronautic Institute of Technology, Brazil)

1. INTRODUCTION

Real-time programs are characterized by the necessity to interact with events occurring in an environment external to the computer in accordance with specific timing constraints. Real-time programs frequently involve some type of parallel execution of processes, being it true parallelism as in the case of multiprocessor systems, or realized by a time-shared processor in the case of single processor systems. This internal parallelism is usually allied with the necessity of responding to a variety of nondeterministic requests from the environment. Therefore in modelling real-time processing systems, the ability to characterize the temporal behavior of the events taking place outside the computer, and the ability to describe the mechanisms for scheduling and synchronization of processes as well the timing properties of the program are of fundamental importance.

For nonreal-time sequential or parallel programming, abstract models have been proposed for the derivation of general properties of programs [1,2,3]. Also flowchart programs and fragments of sequential and parallel programming languages have been succesfully used for the development of formal verification techniques [4,5,6]. However, these models are not suitable for real-time programming for two main reasons: first, execution times in the program are not considered; next, external events with its corresponding time constraints are not represented. Although some techniques have been presented to express timing requirements in real-time systems [7,8,9,10], a model for real-time computation representing the program and its external environment in an integrated form is still being pursued. Such model should be powerful enough to represent important aspects of actual systems and at the same time simple enough to be manageable.

In this paper we propose a new model for real-time programs. In the model a set of concurrent processes executed by a time-shared single processor interacts with a set of input tapes and a set of output tapes representing their environment. Each process in the program is associated to one input tape and one output tape. The input tapes represent temporal sequences of stimuli to the program, and the output tapes represent the temporal sequences of responses from the program to the environment. The flow of time is made explicit by the introduction of a nonnegative integer variable to represent the computation state of the program as a function of time. Processes access their tapes through reading heads and writing heads which move unidirectionally scanning sequentially the cells of the tapes according to the flow of time. Stimuli for a process are modeled as a sequence of symbols representing data, interspersed with subsequences of a special symbol called blank which act as time intervals separating successive stimuli. Each output tape records the responses generated by its corresponding process as a sequence constituted of symbols written by the process interspersed with subsequences of blanks corresponding to time intervals during which the process does not execute output statements.

In the proposed model the correct behavior of a real-time program is stated in terms of generation of correct temporal sequences of responses for given sequences of stimuli; both terminating and cyclic computations are unified in a single concept.

The rest of the paper is composed as follows: In Section 2 the syntax of the model, hereafter referred to as RTPP (Real-Time Parallel Program), is introduced. In Section 3 we define the semantics of RTPPs using the concepts of computation state and computation sequence. We also introduce important

properties for RTPPs such as termination, data overrun, deadlock and correctness. In Section 4 we discuss in detail, as a case study, a real-time version of the Producer-Consumer problem [11]. Section 5 ends with a summary and some remarks.

2. RTPPs SYNTAX

We define a Real-Time Parallel Program (RTPP) as a pair (P,F), where $P = (P_1,\ldots,P_n)$ is an n-tuple of processes and $F = \{f_1,\ldots,f_m\}$ is a set of m symbols called flags. Each process in P is a finite sequence of instructions I_i. Each instruction has the form i.<statement>, where i is the position of the instruction in the sequence and <statement> is one of the following.

assignment:
y_j <-- $f(y_1,\ldots,y_k)$

test:
if $p(y_1,\ldots,y_k)$ then a

delay:
suspend($f(y_1,\ldots,y_k)$)

synchronization:
signal(g)
wait(g)

input/output:
input(y_j)
output(y_j)

termination:
halt

In the statements above, $y_1,\ldots,y_k$ are the program variables of the RTPP; the domain of the variables is **Z**, where **Z** is the set of integer numbers; $f(y_1,\ldots,y_k)$ is a total function from $\mathbf{Z}^k$ to **Z**; $p(y_1,\ldots,y_k)$ is a total predicate from $\mathbf{Z}^k$ to {false,true}; g ∈ F; and y_j ∈ $\{y_1,\ldots,y_k\}$. In a test statement "if $p(y_1,\ldots,y_k)$ then a", a is called "jump address" of the statement. Every process of an RTPP must satisfy the following conditions.

(i) The jump address of a test statement must exist.
(ii) The last instruction of a process must consist of either a termination or a test statement.

[Example 1]: An example of RTPP is shown in Figure 1. The RTPP consists of processes "Producer" and "Consumer", and flag "f". Producer reads symbols from its input tape (see 3.1(4) for input tapes) and inserts it in a circular queue. Consumer transfers symbols from the queue to its output tape (see 3.1(4)). Producer and Consumer are synchronized by flag f. The program variables are n, x, y, hd, tl, and the array b. B and C are positive integer constants representing the capacity of the queue and a characteristic delay in Consumer respectively. The symbol 'e' is used as a kind of end marker.

Although for readers familiar with usual concurrent program models ProducerConsumer may seem to work correctly, it does so only under some conditions. The program is different from the conventional one at the point that the input may be continuously fed to Producer. In Section 4 we will return to this example.

ProducerConsumer = ((Producer,Consumer),{f})

Producer:	Consumer:
1. signal(f)	1. if n=0 then 1
2. input(x)	2. wait(f)
3. if n=B then 3	3. y <-- b[tl]
4. wait(f)	4. tl <-- (tl+1) mod B
5. b[hd] <-- x	5. n <-- n-1
6. hd <-- (hd+1) mod B	6. signal(f)
7. n <-- n+1	7. suspend(C)
8. signal(f)	8. output(y)
9. if x≠'e' then 2	9. if y≠'e' then 1
10. halt	10. halt

Fig. 1. ProducerConsumer: an RTPP for a real-time version of the producer-consumer problem.

3. EXECUTION OF RTPPs

In this Section the semantics of RTPPs will be presented. First we give an intuitive introduction and afterward we define the semantics formally using an interpretive model.

3.1. RTPP Processes and their Environment

(1) Time and Processes' States

RTPPs are executed in connection to the flow of time represented by a running time variable t. There is one location counter associated to each process of an RTPP. Execution begins at time instant t=0 with each location counter pointing at the first statement of its corresponding process. During each time interval (t,t+1), where t is a nonnegative integer value of the time variable (hereafter called time slices), the RTPP machine selects one process for execution and executes the statement pointed at by its corresponding location counter. At the time of selection each process can be in one of the following "execution states":

"r": ready;

"s": suspended for a time interval;

"f_i": waiting for flag f_i, where $i=1,\ldots,m$;

"d": waiting for data;

"h": terminated.

The RTPP machine selects for execution the process in "r" state which does not have executed statements for the longest time, i. e. the RTPP machine selects processes for execution according the round robin scheduling mechanism; if there is more than one process in the same condition, the choice occurs at random; if there is no process in "r" state, the RTPP machine only executes an imaginary NOP (no-operation) instruction in an imaginary process. The diagram in Fig. 2 shows the possible transitions of execution states for an RTPP process.

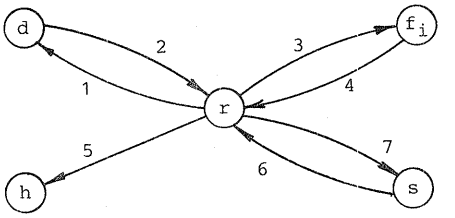
(2) Assignment and Test Statements

Execution of an assignment statement "y_i <-- $f(y_1,\ldots,y_k)$" has the effect of changing the value of the variable y_i to the new computed value $f(y_1,\ldots,y_k)$ and incrementing the location counter of the process which executed it.

A test statement "if $p(y_1,\ldots,y_k)$ then a", if the predicate p is false at the time of execution, has the effect of incrementing the location counter of the process; otherwise the value of the location counter becomes a.

(3) State Control Statements

When a process executes a delay statement "suspend($f(y_1,\ldots,y_k)$)" in the time slice (t,t+1) with $f(y_1,\ldots,y_k)>0$ the effect of the statement is to



1: input statement
2: data available
3: wait(f_i) statement
4: signal(f_i) statement
5: termination statement
6: delay statement
7: end of time

Fig. 2. State transition diagram for an RTPP process.

increment the location counter and to put the process in "s" state until the time instant $t+1+f(y_1,\ldots,y_k)$; at $t+1+f(y_1,\ldots,y_k)$ the RTPP machine puts the process in "r" state. In the case $f(y_1,\ldots,y_k)\leq 0$ the statement just increments the location counter of the process.

The effect of synchronization statements in connection with its arguments (flags) is analogous to operations P and V on binary semaphores. During execution of an RTPP each of its flags is either in "on" state or "off" state. When a "wait(g)" statement is executed in a time slice in which g is in "on" state the location counter of the process is incremented and g is put in "off" state; otherwise the location counter is incremented and the process is put in "g" state until the execution of a "signal(g)" statement by other process. When a "signal(g)" statement is executed the location counter of the process which executed it is incremented and if there are processes in "g" state the one that has been in this state for the longest time is put in "r" state; if there is no process in such situation the flag g is put in "on" state.

Execution of a termination statement puts the process which executed it in "h" state.

(4) Input/Output Statements

The input/output structure of RTPP machines is shown in Fig. 3. There is one "input tape", one "input register" and one "output tape" corresponding to each process of an RTPP. Input and output statements in a process are related to its corresponding input tape and output tape respectively.

The tapes are divided into cells and are infinite to the right. Each cell of an input tape contains one symbol of the set $TS = \mathbb{Z} \cup \{\not\emptyset\}$, where $\not\emptyset \notin \mathbb{Z}$ is a

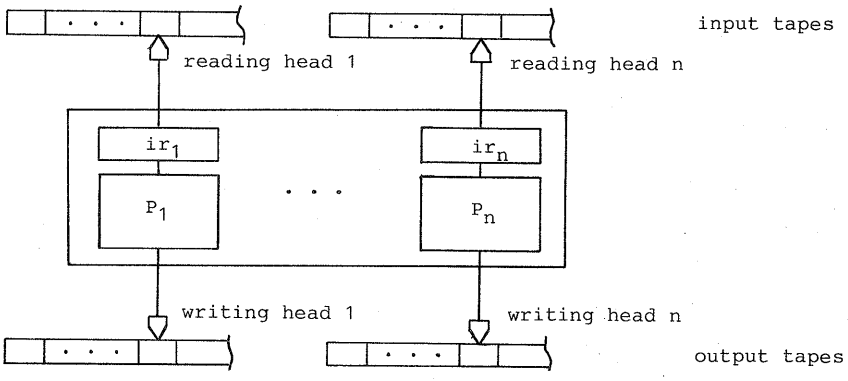


Fig. 3. Input/output structure of RTPPs.

special symbol called "blank". The contents of the input tapes don't change during the execution of an RTPP. When the execution of an RTPP starts, all cells of its output tapes are empty. To scan the cells of each input tape there is one "reading head", and to write symbols on the output tapes there is one "writing head" for each tape.

At t=0, when the RTPP starts to execute, each head is over the leftmost cell of its corresponding tape. At all subsequent values of the time variable t all heads move one cell right. Hence in the time slice (t,t+1) each head scans the (t+1)-th cell of its tape. At the beginning of a time slice the cells that are under the reading heads are analysed and all symbols different from $\not b$ are transferred to the corresponding input registers. At t=0 all input registers contain the blank symbol.

Execution of input statements proceeds in the following way. Let 'c' be the contents of the input register of a process which is going to execute the statement "input(x)". If $c \neq \not b$,i.e., c is nonblank, the effect of the statement is the assignment of the value c to the variable x, the assignment of $\not b$ to the input register and the incrementing of the location counter of the process. If $c = \not b$ the process is put in "d" state until the beginning of a time slice in which the reading head corresponding to the input register scans a valid symbol and transfers it to the input register; at this time the RTPP machine puts the process in "r" state, and the next time it is selected for execution the statement is re-executed as in the former case. Note that in the later case, if before the re-execution of the statement another nonblank symbol is scanned by the reading head, the symbol in the input register is written over by the new one; this phenomenon is called "data overrun".

The behavior of RTPPs concerning output statements and output tapes is as follows. In time slices in which no output statement is executed the RTPP machine writes blank symbols on the cells of the output tapes that are under the writing heads. In case a statement "output(x)" is executed, the value of the variable x is written on the cell of the output tape of the corresponding process which is under the writing head, the location counter of the process is incremented and blanks are written on the cells of the other processes.

Input tapes and output tapes constitutes the external environment of an RTPP. The input tapes act as temporal sequences of stimuli for the RTPP and the output tapes record the corresponding temporal sequences of responses to the environment generated by the RTPP. The blank symbols represent time intervals between successive stimuli or responses.

3.2 Interpretive Model for RTPPs

In this Section, we formally define the effect of an RTPP by giving an abstract interpreter. The interpreter executes an RTPP by changing the current computation state of the RTPP into the next computation state. We start with the definition of Computation state.

Computation state of an RTPP $R = ((P_1,\ldots,P_n),\{f_1, \ldots,f_m\})$ is an 8-tuple (LC,TC,PS,FG,IR,OS,Y,t), where

(i) $LC = (lc_1,\ldots,lc_n)$ is the vector of location counters corresponding to the n processes;

(ii) $TC = (tc_1,\ldots,tc_n)$ is the vector of time counters corresponding to the n processes. The domain of values for $tc_1,\ldots,tc_n$ is Z;

(iii) $PS = (ps_1,\ldots,ps_n)$ is the vector of execution states of the n processes. The domain of values for $ps_1,\ldots,ps_n$ is $\{r,s,d,h,f_1,\ldots,f_m\}$, where "r" stands for ready, "s" for ssuspended for a time interval, "d" for waiting for data, "h" for halt, and f_i for waiting for flag f_i;

(iv) $FG \subset F$ is the set of on flags;
(v) $IR = (ir_1,...,ir_n)$ is the vector of input registers corresponding to the n processes. The domain of values for $ir_1,...,ir_n$ is $\mathbf{Z} \cup \{\not\emptyset\}$;
(vi) $OS = (os_1,...,os_n)$ is the vector of output sequences corresponding to the n processes, where $os_1,...,os_n$ are strings of symbols of the set $\mathbf{Z} \cup \{\not\emptyset\}$ such that $|os_1| = ... = |os_n| = t$ (see viii);
(vii) $Y = (y_1,...,y_k)$ is the vector of values of the variables of the RTPP. The domain of values for Y is $\mathbf{Z}^k$;
(viii) t is a nonnegative integer called time instant.

The computation state $c_0=((1,...,1),(0,...,0),(r,...,r), \emptyset ,(\lambda,...,\lambda),(0,...,0),0)$, where λ represents the empty string, is called initial computation state of the RTPP.

Vector of input tapes for an RTPP $R = ((P_1,...,P_n), \{f_1,...,f_m\})$ is an n-tuple $(it_1,...,it_n)$ of infinite sequences of symbols of the set $\mathbf{Z} \cup \{\not\emptyset\}$. Let $IT=(it_1, ...,it_n)$ be a vector of input tapes for an RTPP $R=((P_1,...,P_n),\{f_1, ...,f_m\})$. Computation sequence for R with IT is an infinite sequence $c_0,c_1,...$ of computation states of R, where c_0 is the initial computation state of R and for any $j>0$ c_{j+1} is obtained from c_j and IT by applying them to procedure "next" given below.

The procedure next(c,IT) is presented in a notation similar to the programming language Pascal [12]. The symbols "(*" and "*)" are used to enclose comments. The curly brackets "{" and "}" are used to indicate variables of the type set. An assignment of the form "x := select(S)" means the assignment at random of one of the elements of the set S as a value for the variable x (this type of statement introduces nondeterminism in the computation state transitions, therefore different computation sequences may exist for an RTPP with the same vector of input tapes).

```
procedure next(var c: computation state;
          IT: vector of input tapes);

  procedure exec(x:integer;
          var c:computation state;
          IT: vector of input tapes);
    (*P_x is the process to be
    executed, where P_0 is the
    imaginary process*)

    begin

      (*if x=0 just append one blank at
      the right of each output sequence
      and exit*)
      if x=0 then
        for j:=1 to n do os_j := os_j . '∅'

      else
        begin
          (*if the statement to be
          executed is not an output
          statement then append one blank
          at the right of each output
          sequence*)
          if statement(lc_x)≠
                'output(y_i)' then
            for j:=1 to n do
              os_j := os_j . '∅';

          case statement(lc_x) of

            'y_i <-- f(y_1,...,y_k)':
              begin
                y_i := f(y_1,...,y_k);
                lc_x := lc_x + 1;
                tc_x := 1
              end

            'if p(y_1,...,y_k) then a':
              begin
                if p(y_1,...,y_k)=true
                 then
                  lc_x := a
                else lc_x := lc_x + 1;
                tc_x := 1
              end

            'suspend(f(y_1,...,y_k))':
              begin
                if f(y_1,...,y_k)>0 then
                  begin
                    ps_x := s;
                    tc_x:=f(y_1,...,y_k)+1
                  end;
                lc_x := lc_x + 1
              end

            'signal(g)':
              begin
                (*find processes that
                have been waiting the
                longest time*)
                PF := {j|P_j ∈ P and
                    ps_j=g and
                    (∄u|P_u ∈ P and
                    ps_u=g and
                    tc_u<tc_j)};
                if PF≠∅ then
                  begin
                    j := select(PF);
                   (*put P_j in r state*)
                    ps_j := r
                  end
                else FG := FG U {g};
                 (*put g in on state*)
                lc_x := lc_x + 1;
                tc_x := 1
              end

            'wait(g)':
              begin
                if g ∈ FG then
                  FG:=FG-{g} {*g off*)
                else ps_x := g;
                lc_x := lc_x+1;
                tc_x := 1
              end

            'input(y_i)':
              begin
                if ir_x=∅ then
                  ps_x := d
                else
                  begin
```

```
                    y_i := ir_x;
                    ir_x := ∅;
                    lc_x := lc_x + 1;
                  end;
                tc_x := 1
               end

          'output(y_i)':
             begin
               os_x := os_x . y_i;
               for j:=1 to n do
                 if j≠x then
                   os_j:=os_j.'∅';
               lc_x := lc_x + 1;
               tc_x := 1
             end

          'halt': ps_x := h
        end (*end of case*)
      end
  end (*end of exec*)

begin (*begin of next*)

  (*update input registers*)
  for j:=1 to n do
    begin
      ir_j := it_j[t];
      (*if P_j is waiting for data
      activate it*)
      if ps_j=d then ps_j := r
    end;

  (*find processes that have been
  waiting the longest time*)
  PE := {j|P_j ∈ P and ps_j=r and
    (∄u|P_u ∈ P and ps_u=r and
    tc_u<tc_j)};
  if PE≠∅ then x := select(PE)
  else x := 0;

  (*execute statement*)
  exec(x,cs,IT);

  (*decrement time counters of
  processes which are in r, s, d, f_1,
  ..., f_m state*)
  for j:=1 to n do
    if ps_j ∈ {r,s,d,f_1,...,f_m} then
      tc_j := tc_j -1;

  (*reactivate suspended processes
  where time counters have reached
  zero*)
  for j:=1 to n do
    if ps_j=s and tc_j=0 then ps_j := r;

  (*increment time instant*)
  t := t + 1;

end (*end of next*)
```

3.3 SOME PROPERTIES OF RTPPs

Here we summarize some consequences of the mechanisms for process scheduling and handling of flags adopted in procedure "next". Also the concepts of "termination", "data overrun", "deadlock", and "correctness" for RTPPs are introduced.

(1) Process Schedulling and Handling of Flags

The algorithm used by procedure "next" to select a processes for execution in a given time slice gives highest priority to processes which have been waiting for the longest time. The following property is an immediate consequence of this observation.

[Property 1]: For any given process in "r" state, the maximum number of time units between two successive selections for execution is the number of the remaining processes which are not in "h" state, if the process remains in "r".

Flags are handled according to the same priority scheme adopted for process scheduling, i.e., when two or more process are waiting for the same flag g and other process executes a "signal(g)" statement, the process that has been waiting for the longest time is put in "r" state; therefore we have the following property:

[Property 2]: For any given process waiting for a flag g, the maximum number of times a "signal(g)" statement has to be executed to put the process in "r" state is equal to the number of the remaining processes which are not in "h" state.

(2) Termination, Deadlock, Data Overrun

A computation state in a computation sequence of an RTPP is a final computation state if and only if its vector of execution states is equal to (h,...,h). An RTPP R terminates for a vector of input tapes IT if and only if in any computation sequence for R with IT there exists a final computation state.

A computation state in a computation sequence of an RTPP is said to be a deadlocked computation state if and only if its vector of execution states is an element of $\{h,f_1,...,f_m\}^n - \{(h,...,h)\}$. Note that, by the definition of "next", if the current computation state is final or deadlocked the following computation state remains in the same situation.

Let CS be a computation sequence for an RTPP R with vector of input tapes IT. We say that data overrun occurs in CS if and only if there is a computation state cs_j in CS such that the contents of the input register of some process is not a blank and the symbol in the (j+1)-th cell of the input tape corresponding to the process is not a blank. Occurrence of data overrun means that a symbol in an input register is written over by another symbol before being removed by the corresponding process. Data overrun is a typical phenomenon which may occur in an incorrect real-time program.

(3) Correctness

Let $IT = (it_1,...,it_n)$ be a vector of input tapes for an RTPP R, and let $OT=(ot_1,...,ot_n)$ be a vector of infinite sequences of symbols of $\Sigma \cup \{\emptyset\}$. OT is a vector of output tapes generated by R with IT if and only if there exists a

computation sequence for R with IT such that for any computation state c_j the components of the vector of output sequences are prefixes of the corresponding components of OT.

Let A and B be total predicates about vectors of infinite sequences of $\Sigma \cup \{\not\emptyset\}$ compatible with an RTPP R. We say that R is correct with respect to A and B if and only if for any IT such that A(IT) is true then, for any OT generated by R with IT, B(OT) is true. The predicates A and B are called input predicate and output predicate of the RTPP respectively.

4. CASE STUDY: THE PRODUCER-CONSUMER RTPP

Now we return to the RTPP ProducerConsumer of Example 1. In this Section, we mainly discuss about data overrun and conditions in which ProducerConsumer works correctly.

4.1. The "ProducerConsumer" RTPP

Our program (Fig. 1) works with processes Producer and Consumer being executed in parallel. Producer reads symbols from its input tape and inserts it into a queue. Consumer repeatedly takes a symbol from the queue, suspends itself for C time units and transfers the symbol to its output tape. Accesses to the queue occur in a mutual exclusion fashion by the use of the flag f. Both processes halt after the symbol 'e' is processed.

The queue of size B is implemented using the array b[0],..., b[B-1]; Variables hd and tl are used as indexes of the head (insertion extremity) and tail (removal extremity) of the queue respectively; n is used as a counter for the number of symbols in the queue; and B and C are positive integer constants. Note that at t = 0, by the definition of initial computation state, all the variables have value zero, all processes are in "r" state and all flags are in "off" state.

The input tapes for Producer are supposed to be constituted of the nonblank symbols 's_1', ..., 's_m', and 'e' in this order and interspersed with strings of blanks (data separators); Consumer is expected to generate output tapes similar to the input tapes of Producer. The output tape of Producer and the input tape of Consumer are not referred to in the program and will be ignored in what follows.

4.2. Conditions to Avoid Data Overrun

One necessary condition for the correct behavior of ProducerConsumer is the execution of the main loop of Producer (instructions 2-9) to occur within a number of time slices small enough to assure that every symbol in the sequence $s_1,\ldots,s_m$ is read by Producer (instruction 2) before being overwritten in the input register by its nonblank successor in the input tape.

Here we determine the relations that must exist between the number of nonblank symbols in the input tapes of Producer, the data separation, and the constants of the program, to assure the nonoccurrence of data overrun in the input register of Producer.

We assume input tapes of the form

$$it = \not\emptyset^{q_1} s_1 \not\emptyset^{q_2} s_2 \ldots s_m \not\emptyset^{q_{m+1}} e \not\emptyset\not\emptyset\ldots\ldots,$$

where:

$$s_1, \ldots, s_m \text{ and } e \text{ are nonblank;}$$

$$e \notin \{s_1,\ldots,s_m\};$$

$$q_1, \ldots, q_{m+1} \geq n; \qquad (1)$$

and perform a case analysis according to the value of m, the number of nonblank symbols in the input tape.

(1) m arbitrarily large.
[Notation]: We indicate that the j-th instruction of process P_i is executed in the time slice (t,t+1) by "select(t)=(P_i,j)"; "select(t)=(0,0)" means that there is no process in "r" state during (t,t+1).

Let (t_0,t_0+1) be the time slice at the beginning of which a symbol s_j is transferred from the input tape to the input register of Producer, $t_r > t_0$ be the minimum value of t such that select(t_r)=(Producer,2), and (t_i,t_i+1) be the time slice at the beginning of which the next nonblank symbol, s_{j+1}, is transferred to the input register. Clearly, in order to prevent s_j from being overwritten by s_{j+1} it is necessary and sufficient that

$$t_i > t_r. \qquad (2)$$

From t_0 to t_i the contents of ir_P (Producer's input register), lc_P and lc_C (location counters of Producer and Consumer respectively), and the contents of the cell scanned by the reading head of Producer, have the temporal behavior shown in Fig.4.
From the format of the input tape we have

$$n > t_r - t_0 - 1. \qquad (3)$$

Therefore the minimum value of n for which no symbol is overwritten is given by

$$n_{min} = (t_r - t_0)_{max} = (t_r)_{max} - t_0. \qquad (4)$$

To determine $(t_r)_{max}$ we have to consider all possible values of the computation state at t_0 and, for each of them, all possible subsequences of computation states from t_0 to t_r. To avoid the

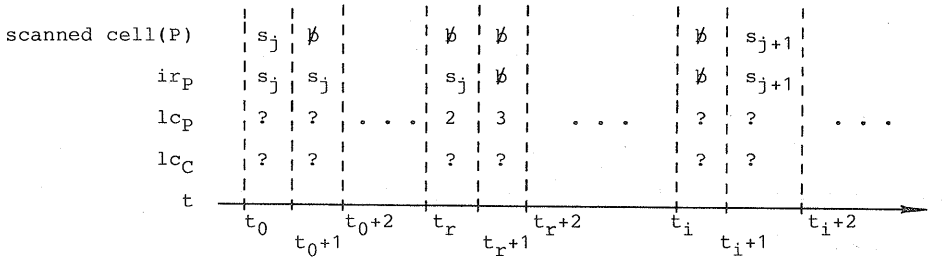


Fig. 4. Temporal behavior of ProducerConsumer from t_0 until t_i.

investigation of all possible cases, instead of the exact value of $(t_r)_{max}$, we estimate an upper bound for it by supposing that at $t=t_0$ we have: select(t_0) = (Producer,3); $lc_C = 6$; and n = B, i. e. the queue is full.

Let t'_r be the value of t_r under the hypothesis stated above. Although we cannot assure that there exists a computation sequence containing a computation state satisfying such hypothesis it is clear that for any possible computation sequence we have

$$(t_r)_{max} \leqslant (t'_r)_{max}. \qquad (5)$$

To estimate $(t'_r)_{max}$ we use the graphs in Fig.5, where the nodes represent the values of the location counters and the time instants on the edges are the time instants at which the statements corresponding to the source nodes are selected for execution. By Property 1 and the treatment given to delay statements in the procedure "next" we get the following relations

$$t_1 \leqslant t_0 + 3 + C + 1;$$

$$t_2 \leqslant t_1 + 2;$$

. . . . .

$$t_8 \leqslant t_7 + 2;$$

$$t_9 \leqslant t_8 + 3;$$

$$t_{10} \leqslant t_9 + 2;$$

. . . . .

$$t'_r \leqslant t_{13} + 2;$$

and consequently

$$t'_r \leqslant t_0 + 31 + C. \qquad (6)$$

Considering the inequalities (4), (5) and (6) we get

$$n_{min} \leqslant 31 + C. \qquad (7)$$

Therefore the condition $n \geqslant 31 + C$ is sufficient to assure that every s_j, $1 \leqslant j \leqslant m$, is read by Producer. The symbol 'e' is clearly read because it is the last symbol transferred to the input register of Producer. Note that this result holds for any value of B, the capacity of the queue.

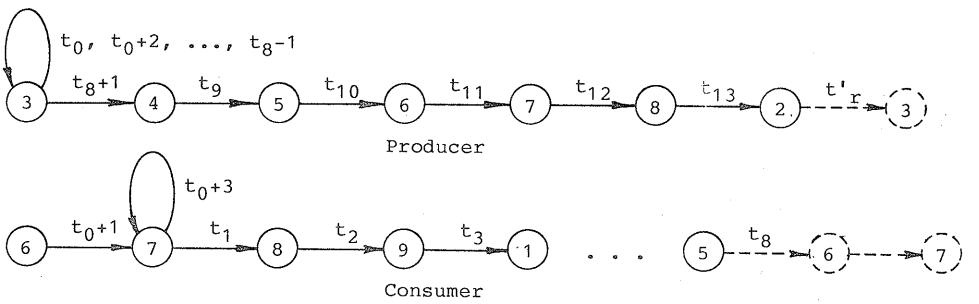


Fig. 5. Graphs for estimation of $(t'_r)_{max}$ (case (1)).

(2) $m + 1 \leqslant B$.

The input tape of Producer has $m + 1$ nonblank symbols, and the capacity of the queue is enough to accomodate all of them. Therefore the test statement in line 3 of Producer never succeeds.

Similarly to what was done in case (1), we estimate an upper bound for $(t_r)_{max}$. Here the worst case for the computation state at t_0 is: select(t_0)= (Consumer,1); $lc_P = 3$; and $n \neq 0$. The graphs for estimation of $(t'_r)_{max}$ are shown in Fig.6. The following inequalities hold

$$t_1 \leqslant t_0 + 11 + 2;$$

$$t_2 \leqslant t_1 + 2;$$

$$t_3 \leqslant t_2 + 2;$$

$$t'_r \leqslant t_3 + 2;$$

and consequently

$$t'_r \leqslant t_0 + 19. \qquad (8)$$

Finally, considering (4), (5) and (8) we get

$$n_{min} \leqslant 19. \qquad (9)$$

Therefore if the capacity of the queue is greater or equal to the number of nonblank symbols in the input tape of Producer, the condition $n \geqslant 19$ is sufficient to assure that all of them are read.

(3) $m + 1 \leqslant M$, where $M > B$ is an integer constant.

Here we estimate the minimum data separation n sufficient to prevent symbol overwriting in the input register of Producer as a function of the maximum number of nonblank symbols (M + 1).

The rate of transfer of symbols to the variable x of Producer given by

$$v_P = \frac{1}{n + 1} \text{ (symbols/time unit).} \qquad (10)$$

Supposing the queue never becomes empty while Producer is active, the number of time slices required by the main loop of Consumer to remove one symbol from the queue is upper bounded by the constant 22 + C (this value can be obtained like in 1). Therefore symbols are removed from the queue at a rate not less than

$$v_C = \frac{1}{22 + C} \text{ (symbols/time unit).} \qquad (11)$$

Consequently, after the first symbol is inserted in the queue, the number of symbols in the queue increase at a rate not greater than

$$v_{QI} = v_P - v_C. \qquad (12)$$

By (10),(11), and (12), considering also the first symbol inserted in the queue, we get the following sufficient condition for the queue never becoming full

$$1 + v_{QI} * \frac{M}{v_P} \leqslant B \qquad (13)$$

or

$$n \geqslant \frac{(C + 22) * (M - B + 1)}{M} - 1. \qquad (14)$$

Therefore, condition (14) is sufficient to assure that all nonblank symbols are read by Producer.

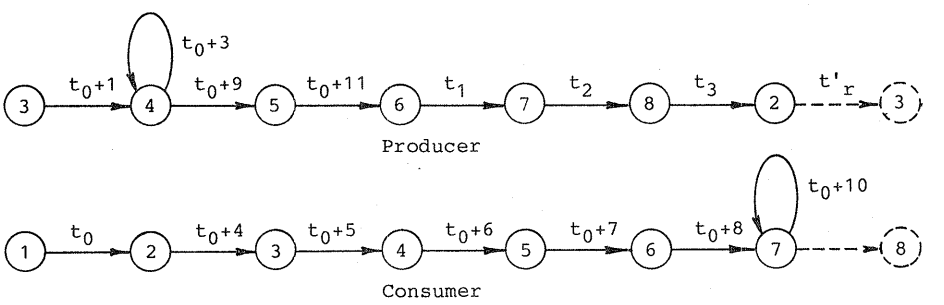


Fig. 6 Graphs for estimation of $(t'_r)_{max}$ (case (2)).

4.3. Correctness

Here we outline a proof that the RTPP ProducerConsumer is correct with respect to the following predicates.

A (Input Predicate): The input tapes for Producer have the form (1) with the additional conditions $m+1 \leq B$, and $n \geq 19$. These conditions correspond to case (2) in Subsection 4.2; for cases (1) and (3) the proof is practically the same.

B (Output Predicate): The output tapes generated by Consumer are such that the nonblank symbols are 's_1', ..., 's_m', 'e'; their order is the same as in the input tapes of Producer; and the minimum data separation between two consecutive ones is constituted by at least C blanks.

[Lemma 1] For any computation sequence CS of ProducerConsumer, with any input tape of Producer satisfying the input predicate, all nonblank symbols of the input tape of Producer are read and inserted in the queue.

Proof: The conditions $m+1 \leq B$ and $n \geq 19$ in the input predicate correspond to case (2) in Subsection 4.2. Therefore the RTPP is overrun free with respect to any input tape of Producer satisfying the input predicate, and all nonblank symbols are read (transferred to variable x of Producer) and subsequently inserted in the queue.

[Lemma 2] Let (t,t+1) be a time slice and Q(t) be the sequence of symbols defined by

$Q(t) = \lambda$, if $n(t) = 0$;

$Q(t) = b[tl(t)]...b[(tl(t)+n(t)-1) \bmod B]$, if $n(t) \neq 0$.

For a given computation sequence CS let T'(CS) be the set of values the time instant t such that Producer is not in the middle of an insertion in the queue and Consumer is not in the middle of removing a symbol from the queue, i. e.,

$$T'(CS) = \{t \in \mathbf{Z} \mid \text{select}(t) \notin \{(\text{Producer},6),(\text{Producer},7),(\text{Consumer},4),(\text{Consumer},5)\}\}.$$

Let X(t), with $t \in T'(CS)$, be the sequence of symbols inserted in the queue until t; and let Y(t), with $t \in T'(CS)$, be the sequence of symbols removed from the queue until t. Then, for any computation sequence CS and any $t \in T'(CS)$

$$Y(t) \cdot Q(t) = X(t),$$

where $Y(T) \cdot Q(t)$ represents the concatenation of the sequences Y(t) and Q(t).

Proof: For any $t \in T'(CS)$ claims (a) and (b) below can be proved by induction on $t \in T'(CS)$.

Claim (a): $hd(t) = (tl(t)+n(t)) \bmod B$.

Claim (b): $|Y(t)|+n(t) = |X(t)|$.

With claims (a) and (b), lemma 2 can be proved by induction on $t \in T'(CS)$ with a case analysis according to the statement selected for execution at a given $t \in T'(CS)$. Note that only the cases select(t) = (Producer,5) and select(t) = (Consumer,3) must be considered, in the remaining ones the values of X, Q and Y are not affected.

[Lemma 3] For any computation sequence CS, all symbols inserted in the queue by Producer are removed from the queue by Consumer and transferred to its output tape.

Proof: By lemma 1 all nonblank symbols in the input tape of Producer are read and inserted in the queue. After the symbol 'e' is inserted in the queue, the test statement in line 9 of Producer is executed with x = 'e'; the test fails and the location counter of Producer is incremented. Consequently, the next time Producer is scheduled for execution, a termination statement is executed and Producer is put in "h" state.

Let (t_P, t_{P+1}) be the time slice in which Producer is put in "h" state; for any $t \geq t_P$ we have $X(t) = s_1...s_m e$. Let $Q(t_P) = Z$ and $Y(t_P) = W$. By induction on the length of Z it can be shown that there exists $t_Q \geq t_P$ such that $|Q(t_Q)| = 0$. Therefore, for any computation sequence CS there exists a time instant t such that $Y(t) = X(t) \cdot \lambda = s_1...s_m e$.

[Theorem] RTPP ((Producer,Consumer),{f}) is correct with respect to predicates A and B.

Proof: By lemmas 1, 2 and 3 all the nonblank symbols in the input tape of Producer are transferred to the output tape of Consumer preserving their order. After removing a symbol from the queue, and before writing it on the output tape, Consumer always executes the statement "suspend(C)" in its 7-th line, and consequently each nonblank symbol in the output tape is always preceded by least C + 1 blank symbols. After the symbol 'e' is written on the output tape, the test statement in line 9 of Consumer is executed with y = 'e', the test fails and the location counter of Consumer is incremented. The next statement executed by Consumer is a termination statement, therefore all the symbols following 'e' in the output tape of Consumer are blanks.

5. Summary and Final Remarks

A program model for real-time processing systems has been presented. The model formalizes a real-time program as a set of concurrent processes time-sharing a single processor and communicating between themselves by use of shared variables and synchronization signals. The environment of the program is conceptualized as a set of temporal sequences of stimuli to the processes, and a set of temporal sequences of responses generated by the processes. The correctness of the programs is stated in terms of generation of correct sequences of responses to given sequences of stimuli from the environment.

Given a description of the temporal behavior of the environment, in the form of sets of input tapes, the behavior of the RTPP can be analysed to verify if it generates output tapes which correspond to the required sequences of responses.

An important topic with deserves further research is the development of verification methods for RTPPs. In connection with the verification methods, two problems that deserve a more systematic approach is the verification of nonoccurrence of data overrun and termination of processes.

This work was partially supported by a Scientific Research Grant-in-Aid from the Ministry of Education, Science and Culture, Japan.

References

[1] Luckham, D., Park, D. and Paterson, M.: "On Formalized Computer Programs", J. Comput. & Syst. Sci., 4, 3, pp. 220-249 (1970).

[2] Constable, R. L. and Gries, D.: "Classes of Program Schemata", SIAM Journal on Computing, 1, 1, pp. 66-118 (1972).

[3] Yamashita, M., Inagaki, Y. and Honda, N.: "A Concurrent Program Scheme with Finite State Schedulers", Trans. IECE Japan, Vol. 63-D No. 8, pp. 634-641 (1980), in japanese.

[4] Floyd, R. W.: "Assigning Meaning to Programs", in Schwartz, J. T., ed. Mathematical Aspects of Computer Science, Proc. Symposia in Applied Mathematics 19, pp. 19-32, Amer. Math. Soc. (1967).

[5] Owicki, S. S.: "Axyomatic Techniques for Parallel Programs", Ph. D. thesis, Cornell University (1975).

[6] Francez, N., Pnueli, A.: "A Proof Method for Cyclic Programs", Acta Informatica 9, pp. 133-157 (1978).

[7] Dasarathy, B.: "Timing Constraints for Real-Time Systems: Constructs for Expressing Them, Methods for Validating Them", in Proc. IEEE Real-Time Systems Symp. (Dec. 1982).

[8] Coolahan, J. E. and Ropssopoulos, N.: "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets", IEEE Trans. Software Eng., Vol. SE-9, pp. 603-616 (Sept. 1983).

[9] Haase, V. H.: "Real-Time Behavior of Programs", IEEE Trans. Software Eng., Vol. SE-7, pp. 494-501 (Sept. 1981).

[10] Ramamoorthy, C. V. and Ho, S. G.: "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets", IEEE Trans. Software Eng., Vol. SE-6, pp. 440-449 (Sept. 1980).

[11] Hoare, C. A. R.: "Monitors, An Operating System Structuring Concept", Communications of the ACM, Vol. 15, No. 10, pp. 549-557 (Oct. 1974).

[12] Wirth, N.: "The Programming Language Pascal", Acta Informatica, 1, 1, pp. 35-63 (1971).