VALIRA/VALISYN
Protocol Validator/Synthesizer
User's Manual (Version 1.2)

S.T. Vuong and T. Lau

Technical Report 88-19

Department of Computer Science
University of British Columbia
Vancouver, B.C, CANADA V6T 1W5

August 1988

VALIRA User's Manual - Version 1.2 (1988)

Computer Science Department
University of British Columbia
Vancouver, B.C., CANADA V6T 1W5

I. INTRODUCTION

VALIRA (VALIdation via Reachability Analysis) is a protocol validation tool that accepts a given protocol specification in a communicating finite state machine (CFSM) model and performs a validation via reachability analysis. The syntactic properties of the protocol, including state ambiguities, state deadlocks, unspecified receptions, non-executable interactions, and unbounded channels, are analyzed and reported.

The program can validate protocols with:

1. up to ten processes (CFSMs).
2. priority channels. Priorities for messages can be set individually by assigning a priority value to each message type. The higher the priority value, the higher the priority.
3. FIFO channels. Two approaches for validating such protocols are provided: the conventional approach (which works for any FIFO protocol) and the R-approach (which guarantees a finite reachability tree, but only works for well-ordered FIFO protocols).
4. non-FIFO channels.

As a side note, VALIRA handles the flow of messages among processes in different ways for different types of protocols. For protocols with conventional FIFO or priority channels, each message from a process can only have one destination process; i.e. no multi-casting. For non-FIFO protocols or FIFO protocols using R-state approach, a message can have multiple destinations.

The commands used in VALIRA are self-explanatory. Basically, a user can apply the package without reading the command explanation part of the manual. However, the output representation part should be read. Commands can be entered in either upper or lower case. Illegal commands or inputs will be signaled by a "beep" sound for re-entering.

This manual is intended as a user reference. As for the internal structures or background theories of VALIRA, interested readers are referred to papers such as [Vuong86], [Zafi80].

II. HOW TO RUN VALIRA

VALIRA currently runs on SUN 3 workstation, IBM-PC and is readily portable to any machine suporting C. Its source consists of around 7,000 lines of commented C code (occupying around 550Kbytes) and its executable object code is under 100Kbytes.

On the faculty SUN 3 machine, it is invoked with the command

~vuong/VALIRA/package

VALIRA provides a two-level interactive environment which includes :

1. The interpreting level, where, in general, the user can monitor the validation process.
2. The editing level, where the user can use the editing commands provided to make changes on the entered transitions. The editing level is invoked on the interpreting level, so it is one level below.

II.A. The invocation phase

Upon invocation of the package, the user automatically enters the interpreting level, and will be prompted by the interpreter to respond with either "y" or "n" for information required by the package. These questions allows the user to choose the type of protocol to work on. First, the

user is greeted with the following heading and a prompt:

```
*****************************************

      VALIRA  version 1.2 (1988)

    Department of Computer Science
    University of British Columbia

*****************************************
```

```
Default settings :

   FIFO and conventional approach
   number of processes     =    2
   channel bound           =    3
   node limit of the tree  = 1000

Do you want to change any of the above ?  [y|n|q] _
```

If the default setting above is not suitable, the user can set up the program himself or herself by answering "n" at this point. In this case, the interpreter will prompt for more answers to several other questions. The examples below show the requests and answers for some types of protocols.

Example 1 Running a two-process priority protocol

```
Number of processes : 2
Priority channels ? [y|n|q] y
Channel bound : 3
```

Example 2 Running a two-process non-FIFO protocol

```
Number of processes : 2
Priority channels ? [y|n|q] n
FIFO channels ? [y|n|q] n
```

Example 3 Running a three-process FIFO protocol

```
Number of processes : 3
Priority channels ? [y|n|q] n
FIFO channels ? [y|n|q] y
```

At this point, the user can choose between using the R-state approach or the conventional approach. In general, the R-state approach should be applied, since it is more efficient and guarantees a finite reachability tree. However, the R-state approach cannot validate every type of FIFO protocols. The message "non-well-ordered protocol" will appear if the R-state approach fails to work. In that case, the conventional approach has to be used.

After setting up the protocol type, the program will ask for the node limit of the reachability tree:

```
Node limit of the tree :
```

This entry would be used as the global state limit of the reachability tree.

II.B. The input phase

When the queries in the invocation phase have been responded to, the interpreter will enter the input phase where the user is prompted to input the transitions for each process.

```
Enter transitions for process 1 :-
>0 1 -1
>1 0 1
>
```

The prefix character in the input phase is a ">", and the transitions are entered in the form:

```
0 1 -1
```

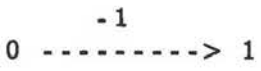
where

0 is the initiating process state of a transition

1 is the resulting process state of a transition

-1 is the transition arc between the two process states

Thus, the above input corresponds to the transition:



The input phase is terminated by entering a null line.

The input of the process states and the message types have the following restrictions:

— process state must be numbered from 0 up to at the maximum of 35. *Note that the initial state of each process must be state 0./fR*

— message type must be numbered within 1 and 35.

II.C. The editing phase

Every time an input phase is terminated, the interpreter will automatically invoke the transition editor so that changes are possible. The prefix character given by the transition editor is a ":", which is followed by entering one of the following commands:

```
h ...  print the message
s ...  stop editing
c ...  clear all transitions
p ...  print all transitions
r n .  replace transition n
d n .  delete transition n
i n .  insert transition(s) after transition n
```

The usage of the above commands is like in any simple line editor.

II.D. The execution phase

The interpreter starts the execution phase when the transitions of all processes are entered. The information entered are echoed to the user if the user reponds to the following with "y"

```
Do you want to view your input ? [y|n|q]
```

Then the following request appears:

```
Do you want the reachability tree printed ? [y|n|q]
```

It should be replied accordingly.

The generation of the reachability tree starts at this point. Shown on the screen will be the print-out of a reachability tree, a node table, and an error summary. Some special notations used in the output will be explained further in section III.

II.E. The command phase

The interpreter enters the command phase after the execution on generating the reachability tree. A list of commands will be printed for selection, as shown below:

```
1. Print the reachability tree
2. Print the node table
3. Print the summary
4. Edit transitions
5. Run
6. Quit

Item number ? _
```

These commands provide the user with an environment in which the protocol entered can be modified and rerun. The commands are selected by entering its item number. Say, if "4" is entered, the interpreter will prompt

process number ?

The process number of the desired process should then be entered. The transition editor (Section II.C) will be invoked to edit the transitions of that process. Upon termination (by the "s" command in the transition editor), the control will be returned to the interpreter in the command phase.

III. Output representations

This section describes the representations used in printing the reachability tree, the node table, and the error summary. To illustrate this, the sample run of an erroneous protocol shown in Figure 1 is given in Figure 2.

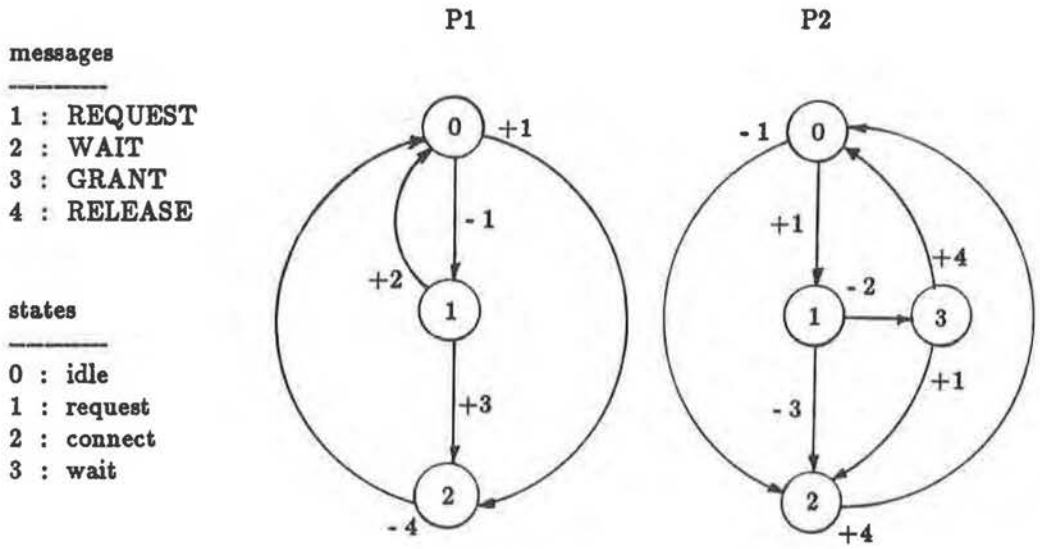


Figure 1. An erroneous protocol.

/** **Figure 2.** A sample run of the erroneous protocol of Figure 1 **/

```
****************************************

        VALIRA  version 1.2 (1988)

        Department of Computer Science
        University of British Columbia

****************************************

Default settings :

   FIFO and conventional approach
   number of processes     =    2
   channel bound           =    3
   node limit of the tree  = 1000

Do you want to change any of the above ?  [y|n|q] n


Enter transitions for process 1 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 mesg

(terminate with a null line)
>0 1 -1
>0 2 1
>1 0 2
>1 2 3
>2 0 -4
>
Enter edit command (or type h for help)
:s

Enter transitions for process 2 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 mesg

(terminate with a null line)
>0 1 1
>0 2 1
>1 2 -3
>1 3 -2
>3 0 4
>3 2 1
>2 0 4
>
Enter edit command (or type h for help)
:p
 1.     0    1   +1
 2.     0    2   +1
 3.     1    2   -3
```

```
 4.     1    3   -2
 5.     3    0   +4
 6.     3    2   +1
 7.     2    0   +4
:r h
  h ... print this message
  s ... stop editing
  c ... clear all transitions
  p ... print transitions
  r n   replace transition n
  d n   delete  transition n
  i n   insert transition(s) after transition n
( i 0   for inserting before transition 1 )
( sample input :  0  1 -1
  meaning : state 0 to state 1 with the transmission of type 1 message )
:r 2
 2.     0    2   +1
>0 2 -1
:s

Do you want to view your input ?  [y|n|q] y


Reachability analysis of protocol with FIFO channels

Process 1 :
  0 -> 1   -1
  0 -> 2   +1
  1 -> 0   +2
  1 -> 2   +3
  2 -> 0   -4

Process 2 :
  0 -> 1   +1
  0 -> 2   -1
  1 -> 2   -3
  1 -> 3   -2
  3 -> 0   +4
  3 -> 2   +1
  2 -> 0   +4

process 1 transmits 2 message(s) :   1  4
process 2 transmits 3 message(s) :   1  3  2


Do you want the reachabilty tree printed ?  [y|n|q] y


-------------------------------- level --------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

 0    1    2    (P1+1)[UR]
                (P2+1)[UR]
```

```
          3    4    5    6    7    (P2+4)[ 1]
                                    (P2+4)[ 0]
               8    9    10   11
     12   (P1-1)[ 2]
          (P1+1)[ 5]

There are a total of 12 different node(s)
Do you want the node table ?  [y|n|q] y


Node configuration :
(s1,s2;t1,t2;t3,t4,t5)

Message counter :  t1  t2  t3  t4  t5
Message type    :   1   4   1   3   2

  0|    (root)00.....   0(P1-1)101....   1(P2-1)121.1..   1(P2+1)11.....
  4|   3(P2-3)12...1.   4(P1+3)22.....   5(P1-4)02.1...   6(P1-1)1211...
  8|   3(P2-2)13....1   8(P1+2)03.....   9(P1-1)131....  10(P2+1)12.....
 12|   0(P2-1)02..1..

Hit return for the summary
```

ERROR SUMMARY

```
1. The list of deadlock node(s) :-
   Configuration : node( s1, .., s2 )

     11( 1, 2)

2. Stable state(s) :-

   P1  P2
  --------
    0   0     (node 0)
    1   1     (node 3)
    2   2     (node 5)
    0   3     (node 9)
    1   2     (node 11)

3. Non-executable interaction(s) :-
   Configuration :   s ->  s' message

   process  2 :    3 -> 0  +4


4. The first unspecified reception is detected at node 2.
   The lowest level UR is detected at node 2 (level 3).

   The list of UR node(s) :-
   Configuration : node( s1, .., s2; reception )
```

```
   process  1 :     2( 1, 2; +1 )

   process  2 :     2( 1, 2; +1 )

5. The reachability tree is bounded at level 6.

6. Max channel queue length = 2.


Choose one item from the list below :
1. Print the reachability tree
2. Print the node table
3. Print the summary
4. Edit transitions
5. Run
6. Quit

Item number ? 6


Total CPU time     :  0.220000 second(s)
Total System Call  :  0.200000 second(s)


                          ***** End of execution *****
```

Figure 2. A sample run of the erroneous protocol

III.A. Representation of the reachability tree

As noted in Figure 2, the reachability tree is printed horizontally from left to right, with each node in the tree represented by a node number. Figure 3 shows the same reachability tree along with branches connecting between every pair of parent and child nodes. For simplicity, these branches are not shown in the actual print out.

```
------------------- level -----------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10

 0---1---2---(P1+1)[UR]
 |   |   ---(P2+1)[UR]
 |   ---3---4---5---6---7---(P2+4)[ 1]
 |       |               ---(P2+4)[ 0]
 |       ---8---9---10--11
 ---12--(P1-1)[ 2]
     ---(P1+1)[ 5]
```

Figure 3. A reachability tree with branches shown.

In most cases, the reachability tree printed would go beyond the right margin. To solve this problem, the portions which go beyond the right margin will be restarted printing from the left margin, on a new level scope. An example of this is shown in Figure 4 and Figure 5.

```
                                                    margin-->|
------------------- level ------------------                 |
|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10       |
                                                            |
 0    1    2    (P1+1)[UR]                                  |
                (P2+1)[UR]                                  |
           3    4    5    6    7    8    9    10   11  |12  13
                                                        |14
           15   16   17   18   19                          |
      20   (P1-1)[ 2]                                      |
           (P1+1)[ 5]                                      |
```

Figure 4. A reachability tree that goes beyond the right margin.

```
------------------- level ------------------
|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10

 0    1    2    (P1+1)[UR]
                (P2+1)[UR]
           3    4    5    6    7    8    9    10   11

------------------- level ------------------
|11  |12  |13  |14  |15  |16  |17  |18  |19  |20  |21


 12   13
 14

------------------- level ------------------
|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10

           15   16   17   18   19
      20   (P1-1)[ 2]
           (P1+1)[ 5]
```

Figure 5. A reachability tree printed in different scopes of levels.

To represent the different kind of nodes in the reachability tree the following notations are being used:

1. *Unspecified node*

   An unspecified node indicates an unspecified reception. It is represented by the notation:

   (Pi+m)[UR]

   where i is the process number, m is the message type, and (Pi+m) represents the transition that initiates the unspecified reception. For example, the "(P1+1)[UR]" node shown in

Figure 3 represents that "process 1 receives message 1 leading to an unspecified reception".

2. *Repeated nodes*

A repeated node is represented by the notation:

(Pi+m)[n]

where n is the node number of the duplicated node, and (Pi+m) has the same meaning as in the unspecified node representation. Thus, the node "(P1-1)[^2]" represents that "process 1 transmits message 1 resulting in a duplicated node having the same global state value as node 2".

3. *Unbounded nodes*

An unbounded node appears when the number of messages in a channel exceeds a prespecified channel bound. It is represented by the notation:

(Pi-m)[UC]

where (Pi-m) has the same meaning as in the unspecified node representation. Thus, the node "(P1-2)[UC]" represents that "process 1 transmits message 2" resulting in the channel bound being exceeded".

4. Others

Other than the above, all the nodes in the reachability tree are represented by sequentially ordered node numbers. The global state value of these nodes can be found in its corresponding entry within the node table, as will be described shortly.

III.B. Representation of the node table

It is assumed the user has a general understanding of the following terminology and notations:[1]

$$\text{configuration } <S;T> = <s_1, \ldots, s_N; t_1, \ldots, t_M>$$

where:

- N is the total number of processes and M is the total number of message types.
- local state s_i is the current state of process i.
- message counter t_k is the number of type-k messages.

As illustrated in Figure 2, the node table consists of all the nodes in the reachability tree. They are ordered according to their node numbers, from node 0 (root node) to the last node. Each node entry in the node table is composed of three parts:

<parent node number><transition><global state>

For example, node 1 in Figure 2:

0(P1-1)1010000

represents the transition "process 1 transmits message 1" initiated at node 0 resulting in a global state 101000 (corresponding to node 1).

The general configuration of the global states (i.e. node representation) is printed in the first line of the node table. In this example, the configuration <s1,s2;t1,t2;t3,t4,t5> is used. To avoid confusions, the local states and message types are limited to a maximum number of 35. The numbers from 10 to 35 are printed as A to Z, respectively. Thus, a global state having s1=11 and s2=2 will be printed with s1,s2=B2 rather than s1,s2=112 which is ambiguous. The message type associated with each message counter is also printed as in Figure 2. Also for easier reading, if a message counter is zero, it is represented by a "." instead of a "0" so that the counters with positive values become more obvious.

---

[1] Detailed explanations of the terminology and notations can be found in [Vuong86,83, Hui85].

When the R-state approach is used, there will be an extra set of state pointers — the r-state pointers — included in the configuration (i.e.<S;R;T>). Each message channel, in this case, is associated to a r-state pointer. A r-state pointer indicates the next message to be received from the corresponding channel by pointing to the state (in the sending process) where the message was sent.

As a final note, if the symbol ω appears in a message counter, the message counter is unbounded and so is the corresponding channel.

III.C. The error summary

The error summary is self-explanatory. However, some attention should be paid to the stable-state table. If there exists a column in the stable-state table such that two entries containing the same state, then state ambiguity occurs. For example, the pair (0,0) and (0,3) in the stable-state table shown in Figure 2 indicates that state 0 in process 1 can coexist stably with either state 0 or state 3 in process 2. The above observation can be extended to the multi-process case.

==========

APPENDIX A

Example 1 - Protocol with Priority Channels

For protocols with priority channels, the program will ask for the priorities for all the messages from all the processes. The priority of a message is assigned by giving a positive integer when asked for the priority — the larger the number, the higher the priority.

```
*****************************************

        VALIRA version 1.21 (1988)

      Department of Computer Science
      University of British Columbia

*****************************************

Default settings :

   FIFO and conventional approach
   number of processes     =    2
   channel bound           =    3
   node limit of the tree = 1000

Do you want to change any of the above ?  [y|n|q] y

Number of processes : 2

Priority channels ?  [y|n|q] y

Channel bound : 3

Node limit of the tree : 100


Enter transitions for process 1 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 messag

(terminate with a null line)
>0 1 -1
>1 0 1
>1 2 -2
>2 1 2
>2 3 1
>3 0 2
>
Enter edit command (or type h for help)
:s
Please enter priority for each message in Process 1:
Priority of message 1 ? 1
Priority of message 2 ? 2
```

```
Enter transitions for process 2 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 messag

(terminate with a null line)
>0 1 1
>1 0 -1
>1 3 2
>3 1 -2
>0 2 2
>2 0 -2
>2 3 1
>
Enter edit command (or type h for help)
:s
Please enter priority for each message in Process 2:
Priority of message 1 ? 1
Priority of message 2 ? 2

Do you want to view your input ?  [y|n|q] n


Do you want the reachabilty tree printed ?  [y|n|q] y


-------------------------------- level --------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

 0   1   2   3   4   (P1+2)[ 1]
                     5   6   7   8   9   10  11  (P1+2)[ 0]
                                     12  (P1+1)[10]
                                         13  14  (P1-2)[ 8]
                                                 (P1+1)[ 0]
                                 15  (P2-2)[ 5]
                             (P2-1)[14]
                         (P2-1)[13]
                     (P2+1)[15]
             (P2+1)[ 6]

There are a total of 15 different node(s)
Do you want the node table ?  [y|n|q] y


Node configuration :
(s1,s2;t1,t2;t3,t4)

Message counter :  t1  t2  t3  t4
Message type    :   1   2   1   2

  0|    (root)00....   0(P1-1)101...   1(P1-2)2011..   2(P2+2)221...
  4|   3(P2-2)201..1   4(P2+1)21...1   5(P1+2)11....   6(P1-2)21.1..
  8|   7(P2-1)20.11.   8(P1+1)30.1..   9(P2+2)32....  10(P2-2)30...1
 12|   8(P2+2)22..1.  12(P2-2)20..11  13(P1+2)10..1.   7(P2+2)23....
```

```
Hit return for the summary
```

E R R O R S U M M A R Y

```
1. No state deadlock.

2. Stable state(s) :-

   P1  P2
  --------
    0   0     (node 0)
    1   1     (node 6)
    3   2     (node 10)
    2   3     (node 15)

3. No non-executable interaction.

4. No unspecified reception.

5. The reachability tree is bounded at level 11.

6. Max channel queue length = 2.


Choose one item from the list below :
1. Print the reachability tree
2. Print the node table
3. Print the summary
4. Edit transitions
5. Run
6. Quit

Item number ? 6


Total CPU time     :  0.140000 second(s)
Total System Call  :  0.040000 second(s)


                      ***** End of execution *****
```

APPENDIX B

Example 2 - Protocol with 3 processes

This example shows a protocol with more than 2 processes and the protocol is validated with the R-approach. Note that the behaviour of the program varies with the validation method for multi-process protocols. For FIFO protocols under R-approach validation or non-FIFO protocols, multi-casting is performed (though there may be excessive unspecified receptions reported). For FIFO protocols under conventional validation or protocols with priority channels, no multi-casting is performed; each message can have only one destination, namely the process (excluding the process sending the message) with the smallest process number.

```
****************************************

        VALIRA version 1.21 (1988)

      Department of Computer Science
      University of British Columbia

****************************************

Default settings :

   FIFO and conventional approach
   number of processes     =    2
   channel bound           =    3
   node limit of the tree = 1000

Do you want to change any of the above ?  [y|n|q] y

Number of processes : 3

Priority channels ?  [y|n|q] n

FIFO channels ?  [y|n|q] y

Do you want the R-state approach ?  [y|n|q] y

Node limit of the tree : 100


Enter transitions for process 1 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 messag

(terminate with a null line)
>0 1 -1
>1 0 2
>1 2 3
>2 0 -4
>
Enter edit command (or type h for help)
```

```
:s

Enter transitions for process 2 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 message.

(terminate with a null line)
>0 1 -5
>1 0 6
>1 2 7
>2 0 -4
>
Enter edit command (or type h for help)
:s

Enter transitions for process 3 :-
   Sample input :  0  1 -1
   Meaning      : state 0 to state 1 with the transmission of type 1 message.

(terminate with a null line)
>0 1 1
>1 0 -2
>1 1 5
>1 3 -3
>0 2 5
>2 0 -6
>2 2 1
>2 3 -7
>3 3 1
>3 3 5
>3 0 4
>
Enter edit command (or type h for help)
:s

Do you want to view your input ?  [y|n|q] n

Do you want the r-equivalent states (for advanced users only) ?  [y|n|q] y


Process states :-
process 1 :
to process 2
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  1  2  ;
   1=1        1  0  2  ;
   2=1        2  0  1  ;
to process 3
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  ;  (1,1)
   1=1        1  0  2  ;  (4,0)  (1,1)
   2=2        2  ;  (4,0)

process 2 :
```

```
to process 1
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  1  2  ;
   1=1        1  0  2  ;
   2=1        2  0  1  ;
to process 3
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  ;  (5,1)
   1=1        1  0  2  ;  (4,0)  (5,1)
   2=2        2  ;  (4,0)

process 3 :
to process 1
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  1  2  3  ;  (3,0)  (2,0)
   1=1        1  ;  (3,0)  (2,0)
   3=0        3  0  1  2  ;  (3,0)  (2,0)
   2=0        2  0  1  3  ;  (3,0)  (2,0)
to process 2
   state    ( rstates ; %r(mess,nextr) )
   0=0        0  1  3  2  ;  (7,0)  (6,0)
   1=0        1  0  2  3  ;  (7,0)  (6,0)
   3=0        3  0  1  2  ;  (7,0)  (6,0)
   2=2        2  ;  (7,0)  (6,0)

Do you want the reachabilty tree printed ?  [y|n|q] y


-------------------------------- level --------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

 0   1   2   3   4   5   (P1-1)[ 2]
                         6   7   8   (P2+6)[ 1]
                                     9   10  (P2-5)[ 3]
                                             11  (P1+2)[ 0]
                                                 (P2-5)[ 4]
                                             12  13  14  15  16  17  18

-------------------------------- level --------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 19  20  21  (P1-1)[18]
     22  23  24  (P1-1)[17]
                 (P3+4)[21]


-------------------------------- level --------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

                                                             (P3+4)[ 2]
                                                         (P3+4)[ 1]
                                                     25  (P1-1)[16]
```

```
                                                                (P3+5)[24]
                                                                (P3+4)[ 5]
                                                            (P3+4)[ 0]
                                                        26  (P1-4)[25]
                                                            (P3+5)[23]
                                                    27  (P1+3)[26]
                                                        (P3+5)[22]
                                            28  29  (P1-1)[ 8]
                                                    (P2+6)[ 0]
                                                30  (P1+2)[ 0]
                                                    31  (P1+2)[ 5]
                                                        32  (P1+2)[ 6]
                                                            33  34  35  (P2+6)[
                                                                        36

---------------------------------- level ----------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 (P2+6)[10]
 (P3-2)[33]
 37  38  39  40  (P2+6)[15]
                 (P3+4)[35]
             (P2+6)[14]
             (P3+4)[34]
         (P2+6)[13]
     41  (P1+3)[13]
         42  (P1+3)[26]
             43  (P1+3)[23]


---------------------------------- level ----------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

                                                                (P2+6)[ 0]
                                                            (P2+6)[30]
                                                        44  45  46  47

---------------------------------- level ----------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 48  49  50  51  52  53  54  (P2-5)[51]
                         55  56  57  (P2-5)[50]
                                     (P3+4)[54]
         (P3+4)[ 2]
     (P3+1)[57]
     (P3+4)[ 1]
 (P3+1)[56]


---------------------------------- level ----------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17
```

```
---------------------------------- level ----------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 (P2+7)[56]


---------------------------------- level ----------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

                                                                59  (P1-1)[
                                                                    60


---------------------------------- level ----------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 (P1-1)[48]
 61  (P1-1)[49]
     (P3+4)[ 5]
 (P3+4)[ 0]


---------------------------------- level ----------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

                                                          62  (P1+2)[59]
                                                          63  (P1+2)[60]
                                                              64

---------------------------------- level ----------------------------------
|18 |19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35

 (P1+2)[61]
 (P3+4)[31]


---------------------------------- level ----------------------------------
|0  |1  |2  |3  |4  |5  |6  |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17

                                                                (P3+4)[

                                        65  66  67  68  (P2+6)[15]
                                                        (P3+4)[ 8]
                                                (P2+6)[14]
                                                (P3+4)[29]
                                            (P2+6)[13]
                                        (P2+6)[41]
                                    69  (P2+7)[47]
                                        (P3+1)[55]
                                    (P3+1)[52]
                                (P3-6)[29]
                                70  (P1-1)[69]
                                    (P2+7)[59]
                        71  (P1+2)[ 6]
                            72  (P1+2)[34]
                                (P2+6)[11]
```

```
                                73  (P1+2)[45]
                                    74  (P1+2)[59]
                                        75  (P1+2)[60]
                                            76  (P1+2)[61]
                                                (P3+4)[ 4]
                                            (P3+4)[11]
                        (P3-3)[27]
                        (P3+5)[19]
                    (P3+5)[ 7]
                (P3+1)[10]
            (P2-5)[ 5]

There are a total of 76 different node(s)
Do you want the node table ?  [y|n|q] y


Node configuration :
(s1,s2,s3;r12,r13;r21,r23;r31,r32;t1,t2;t3,t4;t5,t6,t7,t8)

Message counter :  t1  t2  t3  t4  t5  t6  t7  t8
Message type    :   1   4   5   4   2   3   6   7

  0|    (root)000000000.......   0(P1-1)1000000001.......
  2|   1(P2-5)1100000001.1.....   2(P3+1)111000000..1.....
  4|   3(P3-2)110000010..1.1...   4(P1+2)010000000..1.....
  6|   5(P3+5)012000000.......   6(P1-1)1120000001.......
  8|   7(P3-6)1100000021.....1.   8(P3+1)111000002......1.
 10|   9(P2+6)101000000.......  10(P3-2)100000010....1...
 12|  10(P3-3)103000010.....1..  12(P1+3)203000000.......
 14|  13(P1-4)003020000.1.....  14(P1-1)10302000011......
 16|  15(P2-5)113020000111.....  16(P3+5)11302000011......
 18|  17(P3+4)1100000001.......  18(P3+1)111000000.......
 20|  19(P3-2)110000010....1...  20(P1+2)010000000.......
 22|  19(P3-3)113000010.....1..  22(P1+3)213000000.......
 24|  23(P1-4)013020000.1.....  14(P2-5)013020000.11.....
 26|  13(P2-5)213000000..1.....  12(P2-5)113000010..1..1..
 28|   9(P3-2)110000002....1.1.  28(P1+2)010000002......1.
 30|  28(P2+6)100000000....1...  30(P2-5)110000000..1.1...
 32|  31(P3+5)112000000....1...  32(P3-6)110000000....1.1.
 34|  33(P1+2)010000000......1.  34(P1-1)1100000001.....1.
 36|  35(P3+1)111000000......1.  36(P3-3)113000000.....11.
 38|  37(P1+3)213000000......1.  38(P1-4)013020000.1....1.
 40|  39(P1-1)11302000011....1.  37(P2+6)103000000.....1..
 42|  41(P2-5)113000000..1..1..  42(P3+5)113000000.....1..
 44|  32(P3-7)113000000....1..1  44(P1+2)013000000.......1
 46|  45(P1-1)1130000001......1  46(P2+7)1230000001.......
 48|  47(P2-4)1030002001..1....  48(P2-5)1130002001.11....
 50|  49(P3+1)113000200..11....  50(P3+4)110000000..1.....
 52|  51(P3+5)112000000.......  52(P3-6)110000002......1.
 54|  53(P2+6)100000000.......  52(P3-7)113000002.......1
 56|  55(P2+7)123000000.......  56(P2-4)103000200...1....
 58|  46(P3+1)113000000.......1  45(P2+7)023000000.......
 60|  59(P2-4)003000200...1....  60(P2-5)013000200..11....
 62|  44(P2+7)123000000....1...  62(P2-4)103000200...11...
```

```
 64|   63(P2-5)113000200..111...    9(P3-3)113000002.....11.
 66|   65(P1+3)213000002......1.   66(P1-4)013020002.1....1.
 68|   67(P1-1)11302000211....1.    7(P3-7)1130000021......1
 70|    6(P3-7)013000002.......1    4(P3+5)112000010....1...
 72|   71(P3-6)110000010....1.1.   71(P3-7)113000010....1..1
 74|   73(P2+7)123000010....1...   74(P2-4)103000210...11...
 76|   75(P2-5)113000210..111...
```

Hit return for the summary

E R R O R S U M M A R Y

1. No state deadlock.

2. Stable state(s) :-

P1	P2	P3	
0	0	0	(node 0)
0	1	2	(node 6)
1	0	1	(node 10)
2	0	3	(node 13)
1	1	1	(node 19)
0	1	0	(node 21)
2	1	3	(node 23)
1	1	2	(node 52)
1	0	0	(node 54)
1	2	3	(node 56)
0	2	3	(node 59)

3. No non-executable interaction.

4. No unspecified reception.

5. The reachability tree is bounded at level 25.

```
Choose one item from the list below :
1. Print the reachability tree
2. Print the node table
3. Print the summary
4. Edit transitions
5. Run
6. Quit

Item number ? 6


Total CPU time     :  1.020000 second(s)
Total System Call  :  0.080000 second(s)
```

VALISYN User's Manual - Version 1.2 (1988)

Computer Science Department
University of British Columbia
Vancouver, B.C., CANADA V6T 1W5

1. INTRODUCTION

VALISYN (VALIdator-SYNthesizer) is a software package which assists its user in synthesizing and validating communication protocols between two communication finite state machine (CFSM) by applying production rules [Zafi80], assuming error-free FIFO channels. VALISYN makes sure that the synthesized protocols do not contain state-deadlocks, unspecified receptions, and non-executable interactions, and will report on these errors plus the existence of state ambiguities if they occur on the original inputted protocol. As VALISYN and VALIRA handles the same protocol design errors, they may substitute one another in a number of typical cases, e.g. validation of two-process protocols. But their different features and modes of of operation make them complementary to one another in many aspects. For example, VALIRA should be used to validate multi-process protocols, whereas for interactive design of protocols, VALISYN should be employed to give the users better insight to the potential design errors at hand.

The VALISYN package provides the following features:

1. Two modes of operation: interactive and non-interactive. In the interactive mode, a user can either design a correct protocol by interacting with the package, or validate an existing protocol by interactively synthesizing the correct version of the given protocol. In the non-interactive mode, validation of an existing protocol can be performed by entering the entire protocol specification altogether.
2. An erase feature which allow erasure of any previously entered transmission arcs, for the interactive mode.
3. A timer to enable termination of the program in case of an unbounded protocol.
4. Two levels of debugging information to reveal the underlying mechanism of the program. This feature is useful for further development work on VALISYN, and should not concern a typical user.

VALISYN is easy to use. The program guides the user through the whole process of synthesis or validation of a protocol. The following sections of the manual give a brief explanation on how to operate the program and the format of both input data and output.

2. RUNNING VALISYN

2.1 Starting VALISYN

The program currently runs on SUN workstation and IBM PC, and is easily portable to any machine supporting C. Its source consists of over 11,000 lines of C code (occupying over 600Kbytes) and its executable object code is around 164Kbytes.

On the faculty SUN-3/160 machine, it can be invoked with the command

```
~vuong/VALISYN/package
```

The heading of the program will appear as follows:

```
****************************************

        VALISYN version 1.2 (1988)

        Department of Computer Science
        University of British Columbia

****************************************
```

which indicates that the program has started.

The program first asks for the mode of operation. Depending on the user's choice, it then accepts descriptions of the protocol in the form of transition arcs. Upon completion of the input process, the synthesized protocol and an error summary are produced.

2.2 Debugging information

VALISYN has the ability to provide debugging information if the user is interested in or wants to verify the internal mechanism of the program. Thus the following request will appear:

```
Do you want debugging information  [y|n]
-
```

Under normal operation, the answer should be "n", for the program slows down significantly with the debugging feature turned on. Answer "y" if debugging information is wanted.

There are two levels of debugging information. The first level shows the rule or replication applied. For example, the following may appear in the output if level 1 debugging information is turned on:

```
Rule 1B is called
```

The second level shows every arc (or branch of a tree, the internal representation of a process) generated and where applied, the reason for its generation. For example

```
2 1.1 2.0 0.3 3.1
```

means that for the tree of process 2, the reception arc +2.0 with subscript 3.1, which departs from state node 1.1 to entry state node 0.3, is created.

Thus the program will give one more request:

```
Level 1, level 2 or both  [1|2|b]
-
```

The user should choose either "1", "2" or "b" (for both) depending on his/her need.

For more information on the internal representation of processes and production rules, the reader is referred to [Zafi80] and [Tong85].

2.3 Modes of operation

The program has two modes of operation: interactive and non- interactive.

The interactive mode allows the user, with the help of the program, to design protocols which contain no state deadlocks, unspecified receptions, and non-executable interactions. The user needs only to enter the transmission arcs for the protocol. The program will try to generate the reception arcs according to production rules. When a new reception arc is generated, the user will be asked to enter the entry state for the arc, based on his semantic consideration. In the case of protocol validation, the user enters the transmission arcs and the entry states for the generated reception arcs according to the given protocol specification. The validation is realized by comparing the synthesized correct version with the given one.

The non-interactive mode accepts both transmission arcs and reception arcs for both processes, i.e. the entire protocol specification to be validated is entered at the beginning. It then works in the same way as the interactive mode except that the process of resynthesizing the correct protocol and checking it with the given one is automated and is executed without the user's intervention and "babysitting" of the terminal.

The operating mode can be chosen by answering the following request:

```
Do you want Interactive or Non-interactive    [i|n]
-
```

2.3.1 Interactive mode

In the interactive mode, the user will see the following prompt:

```
Please enter a transmission event (or type h for help)
-
```

At this point, either a command or a transmission arc is entered. The acceptable commands are:

a. "h" or "help"

This command brings out the list of valid values for all the entries in a transmission arc, and other legal commands.

b. "erase"

This command allows the erasure of previously entered transmission arcs and the reception arcs generated for them. The user can either erase a transmission arc by entering the arc, as described above, or by entering "last", which instructs the program to erase the last transmission arc entered. The "help" command can be used when in doubt, and the "q" command can be used to leave the erase mode.

c. "end"

This command signifies the end of data entry. The program will then produce a complete list of transmission and reception arcs for both processes and an error summary.

To enter a transmission arc, the following format is used:

```
[process number], [departure state] [entry state] [transmission]
```

For example

```
1,0 1 -1
```

means that process 1 moves from departure state 0 to entry state 1 by transmitting message 1; or, in a graphical form

```
Process 1

   -1
0 ----> 1
```

After receiving a transmission arc from the user, the program will prompt the user for semantic information (i.e. entry states) needed for generating reception arcs according to production rules. A prompt similar to the ones below will appear:

```
2, 0  ?  +1    enter the "entry state" (or type h for help)
-
```

or

```
2, 3  ?  +5(4)    enter the "entry state" (or type h for help)
-
```

The user is required to enter the entry state (i.e. the state at the question mark) for the reception arc. For reception arcs, a number in a bracket represents the occurrence of a collision in the reception. In the example above, message 5 collides (crosses) with message 4 (i.e. message 5 is received in process 2 while message 4 is "in transit" to process 1). After all the reception arcs are generated for the transmission arc, the user can view the stable states by answering "y" to the following request:

```
Do you want to display the stable state pairs    [y|n]
-
```

The structure of stable state tables is explained in Section 3.2 of the manual.

The program will again ask for transmission arcs until the command "end" is received. At that time, the following request will appear:

```
Do you want the list of executed transition arcs  [y|n]
-
```

The whole listing of all the arcs will be printed if "y" is received.

An error summary is then printed. Since protocols generated with the help of interactive mode are almost error-free, the summary is only a subset of the one generated in the non-interactive mode. Further details are presented in Section 3.3.

2.3.2 Non-interactive mode

When non-interactive mode is chosen, the following request will appear:

```
Do you want to have the default CPU limit 50 sec.      [y|n]
-
```

The reason for setting a time limit is obvious -- to avoid running the program infinitely for certain large and complex protocols. To change the default time, answer "n" and enter the required time.

The program is now ready to receive the lists of transmission and reception arcs for both processes. There are a total of 4 lists: a transmission list and a reception list for each of the two processes. Each list consists of arcs on separate lines. A blank line ends the list. An example of a transmission list is

```
0 0 -1
2 0 -4
                    <- Note: blank line
```

The format for an arc is:

```
[departure state] [entry state] [message]
```

Transmission message is represented by a negative message number. Reception message is represented by a positive message number of which the plus sign "+" can be neglected. For example, 0 0 1 is a reception arc, but 0 2 -2 is a transmission arc.

A prompt of the form

```
Please enter the transmission arc(s) for process 1
(end with a NULL line)
-
```

will appear to inform the user which list should be entered. After entering the four lists, a request will appear:

```
Do you want the list of executed transition arcs  [y|n]
-
```

The listing of all the arcs for the (re)synthesized protocol will be printed if "y" is received. An error summary is then printed. Please refer to Section 3 for details on output formats.

2.4 Limits for input

For both interactive and non-interactive mode, there are limits to the values of input which are considered valid. The range of valid values are listed below:

a. Process number : 1 and 2

b. Departure states and entry states : 0 to 98

c. Transmission arcs : -1 to -58

d. Reception arcs : 1 to 58

For non-interactive mode, the dummy state 99 for a process may appear in the output (stable state table and error summary). This dummy process state serves as the entry state for all unspecified receptions occurred in this process during the resynthesis (validation) process. This dummy state "absorbs" all the unspecified receptions to allow the synthesis (the validation) to execute until completion.

2.5 Special cases

As stated in Section 1, a timer is provided so that the program will stop automatically if a certain time limit is exceeded.

In interactive mode, if a protocol is suspected to be unbounded or it is very large and complex, the program will be interrupted by the timer and the user will be informed of the situation

```
Total CPU time     : 15.880000 second(s)
Total System Call :  0.160000 second(s)

*** Warning -- the protocol may be unbounded
```

The user can carry on with the synthesis if (s)he believes that the protocol is valid. If not, (s)he can stop the synthesis by answering "n" when the following occurs:

```
Do you want to carry on with the synthesis    [y|n]
-
```

In non-interactive mode, the program will stop with the same warning as above. A sample run of an unbounded protocol in interactive mode is included in Section 4 of the manual.

3. OUTPUT

The outputs of VALISYN include the list of all arcs in the final protocol (called the arcs list), stable state tables, and an error summary.

Special attention must be paid to the dummy process state 99. This state number is larger than the highest process state number allowed to be inputted. This dummy state is used exclusively by the program to act as the entry state for unspecified receptions. Reported errors (unspecified receptions, state deadlocks and state ambiguities) involving state 99 should, therefore, all be ignored.

3.1 Arcs list

The list consists of six sub-lists, three for each process:

a. The list of transmission arcs.

b. The list of reception arcs.

c. The list of reception arcs with collision.

An example of an arcs list is

```
Process 1 has the following transmission arc(s)
2 -> 0   -5
0 -> 1   -1
3 -> 1   -6

Process 1 has the following reception arc(s)
1 -> 0   +2
1 -> 2   +3
2 -> 3   +4

Process 1 has the following reception arc(s) with collision
0 -> 0   +4(5)
1 -> 1   +4(1)


Process 2 has the following transmission arc(s)
1 -> 0   -2
1 -> 2   -3
2 -> 3   -4

Process 2 has the following reception arc(s)
2 -> 0   +5
0 -> 1   +1
3 -> 1   +6

Process 2 has the following reception arc(s) with collision
3 -> 0   +5(4)
0 -> 1   +1(4)
```

The general form for an arc is

```
[departure state] -> [entry state] [message] ([collision])
```

The [collision] field will only appear in sub-list 3. For example, the arc

```
1 -> 2   +3(4)
```

means that by receiving 3 (in collision with 4), the state changes from 1 to 2.

Again positive messages indicate reception, while negative ones indicate transmission.

3.2 Stable state table

A stable state table shows all the stable state pairs in a matrix form. For example

```
          Stable States
   Process 1    Process 2
```

	0	1	2
0	1	0	0
1	1	1	0
2	0	0	1

The row represents the states for process 1, and the column represents the states for process 2. A "1" in the matrix means that the corresponding pair of states forms a stable state pair. For example, the above table shows that there are 4 stable state pairs: (0,0), (1,0), (1,1) and (2,2). Note that if the sum of all the numbers (either 1 or 0) in any column or row is greater than 1, state ambiguity is present; for example, the first column and the second row of the matrix.

3.3 Error summary

The error summary summarizes all the errors or potential errors in a protocol. The error summary for a protocol generated using the interactive mode is less detailed than the one generated using non-interactive mode because in the former, the synthesized protocol is guaranteed to contain no errors such as unspecified reception. An example of the error summary generated using non-interactive mode is given below.

```
E R R O R   S U M M A R Y
═══════════════════════════

1.  The list of deadlock state(s) :-
    format : ( s1, s2)

    ( 1, 0)     ( 1,99)     ( 3, 0)     (99, 0)


2.  The stable state(s) table :-

          Stable States
    Process 1     Process 2
                  0   1   2  99

              0   1   1   0   1
              1   1   1   0   1
              2   1   1   1   1
              3   1   0   1   0
             99   1   1   1   0


3.  No non-executable transmission arc.


4.  Non-executable reception arc(s) :-
    format :    s -> s'  reception (collision)

    process  1 :     3 -> 0  +2


5.  Unspecified reception arc(s) :-
    format :    s -> s'  reception (collision)
```

```
process  1 :    0 ->99  +2 ( 4)      3 ->99  +5 ( 4)
                2 ->99  +2           2 ->99  +3
                99->99  +2 ( 4)      99->99  +3 ( 4)
                99->99  +5

process  2 :    1 ->99  +1           2 ->99  +1 ( 3)
                99->99  +4
```

Note that the representation of arcs is the same as the ones in the "Arcs list".

The deadlock nodes are represented by an ordered pair (s1, s2), which means that a deadlock state occurs when process 1 is at state s1 and process 2 is at s2. They are listed in Section 1 of the Error Summary. Section 2 contains the stable state table as explained previously.

Non-executable arcs (either for transmission or reception) refer to the arcs that are entered by the user but cannot be executed. They are listed in Sections 3 and 4.

When reception arcs are not specified for some of the executable transmission arcs, those arcs are generated by VALISYN so that every executable transmission arc has at least one reception arc in the other process. These arcs are listed in Section 5.

The user is reminded to ignore all errors involving dummy state 99 which is introduced merely to absorb all unspecified receptions.

To illustrate the usage and response of VALISYN, three sample runs are provided in the Appendix for the case of (i) interactive mode, (ii) non-interactive mode, and (iii) potential unbounded channel.

APPENDIX

SAMPLE RUNS

Three sample runs of VALISYN are included in this appendix to illustrate the usage and response of the package under various operating modes and situations: (i) interactive mode, (ii) non-interactive mode, and (iii) potential unbounded channel.

A.1 Interactive Mode

```
*****************************************

        VALISYN version 1.2 (1988)

      Department of Computer Science
      University of British Columbia

*****************************************

Do you want debugging information  [y|n]
n

Do you want Interactive or Non-interactive   [i|n]
i

Please enter a transmission event (or type h for help)
h

format : process no , dept. state  entry state  trans. arc

         e.g. 1, 2 3 -4

  Where  process no  :  1 to 2
         dept. state :  0 to 98
         entry state :  0 to 98
         trans. arc  : -1 to -58

  enter END   to end the process
  enter ERASE to erase a transmission arc

Please enter a transmission event (or type h for help)
1,0 1 -1

2, 0  ?  +1    enter the "entry state" (or type h for help)
1

Do you want to display the stable state pairs    [y|n]
y

          Stable States
```

```
    Process 1     Process 2
                  0  1

              0   1  0
              1   0  1

Please enter a transmission event (or type h for help)
2,1 2 -3

1, 1  ?  +3     enter the "entry state" (or type h for help)
h

Valid entry states : 0 to 98

1, 1  ?  +3     enter the "entry state" (or type h for help)
2

Do you want to display the stable state pairs     [y|n]
y

          Stable States
    Process 1     Process 2
                  0  1  2

              0   1  0  0
              1   0  1  0
              2   0  0  1

Please enter a transmission event (or type h for help)
2,2 3 4

*** Illegal transmission arc  4  (valid input : -1 to -58)

Please enter a transmission event (or type h for help)
2,2 3 -4

1, 2  ?  +4     enter the "entry state" (or type h for help)
3

Do you want to display the stable state pairs     [y|n]
n

Please enter a transmission event (or type h for help)
erase

Enter the transmission arc to be ERASED (or type h for help)
2,1 2 -3

Do you really want to ERASE the arc     [y|n]
y

Transmission arc   2, 1  2  -3   has been removed

Do you want to display the stable state pairs     [y|n]
```

```
y

          Stable States
    Process 1    Process 2
                 0  1

              0  1  0
              1  0  1

Do you want the list of executed transition arcs  [y|n]
y


Process 1 has the following transmission arc(s)
0 -> 1  -1

Process 1 has the following reception arc(s)

Process 1 has the following reception arc(s) with collision



Process 2 has the following transmission arc(s)

Process 2 has the following reception arc(s)
0 -> 1  +1

Process 2 has the following reception arc(s) with collision


Please enter a transmission event (or type h for help)
2,1 2 -3

1, 1  ?  +3    enter the "entry state" (or type h for help)
2

Do you want to display the stable state pairs    [y|n]
n

Please enter a transmission event (or type h for help)
2,2 3 -4

1, 2  ?  +4    enter the "entry state" (or type h for help)
3

Do you want to display the stable state pairs    [y|n]
n

Please enter a transmission event (or type h for help)
2,1 0 -2

1, 1  ?  +2    enter the "entry state" (or type h for help)
0
```

```
Do you want to display the stable state pairs     [y|n]
n

Please enter a transmission event (or type h for help)
1,3 1 -6

2, 3  ?  +6     enter the "entry state" (or type h for help)
1

Do you want to display the stable state pairs     [y|n]
n

Please enter a transmission event (or type h for help)
1,2 0 -5

2, 2  ?  +5     enter the "entry state" (or type h for help)
0

2, 3  ?  +5(4)     enter the "entry state" (or type h for help)
0

1, 0  ?  +4(5)     enter the "entry state" (or type h for help)
0

2, 0  ?  +1(4)     enter the "entry state" (or type h for help)
1

1, 1  ?  +4(1)     enter the "entry state" (or type h for help)
1

Do you want to display the stable state pairs     [y|n]
y

          Stable States
    Process 1     Process 2
                  0  1  2  3

              0   1  0  0  0
              1   0  1  0  0
              2   0  0  1  0
              3   0  0  0  1

Please enter a transmission event (or type h for help)
end

Do you want the list of executed transition arcs  [y|n]
y


Process 1 has the following transmission arc(s)
2 -> 0  -5
0 -> 1  -1
3 -> 1  -6
```

```
Process 1 has the following reception arc(s)
1 -> 0  +2
1 -> 2  +3
2 -> 3  +4

Process 1 has the following reception arc(s) with collision
0 -> 0  +4(5)
1 -> 1  +4(1)


Process 2 has the following transmission arc(s)
1 -> 0  -2
1 -> 2  -3
2 -> 3  -4

Process 2 has the following reception arc(s)
2 -> 0  +5
0 -> 1  +1
3 -> 1  +6

Process 2 has the following reception arc(s) with collision
3 -> 0  +5(4)
0 -> 1  +1(4)
```

ERROR SUMMARY

1. No deadlock state.

2. The stable state(s) table :-

	Stable States			
Process 1	Process 2			
	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

```
Total CPU time     :  0.860000 second(s)
Total System Call  :  0.380000 second(s)
```

A.2 Non-interactive Mode

```
*****************************************

        VALISYN version 1.2 (1988)

      Department of Computer Science
      University of British Columbia

*****************************************

Do you want debugging information  [y|n]
n

Do you want Interactive or Non-interactive   [i|n]
n

Do you want to have the default CPU limit 50 sec.    [y|n]
y

Format for inputting the arc :

e.g.  Transmission arc -- 1 2 -3
      i.e. from dept. state 1 to entry state 2 with trans. arc 3
      Note -- the FIRST trans. arc in Process 1 will be executed first

      Reception arc -- 1 2 3
      i.e. from dept. state 1 to entry state 2 with recpt. arc 3

Please enter the transmission arc(s) for process 1
(end with a NULL line)
0 0 1 -1
2 0 -4


Please enter the reception arc(s) for process 1
(end with a NULL line)
1 0 2
1 2 3
2 1 5
0 0 5
1 1 5
0 3 3
3 0 2


Please enter the transmission arc(s) for process 2
(end with a NULL line)
1 0 2
*** Illegal transmission arc  2  (valid input : -1 to -58)

1 0 -2
```

```
1 2 -3
2 1 -5


Please enter the reception arc(s) for process 2
(end with a NULL line)
2 0 4
0 0 4
1 0 4
0 1 1


Do you want the list of executed transition arcs  [y|n]
y


Process 1 has the following transmission arc(s)
2 -> 0  -4
0 -> 1  -1

Process 1 has the following reception arc(s)
1 -> 0  +2
2 -> 1  +5
1 -> 2  +3
0 -> 3  +3
0 ->99  +2
2 ->99  +2
2 ->99  +3
3 ->99  +5
99->99  +5

Process 1 has the following reception arc(s) with collision
0 -> 0  +5(4)
1 -> 0  +2(1)
1 -> 1  +5(1)
2 -> 1  +5(1)
1 -> 2  +3(1)
0 -> 3  +3(4)
0 ->99  +2(4)
3 ->99  +5(4)
99->99  +2(4)
99->99  +3(4)



Process 2 has the following transmission arc(s)
1 -> 0  -2
2 -> 1  -5
1 -> 2  -3

Process 2 has the following reception arc(s)
1 -> 0  +4
2 -> 0  +4
0 -> 1  +1
```

```
1 ->99  +1
99->99  +4

Process 2 has the following reception arc(s) with collision
0 -> 0  +4(2)
1 -> 0  +4(2)
1 -> 0  +4(3)
1 -> 0  +4(5)
2 -> 0  +4(3)
0 -> 1  +1(2)
0 -> 1  +1(3)
0 -> 1  +1(5)
1 ->99  +1(5)
2 ->99  +1(3)
99->99  +4(5)
```

E R R O R S U M M A R Y

1. The list of deadlock state(s) :-
   format : (s1, s2)

   (1, 0) (1,99) (3, 0) (99, 0)

2. The stable state(s) table :-

Process 1 \ Process 2	0	1	2	99
0	1	1	0	1
1	1	1	0	1
2	1	1	1	1
3	1	0	1	0
99	1	1	1	0

3. No non-executable transmission arc.

4. Non-executable reception arc(s) :-
   format : s -> s' reception (collision)

   process 1 : 3 -> 0 +2

5. Unspecified reception arc(s) :-
   format : s -> s' reception (collision)

```
process  1 :    0 ->99  +2 ( 4)      3 ->99  +5 ( 4)
                2 ->99  +2           2 ->99  +3
```

```
                    99->99  +2 ( 4)      99->99  +3 ( 4)
                    99->99  +5

    process  2 :    1 ->99  +1          2 ->99  +1 ( 3)
                    99->99  +4

Total CPU time     :  0.900000 second(s)
Total System Call  :  0.140000 second(s)
```

A.3 Potential Unbounded Channel

```
*****************************************

        VALISYN version 1.2 (1988)

        Department of Computer Science
        University of British Columbia

*****************************************

Do you want debugging information  [y|n]
n

Do you want Interactive or Non-interactive   [i|n]
i

Please enter a transmission event (or type h for help)
1,0 0 -1

2, 0  ?  +1    enter the "entry state" (or type h for help)
0

Do you want to display the stable state pairs    [y|n]
n

Please enter a transmission event (or type h for help)
1,0 0 -2

2, 0  ?  +2    enter the "entry state" (or type h for help)
0

Do you want to display the stable state pairs    [y|n]
n

Please enter a transmission event (or type h for help)
2,0 0 -1

1, 0  ?  +1    enter the "entry state" (or type h for help)
0

1, 0  ?  +1(1)    enter the "entry state" (or type h for help)
0

1, 0  ?  +1(2)    enter the "entry state" (or type h for help)
0

Do you want to display the stable state pairs    [y|n]
n

Please enter a transmission event (or type h for help)
12,0 0 -2
```

```
1, 0  ?  +2     enter the "entry state" (or type h for help)
0

1, 0  ?  +2(1)     enter the "entry state" (or type h for help)
0

1, 0  ?  +2(2)     enter the "entry state" (or type h for help)
0

Do you want to display the stable state pairs     [y|n]
y

          Stable States
    Process 1     Process 2
                  0

              0   1

Please enter a transmission event (or type h for help)
1,0 0 -3

2, 0  ?  +3     enter the "entry state" (or type h for help)
0

2, 0  ?  +3(1)     enter the "entry state" (or type h for help)
0

2, 0  ?  +3(2)     enter the "entry state" (or type h for help)
0

Total CPU time     : 15.880000 second(s)
Total System Call :  0.160000 second(s)

*** Warning -- the protocol may be unbounded

Do you want to display the stable state pairs     [y|n]
y

          Stable States
    Process 1     Process 2
                  0

              0   1

Do you want the list of executed transition arcs  [y|n]
y


Process 1 has the following transmission arc(s)
0 -> 0  -1
0 -> 0  -2
0 -> 0  -3

Process 1 has the following reception arc(s)
```

```
0 -> 0  +1
0 -> 0  +2

Process 1 has the following reception arc(s) with collision
0 -> 0  +1(1)
0 -> 0  +1(2)
0 -> 0  +1(3)
0 -> 0  +2(1)
0 -> 0  +2(2)
0 -> 0  +2(3)



Process 2 has the following transmission arc(s)
0 -> 0  -1
0 -> 0  -2

Process 2 has the following reception arc(s)
0 -> 0  +1
0 -> 0  +2
0 -> 0  +3

Process 2 has the following reception arc(s) with collision
0 -> 0  +1(1)
0 -> 0  +1(2)
0 -> 0  +2(1)
0 -> 0  +2(2)
0 -> 0  +3(1)
0 -> 0  +3(2)


Do you want to carry on with the synthesis   [y|n]
y

Total CPU time     : 26.460000 second(s)
Total System Call :  0.240000 second(s)

*** Warning -- the protocol may be unbounded

Do you want to display the stable state pairs     [y|n]
y

          Stable States
    Process 1    Process 2
                 0

             0   1

Do you want the list of executed transition arcs  [y|n]
y


Process 1 has the following transmission arc(s)
0 -> 0  -1
```

```
0 -> 0  -2
0 -> 0  -3

Process 1 has the following reception arc(s)
0 -> 0  +1
0 -> 0  +2

Process 1 has the following reception arc(s) with collision
0 -> 0  +1(1)
0 -> 0  +1(2)
0 -> 0  +1(3)
0 -> 0  +2(1)
0 -> 0  +2(2)
0 -> 0  +2(3)



Process 2 has the following transmission arc(s)
0 -> 0  -1
0 -> 0  -2

Process 2 has the following reception arc(s)
0 -> 0  +1
0 -> 0  +2
0 -> 0  +3

Process 2 has the following reception arc(s) with collision
0 -> 0  +1(1)
0 -> 0  +1(2)
0 -> 0  +2(1)
0 -> 0  +2(2)
0 -> 0  +3(1)
0 -> 0  +3(2)


Do you want to carry on with the synthesis   [y|n]
n

Do you want the list of executed transition arcs  [y|n]
n
```

E R R O R S U M M A R Y

1. No deadlock state.

2. The stable state(s) table :-

Stable States	
Process 1	Process 2
	0

```
              0    1

Total CPU time     : 28.780000 second(s)
Total System Call  :  0.300000 second(s)
```