4de th

SUID-AFRIKAANSE REKENAARSIMPOSIUM
SOUTH AFRICAN COMPUTER SYMPOSIUM

HOLIDAY INN PRETORIA
JULIE 1 – 3 JULY 1987

Proceedings

of the

4th South African Computer Symposium

Holiday Inn, Pretoria

1 – 3 July 1987

edited by

Pieter Kritzinger

Computer Science Department
University of Cape Town

PREFACE

Computer science is an emerging discipline which is having difficulty in being recognised as a worthy member of the sciences. I will paraphrase John Hopcroft, co-winner of the 1986 Turing Award, when, during a recent interview, he said that the primary reason for the lack of recognition, is the age of our researchers. Probably not one of the researchers who presented their work at this symposium is older than 45. I know of no computer scientist in South Africa who is in a position where (s)he can affect funding priorities. As far as I know we have no representation on any of the committees of the Foundation for Research Development and for our Afrikaans speaking fraternity, none who is a member of the *Akademie vir Wetenskap en Kuns*. It will take time and conscious effort to establish our presence. The same is true of course for our universities. Again, with one exception, I know of no dean of a science faculty, vice-principal or principal who is a computer scientist. We consequently spend an enormous amount of time trying to explain the needs of computer science and its difficulties. I believe this symposium is a further step towards accreditation by our peers and superiors from the other sciences.

The total number of papers submitted to the Programme Committee for consideration was 34. Each paper was reviewed by three persons knowledgeable in the field it represents. Of those submitted, 23 were finally selected for inclusion in the symposium. As a result the overall quality of the papers is high and as a computer science community in Africa we can be justly proud of the final programme.

This is the fourth in the series of South African computer symposia. This year the symposium is sponsored by the Computer Society of South Africa (CSSA), the South African Institute for Computer Scientists and the local IFIP Committee. The executive director of the CSSA and his staff deserve warm thanks for handling the organisation as well as they have, while the Organising Committee provided Derrick and I with very valuable advice.

Finally I would like to express my sincere appreciation to the authors, to the members of the Programme Committee and particularly the reviewers. Without the kind cooperation of everyone, this symposium would not have taken place.

Pieter Kritzinger
July 1987.

SYMPOSIUM CHAIRMAN: PS Kritzinger, University of Cape Town.

SYMPOSIUM CO-CHAIRMAN: D Kourie, University of Pretoria.

MEMBERS OF THE PROGRAMME COMMITTEE

Judy Bishop, Witwatersrand University

Chris Bornman, UNISA

Hannes de Beer, Potchefstroom University

Gideon de Kock, Port Elizabeth University

Jaap Kies, Western Cape University

Derrick Kourie, Pretoria University

Pieter Kritzinger, Cape Town University

Tony Krzesinski, Stellenbosch University

Michael Laidlaw, Durban Westville University

Peter Lay, Cape Town University

Ken MacGregor, Cape Town University

Theo McDonald, Orange Free State University

Jan Oosthuizen, University of the North

Dennis Riordan, Rhodes University

Alan Sartori-Angus, Natal University

John Shochot, Witwatersrand University

Theuns Smith, Rand Afrikaans University

Trevor Turton, ISM (Pty) Ltd

Gerrit Wiechers, Infoplan.

LIST OF REVIEWERS

BERMAN Sonia
BISHOP Judy
BORNMAN Chris
CAREY Chris
CHERENACK Paul
DE BEER Hannes
DE VILLIERS Pieter
GORRINGE Pen
KIES Jaap
KOURIE Derrick
KRITZINGER Pieter
KRZESINSKI Tony
LAIDLAW Michael
LAY Peter
MacGREGOR Ken
MATTISON Keith
McDONALD Theo
RENNHACKKAMP Martin
RIORDAN Denis
SATORI-ANGUS Alan
SCHOCHOT John
SMITH Theuns
TURTON Trevor
VAN DEN HEEVER Roelf
VAN ROOYEN Hester
VON SOLMS Basie
VOS Koos

TABLE OF CONTENTS

Computer Languages III

Tutorial

DAY 1

07h30 Registration and Coffee.

08h45 Welcoming address, President of the South African Institute of Computer Scientists, Dr. G. Wiechers.

09h00 Invited Lecture.
Professor D. Teichrow, University of Michigan.
Software Engineering, ... What Can We Expect in the Future.

10h00 COFFEE

Computer Languages I. Chairman: G. Wiechers.

10h15 S. Berman, University of Cape Town.
SPS-Algol: Semantic Constructs for a Persistent Programming Language.

10h50 A. Watson, University of the Witwatersrand.
Petri Net Topologies for a Specification Language.

11h25 R. Mori, University of Cape Town.
Towards a Programming Environment Standard in USP.

11h50 J. Bishop, University of the Witwatersrand.
ADA for Multiprocessors: Some Problems and Solutions.

12h30 LUNCH

	Computer Graphics. Chairman: D. Kourie	**Operating Systems.** Chairman: K. MacGregor.
14h00	C. F. Scheepers, CSIR. *Polygon Shading on Vector Type Devices.*	M. Morris, UNISA. *The Development of a Fault Tolerant System for a Real-time Environment.*
14h35	P. Gorringe, CSIR. *Hidden Surface Elimination in Raster Graphics Using Visigrams.*	B. H. Venter, CSIR. *A New General-purpose Operating System.*
15h15	COFFEE	COFFEE
	Database Systems I. Chairman: B. von Solms.	**Computer Languages II.** Chairman: J. Bishop.
15h30	M.E. Orlowska, UNISA. *On Syntax and Semantics Related to Incomplete Information Databases.*	E.M. Ehlers and B. von Solms, Randse Afrikaanse Universiteit. *The Representation of Chemical Structures by Random Context Structure Grammars.*
16h05	M.H. Rennhackkamp, Stellenbosch University. *Modelling Distributed Database Concurrency Control Overheads*	W. van Biljon, CSIR. *A Generalised Expression Structure.*

18h00 Cocktail Party in Cullinan Room A.

DAY 2

08h30 Keynote Address by Profesor Niklaus Wirth, Swiss Federal Institute for Technology, Zurich.
An Extensible System and a Programming Tool for Workstation Computers.

Computer Networks and Protocols I. Chairman: P.S. Kritzinger.

09h30 A.E. Krzesinski, University of Stellenbosch.
An Approximate Solution Method for Multiclass Queueing Networks with State Dependent Routing and Window Flow Control.

10h05 J. Punt, University of Cape Town.
A Protocol Validation System.

10h30 COFFEE

Computer Networks and Protocols II. Chairman: R. van der Heever.

11h00 P.S. Kritzinger, University of Cape Town.
Protocol Performance using Image Protocols.

11h35 Invited Lecture by Professor G. Lasker, University of Windsor, Ontario.
The Relationship of Natural and Artificial Intelligence.

12h30 LUNCH

	Artificial Intelligence. Chairman: G. Lasker.	**Information Systems.** Chairman: D. Teichrow.
14h00	A. Cooper, CSIR *A Data Structure for Exchanging Geographic Information.*	L. du Plessis and C. Bornman, UNISA. *The ELSIM Language: an FSM-based Language for the ELSIM SEE.*
14h35	A. I. Newcombe, University of Cape Town and R. Rada, National Library of Medicine, Maryland. *Strategies for Automatic Indexing and Thesaurus Building.*	J. Mende, University of the Witwatersrand. *Three Packaging Rules for Information System Design.*
15h15	COFFEE	COFFEE
	Database Systems II. Chairman: C. Bornman.	**Computer Languages III.** Chairman: N. Wirth.
15h30	M.J. Philips, CSIR. *An Approach to Direct End-user Usage of Mutiple Databases.*	W. van Biljon, CSIR. *Experience with a Pattern-matching Code Generator.*
16h05	S. Berman, University of Cape Town. *A Semantic Data Model Approach to Logical Data Independence.*	C. Mueller, University of the Witwatersrand. *Set-oriented Functional Style of Programming.*

16h45 Open Forum with professors G. Lasker, D. Teichrow and N. Wirth.
Moderator: Dr. D. Jacobson.

19h30 Symposium Banquet in Cullinan Room.
Guest speaker, Dr. D. Jacobson, Group Executive: Technology, Allied Technologies Limited.

DAY 3

08h00 Registration (Tutorial only).

08h30 Tutorial.
The Tutorial will be given by professor Niklaus Wirth, Division of Computer Science, Swiss Federal Institute of Technology, Zurich.

The use of Modula-2 in Software Engineering.
Topics to be covererd include:

What is Software Engineering?
Data types and structures.
Modularization and information hiding.
Definition and implementation parts.
Separate compilation with type checking.
Facilities to express concurrency.
Pompous programming style.
What could be excluded?

12h15 **Close of Symposium.**

12h30 LUNCH

A PROTOCOL VALIDATION SYSTEM

Janette L Punt
University of Cape Town

Abstract

This paper discusses a protocol validation system. The input to the system is a protocol definition specified in the specification language ESTELLE. The ESTELLE specification is the input to a translator program which extracts the protocol definition and creates an output file which serves as input to the validation routine.

The validation routine uses reachability analysis to validate the protocol. The validation routine reports the following conditions: unspecified receptions, deadlock, channel overflow and all those transitions not exercised during the validation. The routine output includes a trace of events, the set of all system states generated - all errors are marked, a summary of the error conditions, a reachability tree, etc. The validation routine was successfully applied to several smaller protocols as well as the CCITT X.21 protocol. The system is implemented on an IBM compatible PC.

This paper discusses the protocol validation system but with emphasis on the validation routine.

1. Introduction

Protocols are an important aspect of data communications. A protocol is a set of rules governing the interaction between separate processes [ZAFI82]. In order to ensure that a protocol is completely defined, that is without, deadlocks, endless looping, uncspecified receptions, etc. and that the functions performed by the protocol are according to their specification, it is necessary to validate or verify the protocol. However, validation is not the only step in the development process.

In order to deal with all the steps in the development process, the design approach should rely on the following:

- A common language or formal description technique (FDT) to express protocol and service specifications in such a way that the written specifications are complete and unambiguous.
- A validation methodology to analyse and predict the behaviour of a protocol layer during the design stage prior to implementation.
- An implementation methodology to produce in a semi-automatic way a reference implementation of the considered protocol layer (for testing purposes) [COUR86].

The aim of this paper is to present a validation methodology to be applied to a protocol specification written in the ISO subgroup B formal description technique known as ESTELLE (Extended State Transistion Language). A model is extracted from the ESTELLE protocol specification which serves as input to a validation routine. The validation routine uses the state exploration technique to validate the protocol.

In many works on protocol design and analysis the concepts of validation and verification are considered as equivalent or used interchangeably [PUNT86]. For the purpose of this paper validation is defined to be concerned with the determination whether or not the protocol is sound and its logical structure complete. Thus, validation of a protocol will determine whether or not the protocol is deadlock free, has no unspecified receptions, contains no transitions that will never be executed, etc. Verification is concerned with what the function of the protocol is and involves a comparison of particular aspects of the protocol behaviour with those intended by the designer [WEST78B].

The place of validation and verification in the protocol development process is illustrated in figure 1.1 [CAST85].

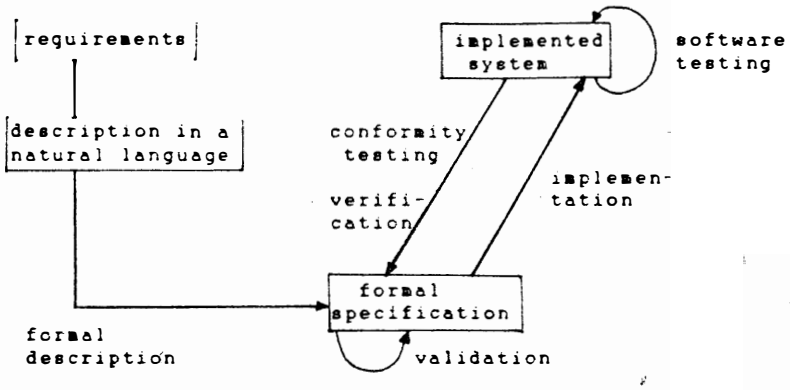


Figure 1.1 Protocol development process.

2. System Overview

The validation system consists of three parts as illustrated in figure 2.1. The first part permits the user to specify the protocol he wishes to validate. The user writes the protocol specification in ESTELLE. To simplify the translator program, some restrictions have been placed on the use of ESTELLE. The restrictions and guidelines of how to use ESTELLE to specify a protocol are discussed in section 3.

The second part of the validation system, the translator program, takes the ESTELLE specification as input and extracts the state names, process names, finite state machines in matrix format, etc. and writes this information as APL statements to a specification file. This specification file serves as input to the third part of the system, that is the part that performs the validation.

The validation routine produces a trace of events during the validation, the execution time (that is the CPU time to validate the protocol), the system states generated as well as the error conditions encountered. The error conditions that are reported are deadlock, unspecified receptions, channel overflow and all transitions that were not exercised during the validation process. The validation system can also perform other functions such as printing the protocol definition and summarizing the error conditions. In general the output is presented in such a way that the user may derive other information useful in understanding the system behaviour.

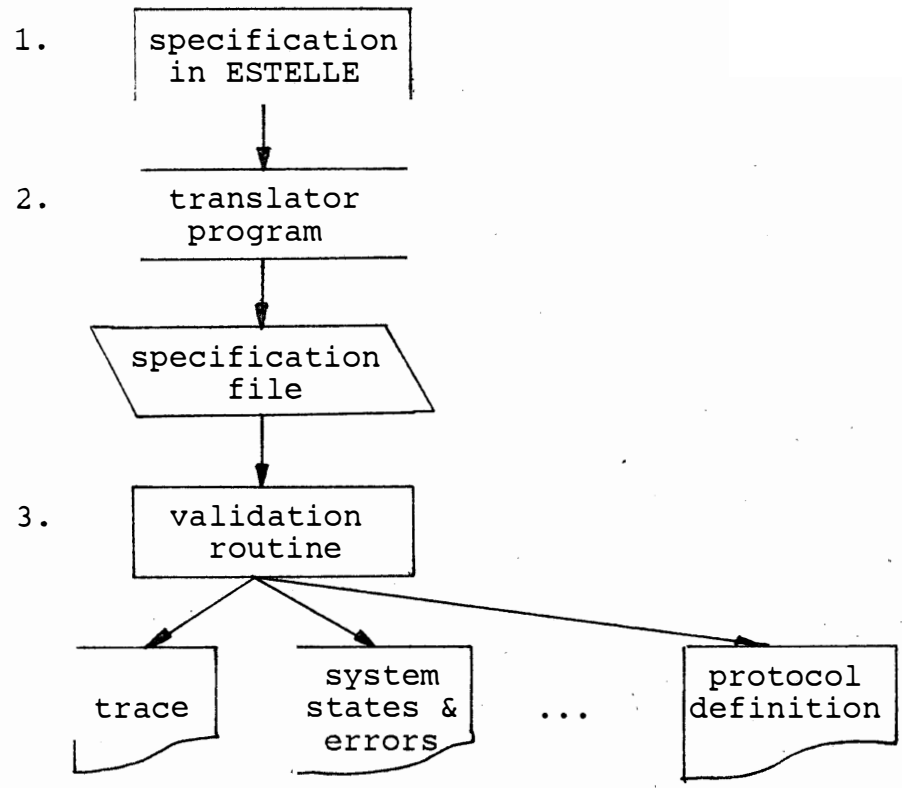


Figure 2.1 Layout of validation system

The translator program is written in PASCAL and the validation routine in APL. The ESTELLE specification, the translator program and the validation routine are now discussed.

3. Specifying the Protocol

An ESTELLE specification intended for the validation of a protocol consists of the description of a set of co-operating processes belonging to different subsystems interconnected by means of a lower layer service. From the ESTELLE point of view these protocol processes, as well as the lower layer service, are considered as co-operating modules [COUR86].

The **module** is the basic component of an ESTELLE description and is defined through a module header definition and a module body definition. The module header definition defines the interaction points through which the module may exchange interactions with its environment and the set of variables exported by the module. The module body definition describes

the internal behaviour of any module instance associated with this body in terms of a state transition model.

This state transition model is based on a finite state machine extended with the addition of variables to the states, parameters to the interactions, time constraints and priorities to the transitions. A transition or set of transitions is introduced by the keyword **trans**. The following fragment demonstrates the general format of a transition:

```
trans
  priority  expression      {relationship to other trans.}
  from      state_a         {current state}
  to        state_b         {next state}
  provided  predicate       {boolean expression}
  when      ip.event        {input required}
    begin

      {transition block}

    end


end
```

The **priority** and **provided** clauses are not allowed in this validation system. The **priority** feature is lost but the **provided** feature can be handled as follows. The transition "**from** state_a **to** state_b **provided** x ..." can be re-written as "**from** state_a' **to** state_b ..." where state_a' is a new state formed by combinding state_a and condition x. Note that combining states in this manner will result in more states and thus in a bigger finite state machine.

The **when** clause introduces an input interaction. Spontaneous transitions do not have an input associated with them, that is, they do not have a **when** clause.

All the states (names) in a module are defined in a **state** statement. The initial state of a module is specified in a **to** clause preceded by the keyword **initialize** in the initialization part of the module body. The keyword **channel** introduces a channel type definition.

A protocol which consists of co-operating processes which communicate by exchanging messages over channels will be specified in ESTELLE as a set of modules attached (connected) via channels.

An example of a protocol specified in ESTELLE can be found in appendix A.

4. The Translator Program

The input to the translator program is an ESTELLE specification of a protocol and the output a specification file containing a set of APL assignment statements. The APL statements should define the following:

1. The names and number of processes that make up the protocol.
2. The number of channels, what processes do they link and the capacity (maximum number of messages in transit) of every channel.
3. The number and the names of the events that may be exchanged between processes via the channels.(An event is the basic unit of communication between processes for example an ACK or MESSAGE_1.)
4. The state names within every process within the system.
5. The transitions between the states in every process.
6. The initial system state.

From the ESTELLE statements discussed in section 3 it is clear from where the translator program will obtain most of its output. The translator program performs no syntax checking on the ESTELLE code. However, if the translator program is unable to obtain all the information to create a complete definition (1 - 6 as describe above), it will report an error message indicating which item definition(s) it could not built.

The specification file consists of APL statements and look like:

```
NAME ← 'Read / Write Protocol'          'protocol name
NP ←  2                                 'no of processes
NC ←  2                                 'no of channels
   .
   .
 state transition matrices
ST[1;1;1;] ←  0 1 0 0
ST[1;1;2;] ←  4 0 3 0
ST[1;1;3;] ←  0 0 0 2
ST[1;1;4;] ←  3 0 4 0
   .
   .
```

The specification file contains statements defining the number of processes, number of channels, number of events and their names. For every process there is a state transition matrix. The state transition matrix indicates the transitions between states in a process and the event involved in the transition. The event is represented by a number and the direction, that is send or receive, by the sign of the number. A second matrix (same dimension as the state transition matrix) indicates in its corresponding

entry to which process an event is send or from which process an event is received.

When the translator program is activated, it will prompt the user for:

specify protocol name: ......
file name with specification: ......

After the user replied, the translator program will start building the specification file. Error messages are printed as they are encountered. If no errors were encountered the message "Specification file built. No errors" is displayed followed by "please specify the channel capacities

- channel A to B: ...." and
"channel B to A: ....".

The user then supplies these capacities. (A and B are two process names as defined in the module header.)

5. The Validation Routine

5.1 The Validation Technique

In the validation routine the protocols are defined in terms of interactions between two or more finite state machines. Interactions consists of the exchange of events (or messsages) that are transported via message queues in the communication medium. The validation routine uses the state exploration technique or reachability analysis.

This technique is based on exhaustively exploring all the possible interactions of two or more entities within a layer. A composite or global state of the system is defined as comprising of the states of the individual processes as well as the states from the channels. The general principle of reachability analysis is as follows.

First an executable model of a communication system is developed, that includes two or more communicating machines running the protocol being validated and a model of the communications medium that transports messages between them. A communications medium between (say) process A and process B will be handled as two one way channels, one from process A to process B and the other one from process B to process A. An initial state of the system is defined and all system states reachable from the initial state are determined by systematically exploring all possible transitions (in accordance with the transitions specified in the component process) from each system state reached. All reachable states are analyzed to determine whether they manifest errors. The process is repeated for each of the newly generated states until no new states are generated. (Some transitions lead back to already generated states.) As these

system states are explored a reachability tree is constructed starting at the initial system state.

The following example will illustrate the principle. Consider the read/write protocol in figure 5.1. This protocol consists of two processes A and B. Each process has four states. The model has two one-way channels as indicated. Initially both process A and process B are in the RESET state and both channels are empty - this combination will form the initial system state. The only new state possible from the initial state is when process A places a WRITE event on the channel A->B and enters the PEND.WRITE state. The state of the system will now be:

```
state of process A : PEND.WRITE
state of process B : RESET
channel A->B :       WRITE
channel B->A :       -
```

From this system state another new state can be generated (when process B receives the WRITE). In total nine system states can be generated from the initial state (see figure 5.1(b)). The corresponding reachability tree is illustrated in figure 5.1(c).

The main advantage of reachability analysis is its graphical form and possibility of automization. The major disadvantage is state space explosion, that is, the generation of a number of reachable system states exceeding the capacity of the validation system. The example in appendix B illustrates this disadvantage. The protocol in appendix B consists of two processes, each having four states. Although this is a very simple protocol its corresponding reachability tree consists of 26 system states. A validation of the OSI Session Layer using reachability analysis generated 25000 reachable states [WEST86].

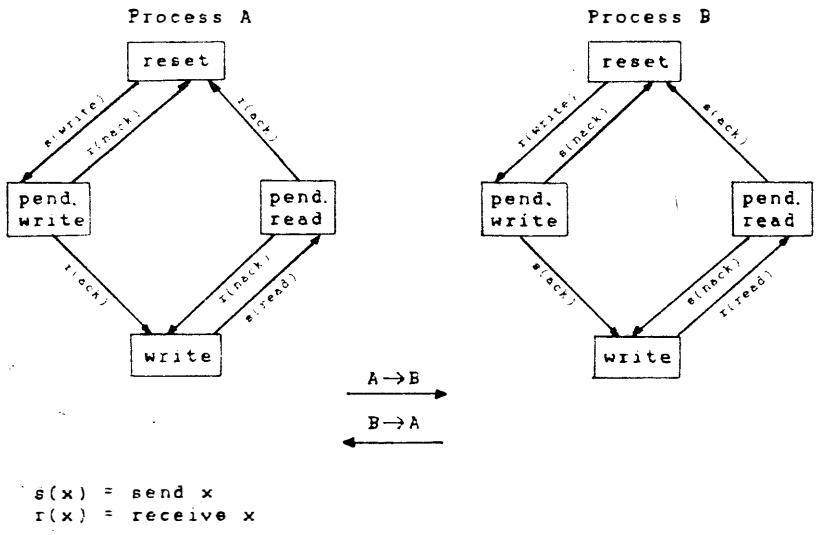


figure 5.1(a). read / write protocol

system state	state of A	channel A→B	state of B	channel B→A
0	reset	-	reset	-
1	pend.write	write	reset	-
2	pend.write	-	pend.write	-
3	pend.write	-	reset	nack
0	reset	-	reset	-
4	pend.write	-	write	ack
5	write	-	write	-
6	pend.read	read	write	-
7	pend.read	-	pend.read	-
8	pend.read	-	write	nack
5	write	-	write	-
9	pend.read	-	reset	ack
0	reset	-	reset	-

figure 5.1(b). system states

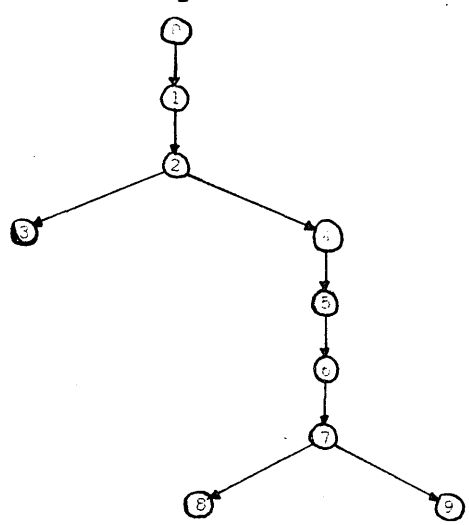


figure 5.1(c). reachability tree

5.2 Protocol Properties Validated

The error analysis emphasize the detection of errors that are common to all protocols. The validation routine reports the following error conditions:

1. Unspecified receptions. That is when the protocol system reaches a state when a process has no specified mechanism for receiving an incoming event. To report a reception error there only need to be one incoming event which can-cannot be received. Possible transmissions or other receptions do not influence this decision.
2. Deadlocks or terminal states of the protocol system which when entered, permit no further transition in the system.
3. Channel overflow which can occur if a process tries to send an event to another process in a way that a pre-defined maximum number of event underway between the processes is exceeded.
4. All transitions within a process that were not exercised during the validation.

This validation routine does not detect any types of errors that are expressible in terms of event sequences but not in terms of the system state at a given instant (for example livelock). Since, in general, there are more executable sequences than reachable states, this type of error detection will add complexity to the analysis [WEST86].

5.3 Input and Output

As discussed in section 4 the input to the validation routine is a specification file containing APL assignment statements which define the protocol. This specification file is imported into the APL workspace after which it will be treated as and APL function (routine). To activate the protocol definition this function must be executed. When the definition function is executed, the validation routine can be started.

The validation routine produces several output reports which are presented in such a way that the user can derive other information useful in understanding the system behaviour. For example the sequences that lead to errors can be derived from the output as well as all stable states, that is states in which the message queues are empty. The latter information is particularly useful in understanding the synchronization of the system. The system does not explicitly report the execution sequences which led to the errors.

An example of the system output can be found in appendic C.

5.3.1 The Trace

The trace is produced as the validation routine proceeds. This continuous output will appear on the screen. (The user has the option to re-route this output to the printer.) The validation routine repeats its actions for the initial and all subsequent system states. The routine attempts to generate as many new system states from each previously generated system state. The actions of the validation routine in its attempt to create new system states are reported. The following fragment reports the actions taken to generate new states from a system state.

```
      i      THE J FOUND WAS NO 2.
             RECEPTION - EVENT 4 FROM 2 TO 1
      ii                       FROM PROCESS 2
                     NOTHING IN CHANNEL TO RECEIVE 2 - 1
1                    NO TRANSITION FROM 2 TO 2
                     RECEPTION - EVENT 3 FROM 2 TO 3
                                 FROM PROCESS 2
                     NOTHING IN CHANNEL TO RECEIVE 2 - 3
      iv             NO TRANSITION FROM 2 TO 4

                     NO TRANSITION FROM 1 TO 1
                     RECEPTION - EVENT 1 FROM 1 TO 2
2     iii                        FROM PROCESS 1
                     ....REST..     ...FIRST...1
                     NO TRANSITION FROM 1 TO 3
                     NO TRANSITION FROM 1 TO 4

      v              PERTURB... 1
      vi             MEMBER ....N ...
```

(i) indicates that the system state under inspection is system state number 2. "1" represents the possible transmissions and receptions in process 1 and "2" the possible transmissions and receptions in process 2. (ii) indicates that process 1 can receive event number 4 from process 2 and thereby advance to state number 1 but the message "NOTHING IN CHANNEL TO RECEIVE 2-1" indicates that there is nothing in the channel to receive which will enable process 1 to go from state 2 to state 1. In (iii) we see that process 2 can receive event number 1 (" ... FIRST 1") and thereby go from state 1 to state 2. The message "NO TRANSITION FROM 2 TO 4" (iv) indicates that process 1 can never (nor by transmission or reception) advance from state 2 to state 4.

"PERTURB ... 1" (v) indicates that one new system state was formed from system state number 2. "MEMBER ... N .." (vi) indicates that this new system state is not a member of the set of existing system states. The new system state is now added to the set of existing system states. The total number

of system states and the validation time are reported at the end of the trace. An example of a trace can be found in appendix C, section C.1.

5.3.2 The System States

The APL function PRTSYS lists the set of system states. The state matrix and the associated channel contents for each system state is listed. If a system state contains an error an error message is displayed. For example the following information will be listed for a system state.

```
 STATE NO. 2
 STATE MATRIX
2  1
0  1
 CHANNEL CONTENTS
  CHANNEL NO. 1 ..... 1E
  CHANNEL NO. 2 .....
```

System state 2 represents the system state in which process 1 is in state 2, process 2 is in state 1 and there is one message on the channel from process 1 to process 2. (The "E" is an eliminator, for example 14E4 represents event 14 followed by event 4.)

Section C.2, appendix C contains an example of a listing of system states.

5.3.3 Transitions not exercised

The APL function DEADTRANS lists for every process all those transitions which were not exercised during the validation process. Section C.3, appendix C contains an example of such a listing.

5.3.4 Protocol Definition

The APL function PRINTDEF lists the protocol definition. It lists the number of processes and their names, the state names for every process, the event names and the channel numbers and what processes they connect. Section C.4, appendix C contains a protocol definition listing.

5.3.5 Error Messages

Typical error messages are:

```
*** ERROR IN THIS STATE - UNSPEC. RECEP.
*** ERROR IN THIS STATE - DEADLOCK/TERM ST.
*** ERROR IN THIS STATE - CHANNEL OVERFLOW
```

These messages are displayed directly following the state matrix and channel contents for the particular system state. Appendix D contains examples of error messages.

5.4 Additional Functions

Apart from the APL functions discussed in 5.3.2, 5.3.3 and 5.3.4 there exist other functions to provide additional information about the validated protocol.

The function NUMERR summarizes the error conditions found during the validation. It lists the number of system states, deadlocks, unspecified receptions and channel overflows.

The function PRTTREE produces output which can be used in constructing the reachability tree. The function lists the number of every system state followed by the numbers of the system states which were generated from it, that is, connected to it in the tree.

The function PRTERR lists all those system states with errors. An error message will indicate the type of error.

6. Applications

The validation routine was tested on several examples. Most of these examples were obtained from articles and in all cases the system state spaces and errors as described in the articles were reproduced [WEST78B, GOUD85, ZAFI82].

The validation routine was also applied to the CCITT X.21 Recommendation. The X.21 interface is a recommendation for a standard means of connecting Data Terminal Equipment (DTE) to Data Circuit-termination Equipment (DCE) in a public data network. The X.21 interface is formally defined in a state diagram for the combined DTE-DCE [X21-76]. The validation routine can only be applied to a pair of separately defined communicating processes. It is therefore necessary to derive from the combined state diagram the logical structure of the DCE and DTE. West used the combined state diagram to derived two separate state diagrams (see appendix E) [WEST78]. These two state diagrams were used in writing the ESTELLE specification for the protocol. The protocol definition includes two processes (DTE and DCE) with 23 states each and 12 events. There are two channels, one from the DTE to the DCE and one from the DCE to the DTE. The channel capacities were taken as 1.

The validation routine generated 331 system states. There were 18 unspecified receptions and 129 channels overflows. (The channel overflows should not be considered as errors, but rather as states in which one or both of the processes could proceed to another state but only if a channel was empty.) The validation time on an Olivetti M24 was 1 hour 15 minutes 38,5 seconds.

Most of these unspecified receptions are the result of collisions resulting when either the DCE or the DTE

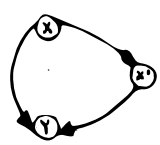
indicates a transition to a NOT READY state at the same time that the other is initiating a call establishment procedure or is itself making a transition to a NOT READY state. These errors are the result of incomplete specification of the interface. These errors were also found in the validation by West and Zafiropulo in [WEST78].

7. Limitations of the Implementation

The main drawback of this validation system is the level of sophistication of the translator program. This program dictates how the ESTELLE specification should be written. For example, the **provided** and **priority** clauses are not allowed. No syntax checking of the ESTELLE code is done. A translator program which places no restrictions on how the specification is written will be complex. In this validation system the emphasis is on the validation routine and therefore the level of sophistication of the translator program is acceptable.

In the validation routine the choice of the data structure (matrices) to represent the state transitions place a limitation on the structure of the directed graph which represents the protocol. For example the following subgraph

will have to be represented as



Although this is no major limitation it can be removed by using adjacency lists to represent the state transitions. This limitation will be removed in future versions of the validation routine.

8. Conclusion

Although this validation system is limited in some ways it proved to worked well on the examples tested. This method has also been applied succesfully to the ISO Session Layer specification [WEST86]. Numerous variations of the basic reachability analysis method are known.

It should be noted that reachability analysis is only one of many different validation methods and the protocol properties validated in this implementation only a subset of all protocol properties [PUNT86].

9. References

[CAST85] R Castanet, P Guitton and O Rafiq, "An Automatic System for the study of Protocols: A Presentation and Critque based on a worked example", Symposium on Protocol Specification, Testing and Verification IV, Y Yemini, R Strom, S Yemini (eds), North-Holland, 1985,pp 111-126.

[COUR86] J P Courtiat et al, "A Simulation Environment for Protocol Specifications described in ESTELLE", Symposium on Protocol Specification, Testing and Verification V, M Diaz (ed), North-Holland, 1986, pp 297-312.

[GOUD85] M G Gouda and Ji-Yun Han, "Protocol Validation by Fair Progress State Exploration", Computer Networks and ISDN Systems, No 9, 1985, pp 353-361.

[PUNT86] J L Punt, "Protocol Validation Methods", Technical Report CS-86-05-00, December 1986, Department of Computer Science, University of Cape Town.

[WEST78] C H West and P Zafiropulo, "Automated Validation of a Communications Protocol: The CCITT X.21 Recommendation", IBM J of Res. Develop., Vol 22, No 1, January 1978, pp 60-71.

[WEST78B] C H West, "General Technique for Communications Protocol Validation", IBM J of Res. Develop., Vol 22, No 4, July 1978, pp 393-404.

[WEST86] C H West, "Protocol Validation by Random State Exploration", IBM Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, February 1986.

[ZAFI82] P Zafiropulo et al, "Protocol Analysis and Synthesis using a State Transition Model", Computer Network Architectures and Protocols, P E Green (ed), Plenum Press, 1982.

[X21-76] Recommendation X.21, "General purpose interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) for synchronous operation on public data networks", CCITT, 1976, pp.38-56.

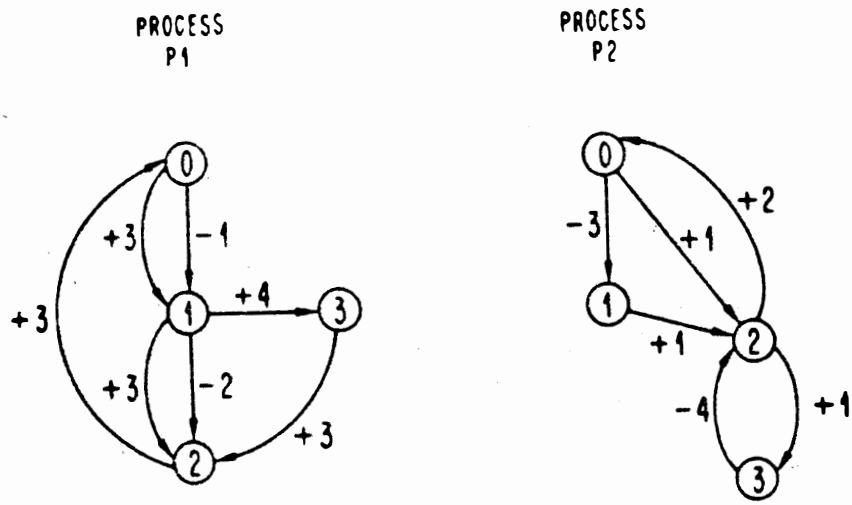
Appendix A - An ESTELLE Specification

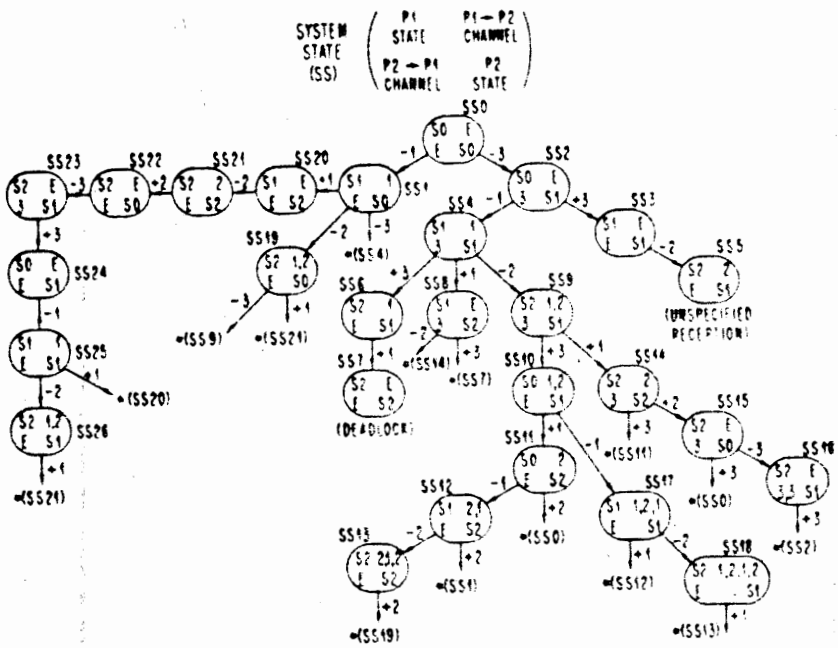
```
(* ********************************************** *)
(*     Read / Write Protocol - refer figure 5.1   *)
(*     ---------------------------------------    *)
(*      ,--------,              ,--------,         *)
(*      |  user  |              |  user  |         *)
(*      '--------'              '--------'         *)
(*          |        read/write      |            *)
(*      ,- - - - - - - - - - - - - - --,          *)
(*      |  ,-----,                ,-----,  |       *)
(*      |  | p1  |                | p2  |  |       *)
(*      |  '-----'                '-----'  |       *)
(*      '- - - - - - - - - - - - - - - - --'      *)
(*          |                        |            *)
(*      ,--------,              ,--------,         *)
(*      |network |              |network |         *)
(*      '--------'              '--------'         *)
(*                                                 *)
(*  layout of specification:                       *)
(* Specification Read_Write Example                *)
(* type as necessary                               *)
(* channel definitions to be used between network, *)
(* user and read_write processes                   *)
(* module header definitions for network, user and *)
(* read_write                                      *)
(*                                                 *)
(* body for user as external                       *)
(* body for network as external                    *)
(*                                                 *)
(* body for read_write                             *)
(*   const, type, ...                              *)
(*                                                 *)
(*   module header definitions process1, process2  *)
(*   body for process1                             *)
(*       const, type, var, state, procedures, functn*)
(*       initialization part, transition part      *)
(*   end of process1 body                          *)
(*   body for process2                             *)
(*       const, type, var, state, procedures, functn*)
(*       initialization part, transition part      *)
(*   end of process2 body                          *)
(*   var part of read_write                        *)
(*   initialization part of read_write             *)
(* end of body for read_write                      *)
(*                                                 *)
(* var part of specification                       *)
(* initialization part of specification            *)
(* end of specification                            *)
(* ********************************************** *)

(* channel definitions                             *)
channel U_access_point (user, provider)

channel N_access_point (user, provider)

(* module header definitions                       *)
module user_type process
       .
       .
module read_write_type process
 (U:U_access_point ....;
  N:N_access_point ....;  param ...)

module network_type process
       .
       .
(* body definitions                                *)
body network_body for network_type; external;
body user_body for user_type; external;

body read_write_body for read_write_type;
  const ..
  var ..


       .
       .
(* module definitions within read_write_body       *)
  module process1_type process
  (U:U_access_point ....;
   N:N_access_point ....;)

 module process2_type process
  (U:U_access_point ....;
   N:N_access_point ....;)
```

```
(* body for process1                                          *)
  body process1_body for process1_type;
     const ...
     type ....
     var ....
     state: (reset, write, pend.write, pend.read);
(* procedures & functions                                     *)
     procedure send( )
     procedure remove (   )

   initialize  (* process1 *)
   to reset
      begin
        .
        .
      end;
(* transition part for process 1                              *)
   trans
     from reset
       to pend.write
         begin
           send(WRITE)
             :
         end;
     from pend.write
       when N.data.id = NACK
         to reset
           begin
             remove( )
               :
           end;
     from pend.write
       when N.data.id = ACK
         to write
           begin
             remove( )
               :
           end;
     from write
       to pend.read
         begin
           send(READ)
             :
         end;
     from pend.read
       when N.data.id = NACK
         to write
           begin
             remove(   )
               :
           end;
     from pend.read
       when N.data.id. = ACK
         to reset
           begin
             remove(   )
               :
           end;
    end;  (* end of process1 body  *)

(* body for process2                                          *)
  body process2_body for process2_type;
     const ...
     type ....
     var ....
     state: (reset, write, pend.write, pend.read);
(* procedures & functions                                     *)
     procedure send( )
     procedure remove (   )

   initialize  (* process2 *)
   to reset
      begin
        .
        .
      end;
```

```
(* transition part for process 2                          *)
   trans
     from reset
       when N.data.id = WRITE
         to pend.write
           begin
             remove(   )
             :
           end;
     from pend.write
       to reset
         begin
           send(NACK)
           :
         end;
     from pend.write
       to write
         begin
           send(ACK)
           :
         end;
     from write
       when N.data.id = READ
         to pend.read
           begin
             remove(   )
             :
           end;
     from pend.read
       to write
         begin
           send(NACK)
           :
         end;
     from pend.read
       to reset
         begin
           send(ACK)
           :
         end;
    end;  (* end of process2 body  *)

(* end of module declaration part of read_write body *)
initialize (* initialization part of read_write body *)
begin
   init process1 with process1_body(name);
   init process2 with process2_body(name);

   attach U to ...
   attach N to ...
end;    (* end of read_write body *)
(* read_write body has no transition part              *)
(* variable declaration part of the specification     *)
var

  ...
  ...
(* initialization part of the specification           *)
intialize
  begin
     init ...
     all ...
     begin
        init ...
        init ...
        connect ...
        connect ...
    end;
  end; (* end of module initialization part           *)
end; (* end of specification. Specification has no    *)
(*      transition part.                               *)
```

Appendix B - Protocol and Reachability Tree





-x = send(x)

+x = receive(x)

Appendix C - System Output

C.1 Trace

```
     VALID
 INITIALIZATION COMPLETED .....


 THE J FOUND WAS NO 1.
TRANSMISSION - EVENT '1 FROM 1 TO 2
               TO PROCESS 2

RECEPTION - EVENT 1 FROM 1 TO 2
               FROM PROCESS 1
      NOTHING IN CHANNEL TO RECEIVE 1 - 2

PERTURB... 1
MEMBER....N...1..TEMP....


 THE J FOUND WAS NO 2.
RECEPTION - EVENT 4 FROM 2 TO 1
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 2 - 1
RECEPTION - EVENT 3 FROM 2 TO 3
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 2 - 3

RECEPTION - EVENT 1 FROM 1 TO 2
               FROM PROCESS 1
      ....REST..     ...FIRST...1

PERTURB... 1
MEMBER....N...2..TEMP....


 THE J FOUND WAS NO 3.
RECEPTION - EVENT 4 FROM 2 TO 1
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 2 - 1
RECEPTION - EVENT 3 FROM 2 TO 3
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 2 - 3

TRANSMISSION - EVENT '4 FROM 2 TO 1
               TO PROCESS 1
TRANSMISSION - EVENT '3 FROM 2 TO 3
               TO PROCESS 1

PERTURB... 2
MEMBER....N...3..TEMP....
MEMBER....N...4..TEMP....

                  :
                  :
                  :

 THE J FOUND WAS NO 5.
RECEPTION - EVENT 4 FROM 2 TO 1
               FROM PROCESS 2
      ....REST..     ...FIRST...3
      FIRST IN CHANNEL 3
      NOTHING IN CHANNEL TO RECEIVE 2 - 1
RECEPTION - EVENT 3 FROM 2 TO 3
               FROM PROCESS 2
      ....REST..     ...FIRST...3

RECEPTION - EVENT 2 FROM 3 TO 4
               FROM PROCESS 1
      NOTHING IN CHANNEL TO RECEIVE 3 - 4

PERTURB... 1
MEMBER....N...5..TEMP....


 THE J FOUND WAS NO 6.
TRANSMISSION - EVENT '2 FROM 3 TO 4
               TO PROCESS 2

RECEPTION - EVENT 2 FROM 3 TO 4
               FROM PROCESS 1
      NOTHING IN CHANNEL TO RECEIVE 3 - 4

PERTURB... 1
MEMBER....N...6..TEMP....


 THE J FOUND WAS NO 7.
RECEPTION - EVENT 3 FROM 4 TO 1
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 4 - 1
RECEPTION - EVENT 4 FROM 4 TO 3
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 4 - 3

RECEPTION - EVENT 2 FROM 3 TO 4
               FROM PROCESS 1
      ....REST..     ...FIRST...2

PERTURB... 1
MEMBER....N...7..TEMP....


 THE J FOUND WAS NO 8.
RECEPTION - EVENT 3 FROM 4 TO 1
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 4 - 1
RECEPTION - EVENT 4 FROM 4 TO 3
               FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 4 - 3

TRANSMISSION - EVENT '3 FROM 4 TO 1
               TO PROCESS 1
TRANSMISSION - EVENT '4 FROM 4 TO 3
               TO PROCESS 1

PERTURB... 2
MEMBER....N...8..TEMP....
MEMBER....N...9..TEMP....




 THE J FOUND WAS NO 9.
RECEPTION - EVENT 3 FROM 4 TO 1
               FROM PROCESS 2
      ....REST..     ...FIRST...3
RECEPTION - EVENT 4 FROM 4 TO 3
               FROM PROCESS 2
      ....REST..     ...FIRST...3
      FIRST IN CHANNEL 3
      NOTHING IN CHANNEL TO RECEIVE 4 - 3

RECEPTION - EVENT 1 FROM 1 TO 2
               FROM PROCESS 1
      NOTHING IN CHANNEL TO RECEIVE 1 - 2

PERTURB... 1
MEMBER....Y...1..TEMP....


 THE J FOUND WAS NO 10.
RECEPTION - EVENT 3 FROM 4 TO 1
               FROM PROCESS 2
      ....REST..     ...FIRST...4
      FIRST IN CHANNEL 4
      NOTHING IN CHANNEL TO RECEIVE 4 - 1
RECEPTION - EVENT 4 FROM 4 TO 3
               FROM PROCESS 2
      ....REST..     ...FIRST...4

RECEPTION - EVENT 2 FROM 3 TO 4
               FROM PROCESS 1
      NOTHING IN CHANNEL TO RECEIVE 3 - 4

PERTURB... 1
MEMBER....Y...6..TEMP....
VALIDATION COMPLETED. NUMBER OF S.STATES 10.

** VALIDATION TIME: 0:40:430
```

States

```
 SYSTEM STATES:
STATE NO. 1.
  STATE MATRIX:
 1 0
 0 1
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....

STATE NO. 2.
  STATE MATRIX:
 2 1
 0 1
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....1E
   CHANNEL NO. 2.....

STATE NO. 3.
  STATE MATRIX:
 2 0
 0 2
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....

STATE NO. 4.
  STATE MATRIX:
 2 0
 1 1
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....4E

STATE NO. 5.
  STATE MATRIX:
 2 0
 1 3
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....3E

STATE NO. 6.
  STATE MATRIX:
 3 0
 0 3
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....

STATE NO. 7.
  STATE MATRIX:
 4 1
 0 3
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....2E
   CHANNEL NO. 2.....

STATE NO. 8.
  STATE MATRIX:
 4 0
 0 4
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....

STATE NO. 9.
  STATE MATRIX:
 4 0
 1 1
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....3E

STATE NO. 10.
  STATE MATRIX:
 4 0
 1 3
  CHANNEL CONTENTS:
   CHANNEL NO. 1.....
   CHANNEL NO. 2.....4E
```

C.3 Transitions not exercised

```
TRANSMISSIONS/RECEPTIONS NOT TAKEN
----------------------------------

   PROCESS 1
   ---------
    NO TRANSMISSION/RECEPTION FROM STATE 3 TO 2
    NO TRANSMISSION/RECEPTION FROM STATE 6 TO 5

   PROCESS 2
   ---------
    NO TRANSMISSION/RECEPTION FROM STATE 1 TO 6
    NO TRANSMISSION/RECEPTION FROM STATE 4 TO 3
  ==========================================
```

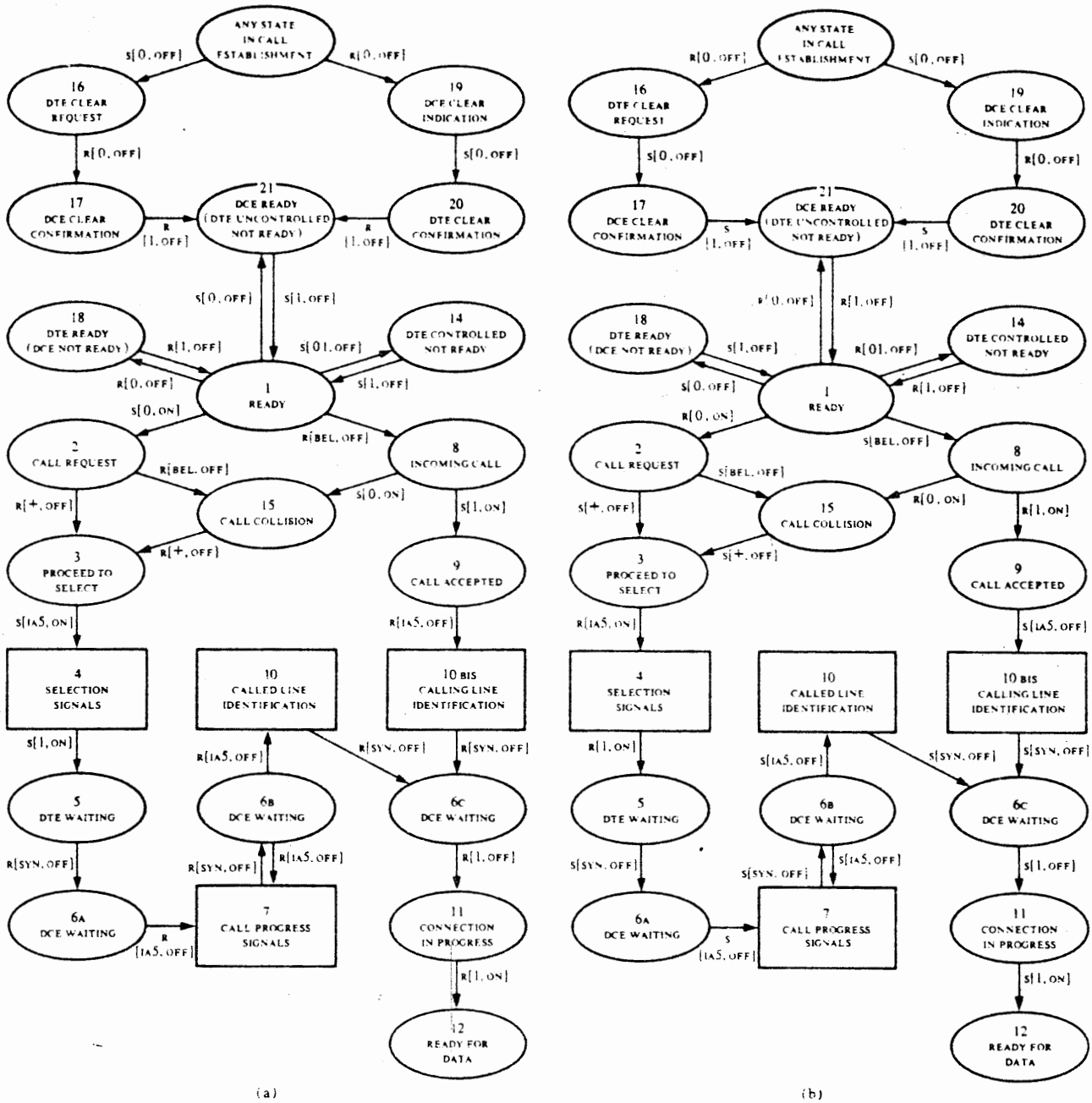
C.4 Protocol Definition

```
            PROTOCOL DEFINITION  ( READ/WRITE )
            ===================

NUMBER OF PROCESSES: 2
   1. PROC_A
   2. PROC_B

NUMBER OF CHANNELS: 2
 NO. 1   FROM PROCESS  PROC_A  TO  PROC_B
 NO. 2   FROM PROCESS  PROC_B  TO  PROC_A

STATE NAMES FOR PROCESS 1  ( PROC_A )
  STATE NO. 1 - RESET
  STATE NO. 2 - PEND.WRITE
  STATE NO. 3 - WRITE
  STATE NO. 4 - PEND.READ

STATE NAMES FOR PROCESS 2  ( PROC_B )
  STATE NO. 1 - RESET
  STATE NO. 2 - PEND.WRITE
  STATE NO. 3 - WRITE
  STATE NO. 4 - PEND.READ

NUMBER OF EVENTS: 4
  EVENT NO. 1 - WRITE
  EVENT NO. 2 - READ
  EVENT NO. 3 - ACK
  EVENT NO. 4 - NACK
            ============================
```

Appendix D - Error Messages

```
 THE J FOUND WAS NO 5.
TRANSMISSION - EVENT '1 FROM 1 TO 2
               TO PROCESS 2
**** OVERFLOW:1 - 2 EV '1 P 2

TRANSMISSION - EVENT '1 FROM 2 TO 1
               TO PROCESS 3
       CANNOT RECEIVE..4..CHANNEL..1
**** RECEPTION ERROR ------

RECEPTION - EVENT 3 FROM 1 TO 1
                FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 1 - 1
RECEPTION - EVENT 1 FROM 1 TO 2
                FROM PROCESS 2
      NOTHING IN CHANNEL TO RECEIVE 1 - 2

PERTURB... 2
MEMBER....N...6..TEMP....
MEMBER....N...7..TEMP....



PERTURB... 0
 ****  THE 14TH STATE: DEADLOCK/TERMINAL STATE !!!!!



STATE NO. 7.
  STATE MATRIX:
2 2 0
0 2 0
0 0 1
 CHANNEL CONTENTS:
  CHANNEL NO. 1.....4E1E
  CHANNEL NO. 2.....
  CHANNEL NO. 3.....
*** ERROR IN THIS STATE - CHANNEL OVERFLOW

STATE NO. 8.
  STATE MATRIX:
1 1 0
0 1 1
0 0 1
 CHANNEL CONTENTS:
  CHANNEL NO. 1.....4E
  CHANNEL NO. 2.....1E
  CHANNEL NO. 3.....
*** ERROR IN THIS STATE - UNSPEC. RECEP.


STATE NO. 14.
  STATE MATRIX:
3 0 0
0 1 0
0 0 3
 CHANNEL CONTENTS:
  CHANNEL NO. 1.....
  CHANNEL NO. 2.....
  CHANNEL NO. 3.....
*** ERROR IN THIS STATE - DEADLOCK/TERM.ST



ERROR CONDITIONS - SYSTEM STATES
================================

NUMBER OF SYSTEM STATES: 15
NUMBER OF DEADLOCKS:      1
NUMBER OF UNSPEC. RECEP:  1
NUMBER OF OVERFLOWS:      4
```

Appendix E - X.21 Recommendation State Diagrams



Derived state diagrams for the DTE (a) and DCE (b), where S indicates send and R receive.