INSPECTING THE SOURCE CODE THAT IMPLEMENTS THE PPP PROTOCOL IN LINUX

By

SRDJAN RUSOVAN, B.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

Mc Master University

Copyright by Srdjan Rusovan, October 2003.

MASTER OF SCIENCE (2003) Mc Master University

(Computing & Software) Hamilton, Ontario

TITLE: Inspecting the Source Code that Implements the PPP Protocol in Linux

AUTHOR: Srdjan Rusovan, B.Sc. (University of Belgrade, Serbia and Montenegro)

CO-SUPERVISOR: Dr. David L. Parnas

CO-SUPERVISOR: Dr. Mark Lawford

NUMBER OF PAGES: viii, 145

ABSTRACT

The Point-to-Point Protocol (PPP) is a widely accepted standard used by almost everyone who connects to a server over a telephone line. An open source implementation of the PPP protocol, pppd – the PPP daemon, is included in various Linux distributions and widely used by Linux users. Version 2.4.1 of the pppd is typical of many software products in that the only documentation available is the C programming language code and a high-level requirements document containing an English language description of the system's required behaviour, together with the state transition table for the protocol's abstract state machine. In order to perform a rigorous inspection of the pppd protocol state machine code based upon Parnas' Display method, we required detailed design documents that simply did not exist. In [Parnas 1994] a rigorous inspection method which manually generates the absent design documentation and performs the inspection, is presented. This thesis shows how parts of the process can be automated using theorem proving techniques. Specifically, the reverse engineering of the design documentation was primarily done in the PVS Specification and Verification System. The resulting combination of inspection based upon the Display method and theorem proving was used to find several mistakes in the PPP implementation code. Inconsistency in coding style and significant issues regarding the readability of the code were also discovered. Thus the thesis also provides a detailed example of successful application of the Displays inspection method to a non-trivial application.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. David L. Parnas and Dr. Mark Lawford, my thesis supervisor and my thesis co-supervisor, for their guidance, support, and faith throughout the research of this thesis.

I wish to thank Dragana Stasic, Irena Rancic and Doris Burns for their help and support during the research. I would also like to thank to all members of the Software Quality Research Laboratory, especially Dr. Ridha Khedri, Dr. Robert Baber, Dr. Ryszard Janicki and Dr. Emil Sekerinski for stimulating discussions and support.

Last, but not least, I wish to thank my parents, my sister and my friends for the love and support, and the faith they have always shown in me.

CONTENTS

Abstract iii

Acknowledgments iv

1 WHAT IS THE PURPOSE OF SOFTWARE INSPECTION? **1**

1.1. What is important to know about software inspection? 1

1.1.1 What is software inspection?…………………………………………….…. 1
1.1.2 What are the main characteristics of software inspection? …………...…… 1
1.1.3 What is theorem proving?…………………………………………………1
1.1.4 What are the main advantages and disadvantages of theorem proving?……..2
1.1.5 Why do we need software inspection?…………………………………….2
1.1.6 What criteria should a systematic inspection process meet?………………..3
1.1.7 What criteria should the documentation in an inspection process meet?…..4

1.2. What is our motivation to implement our software inspection method?…………5

1.3. What is already known about inspection ?................……………………………8

1.4. What is the contribution of this work?……………………………………………11

1.5 What is the structure of the rest of this thesis?……………………………………13

2 WHICH INSPECTION PROCESS (IN GENERAL) DO WE PROPOSE? **15**

2.1. What are the steps in the proposed process? ………………………………………15
2.2. What software development process do we assume and why?……………………19
2.2.1 What is the role of program function tables in this procedure?……………..19
2.2.2 What software development process do we assume?……………………….21

2.3. How do we model the system requirements (REQ) and the software design (SOF)?21

2.3.1 What is the 4 variable model?………………………………………………….21
2.3.2 How is it related to our work?…………………………………………………23

3 WHAT IS THE ROLE OF THE THEOREM PROVER SYSTEMS IN OUR WORK?

3.1 What is important to know about formal methods and the theorem prover systems? 24

3.2. Which are the theorem prover systems most related to our work?………………..25

3.3. What are the advantages of PVS that led us to choose PVS?……………………...28
3.3.1 Why did we chose PVS?…………………………………………………28
3.3.2 How does PVS support tables?……………………………………………..29

4 WHAT IS THE ROLE OF THE DISPLAY METHOD IN OUR WORK? **33**

4.1. What is the main concept of the Display method?…………………………………33

4.1.1 What is a display?………………………………………………………….33
4.1.2 What is a lexicon?………………………………………………………....34
4.1.3 What is an index?…………………………………………………………...34
4.1.4 What are completeness and correctness?…………………………………...34
4.1.5 What is the role of mathematical.notation and mathematical methods in the Display method?……………………………………………...35
4.1.6 What is the role of tables and tabular notation in the Display method?………36

4.2. What is the motivation to use the Display method in our inspection?………………37

4.3. What is the Display Management System (DMS)…………………………………..37

5 WHAT IS THE PURPOSE OF THE PPP PROTOCOL AND HOW DOES IT WORK? **39**

5.1. What is the purpose of PPP?…………………………………………………………...39

5.2. What is the structure of PPP?……………………………………………………39

5.3. What is the Option Negotiation Automaton?……………………………………….40

5.4. What does the PPP State Transition Table look like? ………………………………41

5.5. What are the Counters and Timers within PPP?……………………………………44

5.5.1 What is the restart timer (within PPP)?……………………………………44
5.5.2 What are Max Terminate, Max Configure, Max Failure?……………………44

5.6. What are the main characteristics of each state in the PPP State Transition Table?……………………………………………………………………………45

5.7. What are the main characteristics of each event in the PPP State Transition Table?.47

5.8. What are the main characteristics of each action in the PPP State Transition Table?50

6 HOW WAS OUR INSPECTION TECHNIQUE APPLIED TO THE LINUX PPP DAEMON ? **54**

6.1 What part of the PPP code did we inspect?…………………………………………...54

6.2 How did we use PVS in inspecting the Linux PPP daemon?………………………54

6.3 How did we use the Display method in inspecting the Linux PPP daemon?……...57

6.4 Concrete examples……………………………………………………………………58

6.4.1 Example 1 with comments…………………………………………………59
6.4.2 Example 2 with comments…………………………………………………63

7 WHICH MISTAKES AND INCONSISTENCIES DID WE FIND WHEN INSPECTING THE PPP CODE ? **69**

7.1. How can we classify the mistakes that we found?………………………………..69

7.2. What are the concrete examples of the mistakes we have found?………………...70

8 CONCLUSION…………………………………………………………..……**76**

8.1 What can we conclude about the method we have proposed?................................76

8.2 Limitations and future work…………………………………………………………78

BIBLIOGRAPHY……………………………………………………………………..**79**

APPENDIX A : PVS files for the Inspection of the PPP FSM code…. ………………**84**

APPENDIX B: Displays for the Inspection of the PPP FSM Code………..…………**119**

LIST OF FIGURES

2.1 Inspection process (Diagram)……………………………..16

2.3.1 Commutative diagram for the 4-variable model …………..22

4.1.6 Conventional predicate specifying a search program……...36

5.1.4 PPP State Transition Table………………………………….43

1 WHAT IS THE PURPOSE OF SOFTWARE INSPECTION?

1.1 What is important to know about software inspection?

1.1.1. What is software inspection?

Software inspection is an efficient technique to detect faults throughout the software development process. The main purpose of inspection is to see if software is ready for release [Laitenberger, and Kohler 2001]. A more detailed discussion of the inspection methods used to achieve these goals follows in section 1.3.

1.1.2. What are the main characteristics of software inspection?

Good software inspection means a careful and systematic analysis and evaluation of every section of the code - one that overlooks no cases or parts of the program. Inspection is an efficient and reliable way to either find errors or gain confidence that there are no errors.

1.1.3. What is theorem proving?

Theorem proving is a mechanical process in which mathematical expressions describing the program and its states are manipulated using a fixed set of inference rules. Theorem proving establishes properties of hardware or software designs using "rigorous mathematics", rather than using testing or informal arguments alone. Formal methods are an analytical approach relying on mathematical models for excluding design errors in software. This involves formal specification of the requirements, formal modeling of the implementation, and precise rules of inference to demonstrate, say that the implementation satisfies the specification.

1.1.4. What are the main advantages and disadvantages of theorem proving?

Theorem proving is intended to demonstrate that a specification and an implementation agree or disagree on all possible input sequences (guaranteed domain coverage). Theorem proving can also be used to perform “refutation” by proving a theorem demonstrating that the specification and implementation are not consistent or compatible for a specific input.

While theorem proving may give a greater degree of certainty of the correctness of a software system, it is typically too time-consuming and expensive to apply to all parts of every project. In addition, we often do not have the axiomatic definitions of the programming constructs that would be required for full formal verification.

Also, theorem proving only looks at correctness, not at other issues such as structure and robustness or adherence to coding standards.

1.1.5. Why do we need software inspection?

Software is used increasingly in safety-critical and mission-critical systems. The software that controls a nuclear plant, automobile, chemical plant or airplane is obviously critical to the safety of the equipment and people near it. Therefore, these software products must be carefully documented and inspected before they enter service. We cannot rely upon *ad hoc* inspections but rather need a rigorous approach to software inspection and verification that has a sound mathematical foundation. The benefits of a good inspection process are significantly improved safety and reliability and reduced maintenance cost of the inspected code. Inspection is not just important in so called mission-critical applications. It is important whenever the cost of a failure or a correction

is high. For example, if we buy a car with a radio and the software causes the radio to function improperly (losing stations etc.), the manufacturer might have to recall it or develop a bad reputation. Inspection is necessary in all areas of software application.

1.1.6. What criteria should a systematic inspection process meet?

We need a reliable inspection method in which it is possible to be sure that we have considered all possible operating conditions or input cases and all possible paths and all possible data states for the program. A good inspection process will apply the "divide and conquer" principle - the principle of dividing something large into smaller units, so that it can be dealt with more easily. We must decompose a long program into small parts and assign a "function" to each part. We have to be sure that (1) if each part implements its assigned function the whole program will be correct and (2) each part implements its assigned function [Parnas 1995]. We can use a combination of inspection and formal verification to demonstrate both (1) and (2), using formal verification where practical in support of the inspection process.

1.1.7. What criteria should the documentation in an inspection process meet?

Software documentation should describe the program's required behaviour under all operating conditions and explain any assumptions about the inputs that are required for correct operation of the program. Proper formal documentation facilitates the inspection process, guiding the inspection team's decomposition of the inspection into subtasks. In the previous section we said that we must assign a function to each component. It is very important to document that function.

Software documentation must support the “divide and conquer” approach to software inspection by describing both the structure of the software and its partitioning into functional units which can then be inspected separately on the assumption that the other programs correctly implement their required functionality.

Good documentation contributes to increasing the reliability and accuracy of software products. It plays an essential role in the software development process. A general approach to software documentation called “functional documentation” is described in [Parnas, and Madey 1995]. The documents described in [Parnas, and Madey 1995] include: the “system requirements document”, the “system design document”, the “software requirements document”, the “module interface specification”, and the “module internal design document”. If software function is not documented it cannot be inspected. Hence all design decisions should be documented and inspected before being implemented.

1.2. What is our motivation to implement our software inspection method?

As we have already noted, inspection and formal verification are very useful for safety-critical software but they require detailed design documents. For example, the Display method of inspection typically requires detailed Program Function Tables against which the code is inspected [Parnas, Madey, and Iglewski 1994]. These Program Function Tables should appear in a detailed design document, the Module Internal Design (MID). The MID should describe the module's data structure, state the intended interpretation of that data structure (in terms of external interface), and specify the effect of each access-program on the modules data structure [Parnas, and Madey 1991]. To ensure that the program function tables properly describe the requirements for individual programs, we would ideally like to perform a formal verification of the design. This verification requires mathematical descriptions of both the software requirements and the software design.

Ideally, all necessary documents are created and updated from the beginning of the project as a required part of the software development process. This is not always the case for legacy systems or in the case when developers rush to bring a product to market. Another situation in which proper documentation may not exist is when a piece of software becomes critical when incorporated into a larger system. For instance, a distributed control system might rely upon an existing PPP (Point-to-Point Protocol) implementation to receive critical data from a remote central system. PPP is not normally considered critical, but would be so if the control system that uses it is considered critical.

Microsoft Word could be safety-critical if it was used to document a design and the formulae were changed by the format rules.

In practice, inspectors and verifiers will often have serious problems with software documentation; because unfortunately, the detailed design documents required to perform a proper software inspection hardly ever exist. More often than not, only a requirements document (e.g. the Point-to Point RFC document) [Simpson 1994] and the code are available. There is little other intermediate design documentation.

Faced with a similar lack of detailed design documentation when attempting to inspect the first version of the software for the shutdown systems on the Darlington Nuclear Generating Station, Ontario Hydro employed the following manual inspection process. As described in [Parnas 1994], the work involved 4 distinct teams:

1. A requirements team, producing tables of the requirements. This corresponds to a part of the SRS – Software Requirements Specification of the forward going software development process described in Section 2.3.

2. A code inspection team producing program-function tables from the code. In the preferred software development process outlined in Section 2.3, these tables should be created before the code as a central part of the SDD - Software Design Description.

3. A comparison team, establishing the equivalence between the requirements tables and program-function tables. This corresponds to the SDV - Systematic Design Verification of Section 2.3.

4. An audit team that checked the work of team 3 on a "random sample" basis.

This thesis uses an improvement of the above formal software inspection technique that is based upon a partial automation of the above process using theorem proving. From the above process, we can see that in order to perform the Displays inspection in the absence of detailed design documents, we need to: 1) Create Program Function Tables corresponding to the system design 2) make sure the Program Function Tables implement the requirements (there is no point in inspecting against an incorrect design) 3) make sure that the code correctly implements the Program Function Tables.

In our case, in order to inspect the Linux PPP code:

1. We used the RFC to create requirements tables that take into account additional timer information that does not appear explicitly in the abstract state machine transition table that appears as Fig. 4 of the thesis.

2. We created the progam-function tables by looking at the code and the RFC. Initially the tables were in PVS, but later further details were added to create the displays. Creating the displays from the PVS tables was a manual process involving transcribing the PVS table into a spreadsheet and then adding addition details and notation as required by the Display method. The "Inspection" performed once the displays were complete corresponds to the code inspection step 2 in the previous process. Ideally it confirms that the code does as the "implementation" PFTs says it should.

3. We replaced the comparison team by using PVS to prove the equivalence of corresponding requirements and design tables created in steps 1 and 2 above. This comparison in PVS was done before creating the detailed displays. Once completed, the rest of 2 above (creating the displays and reviewing them) was performed.

4. The audit team in our case corresponds to reviewers of earlier drafts of the thesis. This "team" was auditing the work of 2 rather than that of 3, inspecting the code Displays for correctness. If an actual audit team so desired, they could also have us supply the PVS dump file used to perform the comparison 3 and could review that and re-run the proofs to check that work.

1.3. What is already known about inspection?

Software inspection is an efficient technique to detect faults throughout the software development process. Several research papers have reported on the benefits of using inspections [Fagan 1976], [Gilb, and Graham 1993], [Ebenau and Strauss 1993]. Different inspection processes have been proposed since the formalization of the first inspection process [Fagan 1976]. The different research approaches include changes to the inspection process [Parnas, and Weiss 1985], [Bisant, and Lyle 1989], [Martin and Tsai 1990], [Knight, and Myers 1993]. The different research approaches also include support to the inspection process [Basili et al. 1996], [Eick et al.1992] and empirical studies [Porter, Votta, and Basili 1995], [Shull, Ioana, and Basili 2000]. The suggested improvements include active design reviews [Parnas, and Weiss 1985], two-person inspection teams [Bysant, and Lyle 1989], n-fold inspections [Martin, and Tsai 1990], phased-inspections [Knight, and Myers 1993], perspective based reading (PBR) [Basili et

al.1996] and the use of capture-recapture techniques to estimate the remaining number of faults after an inspection [Eick et al.1992].

[Gilb, and Graham 1993] suggest inspecting part of a software artefact in order to determine whether the artefact is ready for the main inspection. They argue that the same type of faults exist in different forms throughout the same document. Both [Gilb, and Graham 1993], [Burr, and Owen 1996] describe sampling as a way to make inspections more efficient and effective. Industry has studied the benefits of conducting software inspections [Weller 1993].

The method discussed in this thesis arises out of work developed over many years and applied in the Darlington Nuclear Power Generating Station. The work was first reported in [Parnas, Asmis, and Madey 1991]. We also used basic principles and ideas developed in [Parnas 1994]. In particular, [Parnas 1994] describes a rigorous approach to software inspection (verification).

The Display method, a method of documenting well-structured programs, was described in [Parnas, Madey, and Iglewski 1994]. Displays are intended to be used by Software engineers as a reference document during inspection and maintenance. In fact, the displays can be viewed as a way to organize the inspection process.

While proponents of formal methods have been advocating their use in the development and verification of safety-critical software for over two decades [Parnas 1977], [Parnas 1995], there have been few full industrial applications utilizing rigorous mathematical techniques. Application of tool-supported tabular methods to the specification and verification of safety-critical software in the Darlington Nuclear Power

generation Station was described in [Lawford, Froebel, and Moum 2000]. Further [Lawford, Froebel, and Moum 2000] also explained the principles of a Systematic Design Verification Procedure and the role of the theorem prover PVS in the procedure. We found the use of a theorem prover to be a great help in our work. The need for a Theorem Prover system and the reasons for choosing PVS as the theorem proving engine we used, is described in [Jing 2000].

Finally, we note that [Alur, and Wang 2001], who considered the problem of establishing consistency of the code, implementing a network protocol with respect to the documentation provided a standard RFC. The problem is formulated as a refinement checking between two models, the implementation extracted from code and the specification extracted from RFC. The methodology is illustrated in a case study involving the same network protocol, PPP that we study in this thesis.

1.4. What is the contribution of this work?

This thesis successfully applies a software inspection method to produce a software inspection document for the Linux PPP (Point-to Point Protocol) daemon (pppd). The pppd implementation version inspected was 2.4.1, an open-source package written in the C programming language that is included in various Linux distributions and widely used by Linux users. PPP is the widely accepted standard used by almost everyone who connects to a server over the telephone.

Using the combination of inspection based upon the Display method and theorem proving employing the PVS Specification and Verification System in our inspection, we found several mistakes in the implementation code. We also found inconsistency in coding style and significant issues regarding the readability of the code. Thus the inspection method we applied enabled us to find more than inconsistencies between the requirements and implementation code. An additional benefit of this method is its documentation of the implementation provided by the PFTs (Program Function Tables) used for inspection. The PFTs are much easier to understand than the PPP daemon code. It is important to understand that while we have used PVS, the same technique could be applied with any other theorem proving tool that can be used to deal with tabular specifications.

We used some ideas and explanations from the papers mentioned in 1.3 but our work still represents a significant contribution. What follows is a list of the contributions:

1) Partial automaton of steps 1-3 in the inspection process of [Parnas 1994] using theorem proving/verification techniques.
2) Providing a detailed example of successful use of the display method a non-trivial part (900+ lines of code and comments) "real world" application.
3) Using Formal Verification to construct system documentation to allow inspection, we found not only functional errors, but also coding style and readability issues.
4) We have used a theorem prover to help reverse-engineer the PPP design documentation and verify its correctness with respect to the requirements.
5) The inspection method that we used has demonstrated its effectiveness by enabling us to discover several mistakes and inconsistencies in the PPP implementation code.
6) The inspection process checks not only the correctness of the code, but also the formal verification work since the PFTs used in the displays for the inspection are also reviewed.

While using a theorem prover to help create missing documentation to bootstrap the inspection may seem like excessive work, we found that this was not the case. The application of the manual process described in [Parnas 1994] to roughly 20,000 lines of code required 60 people working for approximately 1 year! Much of effort in that case was spent creating and debugging the tabular specifications of requirements and design and then proving their equivalence. Although producing correct (complete and consistent) function tables by hand is difficult and prone to error, PVS's basic support for tables allows one to quickly input and debug some basic types of program function tables. By proving the functional equivalence of the PVS models of the requirements and

the design, we further increase our confidence in the Program Function Tables that will be used in the inspection process, and when the functional equivalence proofs fail, it helps to highlight those parts of the code for closer scrutiny by the inspection team. Later the detailed inspection process helps to determine if the PVS models accurately reflect the requirements and the code.

Thus this thesis provides an example of how mathematical methods can be used as an integral part of a computer-aided software inspection process. We combined computer-aided verification with software inspection to compensate for a lack of sufficiently detailed documentation.

1.5. What is the structure of the rest of thesis?

Chapter 2 is a brief description of the inspection process we applied.

Chapter 3 gives an overview of Theorem Prover Systems and the PVS Specification and Verification System.

Chapter 4 serves as a guide to the Display method.

Chapter 5 gives an introduction to the Point to Point protocol.

Chapter 6 is a detailed description of the inspection process of the PPP implementation code.

Chapter 7 details all of the mistakes and inconsistencies we found in the PPP implementation code during our inspection.

Chapter 8 reports our conclusions and suggestions for future work.

Appendix A presents PVS files written for the purpose of verifying the PPP implementation code.

Appendix B presents the Displays which were used in our inspection of the PPP code.

2 WHICH INSPECTION PROCESS (IN GENERAL) DO WE PROPOSE?

2.1. What are the steps in the proposed process?

To perform our inspection we used the PVS Specification and Verification System to create the Program Function Tables (PFT) to help boot strap the inspection process (i.e. to create the absent design documentation), and then used the Display method to finish it. The Display method is a way to organize proofs as a set of simple proofs. The inspection process we have followed can be understood as a “divide and conquer” process. The display “divides” and PVS helps to “conquer”.

Our inspection method uses a theorem prover such as PVS to construct the PFTs for an application by examining the high level requirements document and the code. The requirements are formalized in the language used for input to the theorem prover and then the program function tables corresponding to the system design are created from the code and verified against the requirements to ensure that the design satisfies the requirements. The program function tables are then used in displays for the inspection of the programs.

We now provide a brief step by step explanation of the inspection process. The entire process is illustrated in Figure 1.

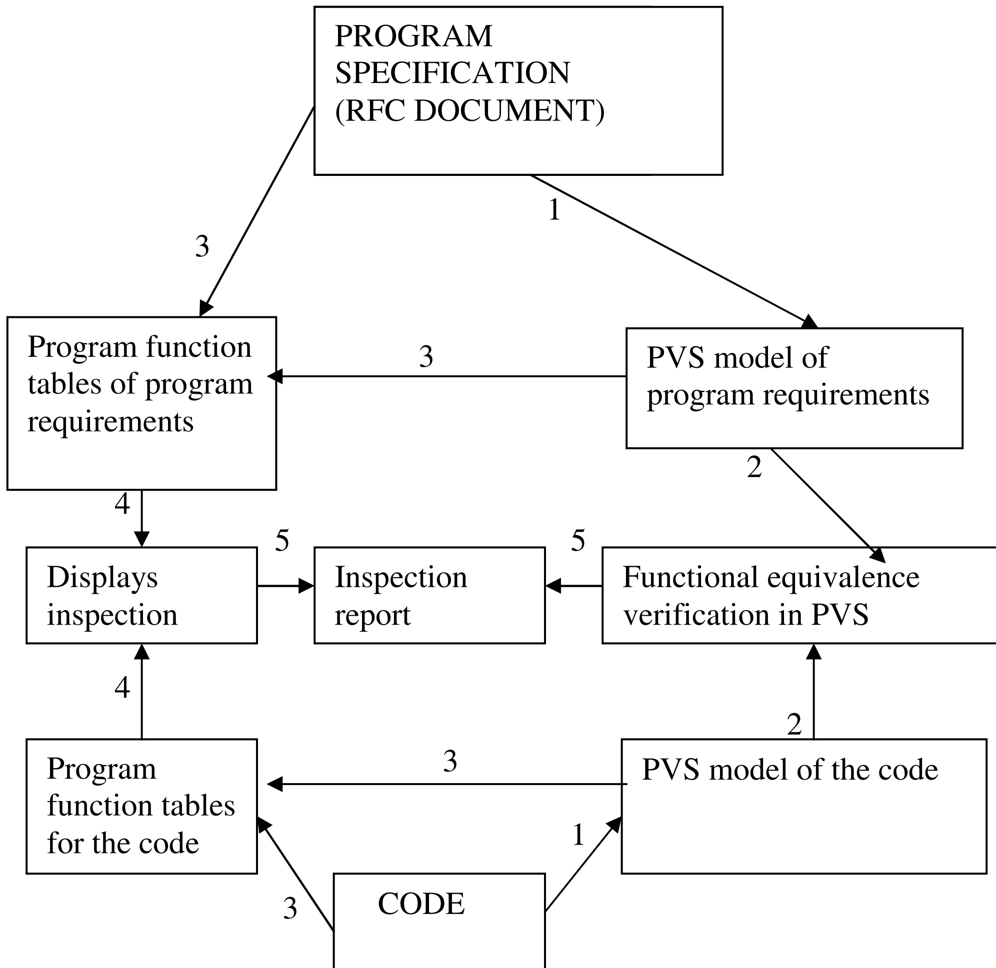

Figure 1: Inspection process (Diagram)

The arrows in the figure represent the information flow of the process and the numbers on the arrows correspond to the sequence of steps outlined below:

1) Creating PVS models of the program requirements and the code

We used the PPP RFC document as a Software Requirements Specification to create the PVS Specification file that contains the formal model of the Program Requirements.

We used the implementation code to create the PVS Implementation file. The PVS Implementation file presents the PVS model of the code using PVS's Table constructs. The tabular definitions are typechecked in PVS to ensure that they represent total functions.

2) Functional equivalence verification in PVS

In this step we compared the PVS tables for the specification and the implementation using the functional 4 variable model. In the process we found mistakes in the implementation and other discrepancies between the specification and implementation.

3) Creating Display Program Function Tables (PFT) of the program requirements and the code

The PVS models of Program Requirements and the code provided considerable help in writing the Display Program Function Tables and applying the Display method in inspecting the code. The Displays were created manually in Excel by looking at the PVS tables, PPP RFC document [Simpson 1994] and the code. Because this manual creation is a possible source of errors, a careful review of the displays is necessary to correct any such problems.

We used the PPP RFC document [Simpson 1994] for creating PFTs of program requirements and we used code for creating PFTs for the code. Having the PVS tables for the Program Requirements and the PVS tables for the Implementation code before we created PFTs put us on the "right track" to prepare PFTs in a proper and precise way. As a result of the functional verification in step (2), we paid particular attention to the problem areas. This flagging of problem areas is one the main advantages of combining

a theorem prover such as PVS with the Display method. Another advantage was increased confidence that the PFTs used for the inspection were correct.

4) Displays inspection

Using the Displays method to compare the PFTs of program requirements and the code enabled us to confirm the errors found in step (2). It also allowed us to do the software inspection on a highly detailed level, revealing some mistakes and details that would have gone undetected using PVS alone. PVS can help to do software inspection at one level but full PFT Displays are much more precise and more detailed. Each PFT for the code (part P1 of the Display - see Chapter 4 for the explanation) presents a direct mapping from the code (part P2 of the Display), and that fact gives us a high level of confidence that our inspection method is reliable.

While PVS (or another formal verification system) can provide models that are very close to the programming code and the original software requirements, it is still open to question whether or not these models are accurate representations. The Display inspection helps to validate the modeling performed in step (1). By combining the Display inspection of step (4) with the functional verification of step (2), we benefited from both PVS and the Display method in a manner that is more than the sum of its parts.

5) Inspection report

After finishing the software inspection in step (4) we wrote a software inspection report detailing the mistakes and inconsistencies we found during our software inspection. This provided an opportunity to compare the results obtained during all of the previous steps.

Summary

PVS mainly helped us to demonstrate that the formal models of the specification and implementation agree (or disagree) on all possible input sequences (guaranteed domain coverage). Theorem provers like PVS can also provide consistency checks of tabular definition of both the system specification and the implementation. Failed proofs generated during the verification process can help to identify counterexamples and provide insight into why a verification proof fails. The Display method allowed us to discover different types of mistakes. It mainly helped us to check if each mode in the specification corresponds to a set of states in the implementation i.e. if all the variables that store state information within the protocol have proper values. The Display method also enabled us to see if the code is written consistently i.e. if the values of each variable were changed properly throughout the code.

2.2. What software development process do we assume and why?

2.2.1. What is the role of program function tables in this procedure?

Any deterministic program can be described by a mathematical function whose domain consists of the set of starting states for which the program will terminate and whose range consists of the set of states in which the program terminates. The function maps a state, S, onto the state in which the program will terminate whenever it is started in state S. We call these functions *program functions*.

Although this thesis deals with deterministic programs, we note that non-deterministic programs cannot be fully described by program functions. First, a program started in a safe state may terminate in one of several distinct final states. This means that

we have a relation that is not a function. Second, a program started in a state that is not a safe one may sometimes terminate and sometimes not.

The program functions that must be described typically have many points of discontinuity [Parnas, Asmis, and Madey 1991], [Parnas, Madey, and Iglewski 1994]. We describe these functions by program function tables (PFTs). In the PFTs used for our displays, the column headings contain predicates that partition the domain of the function. The predicates characterize state classes. Every starting state for which the program's termination is possible must be in one and only one class. The row headings are program variable names; there is a row for every variable that can be changed by a program. The entries in a given row and column are expressions defining the final value of the row's variable when the program is started in one of the states characterised by the column heading. Whenever an entry in a table would be a complex conditional expression, the entry can be another table. Our experience showed that use of these tabular expressions is essential to the success of the inspection of complex programs [Parnas 1994], [Parnas, Asmis, and Madey 1991], [Parnas, Madey, and Iglewski 1994].

The purpose of the software engineering inspection process is to ensure that the implementation meets the requirements. In the Display-based, or program-function table-based method, each program is described by a function or relation that maps from "start states" to "final states". The functional descriptions are organized into a set of displays, each one of which can be inspected without looking at another display. The readability of PFTs makes them the ideal basis of a display based inspection.

Program function tables can also be used as a basis for software verification. The objective of verification is to verify, using mathematical techniques or rigorous

arguments, that the behaviour of every output on the Software Implementation is in compliance with the requirements for the behaviour of that output as specified in the Software Requirements Specification.

2.2.2. What software development process do we assume?

Software development and verification should be performed as a sequence of tasks that result in the production of detailed documents at each stage.

The software development stages relevant to this thesis are governed by the Software Requirements Specification Procedure and the Software Design Description Procedure [Jankowski, and McDougall 1995], [McDougall, and Lee 1995]. These procedures respectively produce the Software Requirements Specification and Software Design Description documents. In addition to other methods, these documents make use of Parnas' tabular representations of mathematical functions to specify the software's behaviour. Tables provide a mathematically precise notation for the Software Requirements Specification and Software Design in a visual format that is easily understood by domain experts, developers, testers, reviewers and verifiers alike.

The underlying models of both the Software Requirements Specification and Software Design are based upon Finite State Machines (FSM) [Lawford, Froebel, and Moum 2000].

2.3. How do we model the system requirements (REQ) and the software design (SOF)?

2.3.1. What is the 4-variable model?

This thesis provides a rigorous method for creating program function tables for the system design that are provably correct in the sense that they have been proved to be

functionally equivalent to a formal model of the requirements. The process of modelling the system requirements (REQ) and the software design (SOF), and verifying that the design implements the requirements is based upon a simplified functional version of the 4-variable model of Parnas [Parnas, and Madey 1995] similar to that employed in [Lawford, Froebel, and Moum 2000]. A diagram representing the proof obligation of the 4 variable model appears in Figure 2.

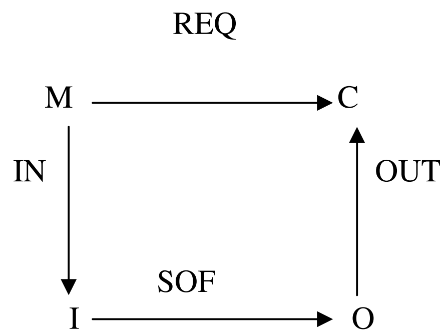

Figure 2. Commutative diagram for 4-variable model

$REQ = OUT \circ SOF \circ IN$

Here REQ represents Software Requirements Specification (SRS) state transition function mapping the monitored variables and state variables, represented by M, to the controlled variables and updated (current) state, represented by C. Monitored variables are those that the user wants the system to measure. Controlled variables are those whose values the system is intended to control.

The function SOF represents the Software Design Description (SDD) state transition function mapping the behaviour of the implementation input variables represented by statespace I to the behaviour of the software output variables represented by the statespace O.

The mapping IN relates the specification's monitored variables to the implementation's input variables, while the mapping OUT relates the implementation's output variables to the specification's controlled variables.

2.3.2 How is it related to our work?

The program function tables for the design are demonstrated to implement the requirements using PVS. The design PFTs are then used to inspect the application code, ensuring that each piece of code correctly implements its program function table, which in turn has been demonstrated to implement the systems requirements.

3 WHAT IS THE ROLE OF THEOREM PROVER SYSTEMS IN OUR WORK?

Note: The material in Chapter 3 is part of the essential background for the research reported in this thesis. It is essential for understanding the thesis, but it is not original. All of the material has been extracted from the sources named and individual sections reference the source. Those who are already familiar with the subjects of this chapter may choose to omit it.

3.1. What is important to know about formal methods and the theorem prover systems?

Central to formal methods is the use of mathematical logic. Mathematical logic serves the computer system designer in the same way that calculus serves the designer of continuous systems: as a notation for describing systems and as an analytical tool for calculating and predicting the behaviour of systems. In both design domains, computers can provide speed and accuracy for the analysis.

Formal methods involve the specification of a system using languages based on mathematical logic. Formal methods provide a means for rigorous specification of desired properties as well as of implementation details. Mathematical proof may be used to establish that an implementation meets the desired abstract properties. One of the most rigorous applications of formal methods is to use semi-automatic theorem provers to ensure the correctness of the proofs. In principle, formal methods can accomplish the equivalent of exhaustive testing, if applied all the way from requirements to implementation. However, this requires a complete verification, which is rarely done in practice.

Formal logic provides rules for constructing arguments that are sound because of their form, and independent of their meaning. Formal logic provides rules for manipulating formulas in such a manner that only valid conclusions are deducible from premises. The manipulations are called a proof. If the premises are true statements about the world, then the soundness theorems of logic guarantee that the conclusion is also a true statement about the world. Assumptions about the world are made explicit, and are separated from rules of deduction.

Theorem proving applied to real-time systems design and verification normally uses definitions and theorems to help to design, implement, validate and verify requirements. The theorem proving methodology can provide a high level specification for software requirements and a detailed description of the implementation. The implementation can then be checked against the specification to ensure that it is correct.

3.2. Which are the theorem prover systems most related to our work?

There are a number of good theorem prover systems such as: IMPS, Isabelle, HOL and TPS, PVS.

1) IMPS (Interactive Mathematical Proof System) provides organizational and computational support for the traditional techniques of mathematical reasoning. The logic of IMPS allows partial functions and undefined terms. IMPS consists of a database of mathematics and a collection of tools for applying and communicating the mathematics in the database [Farmer, Guttman, and Thayer 1993]. IMPS is a strong alternative to PVS and could have been used for the software inspection that is the subject of this thesis.

2) Isabelle is a generic theorem proof system, which provides a framework in which different logics can be introduced by specifying their syntax and rules of inference.

Isabelle has often been seen as a tool for implementing different logics and examining exotic proof systems [Paulson 1993], [Kalvala 1993].

3) The HOL system is a powerful and widely used computer program for constructing formal specifications and proofs in higher order logic [Gordon, and Melhalm 1993]. HOL supports formal reasoning in many different areas, including hardware design and verification, proofs about real-time systems, compiler verification, program correctness and program refinement. It is also used as an open platform for general theorem-proving research.

4) TPS is a theorem prover system for first-order logic and type theory. It can be used to prove theorems of first and higher order logic interactively, automatically, or in a mixture of these modes. It has facilities for searching for expansion proofs, translating these into natural deduction proofs, constructing natural deduction proofs and solving unification problems in higher-order logic [Andrews et al. 1993]. TPS has a formula editor, which facilitates constructing new formulas from others already known to TPS, and a library facility for saving formulas, definitions, and modes. The automatic proof can be done by searching for an expansion proof, and the translates this into a natural deduction proof. TPS could have been used in our work instead of PVS but the latter is more accessible.

5) PVS (Prototype Verification System) provides mechanized support for formal specification and verification. PVS consists of a number of predefined theories, a theorem prover, various utilities and documentation [Owre et al. 1995].

The specification language of PVS is based on classical, typed higher-order logic. The base types include uninterrupted types that may be introduced by the user, and built-

in types such as the booleans, integers, reals; the type-constructors include functions, sets, tuples, records and recursively-defined abstract data types, such as lists and binary trees.

PVS specifications are organized into parameterized theories that may contain assumptions, definitions, axioms, and theorems. PVS expressions provide the usual arithmetic and logical operators, function application, and quantifiers, within a natural syntax. Names may be freely overloaded, including those of the built-in operators such as AND and +. Tabular specifications of the kind advocated by Parnas are partially supported, with automated checks for disjointness and coverage of conditions for a limited number of tabular formats. An extensive prelude of built-in theories provides hundreds of useful definitions and lemmas; user-contributed libraries provide many more [Shankar, Owre, and Rushby 1993].

PVS consists of a language for creating input files containing user-defined theories, and an interactive theorem prover and decision procedures for typechecking and verifying these theories [Lawford, Froebel, and Moum 2000].

While much of the typechecking required to ensure conservative extension of the PVS logic can be done automatically, predicate subtypes and tabular specification of functions can cause PVS to generate proof obligations called Type Correctness Conditions (TCCs).

The proof strategies built into the theorem prover handle many of these proofs automatically, leaving the user to prove the more complex TCCs interactively. The proofs of any theorems in a user input file are considered incomplete until the user-defined theory, and any theories it imports, have been typechecked and any generated TCCs have been proved [Shankar, Owre, and Rushby 1993].

The following section discusses the reasons for choosing PVS as the theorem proving engine in the software inspection method we followed; it also discusses the main characteristics of PVS.

3.3. What are the advantages that led us to choose PVS?

3.3.1. Why did we choose PVS?

Checking the completeness and consistency of tables was an important part of our inspection process. In order to do that in a proper way, we had to prove theorems. We needed to select a good theorem prover system which would meet certain important criteria: 1) It should be able to deal with partial function and different data types 2) The theorems could be formulated easily 3) Many of the theorems could be verified automatically.

We have chosen PVS for the following reasons: 1) The theorems relevant to the completeness and consistency checking of Tabular Specification are automatically

generated by PVS. There are type constructors for functions, tuples and abstract data types. Using these constructors the theorems can be formulated easily. 2) By checking automatically, PVS can verify many theorems relevant to completeness and consistency. 3) The number of PVS users has been growing steadily and PVS is well-supported. 4) PVS uses classical logic and total functions. However it provides predicate and dependant types that can be used to constrain the domains of what would otherwise be partial functions. 5) Because most theorem provers have a steep learning curve, and because we were already familiar with PVS, we could proceed more quickly than if we had selected another theorem prover.

3.3.2. How does PVS support tables?

This section describes PVS's support for tables. Tables can be viewed either as one dimensional tables or as a two dimensional table with the second dimension being 1. PVS does not support full two dimension tables, tables with more than two dimensions, inverted tables, program function tables, etc.

Example:

A function $f : T_1 \times \ldots\ldots \times T_m \rightarrow T_r$ may have a tabular representation :

$f(x_1, \ldots. , x_m) =$

c_1	c_2	...	c_n
e_1	e_2	...	e_n

or

c_1	e_1
c_2	e_2
...	...
c_n	e_n

Here each c_i is a boolean expression and e_i is a term of type T_i. The interpretation is that when c_i is true f returns e_i. In this case, for the table to properly define a (total) function, it is sufficient for it to satisfy the following two conditions:

Disjointness: requires that the columns (rows) do not overlap. i.e., $i \neq j \rightarrow (c_i \wedge c_j \leftrightarrow \bot)$;

Completeness: requires that at least one column (row) is applicable to every input. i.e., $(c_1 \vee c_2 \vee \ldots. \vee c_n) \leftrightarrow \top$

Example: Let x be a real valued variable. Then the function:

sign(x) = { -1, x < 0

0, x = 0

1, x > 0

has the equivalent tabular representation:

x < 0	x = 0	x > 0
-1	0	1

The PVS input for the above table is:

```
sign_htable(x:real): signs = TABLE

%------------------%

 |[x<0 | x=0 | x>0] |

%------------------%

 | -1  | 0   | 1  ||

%------------------%

          ENDTABLE
```

Given this description, PVS automatically generates the following two "Type Correctness Conditions" corresponding to the disjointness and completeness conditions for the table.

```
% Disjointness TCC generated for

  % COND x < 0 -> -1, x = 0 -> 0, x > 0 -> 1

    ENDCOND

 % unfinished

sign_cond_TCC3: OBLIGATION

(FORALL (x: real):

NOT (x < 0 AND x = 0)

AND NOT (x < 0 AND x > 0)

AND NOT (x = 0 AND x > 0));

% Coverage TCC generated for

% COND x < 0 -> -1, x = 0 -> 0, x > 0 -> 1

  ENDCOND
```

```
  % unfinished
sign_cond_TCC4: OBLIGATION
(FORALL (x: real): x < 0 OR x = 0 OR x > 0);
```

In this case, both of these theorems are automatically proved by PVS. When the built-in proof strategies and user intervention fail, the resulting unprovable sequent(s) can often provide useful information regarding the incompleteness or inconsistency of specifications. For further details on PVS's support for tabular notation the interested reader is referred to [Lawford, Froebel, and Moum 2000], [Owre, Rushby, and Shankar 1997].

4 WHAT IS THE ROLE OF THE DISPLAY METHOD IN OUR WORK?

Note: The material in Chapter 4 is part of the essential background for the research reported in this thesis. It is essential for understanding the thesis, but it is not original. All of the material has been extracted from the sources named and individual sections reference the source. Those who are already familiar with the subjects may choose to omit it. All of the definitions in section 4.1.1., 4.1.2, 4.1.3 and 4.1.4 are taken from [Parnas, Madey, and Iglewski 1994]. There are detailed examples illustrating these concepts in [Parnas, Madey, and Iglewski 1994].

Displays are the main method employed by our inspection process to ensure the correctness of the code and to validate our formal models. The remainder of this chapter provides the information the reader requires to understand how displays can play this central role.

4.1 What is the main concept of the Display method?

4.1.1. What is a Display?

A Display is a precise document in which a program is presented in such a way that its correctness can be determined without examining other displays.

A Display consists of the following three parts:

P1: a specification for the program presented in this display,

P2: the program itself. The names of other programs may appear in this text; we say that these programs are invoked in this display,

P3: specifications of all programs invoked in P2 that are not known. (A *known* program is one that does not require a specification. The semantics of known programs are assumed to be understood. Every project should have a list of programs that are considered to be known).

4.1.2. What is a Lexicon?

Definition of Lexicon:

A lexicon is a dictionary containing definitions of any mathematical functions, program constants, types, etc. that are used in more than one display.

4.1.3. What is an index?

Definition of index:

An index is a list of all the variables and, programs, indicating where those items appear in the displays. If some names are used with more than one meaning, we also describe the category of each name.

4.1.4. What are completeness and correctness?

Each display can be reviewed without any reference to other displays; its correctness can be verified without looking at the implementation of either the programs that are invoked in that display or the programs that invoke the program it describes.

Definition of Correctness:

A display is *correct* if the program in P2 will satisfy the specification in P1, provided that the programs invoked in P2 satisfy the specifications given in P3.

Definition of Completeness:

A set of displays is *complete*, if for each specification of a program that is found in P3 of a display, there exist another display in which this specification is in P1.

A set of displays is correct if 1) the set of displays is complete and 2) all displays are correct.

4.1.5. What is the role of mathematical notation and mathematical methods in the Display method?

The Display method is based on a mathematical model of programs and uses mathematical notation to provide precise descriptions of programs. The use of mathematical methods in software engineering emphasize program development or verification.

The method is based on the recognized fact that a well-structured program can always be written as a short text in which the names of other programs may appear and the programs named can also be short. The down-side of such an organization is that there will be many programs and to understand any one of them, one must understand several others. We overcome this by presenting material in displays.

In documentation, notation is very important; documents are to be read by experts from a variety of fields and should be easily understood. We must apply the principle "divide and conquer" when designing notation; readers should not have to parse long expressions.

4.1.6. What is the role of tables and tabular notation in the Display method?

Our approach is based on the use of tables to describe mathematical functions, relations and sets; such tabular notation has already been used in practice and has proved practical. Tabular notation decreases the complexity of expressions in three ways [Parnas, Asmis, and Madey 1991]:

1) The table parses the expression for the reader; many nested pairs of parentheses are eliminated, and the intended structure of expressions is revealed

2) The table eliminates many repetitions of the subexpressions that appear in column headings

3) Because each table entry only applies to a small part of the function's domain, the expression in that entry can be simplified.

Figure 3 has an example of predicate logic expression and its equivalent table that illustrates the above 3 points.

$$(((\exists i, B[i] = x) \wedge B[j'] = x \wedge (present' = \mathbf{true})) \vee ((\forall i, ((i \leq i \leq N) \rightarrow B[i] \neq x)) \wedge (present' = \mathbf{false}))) \wedge (\text{‘}x=x\text{’}) \wedge (\text{‘}B=B\text{’})$$

Figure 3 Conventional predicate specifying a search program

	$(\exists i, B[i] = x)$	$(\forall i, ((i \leq i \leq N) \rightarrow B[i] \neq x))$
$j' \mid$	$B[j'] = x$	**TRUE**
present'=	**TRUE**	**FALSE**

$\wedge$**NC (x, B)**

Tabular expression equivalent to Figure 3

4.2. What is the motivation to use the Display method in our inspection?

Anyone who has ever seriously read a lengthy program produced by others (for example to inspect it or to make changes to it) realizes the importance of documentation. The combination of a large amount of detail with inaccurate or vague descriptions of the structure makes it quite common for serious errors to escape the reviewer's attention. Therefore, a program's document organization should focus attention on: 1) Good organization of the structure of the set of documents, 2) Readability of the documents, 3) Independence of each individual document, 4) Consistency of documents, 5) Completeness of the set of documents.

The Display method was introduced to make program documentation more useful when a program is lengthy. The main idea is to present a program as a set of displays with the property that each display can be examined and verified without looking at any other displays. The displays are the main method used to ensure the correctness of the code.

4.3. What is the Display Management System (DMS)?

The Display Management System (DMS) is a prototype of a tool to support users of the Display method [Wang 1995]. It is intended to support the management aspect of software design projects. DMS lets the manager know which displays have been checked for correctness, whether or not a display that has been checked has been found to be correct, and whether or not a set of displays is complete and correct. The system re-evaluates the status of each display and notifies checkers if a program that they have already checked must be checked again.

The DMS, like some software managers, does not read programs or specifications. It can only manage files and evaluate the content of those files from their names. No syntactic or semantic checks can be done by DMS. A tool like DMS (but improved) could help greatly in applying the inspection method used in this thesis.

Future work should include developing an improved system similar to DMS in order to make our inspection method more efficient. We did not use the DMS because of its prototype nature and lack of features that would make it really useful.

5 WHAT IS THE PURPOSE OF THE PPP PROTOCOL AND HOW DOES IT WORK?

Note: The material in Chapter 5 is part of the essential background for the research reported in this thesis. It is essential for understanding the thesis, but it is not original. All of the material has been extracted from the sources named and individual sections reference the source. Those who are already familiar with the subjects of this chapter may choose to omit it.

5.1. What is the purpose of PPP?

The Point-to-Point Protocol is designed for simple links which transport packets between two peers. These links are assumed to deliver packets in order. PPP provides a common solution for easy connection of a wide variety of hosts, bridges and routers [Simpson 1994].

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. *A datagram* is the unit of transmission in the network layer. A datagram may be encapsulated in one or more packets passed to the data link layer.

5.2. What is the structure of PPP?

PPP has three main components:

1. A method for encapsulating multi-protocol datagrams. The PPP encapsulation provides for multiplexing of different network-layer protocols simultaneously over the same link. The PPP encapsulation has been carefully designed to retain compatibility with most commonly used supporting hardware. The PPP encapsulation is used to

disambiguate multiprotocol datagrams. This encapsulation requires framing to indicate the beginning and end of the encapsulation.

2. A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection. The LCP is used to automatically agree upon the encapsulation format options, handle varying limits on sizes of packets, detect a looped-back link and other common misconfiguration errors, and terminate the link.

3. A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols. Point-to-Point links tend to exacerbate many problems with the current family of network protocols. For instance, assignment and management of IP addresses, which is a problem even in LAN environments, is especially difficult over circuit-switched point-to-point links (such as dial-up modem servers). These problems are handled by a family of Network Control Protocols (NCPs), each of which manages the specific needs required by its respective network-layer protocols.

In order to establish communications over a point-to–point link, each end of the PPP link must first send LCP packets to configure and test the data link. Then, PPP must send NCP packets to choose and configure one or more network-layer protocols. Once each of the chosen network-layer protocols has been configured, datagrams from each network-layer protocol can be sent over the link. The link remains configured for communications until explicit LCP or NCP packets close the link down.

5.3. What is the Option "Negotiation Automaton"?

The option "negotiation automaton" is defined in terms of events, actions and state transitions. *An event* is a class of state transitions. Events include reception of

external commands such as `Open` and `Close`, expiration of the Restart timer, and reception of packets from a peer. *A packet* is the basic unit of encapsulation, which is passed across the interface between the network layer and the data link layer. *A peer* is the other end of the point-to-point link.

Actions include the starting of the Restart timer and transmission of packets to the peer [Simpson 1994].

When initiating a PPP connection, the host first sends a configuration request packet (`scr`) to its peer and waits for the acknowledgment (`RCA` or `RCN`). The peer responds by checking the options sent in the request. If the options are acceptable, the peer sends a positive acknowledgment (`sca`). Otherwise, a negative acknowledgment (`scn`) is sent to the host. In any case, the peer also sends its configuration request packet to the host. They try to negotiate options acceptable to both of them.

After they agree on the options, both move to the `Opened` state and start the authentication phase (or data transmission, if authentication is not required). The communication can be terminated by a `Close` event explicitly or a `Down` event (perhaps due to hardware failure). A termination request (`str`) is sent if the link is closed explicitly. A restart counter is used to monitor the responses to request actions (`scr` and `str`). If the host has not received the acknowledgment from the peer when the timer expires, and if the counter is greater than zero, it sends another request. Otherwise, it stops the connection locally [Simpson 1994], [Alur, and Wang 2001].

5.4. What does the PPP State Transition Table look like?

Column headers are states and row headers are events. State transitions and actions are represented in the form action/new-state. In lists, actions are separated by

commas and they are executed in an arbitrary order. The dash ('-') indicates an illegal transition [Alur, and Wang 2001].

The State Transition Table in Figure 4 differs from the State Transition Table in RFC 1661 [Simpson 1994]. Two events (RXR – Receive Echo Request and RUC – Receive Unknown Code) are not included in the State Transition Table in Figure 4. There is no written code in the fsm.c for the event RXR - Receive Echo Request. In RFC 1661[Simpson 1994] Section "5.8 Echo-Request and Echo-Reply" gives some insight for the RXR event: "Echo-Request and Echo-Reply packets MUST only be sent in the LCP opened state. Echo-Request and Echo-Replay packets received in any state other than LCP Opened state SHOULD be silently discarded". This section of the RFC also says that the event is mainly intended for debugging and some performance analysis, so it is not critical for correct operation of the basic protocol.

RUC event [Simpson 1994] occurs when a Code-Reject or a Protocol-Reject packet is received from the peer. In the fsm.c code this event is implemented by calling function `fsm_sdata(f, CODEREJ, ++f->id, inpacket, len + HEADERLEN)`.

The fact that the RUC event is not presented like the other events in the fsm.c code and as is proposed in RFC 1661[Simpson 1994] does not mean a lack of implementation in this case. RUC event is just implemented differently by using `fsm_sdata` function.

RUC – Receive Unknown Code event and RUC – Receive Unknown Code event were also not included in [Alur, and Wang 2001].

Detailed descriptions of all the states, actions and events can be found in the following subsections.

	0	1	2	3	4	5	6	7	8	9
	Initial	Starting	Closed	Stopped	Closing	Stopping	Req-Sent	Ack-Rcvd	Ack-Sent	Opened
Up	2	irc, scr/6	---	---	---	---	---	---	---	---
Down	---	---	0	tls/1	0	1	1	1	1	tld/1
Open	tls/1	1	irc, scr/6	3	5	5	6	7	8	9
Close	0	tlf/0	2	2	4	4	irc, str/4	irc, str/4	irc, str/4	tld, irc, str/4
TO +	---	---	---	---	str/4	str/5	scr/6	scr/6	scr/8	---
TO --	---	---	---	---	tlf/2	tlf/3	tlf/3	tlf/3	tlf/3	---
RCR +	---	---	sta/2	irc, scr, sca/8	4	5	sca/8	sca, tlu/9	sca/8	tld, scr, sca/8
RCR --	---	---	sta/2	irc, scr, scn/6	4	5	scn/6	scn/7	scn/6	tld, scr, scn/6
RCA	---	---	sta/2	sta/3	4	5	irc/7	scr/6	irc, tlu/9	tld, scr/6
RCN	---	---	sta/2	sta/3	4	5	irc, scr/6	scr/6	irc, scr/8	tld, scr/6
RTR	---	---	sta/2	sta/3	sta/4	sta/5	sta/6	sta/6	sta/6	tld, zrc, sta/5
RTA	---	---	2	3	tlf/2	tlf/3	6	6	8	tld, scr/6
RXJ	---	---	tlf/2	tlf/3	tlf/2	tlf/3	tlf/3	tlf/3	tlf/3	tld, irc, str/5

Figure 4 PPP State Transition Table

5.5. What are the Counters and Timers within PPP?

5.5.1. What is the Restart Timer (within PPP)?

There is one special timer used by the automaton. The Restart timer is used to time transmissions of `Configure-Request` and `Terminate-Request` packets. Expiration of the Restart timer causes a `Timeout` event, and retransmission of the corresponding `Configure-Request` or `Terminate-Request` packet.

5.5.2. What are Max Terminate, Max Configure, Max Failure?

Max-Terminate

There is one required restart counter for `Terminate-Request`. `Max-Terminate` indicates the number of `Terminate-Request` packets sent without receiving a `Terminate-Ack` before assuming that the peer is unable to respond.

Max-Configure

A similar counter is recommended for `Configure-Requests`. `Max-Configure` indicates the number of `Configure-Request` packets sent without receiving a valid `Configure-Ack`, `Configure-Nak` or `Configure-Reject` before assuming that the peer is unable to respond.

Max-Failure

A related counter is recommended for `Configure-Nak`. `Max-Failure` indicates the number of `Configure-Nak` packets sent without sending a `Configure-Ack` before assuming that configuration is not converging.

5.6. What are the main characteristics of each state in the PPP State Transition Table?

`Initial`

In the `Initial` state, the lower layer is unavailable (`Down`), and no `Open` has occurred. The Restart timer is not running in the Initial state.

`Starting`

In the `Starting` state an administrative `Open` has been initiated, but the lower layer is still unavailable (`Down`). The Restart timer is not running in the `Starting` state. When the lower layer becomes available (`Up`), a `Configure-Request` is sent.

`Closed`

In the `Closed` state, the link is available (`Up`), but no `Open` has occurred. The Restart timer is not running in the `Closed` state.

`Stopped`

The `Stopped` state is entered when the automaton is waiting for a `Down` event after the `This-Layer-Finished` action, or after sending a `Terminate-Ack` . The Restart timer is not running in the `Stopped` state.

`Closing`

In the `Closing` state, an attempt is made to terminate the connection. A `Terminate-Request` has been sent and the Restart timer is running, but a `Terminate-Ack` has not yet been received.

Stopping

In the `Stopping` state a `Terminate-Request` has been sent and the Restart timer is running, but a `Terminate-Ack` has not yet been received.

Request-Sent

In the `Request-Sent` state an attempt is made to configure the connection. A `Configure-Request` has been sent and the Restart timer is running, but a `Configure-Ack` has not yet been received nor has one been sent.

Ack-Received

In the `Ack-Received` state, a `Configure-Request` has been sent and a `Configure-Ack` has been received. The Restart timer is still running, since a `Configure-Ack` has not yet been sent.

Ack-Sent

In the `Ack-Sent` state, a `Configure-Request` and a `Configure-Ack` have both been sent, but a `Configure-Ack` has not yet been received. The Restart timer is running, since a `Configure-Ack` has not yet been received.

Opened

In the `Opened` state, a `Configure-Ack` has been both sent and received. The Restart timer is not running.

5.7. What are the main characteristics of each event in the PPP State Transition Table?

`Up`

This event occurs when a lower layer indicates that it is ready to carry packets.

`Down`

This event occurs when a lower layer indicates that it is no longer ready to carry packets.

`Open`

This event indicates that the link is administratively available for traffic. When this event occurs, and the link is not in the `Opened` state, the automaton attempts to send configuration packets to the peer. If the automaton is not able to begin configuration (the lower layer is `Down`, or a previous `Close` event has not completed), the establishment of the link is automatically delayed. When a `Terminate-Request` is received, or other events occur which cause the link to become unavailable, the automaton will progress to a state where the link is ready to re-open. No additional administrative intervention is necessary.

Close

This event indicates that the link is not available for traffic; that is, the network administrator has indicated that the link is not allowed to be `Opened`. When this event occurs, and the link is not in the `Closed` state, the automaton attempts to terminate the connection. Further attempts to re-configure the link are denied until a new `Open` event occurs.

Timeout(TO+event,TO-event)

This event indicates the expiration of the Restart timer. The Restart timer is used to time responses to `Configure-Request` and `Terminate-Request` packets. The TO+ event indicates that the Restart counter continues to be greater than zero, which triggers the corresponding `Configure-Request` or `Terminate-Request` packet to be retransmitted. The `TO-` event indicates that the Restart counter is not greater than zero, and no more packets need to be retransmitted.

Receive-Configure-Request(RCR+,RCR-)

This event occurs when a `Configure-Request` packet is received from the peer. The `Configure-Request` packet indicates the desire to open a connection and may specify Configuration Options. The `RCR+` event indicates that the `Configure-Request` was acceptable, and triggers the transmission of a corresponding `Configure-Ack`. The `RCR-` event indicates that the `Configure-Request` was unacceptable, and triggers the transmission of a corresponding `Configure-Nak` or `Configure-Reject`.

Receive-Configure-Ack (RCA)

This event occurs when a valid `Configure-Ack` packet is received from the peer. The `Configure-Ack` packet is a positive response to a `Configure-Request` packet. An out of sequence or otherwise invalid packet is silently discarded.

Receive-Configure-Nak/Rej(RCN)

This event occurs when a valid `Configure-Nak` or `Configure-Reject` packet is received from the peer. The `Configure-Nak` and `Configure-Reject` packets are negative responses to a `Configure-Request` packet. An out of sequence or otherwise invalid packet is silently discarded.

Receive-Terminate-Request(RTR)

This event occurs when a `Terminate-Request` packet is received. The `Terminate-Request` packet indicates the desire of the peer to close the connection.

Receive-Terminate-Ack(RTA)

This event occurs when a `Terminate-Ack` packet is received from the peer. The `Terminate-Ack` packet is usually a response to a `Terminate-Request` packet. The `Terminate-Ack` packet may also indicate that the peer is in `Closed` or `Stopped` states, and serves to re-synchronize the link configuration.

Receive-Code-Reject,Receive-Protocol-Reject(RXJ)

This event occurs when a `Code-Reject` or a `Protocol-Reject` packet is received from the peer. The `RXJ-` event arises when the rejected value is catastrophic, such as a `Code-Reject` of Configure-Request, or a `Protocol-Reject` of LCP! This event communicates an unrecoverable error that terminates the connection.

5.8. What are the main characteristics of each action in the PPP State Transition Table?

Actions in the automaton are caused by events and typically indicate the transmission of packets and/or the starting or stopping of the Restart timer.

Illegal-Event (-)

This indicates an event that cannot occur in a properly implemented automaton. The implementation has an internal error, which should be reported and logged. No transition is taken, and the implementation should not reset or freeze.

This-Layer-Up(tlu)

This action indicates to the upper layers that the automaton is entering the `Opened` state.

Note: `tlu` action is implemented by

```
if (f->callbacks->up)
      (*f->callbacks->up)(f);
```

This-Layer-Down(tld)

This action indicates to the upper layers that the automaton is leaving the `Opened` state.

Note: `tld` action is implemented by

```
if( f->callbacks->down )
    (*f->callbacks->down)(f);
```

This-Layer-Started(tls)

This action indicates to the lower layers that the automaton is entering the `Starting` state, and the lower layer is needed for the link.

Note: `tls action` is implemented by

```
if( f->callbacks->starting )
    (*f->callbacks->starting)(f);
```

This-Layer-Finished(tlf)

This action indicates to the lower layers that the automaton is entering the `Initial`, `Closed` or `Stopped` states, and the lower layer is no longer needed for the link. The lower layer SHOULD respond with a `Down` event when the lower layer has terminated.

Note: `tlf action is implemented by`

```
if( f->callbacks->finished )
    (*f->callbacks->finished)(f);
```

Initialize-Restart-Count(irc)

This action sets the Restart counter to the appropriate value (`Max-Terminate` or `Max-Configure`). The counter is decremented for each transmission, including the first.

Note: `irc` action is implemented by function

```
TIMEOUT(fsm_timeout, f, f->timeouttime);
```

Zero-Restart-Count(zrc)

This action sets the Restart counter to zero.

Note: `zrc` action is implemented by function

```
UNTIMEOUT(fsm_timeout, f);
```

Send-Configure-Request(scr)

A Configure-Request packet is transmitted. This indicates the desire to open a connection with a specified set of Configuration Options. The Restart timer is started when the Configure-Request packet is transmitted, to guard against packet loss. The Restart counter is decremented each time a Configure-Request is sent.

Note: scr action is implemented by functions

fsm_sconfreq(f, 0) and
fsm_sdata(f, CONFREQ, f->reqid, outp, cilen)

Send-Configure-Ack(sca)

A Configure-Ack packet is transmitted. This acknowledges the reception of a Configure-Request packet with an acceptable set of Configuration Options.

Note: sca action is implemented by function

```
fsm_sdata(f, CONFACK, id, inp, len);
```

Send-Configure-Nak(scn)

A Configure-Nak packet is transmitted, as appropriate. This negative response reports the reception of a Configure-Request packet with an unacceptable set of Configuration Options. Configure-Nak packets are used to refuse a Configuration Option value, and to suggest a new, acceptable value.

Note: scn action is implemented by function

```
fsm_sdata(f, CODEREJ, ++f->id, inpacket, len + HEADERLEN);
```

Send-Terminate-Request(str)

A Terminate-Request packet is transmitted. This indicates the desire to close a connection. The Restart timer is started when the Terminate-Request packet is transmitted, to guard against packet loss. The Restart counter is decremented each time a Terminate-Request is sent.

Note: str action is implemented by function

```
fsm_sdata(f, TERMREQ, f->reqid = ++f->id,
        (u_char *) f->term_reason, f->term_reason_len);
```

Send-Terminate-Ack(sta)

A Terminate-Ack packet is transmitted. This acknowledges the reception of a Terminate-Request packet or otherwise serves to synchronize the automatons.

Note: sta action is implemented by function

```
fsm_sdata(f, TERMACK, id, NULL, 0);
```

6 HOW WAS OUR INSPECTION TECHNIQUE APPLIED TO THE LINUX PPP DAEMON?

6.1. What part of the PPP code did we inspect?

The implementation ppp version 2.4.1 is an open-source package included in various Linux distributions and widely used by Linux users. The package contains several tools for monitoring and maintaining PPP connections. The daemon `pppd` implements the protocol and is our concern here. The file `main.c` uses the subroutines defined in `fsm.c` (~900 lines) to implement the finite state machine. Events and actions described in Chapter 5 have their corresponding subroutines in `fsm.c`. The scope of our inspection is limited to the code implementing the protocol state machine transitions. The code implementing the actions is not inspected.

6.2. How did we use PVS in inspecting the Linux PPP daemon?

Programs such as `fsm.c`, that implement the PPP protocol are required to do three things: 1) update the state of the protocol FSM 2) produce a sequence of output actions 3) possibly start or stop a timer associated with timeout events. In PVS, we represent this with the program requirement state type ProgReq. In this way, we represent controlled variables (see 4-variable model in Chapter 2).

ProgReq: TYPE= [machine: fsm , actions: ActionSeq, timer_op: TimerOp]

The "actions" of type ActionSeq is a sequence of actions such as `irc`, `scr` in the case of `Starting` corresponding to the entry `Up` of the RFC protocol table.

```
Action: TYPE = {tlu,tld,tls,tlf,irc,zrc,scr,sca,scn,str,sta,none};

ActionSeq:TYPE =finseq[Action]

TimerOp: TYPE = {start, stop, none}

none:empty_seq

fsm:TYPE =
      [# state: subrange(0,9),

                  flags: int,
                  id: u_char,
                  reqid: u_char,
                  seenack: u_char,
                  timeouttime: int,
                  maxconfreqtransmits: int,
                  retransmits: int,
                  maxtermtransmits: int,
                  nakloops: int,
                  maxnakloops: int,
                  term_reason: string,
                  term_reason_len: int,
                  callbacks: fsm_callbacks
       #]
```

In PVS, we also represent the implementation state type as output variables according to the 4-variable model. The mapping OUT in the 4 variable model relates the

implementation's output variables to the specification's controlled variables and it was obvious and straightforward in our example.

Several steps were completed during the PVS software inspection and verification procedure. First, PVS was used to create a precisely formalized table for each of the protocol events described in the state transition table and text of [Simpson 1994]. Thus a PVS table was created from each row of the transition table shown in section 5.4. (see points 2 and 3 in Section 6.3. for the more detailed explanation). Second, the ppp implementation code was inspected and PVS tables which correspond to the actual C code were prepared.

Third, comparing the tables from the specification with the tables from the implementation, we found mistakes in the implementation and differences between the specification and the implementation. It enabled us to analyse the ppp code and its weaknesses.

During our inspection four separate PVS files were written: 1) `pppSpec.pvs` - the PVS file containing the Specification tables 2) `pppCode.pvs` – the PVS file containing the Implementation tables 3) `pppCommon.pvs` - the PVS file declaring and defining parameters and variables common to both the PVS Specification file and the PVS Implementation file) 4) `pppInspection.pvs` - the PVS Inspection file in which PVS Specification tables and PVS Implementation tables were compared for each event in the `fsm.c` code. Running PVS on the Inspection file detected all the mistakes, discrepancies, inconsistencies and differences between tables in the PVS Specification file and the PVS Implementation file with only minimal user intervention. Concrete examples are given in subsections 6.4.1 and 6.4.2. Complete versions of the 4 files appear in Appendix A.

6.3. How did we use the Display method in inspecting the Linux PPP daemon?

In order to connect all of the events and to make the inspection document more readable and understandable for maintainers and reviewers, the Display method was applied. Below we outline the steps that were followed during the software inspection of the `fsm.c` ppp protocol code.

Our specification for the ppp code is a table that describes 10 states and the rules for transitions from one state to another (see Fig. 4 of section 5.4). The protocol must be seen as a module (a set of programs rather than a single one) and the state table as a module specification rather than a program specification. The display method starts with a program specification.

To make the transition we need an abstraction function. This is a mapping from the values of the internal variables in the ppp code to the externally visible states. Here is a rough outline of how to do this:

1) Identify all the variables that store state information within the protocol.

2) By studying the code hypothesize a mapping from states in the variables to modes in the specification. Essentially, each mode in the ppp specification corresponds to a set of states in the implementation. We need to write a predicate that characterises the set of internal states corresponding to each mode, or we can write a function that maps from the values of the internal variables to the actual mode of the software.

3) From this characterization of the relation between the low level states and the upper level states, we can prepare the required program functions. We obtain a set of program functions, one for each transition in the table. The domain is the set of states in the start mode for the transition, the range is the set of states in the target mode of the

transition. In most cases this relation will be a function with this relation we can then begin the inspection process.

4) There are lower level programs invoked by the code during execution. These must be identified. Their effect must be documented in the lexicon or in the "programs invoked" section of each display. We are now prepared to start the inspection of the code that implements each transition.

To implement the Display method in a proper way and to cover all possible cases, we had to write a display for every event in the manner explained above. Concrete examples are given in subsections 6.4.1 and 6.4.2. Appendix B contains displays for all events (functions in `fsm.c`).

6.4 Concrete examples

Examples in this section were chosen because they show several interesting facts and occurrences encountered during our software inspection. Some additional comments about example 1 (section 6.4.1) and example 2 (section 6.4.2) are found in Section 7.2 – points 2 and 9, respectively.

6.4.1. Example 1 with comments

The `fsm_lowerup` function corresponds to the first row "Up" of table in fig 4 in section 5.4. The fact that this is a partial function is indicated by the predicate subtype on its statemachine argument, which restricts it to the case when the FSM state is either `INITIAL` or `STARTING` corresponding to the first two columns of the table in Fig 4. in section 5.4.

PVS SPECIFICATION TABLE (UP EVENT):

```
  fsm_lowerup(f:{g:fsm|state(g)=INITIAL OR state(g)=STARTING}):ProgReq
=
       TABLE
%-----------------------+-------------------------------------------||
|state(f)=INITIAL      |(# machine:= f WITH [state:= CLOSED], %|
                          actions:=none, %|
                          timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=STARTING      |(# machine:= fsm_sconfreq(f,FALSE)
                                      WITH [state:= REQSENT], %|
                                      actions:= irc o scr, %|
                                       timer_op:=none #) ||
%-----------------------+-------------------------------------------||

        ENDTABLE
```

The actions in the state(f)=`STARTING` row correspond to the action sequence proscribed by Fig. 4 in section 5.4 in this case. No timer actions are required.

The disjointness and completness obligations for the table are automatically discharged by PVS.

PVS IMPLEMENTATION TABLE (UP EVENT):

```
  fsm_lowerup(f:{g:fsm|state(g)=INITIAL OR state(g)=STARTING}):ProgReq
=
       TABLE
%-----------------------+--------------------------------------------||
|state(f)=INITIAL      |(# machine:= f WITH [state:= CLOSED], %|
                           actions:=none, %|
                           timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STARTING
  & IsSet(flags(f),OPT_SILENT)|(# machine:= f WITH [state:= STOPPED],%|
                                  actions:=none, %|
                                  timer_op:=none #)||
%-----------------------+--------------------------------------------||
|state(f)=STARTING
  & NOT IsSet(flags(f),OPT_SILENT) |(# machine:= fsm_sconfreq(f,FALSE)
                                       WITH [state:= REQSENT], %|
                                       actions:= irc o scr, %|
                                        timer_op:=none #) ||
%-----------------------+--------------------------------------------||

        ENDTABLE
```

COMMENT:

Here we can compare those two tables and denote the difference between specification and implementation. The implementers added an extra condition based upon a flag value in the event `fsm-lowerup` in the state(f)= `STARTING` case. If the flag value is set to `OPT_SILENT` then the next state is `STOPPED`. The implementers did this to write secure code and provide more reliable implementation.

PVS INSPECTION FILE:

The following PVS code extracted from the `pppInspection.pvs` file states the proposition that the functions defined by the specification table and implementation table agree on all of their inputs.

```
pppInspection : THEORY

  BEGIN

  IMPORTING pppSpec, pppCode
  f:VAR fsm
  flag:VAR int


  Up:PROPOSITION pppSpec.fsm_lowerup = pppCode.fsm_lowerup
```

The proof of the above theorem fails in the case when the `OPT_SILENT` flag is set in the implementation since then the state is updated to `STOPPED` rather than `REQ_SENT`.

The display follows for actual code segment modelled by the implementation table above.

DISPLAY 2

Specification

d fsm_lowerup (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME										
	' state = INITIAL	' state = STARTING ∧		' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
		flags&OPT_SILENT	¬ (flags&OPT_SILENT)								
state' =	CLOSED	STOPPED	REQSENT								
callbacks' =	none	none	addci								
timer' =	none	none	start								
flags' =	' flags	OPT_SILENT	' flags								
code' =	' code	' code	CONFREQ								
reqid' =	' reqid	' reqid	id +1								
timeouttime' =	' timeouttime	' timeouttime	DEFTIMEOUT								
retransmits' =	' retransmits	' retransmits	maxconfreqtransmits								
protoname' =	' PROTONAME	' PROTONAME	' PROTONAME	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "
data' =	' data	' data	outp								
datalen' =	' datalen	' datalen	cilen								

Program

```
/*
 * fsm_lowerup - The lower layer is up
 */
void
fsm_lowerup(f)
    fsm *f;
{
    switch( f->state ){
    case INITIAL:
        f->state = CLOSED;
        break;

    case STARTING:
        if( f->flags & OPT_SILENT )
            f->state = STOPPED;
        else {
            /* Send an initial configure-request */
            fsm_sconfreq(f, 0);
            f->state = REQSENT;
        }
        break;

    default:
        FSMDEBUG(("%s: Up event in state %d!", PROTO_NAME(f), f>state));
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)		u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen					
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

FSMDEBUG(PROTO_NAME, state)					
R2 (,) = (PROTO_NAME = "MISTAKE")					

On Display 17

End of Display 2

The display helps to provide a second check of the error discovered in the PVS verification. The display also adds more details allowing a review of the PFT to ensure that it correctly represents what the code needs to do. The action `scr` in the code above is implemented by function `fsm_sconfreq(f, 0)`. The complete list of actions being implemented by the C function calls in the code is at the beginning of Appendix B (see IMPORTANT NOTE 2)

6.4.2 Example 2 with comments

The `fsm_rtermack` function corrresponds to the 12th row "`RTA`" of the table in Fig 4 in Section 5.4. The fact that this is a partial function is indicated by the predicate subtype on its statemachine argument which restricts it to the case when the FSM state is either `CLOSED` or `STOPPED` or `CLOSING` or `STOPPING` or `REQSENT` or

ACKRCVD or ACKSENT or OPENED corresponding to the columns 2, 3 , 4 , 5 , 6 , 7 , 8 , 9 of the table in Fig 4. in section 5.4.

PVS SPECIFICATION TABLE (RTA EVENT):

```
fsm_rtermack(f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}): ProgReq =
        TABLE
%------------------------+---------------------------------------------||
|state(f)=CLOSED        | (# machine:=f,                      %|
                            actions:=none,                   %|
                            timer_op:=none #) ||
%------------------------+---------------------------------------------||
|state(f)=STOPPED       | (# machine:=f,                      %|
                            actions:=none,                   %|
                            timer_op:=none #) ||
%------------------------+---------------------------------------------||
|state(f)=CLOSING       | (#machine:= f WITH  [state:= CLOSED], %|
                                actions:= tlf, %|
                                timer_op:=stop #) ||
%------------------------+---------------------------------------------||
%|state(f)=STOPPING      | (#machine:=f WITH [state:= STOPPED]), %|
                            actions:= tlf, %|
                            timer_op:=stop #)||
%------------------------+---------------------------------------------||
|state(f)=REQSENT       | (# machine:=f,                      %|
                            actions:=none,                   %|
                            timer_op:=none #) ||
%------------------------+---------------------------------------------||

|state(f)=ACKRCVD        | (#machine:=f WITH [state:= REQSENT]), %|
                             actions:= none, %|
                             timer_op:=none #)    ||
%------------------------+---------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f,                      %|
                            actions:=none,                   %|
                            timer_op:=none #) ||
%------------------------+---------------------------------------------||
|state(f)=OPENED          |
            (#machine:= fsm_sconfreq(f,FALSE)WITH [state:=REQSENT]), %|
                           actions:= tld o scr, %|
                           timer_op:=none #) ||
%------------------------+---------------------------------------------||

             ENDTABLE
```

The actions in the state(f)=CLOSED, state(f)=STOPPED, state(f)=CLOSING, state(f)=STOPPING, state(f)=REQSENT, state(f)=ACKRCVD, state(f)=ACKSENT,

state(f)=OPENED rows correspond to the action sequence proscribed by Fig. 4 in section 5.4 in this case. No timer actions are required except for the state(f)=CLOSING, state(f)=STOPPING . The disjointness and completeness obligations for the table are automatically discharged by PVS.

PVS IMPLEMENTATION TABLE (RTA EVENT):

```
fsm_rtermack(f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}): ProgReq =
        TABLE
%-----------------------+---------------------------------------------||
|state(f)=CLOSED        | (# machine:=f,                    %|
                           actions:=none,                   %|
                           timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=STOPPED       | (# machine:=f,                    %|
                           actions:=none,                   %|
                           timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=CLOSING       | (#machine:= f WITH  [state:= CLOSED], %|
                               actions:= tlf, %|
                               timer_op:=stop #) ||
%-----------------------+---------------------------------------------||
%|state(f)=STOPPING      | (#machine:=f WITH [state:= STOPPED]), %|
                           actions:= tlf, %|
                           timer_op:=stop #)||
%-----------------------+---------------------------------------------||
|state(f)=REQSENT       | (# machine:=f,                    %|
                           actions:=none,                   %|
                           timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f,                    %|
                           actions:=none,                   %|
                           timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=ACKRCVD        | (#machine:=f WITH [state:= REQSENT]), %|
                            actions:= none, %|
                            timer_op:=none #)  ||
%-----------------------+---------------------------------------------||
|state(f)=OPENED          |
            (#machine:= fsm_sconfreq(f,FALSE)WITH), %|
                          actions:= tld o scr, %|
                          timer_op:=none #) ||
% HERE AN ERROR OCCURS BECAUSE THE NEW STATE HAS NOT BEEN UPDATED!
% WE DID NOT FIND f->state=REQSENT IN THE IMPLEMENTATION CODE!
%-----------------------+---------------------------------------------||

             ENDTABLE
```

COMMENT: On receiving RTA at state Opened, the automaton should bring down the link (tld), send a configuration request (scr) and go to state Req-Sent. As we can see the implementers made an error in the implementation. In the state(f)= OPENED they did not update the state after it brings down the link and sends the request. We should find code line f→state=REQSENT in the implementation code according to the state transition table (transition table of the automaton) written in the RFC document.

PVS INSPECTION FILE:

The following PVS code extracted from the pppInspection.pvs file states the proposition that the functions defined by the specification table and implementation table agree on all of their inputs.

```
pppInspection : THEORY

  BEGIN

  IMPORTING pppSpec, pppCode
  f:VAR fsm
  flag:VAR int


RTA:PROPOSITION
(lambda (f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}):
pppSpec.fsm_rtermack(f)) =
(LAMBDA (f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR
state(g)=OPENED}):pppCode.fsm_rtermack(f))
```

The display follows for actual code segment modeled by the implementation table above.

DISPLAY 11

Specification

void fsm_rtermack (fsm * f)

int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			CLOSED	STOPPED	CLOSING	STOPPED	REQSENT	REQSENT	ACKSENT	REQSENT
callbacks' =			none	none	finished	finished	none	none	none	addci
timer' =			none	none	stop	stop	none	none	none	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			' code	' code	' code	' code	' code	' code	' code	CONFREQ
reqid' =			' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	id
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	maxconfreqtransmits
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			' data	' data	' data	' data	' data	' data	' data	outp
datalen' =			' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	cilen

Program

```
/*
 * fsm_rtermack - Receive Terminate-Ack.
 */
static void
fsm_rtermack(f)
    fsm *f;
{
    switch (f->state) {
    case CLOSING:
        UNTIMEOUT(fsm_timeout, f);
        f->state = CLOSED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;
    case STOPPING:
        UNTIMEOUT(fsm_timeout, f);
        f->state = STOPPED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;

    case ACKRCVD:
        f->state = REQSENT;
        break;

    case OPENED:
        if (f->callbacks->down)
            (*f->callbacks->down)(f);  /* Inform upper layers */
        fsm_sconfreq(f, 0);
        break; /*HERE IS THE ERROR !
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)		u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen					
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

UNTIMEOUT (fsm_timeout, f)					
R1 (,) = (timer = stop)					

On Display 16

End of Display 11

7 WHICH MISTAKES AND INCONSISTENCIES DID WE FIND WHEN INSPECTING THE PPP CODE?

7.1. How can we classify the mistakes that we found?

Using the combination of the PVS Specification and Verification System and the Display method in our inspection of the Linux PPP daemon (pppd), we found a functional error in the implementation code as described in Section 6.4.2 of the previous chapter. It is further described in point 2 of Section 7.2. In the rest of Section 7.2, we also describe finding inconsistencies in coding style (described in points 1,4,7,8), significant issues regarding the readability of the code (described in points 3,4,8), and differences between the requirements and implementation code (described in points 5,6,9,10) such as the one described in Section 6.4.1. Thus the inspection method we applied enabled us to find more than simple inconsistencies between the requirements and implementation code.

Despite all the facts mentioned above, the code we inspected runs correctly and is much used code. In almost all circumstances, the functional error we found is not significant. The differences we found between requirements and implementation code do not mean that implementers made mistakes. They implemented some parts of the code in a different way than proposed in the requirements, by considering some additional cases and trying to write a more secure and reliable code.

Our inspection also showed that the readability of the code can be significantly improved. Some parts of the code need to be improved by rewriting them to be more consistent with the other parts.

7.2. What are the concrete examples of the mistakes we have found?

During our inspection process we discovered a few mistakes.

1) Variable id (current id), which is a part of an `fsm` structure, is declared inconsistently. Functions `fsm_rconfreq` (rows 7 and 8 of the table in Fig. 4 section 5.4) and `fsm_rconfack` (row 9 of the table in Fig. 4 section 5.4) have the same prototypes (fsm *, int, u_char *, int) but in the first case id is declared as u_char and in the second case it is declared as an integer!

Discussion: This inconsistency was noticed when we declared variables for Display 7 (`fsm_rconfreq`) and Display 8 (`fsm_rconfack`). This illustrates one of the most important points of the process.

We did not find this error with the PVS type of checking. It might have been missed because we cut and pasted the definition of `fsm_rconfreq` to create `fsm_rconfack`. This discrepancy reveals a really important point: that an inspection undertaken with a formal methods tool does not guarantee that nothing can be missed. There is a definite advantage in using the displays over just doing some formal methods stuff; it helps to check our formal model against reality!

Thus, one of the main advantages of this inspection process is that it can discover defects that the formal verification may miss (i.e. it helps to ensure that there is not a single point of failure that allows an error to go undetected). PVS should have found this error but did not, most likely due to an oversight on our part during the formal verification. The point is that on any real project, people and tools will make mistakes. It

is important to have a process that minimizes the potential impact of oversights and errors at any step along the way.

2) In the function `fsm_rtermack` (state(f)= `OPENED`) the implementers made a mistake. They did not update new state f→state=`REQSENT` in the implementation `fsm.c` code. This disagrees with the table in section 5.4 (see row: RTA, column: 9). The implementation does not update the state after it brings down the link and sends the request. In almost all circumstances, the bug is not significant. It can be observed only if the user tries to open the link immediately after the disconnection.

Note: f is the variable containing the data structure for the FSM (see Chapter 6).

Discussion: This mistake was noticed when we tried to apply PVS to inspect the `fsm.c` program (Linux PPP module). Comparing PVS tables `pppSpec.fsm_rtermack` and `pppCode.fsm_rtemack` in the `pppInspection` file it was impossible to prove that they are equal because of the mistake in `pppCode.fsm_termack`. Using the PVS theorem prover we were able to locate the error precisely and determine its nature. This bug was also identified by [Alur and Wang 2001].

This mistake was also discovered by using the Display method. When we prepared Display 11 it was obvious that the variable state' for the `OPENED` state was not set to state'= `REQSENT`.

It is very important to note that this double finding helps to reduce the risk of a single point of failure in the inspection method, thus preventing a mistake from going unnoticed.

3) In the function `fsm_rconfreq` (state(f)= `OPENED,RCR-case`) the implementers have written quite ambiguous implementation code. It is very difficult to

analyse this part of the code and follow it. More detailed explanations and comments would be extremely helpful for better understanding this section of code.

Discussion: This problem was noted by reading `fsm.c` code (Linux PPP module) and trying to understand what the implementers intended to write. The problem was noted while we created the tables in PVS.

4) In the functions `fsm_rconfreq` (state(f)= `STOPPED`) and `fsm_rconfreq` (state(f)= `OPENED`), the implementers included the `RCR+` case using an approach that is very different from and inconsistent with the approach used to implement the other events. Again, this different approach is very difficult to understand and the code could be written much more clearly.

Discussion: This problem was noticed when we tried to prepare PVS tables for the `fsm_rconfreq` function (`RCR` event). It was time consuming to prepare those tables (for the `RCR` event) following the same principles used for the other events. The main problem was badly written code for the `RCR` event.

5) In the function `fsm_rconfnakrej` two more cases that do not occur in the specification (state(f)= `REQSENT` & (ret<0)) and (state(f)= `ACKSENT` & (ret<0)) are included. The implementers did not explain the meaning of the variable ret in either `fsm.c` or in `fsm.h`. It is difficult to discover the purpose of this particular variable, even though understanding its purpose is essential for determining the difference between the specification and implementation in the two cases mentioned above.

Discussion: This was noticed by reading the `fsm.c` code (Linux PPP module) and trying to understand what the implementers wanted to accomplish by using the variable

ret. The PVS method and the Display method (Display 9) revealed the difference between specification and implementation code in the event fsm_rconfnakrej.

6) Certain aspects of the specification are not explicitly present in the implementation. For example, the automaton is able to send a couple of packets in any order if it is in the state STOPPED on event RCR+ or RCR-. Two variables are introduced in the requirements to record which packets have been sent. These variables do not appear in the C program but only in the specification model.

Discussion: This was noticed by reading the RFC specification and comparing it to the implementation. The implementer chose a particular order for the packets in the implementation and hence did not require the variables to keep track of what had been sent

7) Function FSM_DEBUG was not used consistently in the code. The implementers missed calling this function in the code for the following cases: (a)(function fsm_rtermreq (state: INITIAL, state: STARTING) and (b) function fsm_rtermack (state: INITIAL, state: STARTING)). Those cases are denoted in the RFC specification table as Illegal-Event (-). This indicates an event that cannot occur in a properly implemented automaton. If either of these events occur, the implementation has an internal error, which should be reported and logged. An illegal event is denoted in the fsm.c program (Linux PPP module) by invoking the FSM_DEBUG function but the implementers did not call it in the above two cases, making a mistake.

Discussion: This mistake was discovered using the Display method. When we prepared Display 10 and Display 11 it was obvious that the variable protoname' for the INITIAL state and STARTING state was not set to protoname'= "MISTAKE".

8) Implementers have written inconsistent and unreadable code in cases when the fsm stays in the same state. They denoted those cases clearly only two (2) times in the function fsm_rconfreq (state: CLOSING, state: STOPPING), but missed doing it the same way in eighteen (18) other cases. Those cases are: function fsm_open (state: STARTING, state:STOPPED, state: STOPPING, state: REQSENT, state: ACK-RCVD, state: ACK-SENT, state: OPENED) , function fsm_close (state: INITIAL, state:CLOSED, state: CLOSING), function fsm_rconfack (state: CLOSING, state: STOPPING), function fsm_rconfnakrej (state: CLOSING, state: STOPPING), function fsm_rtermack (state: CLOSED, state: STOPPED, state: REQSENT, state: ACKSENT).

Discussion: This problem was noted by reading fsm.c code (Linux PPP module) and trying to understand what the implementers intended to write. The error was also noticed by reading the RFC specification and comparing it to the implementation.

9) Comparing the specification and implementation for function fsm_lowerup(corresponds to Row 1 (Up) of Fig 4 – table in section 5.4) we noted a difference between them. Implementers added one more case in the function fsm_lowerup in the possible state(f) = STARTING (go to state = STOPPED).

Discussion: This difference was noticed when we tried to apply PVS to inspect fsm.c program (Linux PPP module). Comparing the PVS tables for pppSpec.fsm_lowerup and pppCode.fsm_lowerup in the pppInspection file it was

impossible to prove that they are equal because of the difference between specification and implementation code. Using the PVS theorem prover we were able to exactly "locate" the difference and its nature.

This difference between the RFC specification and the implementation code was also revealed by using the Display method. When we prepared Display 2, it was obvious that the variable state' for the `STARTING` state was not only set to state'= `REQSENT` but also to state'= `STOPPED` in an additional case.

10) Comparing the RFC specification and the implementation (function `fsm_open`) we noted a difference between them. Implementers added one more case in the function `fsm_open` in the possible state(f) = `CLOSED` (go to state(f)=`STOPPED`).

Discussion: This difference was noticed when we tried to apply PVS to inspect `fsm.c` program (Linux PPP module). Comparing PVS tables in the `pppSpec.fsm_open` and the `pppCode.fsm_open` in the `pppInspection` file it was impossible to prove that they are equal because of the difference between specification and implementation code. Again, using the PVS theorem prover, we were able to exactly "locate" the difference and its nature.

This difference between the RFC specification and implementation code was also discovered using the Display method. When we prepared Display 4 it was obvious that the variable state' for the `CLOSED` state was not only set to state'= `REQSENT` but also to state'= `STOPPED` in additional case. This disagrees with the table in section 5.4 (see row: Open, column: 2).

8 CONCLUSION

8.1 What can we conclude about the method we have proposed?

The software inspection method that we used has demonstrated its efficiency by enabling us to discover several mistakes and inconsistencies in the state machine module (fsm.c in the implementation code) of a widely used, open source PPP implementation. We used a method based on a mathematical model of programs and used mathematical notation to provide precise descriptions of programs.

The main contributions of the software inspection method we applied are:

1) Using Formal Verification to construct system design documentation of the PPP implementation to enable application of the display inspection technique. In the process, inspection found not only functional errors, but also coding style and readability issues that were ignored by formal verification in PVS.

2) Verifying that the program function table for each event implements PVS requirements table we created for each row.

3) Enabling us to discover several mistakes and inconsistencies in the PPP implementation code.

4) Using our formal software inspection technique to demonstrate that it is feasible to perform the display inspection technique in the absence of detailed design documents for a non-trivial application. The theorem prover helped to make the process possible by speeding up the construction of the required documentation and helping to perform the inspection.

5) Providing an example of how mathematical methods can be used as an integral part of a computer-aided software inspection process. We combined computer aided verification with software inspection to compensate for a lack of sufficiently detailed documentation.

6) Automaton of steps in the inspection process of [Parnas 1994] using theorem proving/verification techniques

7) Providing a detailed example of use of the display method a non-trivial “ real world ” application

PVS mainly helped us to demonstrate that the specification and implementation agree (or disagree) on all possible input sequences (guaranteed domain coverage). PVS also provided almost complete automation of consistency checks of the tabular definition for both the system specification and implementation. Failed proofs generated during the verification process helped us to identify counterexamples and provided insight into why a verification proof fails. PVS allowed us to create the documentation we needed for the inspection and increased our confidence that the documentation was correct.

The Display method allowed us to discover different types of mistakes. It mainly helped us to check if each mode in the specification corresponds to a set of states in the implementation i.e. if all the variables that store state information within the protocol have proper values. It also enabled us to see if the code is written consistently i.e. if the values of each variable were changed properly throughout the code. It also helped to confirm the mistakes PVS discovered – making it more unlikely for a single point of failure to allow an error to go undetected.

The Display inspection checks not only the code, but also the table created in PVS for the inspection. This step is usually lacking in most applications of formal methods. It is also not extra work in this case because we are already doing the displays inspection.

Finally, overall our work reinforces the importance of software inspection and its role in software engineering.

8.2 Limitations and future work

The success of the PPP inspection was in part due to having a good specification in the form of the RFC State Transition Table. The fact that there were no detailed numerical computations or heavy use of pointers also helped to make things easier.

Applying the method and modifying it accordingly to deal with these types of code is one type of possible future work. Future work should include developing an improved Display Management System (DMS) tool that could then be employed in order to make our inspection method more efficient. Perhaps more importantly, tool support to automate transcription of the PVS tables into the spreadsheet format used for the displays would eliminate a significant potential source of error.

A further limitation to what can be concluded from the work comes from the fact that the requirements were easily partitioned and mapped to various parts of the PPP code. The inspection process used does not explicitly handle requirements that are spread throughout the code (e.g. "the system must be secure", timing performance, etc.).

BIBLIOGRAPHY

Alur, R., and Wang, B., Verifying Network protocol Implementations by Symbolic Refinement Checking, Computer Aided Verification 2001, LNCS 2102, pp.169-181

Andrews, P. B., Bishop, M., Issar, S., Nesmith, D., and Pfenning, F., "TPS: An Interactive and Automatic Tool for Proving Theorems of Type Theory", HUG 1993, HOL User's Group Workshop, August 1993, Vancouver, B.C., pp.519-522

Basili, V., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., and Zelkowitz, M., "The Empirical Investigation of Perspective-Based Reading", Empirical Software Engineering: An International Journal, 1(2): 133-164, 1996.

Bisant, D., and Lyle, J., "A Two Person Inspection Method to Improve Programming Productivity", IEEE Transactions on Software Engineering, 15 (10): 1294-1304, 1989.

Burr, A., and Owen, M., "Statistical Methods for Software Quality",International Thomson Computer Press, UK, 1996.

Ebenau, R., and Strauss, S., "Software Inspection Process", Mc Graw – Hill, Inc., 1993.

Eick, S.G., Loader, C.R., Long, M.D., Votta, L.G., and Wiel, S., "Estimating Software Fault Content before Coding", Proceedings of the 14th International Conference on Software Engineering, pages 49-65, 1992.

Fagan, M., "Design and Code Inspections to Reduce Errors in Program Development", IBM System Journal, 15 (3): 182-211, 1976.

Farmer, W. M., Guttman, D., and Thayer, F. J., "An Interactive Mathematical Proof System", J. of Automated Reasoning, 11:213-248,1993.

Gilb, T., and Graham, D., "Software Inspections", Addison-Wesley, 1993.

Gordon, M. J. C., and Melham, T.F., “Introduction to HOL”, Cambridge University Press, 1993.

Jing, M., “A Table Checking Tool”, Mc Master University, Hamilton, Ontario, SERG Report No 384, March 2000.

Jankowski, E., and McDougall, J., "Procedure for the Specification of Software Requirements for Safety Critical Software", CANDU Computer sytems Engineering Centre of Excellence Procedure CE-1001-PROC Rev.1", Jul 1995

Kalvala, S., “Using Isabelle to prove simple theorems”, HUG 1993, HOL User’s Group Workshop, August 1993, Vancouver, B.C., pp.519-522

Knight, J.C., and Myers, A.E., “An Improved inspection Technique”, Communications of ACM, 36(11): 50-69, 1993.

Lawford, M., Froebel, P., and Moum, G., Application of Tabular Methods to the Specification and Verification of a Nuclear Reactor Shutdown System, Submitted to Formal Methods in System Design,2000.

Laitenberger, O., and Kohler, K., “ The systematic Adaptation of Perspective-based Inspections to Software Development Projects”, Proceedings of the 1st Workshop on Inspection in Software Engineering, SQRL, Mc Master University, July 2001, pp.105-114

Martin, J., and Tsai, W., “N-Fold Inspection: A Requirements Analysis Technique”, Communications of ACM, 33 (2): 225-232, 1990.

McDougall, J., and Lee, J., "Procedure for the Software DesignDescription for Safety Critical Software”, CANDU Computer System Engineering Centre of Excellence Procedure CE-1002-PROC Rev.1", Oct 1995.

Owre, S., Rushby, J. and Shankar, N., ”Integration in PVS: Tables, types,and model checking,” in Tools and Algorithms for the Construction and Analysis of Systems (TACAS '97) (E.Brinksma, ed.), vol.1217 of Lecture Notes in Computer Science, (Enschede, The Netherlands), pp.366-383, Springer-Verlag, Apr. 1997.

Owre, S., Rushby, J., Shankar, N. and von Henke, F., Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS," IEEE Transactions on Software Engineering, vol.21, pp.107-125, Feb. 1995.

Parnas, D.L., “Using Mathematical Models in the Inspection of Critical Software”, Application of Formal Methods, 1995

Parnas, D.L., “Inspection of Safety Critical Software using Function Tables”, Proceedings of IFIP World Congress 1994, Volume III, August 1994,pp. 270 - 277.

Parnas, D.L., Software Aging, Proceedings of the 16th InternationalConference on Software Engineering, 1994., IEEE Press, pages 279-287

Parnas, D.L., “The use of precise specifications in the development of software”, Proceedings of the IFIP Congress, 1977., pp.861-867

Parnas, D.L., Asmis, and G.J.K., Madey, J., “Assessment of Safety-Critical Software in Nuclear Power Plants”, Nuclear Safety, vol. 32, no. 2, April-June 1991, pp. 189-198.

Parnas, D.L., Clements, P.C., and Weiss, D.M., The Modular Structure of Complex Systems, IEEE Transactions on Software engineering, Vol SE-11, 1985., pp.259-266

Parnas, D.L., and Madey J., “Functional Documentation of Computer Systems”, Science of Computer Programming (25) 1995. p 41-61

Parnas, D.L., and Madey J., “Functional Documents for Computer Systems Engineering”, CRL Report 237, Mc Master University, CRL, TRIO, Hamilton, Ontario, Canada, September 1991, 14 pp

Parnas, D. L., Madey, J., and Iglewski, M., Precise Documentation of Well-Structured Programs, IEEE Transactions on Software engineering, Vol.20, No 12, Dec 1994., pp.948-976

Parnas, D.L., and Weiss, D., “Active Design Reviews: Principles and Practices”, Proceedings of the 8th International Conference on Software Engineering, 1985, pages 132-136

Paulson, L., “Introduction to Isabelle”, Technical Report 280, 1993, University of Cambridge, Computer laboratory

Porter, A.A., Votta, L.G., and Basili, V.R., “Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment”, IEEE Transactions on Software engineering, 21(6): 563-575, 1995.

Shankar, N., Owre, S., Rushby, J. M., PVS Tutorial, Computer Science Laboratory, SRI International, Menlo Park, CA, Feb 1993.

Shull, F., Ioana, R., Basili, V.R., “How Perspective-Based Reading Can Improve Requirements Inspections”, IEEE Computer, 33 (7): 73-79, 2000.

Simpson, W., The Point-to-Point protocol, Computer Systems Consulting Services, July 1994. STD 51, RFC 1661

Wang, Y., “ Display Management System”, Faculty of Engineering, Mc Master University, Hamilton, Ontario, CRL Report No 297, April 1995

Weller, E.F., “Lessons from Three Years of Inspection Data”, IEEE Software, 10(5): 38-45, 1993.

APPENDIX A: PVS files for the Inspection of the PPP FSM code

1.
$$$pppCommon.pvs

```
pppCommon : THEORY

  BEGIN

    Mode:TYPE = {Initial, Starting, Closed, Stopped, Closing, Stopping,
                Req_Sent, Ack_Rcvd, Ack_Sent, Opened}

%  Action: TYPE = {tlu,tld,tls,tlf,irc,zrc,scr,sca,scn,str,sta,none};

%  ActionSeq:TYPE =finseq[Action]
%  none:empty_seq

Action: TYPE+
ActionSeq: TYPE = [Action->Action]
tlu,tld,tls,tlf,irc,zrc,scr,sca,scn,str,sta,none:ActionSeq

  u_char:TYPE = char
  void: TYPE
%  callback: TYPE = [fsm -> void]


TimerOp: TYPE = {start, stop, none}

  fsm:TYPE =
        [# state: subrange(0,9), % want to make this subrange for
typechecking

               flags: int,
               id: u_char,
               reqid: u_char,
               seenack: u_char,
               timeouttime: int,
               maxconfreqtransmits: int,
               retransmits: int,
               maxtermtransmits: int,
               nakloops: int,
               maxnakloops: int,
               term_reason: string, % This is really a pointer to char
               term_reason_len: int,
               callbacks: fsm_callbacks %This is really a pointer to
fsm_callbacks
        #]
```

```
% Programs that implement the PPP protocol are required to do 3 things:
%       1. update the state of the protocol FSM
%       2. produce a sequence of output actions
%       3. possibly start or stop the timer
% We represent this with the program requirement type ProgReq

ProgReq: TYPE =
        [#      machine: fsm,
                actions: ActionSeq,
                timer_op: TimerOp
        #]

% integer constants declared in header file for states

  INITIAL: int =       0
  STARTING: int =      1
  CLOSED: int =        2
  STOPPED: int =       3
  CLOSING: int =       4
  STOPPING: int =      5
  REQSENT: int =       6
  ACKRCVD: int =       7
  ACKSENT: int =       8
  OPENED:  int =       9

  f:VAR fsm

  fsm2Mode(f):Mode =
    COND
        state(f)=INITIAL       -> Initial,
        state(f)=STARTING      -> Starting,
        state(f)=CLOSED        -> Closed,
        state(f)=STOPPED       -> Stopped,
        state(f)=CLOSING       -> Closing,
        state(f)=STOPPING      -> Stopping,
        state(f)=REQSENT       -> Req_Sent,
        state(f)=ACKRCVD       -> Ack_Rcvd,
        state(f)=ACKSENT       -> Ack_Sent,
        state(f)=OPENED        -> Opened

    ENDCOND

% This is a function currently at the boundary of the inspection.
% We assume that it is equivalent in both the spec and code.
 fsm_sconfreq(f:fsm, retransmit:bool):fsm = f


END pppCommon

$$$pppCommon.prf (PROOFS)
(|pppCommon| (|fsm2Mode_TCC1| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm2Mode_TCC2| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_sconfreq_TCC1| "" (INST 1 "lambda (f:fsm, b:bool):f") NIL NIL))
```

2.
$$$pppSpec.pvs

```
pppSpec  % [parameters]
                : THEORY

  BEGIN
  IMPORTING pppCommon



%Define flag integer constants
%OPT_RESTART:posint =  2
%OPT_SILENT:posint =   4
%OPT_PASSIVE:posint =  1



%  fsm2Mode_alt(mod):Mode = state(mode)
%IsSet(flags:int,flag:int):bool



% CP Codes
CONFREQ:posint = 1
CONFACK:posint = 2
CONFREJ:posint = 4
```

```
fsm_lowerup(f:{g:fsm|state(g)=INITIAL OR state(g)=STARTING}):ProgReq
=
       TABLE
%-----------------------+---------------------------------------------||
|state(f)=INITIAL       |(# machine:= f WITH [state:= CLOSED], %|
                            actions:=none, %|
                            timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=STARTING      |(# machine:= fsm_sconfreq(f,FALSE)
                                        WITH [state:= REQSENT], %|
                                        actions:= irc o scr, %|
                                         timer_op:=start #) ||
%-----------------------+---------------------------------------------||

        ENDTABLE


    fsm_lowerdown (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}):ProgReq =
         TABLE
%-----------------------+---------------------------------------------||
|state(f)=CLOSED        | (# machine:=f WITH [state:=INITIAL], %|
                              actions:= none, %|
                              timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=STOPPED       | (# machine:=f WITH [state:= STARTING], %|
                             actions:=tls, %|
                             timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=CLOSING       | (# machine:=f WITH [state:=INITIAL], %|
                              actions:=none, %|
                              timer_op:= stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=STOPPING      | (# machine:=f WITH [state:=STARTING], %|
                              actions:=none, %|
                              timer_op:= stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=REQSENT       | (# machine:=f WITH [state:=STARTING], %|
                              actions:=none, %|
                              timer_op:= stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=ACKRCVD       | (# machine:=f WITH [state:=STARTING], %|
                              actions:=none, %|
                              timer_op:= stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f WITH [state:=STARTING], %|
                              actions:=none, %|
                              timer_op:= stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=OPENED        | (# machine:=f WITH [state:= STARTING], %|
                              actions:=tld, %|
                              timer_op:=none #)  ||
%-----------------------+---------------------------------------------||


ENDTABLE
```

```
      fsm_open(f:fsm):ProgReq =
       TABLE
%-----------------------+-------------------------------------------||
|state(f)=INITIAL       | (# machine:=f WITH [state:= STARTING],     %|
                            actions:=tls,                            %|
                           timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=STARTING      | (# machine:=f,                %|
                            actions:=none,              %|
                           timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=CLOSED        |(#machine:=fsm_sconfreq(f,FALSE)
                                      WITH [state:=REQSENT], %|
                                      actions:=irc o scr, %|
                                      timer_op:=start #) ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPED       |(# machine:= f, %|
                           actions:=none, %|
                           timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=CLOSING        | (# machine:= f WITH [state:=STOPPING], %|
                                         actions:=none, %|
                                         timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPING      | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=REQSENT       | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=ACKRCVD       | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=OPENED               |(# machine:= f, %|
                                   actions:=none, %|
                                   timer_op:=none #) ||
%-----------------------+-------------------------------------------||

        ENDTABLE
```

```
fsm_close (g:{g:fsm|state(g)=INITIAL OR state(g)=STARTING OR
state(g)=CLOSED
OR state(g)=STOPPED OR state(g)=CLOSING OR state(g)=STOPPING OR
state(g)=REQSENT
 OR state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED},
reason:string):ProgReq =

LET f = g WITH [ term_reason:=reason,
                term_reason_len:=length(reason)] IN

TABLE
%----------------------+--------------------------------------------||
|state(f)=INITIAL      | (# machine:=f,                  %|
                            actions:=none,               %|
                            timer_op:=none #) ||
%----------------------+--------------------------------------------||
|state(f)=STARTING     |(#     machine:= f WITH [state:= INITIAL],   %|
                              actions:= tlf,                         %|
                              timer_op:=none #)                      ||
%----------------------+--------------------------------------------||
|state(f)=CLOSED       | (# machine:=f,                  %|
                            actions:=none,               %|
                            timer_op:=none #) ||
%----------------------+--------------------------------------------||
|state(f)=STOPPED      |(#     machine:=( f WITH [state:= CLOSED]),%|
                              actions:=none, %|
                              timer_op:=none #) ||
%----------------------+--------------------------------------------||
|state(f)=CLOSING      | (# machine:=f,                  %|
                            actions:=none,               %|
                            timer_op:=none #) ||

%----------------------+--------------------------------------------||
|state(f)=STOPPING     |(#     machine:=(f WITH [state:= CLOSING]),%|
                              actions:=none,                       %|
                              timer_op:=none #)                    ||
%----------------------+--------------------------------------------||
|state(f)=REQSENT OR                                    %|
  state(f)=ACKRCVD                                      %|
 OR state(f)=ACKSENT   |(#     machine:=f WITH [state:= CLOSING,%|
                               actions:=irc o str,    %|
                              timer_op:=start #)      ||
%----------------------+--------------------------------------------||
|state(f)=OPENED       |(# machine:=f WITH [state:= CLOSING,%|
                           actions:= tld o irc o str, %|
                           timer_op:=start #)            ||
%----------------------+--------------------------------------------||
ENDTABLE

% NOTE: In the PPP code, the function fsm_sdata is a general function
%that
% is used to send data, in particular the various requests and
% acknowlegements that are required by the protocol actions, e.g:
```

```
% fsm_sdata(f, TERMREQ, f->reqid = ++ f->id,
%         (u_char *) f->term_reason, f->term_reason_len);

% implements the requirements action:

%       str = Send-Terminate-Request

% Similarly action scr,sca,scn,sta are implemented by fsm_sdata.




fsm_timeout_minus(f:{g:fsm|retransmits(g)<=0 & (state(g)=CLOSING OR
state(g)=STOPPING OR
state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT)}):ProgReq =
               TABLE
%------------------------+--------------------------------------------||
| state(f)= CLOSING    | (# machine:=f WITH [state:= CLOSED], %|
                             actions:=tlf, %|
                             timer_op:=none #) ||%THIS IS TO- CASE
%------------------------+--------------------------------------------||
| state(f)= STOPPING    | (# machine:=f WITH [state:= STOPPED], %|
                             actions:=tlf, %|
                              timer_op:=none #) || % THIS IS TO- CASE
%------------------------+--------------------------------------------||

|state(f)=REQSENT       | (# machine:= f WITH [state:=
                                             STOPPED], %|
                                             actions:=tlf, %|
                                timer_op:=none #) || % THIS IS TO- CASE
%------------------------+--------------------------------------------||
|state(f)=ACKRCVD       |(# machine:=f WITH [state:=
                                            STOPPED], %|
                                            actions:=tlf, %|
                                 timer_op:=none #) || %THIS IS TO- CASE
%------------------------+--------------------------------------------||
|state(f)=ACKSENT       |  (# machine:=f WITH [state:=
                                              STOPPED], %|
                               actions:=tlf, %|
                               timer_op:=none #) || % THIS IS TO- CASE
%------------------------+--------------------------------------------||
                 ENDTABLE
```

```
fsm_timeout_plus(f:{g:fsm|retransmits(g)>0 & (state(g)=CLOSING OR
state(g)=STOPPING OR
state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT)}):ProgReq =
               TABLE

%-----------------------+--------------------------------------------||
| state(f)= CLOSING    |(# machine:=f WITH [state:= CLOSING], %|
                             actions:=str, %|
                            timer_op:=start #) || % THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
| state(f)= STOPPING    |(# machine:=f WITH [state:= STOPPING], %|
                            actions:=str, %|
                            timer_op:=start #)||  % THIS IS TO+ CASE

%-----------------------+--------------------------------------------||
|state(f)=REQSENT       |  (# machine:=
                          fsm_sconfreq(f,TRUE)WITH[state:= REQSENT], %|
                             actions:=scr, %|
                             timer_op:=none #)||  % THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD       |  (#
                 machine:=fsm_sconfreq(f,TRUE)WITH[state:= REQSENT], %|
                         actions:=scr, %|
                         timer_op:=none #) || %    THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
|state(f)=ACKSENT       |(# machine:=fsm_sconfreq(f,TRUE)WITH[state:=
ACKSENT], %|
                              actions:=scr, %|
                              timer_op:=start #)|| %THIS IS TO+ CASE
%-----------------------+--------------------------------------------||

                 ENDTABLE
```

```
fsm_rconfreq_minus(f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR
state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR
state(g)=OPENED}):ProgReq  =

               TABLE
%----------------------+---------------------------------------------||
|state(f)=CLOSED       | (# machine:= f WITH [state:=CLOSED], %|
                            actions:=sta, %|
                            timer_op:=none #)       || % THIS IS RCR-
CASE
%----------------------+---------------------------------------------||
|state(f)=STOPPED    |(# machine:=
fsm_sconfreq(f,FALSE)WITH[state:=REQSENT],%|
                                 actions:= irc o scr o scn, %|
                                 timer_op:=start #) ||
% THIS IS RCR- CASE!
%----------------------+---------------------------------------------||
|state(f)=CLOSING      | (# machine:= f WITH [state:=CLOSING], %|
                            actions:=none, %|
                            timer_op:=none #)    || % THIS IS RCR-
CASE
%----------------------+---------------------------------------------||
|state(f)=STOPPING     | (# machine:= f WITH [state:=STOPPING], %|
                            actions:=none, %|
                            timer_op:=none #)    || % THIS IS RCR-
CASE
%----------------------+---------------------------------------------||
|state(f)=REQSENT    |(# machine:=f WITH[state:=REQSENT],%|
                                 actions:= scn, %|
                                 timer_op:=none #) ||
% THIS IS RCR- CASE!
%----------------------+---------------------------------------------||
|state(f)=ACKRCVD  |(# machine:= f WITH [state:= ACKRCVD],%|
                                      actions:= scn, %|
                                      timer_op:=stop #)   ||
% THIS IS RCR- CASE!
%----------------------+---------------------------------------------||

|state(f)=ACKSENT  |(# machine:= f WITH [state:= REQSENT],%|
                                      actions:= scn, %|
                                      timer_op:=none #)   ||
% THIS IS RCR- CASE!
%----------------------+---------------------------------------------||
 |state(f)=OPENED       | (#machine:=fsm_sconfreq(f,FALSE)
WITH[state:=REQSENT], %|
                                 actions:= tld o scr o scn, %|
                                 timer_op:=start #) ||
% THIS IS RCR- CASE!
%----------------------+---------------------------------------------||
                  ENDTABLE
```

```
fsm_rconfreq_plus(f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR
state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR
state(g)=OPENED}):ProgReq  =


               TABLE
%---------------------+--------------------------------------------||
|state(f)=CLOSED       | (# machine:= f WITH [state:=CLOSED], %|
                            actions:=sta, %|
                            timer_op:=none #) || % THIS IS RCR+ CASE
%---------------------+--------------------------------------------||

|state(f)=CLOSING      | (# machine:= f WITH [state:=CLOSING], %|
                            actions:=none, %|
                            timer_op:=none #) || % THIS IS RCR+ CASE
%---------------------+--------------------------------------------||
|state(f)=STOPPING     | (# machine:= f WITH [state:=STOPPING], %|
                            actions:=none, %|
                            timer_op:=none #) || % THIS IS RCR+ CASE
%---------------------+--------------------------------------------||
|state(f)=STOPPED    |(# machine:=f WITH[state:=ACKSENT],%|
                           actions:= irc o scr o sca, %|
                           timer_op:=start #) ||% THIS IS RCR+ CASE
%---------------------+--------------------------------------------||
|state(f)=REQSENT    |(# machine:=f WITH[state:=ACKSENT], %|
                           actions:= sca, %|
                           timer_op:=none #) ||% THIS IS RCR+ CASE!
%---------------------+--------------------------------------------||
|state(f)=ACKRCVD        | (# machine:= f WITH [state:=OPENED],%|
                              actions:= sca o tlu, %|
                              timer_op:=stop #) ||%THIS IS RCR+CASE!
%---------------------+--------------------------------------------||
|state(f)=ACKSENT  |(# machine:= f WITH [state:= ACKSENT],%|
                          actions:= sca, %|
                          timer_op:=none #)   ||% THIS IS RCR+ CASE
%---------------------+--------------------------------------------||
|state(f)=OPENED        | (#machine:=f WITH[state:=ACKSENT], %|
                             actions:= tld o scr o sca, %|
                             timer_op:=start #) ||% THIS IS RCR+ CASE
%---------------------+--------------------------------------------||

ENDTABLE
```

```
fsm_rconfack(f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR state(g)=
CLOSING
OR state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT
OR state(g)=OPENED}):ProgReq  =

               TABLE
%-----------------------+--------------------------------------------||
|state(f)=CLOSED       |(# machine:= f WITH [state:=CLOSED], %|
                            actions:= sta, %|
                            timer_op:=none #)   ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPED       |(# machine:= f WITH [state:=STOPPED], %|
                            actions:=sta, %|
                            timer_op:=none #)   ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSING      | (# machine:=f,                %|
                            actions:=none,             %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPING     | (# machine:=f,                %|
                            actions:=none,             %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=REQSENT      | (# machine:=f WITH [state:=ACKRCVD], %|
                             actions:= irc, %|
                             timer_op:=none #)   ||
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD      |(# machine:=
                         fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                            actions:=scr, %|
                            timer_op:=start #) ||
%-----------------------+--------------------------------------------||
|state(f)= ACKSENT      |(# machine:= f WITH [state:=OPENED],
                             actions:=irc o tlu, %|
                             timer_op:=stop #)    ||
%-----------------------+--------------------------------------------||
|state(f)=OPENED        |(# machine:=
                         fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                            actions:=tld o scr, %|
                            timer_op:=start #) ||
%-----------------------+--------------------------------------------||

               ENDTABLE
```

```
fsm_rconfnackrej(f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKSENT
OR state(g)=ACKRCVD OR state(g)=OPENED}):ProgReq =


                TABLE
%------------------------+--------------------------------------------||
|state(f)=CLOSED        |(# machine:= f WITH [state:=CLOSED], %|
                             actions:=sta, %|
                             timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=STOPPED        |(# machine:= f WITH [state:=STOPPED], %|
                             actions:=sta, %|
                             timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=CLOSING       | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=STOPPING      | (# machine:=f,                %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%------------------------+--------------------------------------------||
| state(f)=REQSENT
            |(#machine:=fsm_sconfreq(f,FALSE) WITH [state:=REQSENT], %|
                              actions:= irc o scr, %|
                              timer_op:=start #) ||
%------------------------+--------------------------------------------||
|state(f)=ACKRCVD
       |(#machine:=fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|

                                actions:= scr, %|
                                timer_op:=start #) ||
%------------------------+--------------------------------------------||
| state(f)=ACKSENT       |(#machine:=fsm_sconfreq(f,FALSE)
                                    WITH [state:=ACKSENT], %|
                                    actions:= irc o scr, %|
                                    timer_op:=start #) ||
%------------------------+--------------------------------------------||
|state(f)=OPENED
            |(#machine:= fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                           actions:= tld o scr, %|
                           timer_op:=start #) ||
%------------------------+--------------------------------------------||
              ENDTABLE
```

```
fsm_rtermreq (f:{g:fsm|state(g)=CLOSED OR state(g)= STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD
OR state(g)=ACKSENT OR state(g)=OPENED}):ProgReq =

               TABLE
%------------------------+--------------------------------------------||
|state(f)=CLOSED      |(#machine:= f WITH [state:=CLOSED], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=STOPPED      |(#machine:= f WITH [state:=STOPPED], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=CLOSING      |(#machine:= f WITH [state:=CLOSING], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=STOPPING      |(#machine:= f WITH [state:=STOPPING], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=REQSENT      |(#machine:= f WITH [state:=REQSENT], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)=ACKRCVD      |(#machine:= f WITH [state:=REQSENT], %|
                           actions:= sta, %|
                           timer_op:=none #) ||
%------------------------+--------------------------------------------||
|state(f)= ACKSENT     | (#machine:= f WITH [state:=REQSENT], %|
                            actions:= sta, %|
                            timer_op:=none #)  ||
%------------------------+--------------------------------------------||
|state(f)=OPENED         | (#machine:=f WITH [state:=STOPPING], %|
                                        actions:= tld o zrc o sta, %|
                                        timer_op:=start #) ||
%------------------------+--------------------------------------------||


               ENDTABLE
```

```
fsm_rtermack(f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}): ProgReq =
       TABLE
%-----------------------+---------------------------------------------||
|state(f)=CLOSED        | (# machine:=f,                   %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=STOPPED       | (# machine:=f,                   %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=CLOSING       | (#machine:= f WITH  [state:= CLOSED], %|
                                actions:= tlf, %|
                                timer_op:=stop #) ||
%-----------------------+---------------------------------------------||
|state(f)=STOPPING      | (#machine:=f WITH [state:= STOPPED], %|
                            actions:= tlf, %|
                            timer_op:=stop #)||
%-----------------------+---------------------------------------------||
|state(f)=REQSENT       | (# machine:=f,                   %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+---------------------------------------------||

|state(f)=ACKRCVD        | (#machine:=f WITH [state:= REQSENT], %|
                             actions:= none, %|
                             timer_op:=none #)   ||
%-----------------------+---------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f,                   %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+---------------------------------------------||
|state(f)=OPENED         |
            (#machine:= fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                           actions:= tld o scr, %|
                           timer_op:=start #) ||
%-----------------------+---------------------------------------------||

             ENDTABLE
```

```
fsm_protreject(f:{g:fsm|state(g)=CLOSED OR state(g)=CLOSING OR
state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR state(g)=STOPPED OR state(g)=OPENED}):ProgReq =

         TABLE

%-----------------------+--------------------------------------------||
|state(f)=CLOSED        |(#machine:=f WITH [state:= CLOSED], %|
                            actions:= tlf, %|
                            timer_op:=none #)    ||

%-----------------------+--------------------------------------------||

|state(f)=CLOSING       |(#machine:=f WITH[state:=CLOSED], %|
                            actions:= tlf, %|
                            timer_op:=stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPING      | (#machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=stop #) ||

%-----------------------+--------------------------------------------||
|state(f)=REQSENT       |  (#machine:=f WITH [state:= STOPPED], %|
                              actions:= tlf, %|
                              timer_op:=stop #) ||

%-----------------------+--------------------------------------------||

|state(f)=ACKRCVD       |  (#machine:=f WITH [state:= STOPPED], %|
                              actions:= tlf, %|
                              timer_op:=stop #) ||

%-----------------------+--------------------------------------------||

|state(f)=ACKSENT       | (#machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=stop #) ||
%-----------------------+--------------------------------------------||

|state(f)=STOPPED       | (#machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=none #)||

%-----------------------+--------------------------------------------||
|state(f)=OPENED         |   (#machine:=f WITH [state:= STOPPING], %|
                              actions:= tld o irc o str, %|
                              timer_op:=start #) ||
%-----------------------+--------------------------------------------||
              ENDTABLE

  END pppSpec
```

```
$$$pppSpec.prf (PROOFS)
(|pppSpec| (|fsm_lowerup_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerup_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerup_TCC3| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_lowerup_TCC4| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC6| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC7| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC8| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC9| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_lowerdown_TCC10| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_open_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC4| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_open_TCC5| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_close_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_timeout_minus_TCC7| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_timeout_plus_TCC7| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC1| "" (SKOLEM!) ((""  (GRIND) NIL NIL)) NIL)
 (|fsm_rconfreq_minus_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC6| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC7| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC8| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_minus_TCC9| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_rconfreq_plus_TCC7| "" (COND-COVERAGE-TCC))
 (|fsm_rconfack_TCC1| "" (SUBTYPE-TCC) NIL NIL)
```

```
(|fsm_rconfack_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC7| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC8| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rtermreq_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermreq_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC3| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_protreject_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC7| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC8| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC9| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_protreject_TCC10| "" (COND-COVERAGE-TCC) NIL NIL))
```

3.
$$$pppCode.pvs

```
pppCode : THEORY

  BEGIN

  IMPORTING pppCommon


%Define flag integer constants
OPT_RESTART:posint =   2
OPT_SILENT:posint =    4
OPT_PASSIVE:posint =   1



%  fsm2Mode_alt(mod):Mode = state(mode)
IsSet(flags:int,flag:int):bool % Need to define this



% CP Codes
CONFREQ:posint = 1
CONFACK:posint = 2
CONFREJ:posint = 4






  fsm_lowerup(f:{g:fsm|state(g)=INITIAL OR state(g)=STARTING}):ProgReq
=
       TABLE
%-----------------------+---------------------------------------||
|state(f)=INITIAL      |(# machine:= f WITH [ state:= CLOSED], %|
                          actions:=none,                    %|
                          timer_op:=none #)                 ||
%-----------------------+-------------------------------------------||
|state(f)=STARTING
  & IsSet(flags(f),OPT_SILENT)|(# machine:= f WITH [state:= STOPPED],%|
                                 actions:=none, %|
                                 timer_op:=none #)||
%-----------------------+-------------------------------------------||
|state(f)=STARTING
  & NOT IsSet(flags(f),OPT_SILENT) |(# machine:= fsm_sconfreq(f,FALSE)
                                      WITH [state:= REQSENT], %|
                                      actions:= irc o scr, %|
                                       timer_op:=start #) ||
%-----------------------+-------------------------------------------||

        ENDTABLE
```

```
   fsm_lowerdown (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}):ProgReq =
         TABLE
%-----------------------+--------------------------------------------||
|state(f)=CLOSED        | (# machine:=f WITH [state:=INITIAL], %|
                             actions:= none, %|
                             timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPED       | (# machine:=f WITH [state:= STARTING], %|
                            actions:=tls, %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSING       | (# machine:=f WITH [state:=INITIAL], %|
                             actions:=none, %|
                             timer_op:= stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPING      | (# machine:=f WITH [state:=STARTING], %|
                             actions:=none, %|
                             timer_op:= stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=REQSENT       | (# machine:=f WITH [state:=STARTING], %|
                             actions:=none, %|
                             timer_op:= stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD       | (# machine:=f WITH [state:=STARTING], %|
                             actions:=none, %|
                             timer_op:= stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f WITH [state:=STARTING], %|
                             actions:=none, %|
                             timer_op:= stop #) ||
%-----------------------+--------------------------------------------||
|state(f)=OPENED        | (# machine:=f WITH [state:= STARTING], %|
                             actions:=tld, %|
                             timer_op:=none #)  ||
%-----------------------+--------------------------------------------||


        ENDTABLE
```

```
      fsm_open(f:fsm):ProgReq =
       TABLE
%-----------------------+--------------------------------------------||
|state(f)=INITIAL       | (# machine:=f WITH [state:= STARTING],     %|
                            actions:=tls,                             %|
                           timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STARTING      | (# machine:=f,                 %|
                            actions:=none,               %|
                           timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSED
  & IsSet(flags(f),OPT_SILENT)|(# machine:=f WITH [state:= STOPPED], %|
                                  actions:=none, %|
                                  timer_op:=none #)||
%-----------------------+--------------------------------------------||
|state(f)=CLOSED
  & NOT IsSet(flags(f),OPT_SILENT)|(#machine:=fsm_sconfreq(f,FALSE)
WITH [state:=REQSENT], %|
                                      actions:=irc o scr, %|
                                      timer_op:=start #) ||
%-----------------------+--------------------------------------------||

|state(f)=STOPPED
  & IsSet(flags(f),OPT_RESTART)        |(# machine:=
machine(fsm_lowerup(machine(fsm_lowerdown(f)))), %|
actions:=actions(fsm_lowerdown(f)) o
actions:= actions(fsm_lowerup(machine(fsm_lowerdown(f)))), %|
                                         timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPED
  & NOT IsSet(flags(f),OPT_RESTART)    |(# machine:= f, %|
                                          actions:=none, %|
                                          timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSING
  & IsSet(flags(f),OPT_RESTART)        | (# machine:=
machine(fsm_lowerup(machine(fsm_lowerdown(f WITH
[state:=STOPPING])))),%|
actions:=actions(fsm_lowerdown(f WITH [state:=STOPPING] )) o
actions(fsm_lowerup(machine(fsm_lowerdown(f WITH [state:=STOPPING])))),
%|
                                          timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSING
  & NOT IsSet(flags(f),OPT_RESTART) | (# machine:= f WITH [
state:=STOPPING], %|
                                         actions:=none, %|
                                         timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPING      | (# machine:=f,                 %|
                            actions:=none,               %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=REQSENT       | (# machine:=f,                 %|
                            actions:=none,               %|
                            timer_op:=none #) ||
```

```
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD       | (# machine:=f,                     %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=ACKSENT       | (# machine:=f,                     %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=OPENED
  & IsSet(flags(f),OPT_RESTART)       |(# machine:=
machine(fsm_lowerup(machine(fsm_lowerdown(f)))), %|
actions:= actions(fsm_lowerdown(f)) o
actions(fsm_lowerup(machine(fsm_lowerdown(f)))), %|
                                          timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=OPENED
  & NOT IsSet(flags(f),OPT_RESTART)   |(# machine:= f, %|
                                          actions:=none, %|
                                          timer_op:=none #) ||
%-----------------------+--------------------------------------------||

        ENDTABLE
```

```
fsm_close (g:{g:fsm|state(g)=INITIAL OR state(g)=STARTING OR
state(g)=CLOSED OR state(g)= STOPPED OR state(g)=CLOSING OR
state(g)=STOPPING OR
state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT OR
state(g)=OPENED}, reason:string):ProgReq =

LET f = g WITH [ term_reason:=reason,
                term_reason_len:=length(reason)] IN

TABLE
%-----------------------+--------------------------------------------||
|state(f)=INITIAL      | (# machine:=f,                    %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STARTING     |(#     machine:= f WITH [state:= INITIAL],   %|
                               actions:= tlf,                        %|
                               timer_op:=none #)                     ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSED       | (# machine:=f,                    %|
                            actions:=none,                 %|
                            timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=STOPPED      |(#     machine:=( f WITH [state:= CLOSED]),%|
                               actions:=none, %|
                               timer_op:=none #) ||
%-----------------------+--------------------------------------------||
|state(f)=CLOSING      | (# machine:=f,                    %|
                            actions:=none,                 %|
                            timer_op:=none #) ||

%-----------------------+--------------------------------------------||
|state(f)=STOPPING     |(#     machine:=(f WITH [state:= CLOSING]),%|
                               actions:=none,                      %|
                               timer_op:=none #)                   ||
%-----------------------+--------------------------------------------||
|state(f)=REQSENT OR                                    %|
  state(f)=ACKRCVD                                      %|
 OR state(f)=ACKSENT   |(#     machine:=f WITH [state:= CLOSING,%|
                               actions:=irc o str,     %|
                               timer_op:=start #)      ||
%-----------------------+--------------------------------------------||
|state(f)=OPENED       |(# machine:=f WITH [state:= CLOSING,%|
                           actions:= tld o irc o str, %|
                            timer_op:=start #)           ||
%-----------------------+--------------------------------------------||
ENDTABLE

% NOTE: In the PPP code, the function fsm_sdata is a general function
%that
% is used to send data, in particular the various requests and
% acknowlegements that are required by the protocol actions, e.g:

% fsm_sdata(f, TERMREQ, f->reqid = ++ f->id,
%         (u_char *) f->term_reason, f->term_reason_len);

% implements the requirements action:
```

```
%       str = Send-Terminate-Request

% Similarly action scr,sca,scn,sta are implemented by fsm_sdata.



fsm_timeout(f:{g:fsm|state(g)=CLOSING OR state(g)=STOPPING OR
state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT}):ProgReq =
               TABLE
%-----------------------+--------------------------------------------||
| state(f)= CLOSING
  & (retransmits(f)<=0)    | (# machine:=f WITH [state:= CLOSED], %|
                             actions:=tlf, %|
                             timer_op:=none #) ||%THIS IS TO- CASE
%-----------------------+--------------------------------------------||
| state(f)= CLOSING
  & NOT(retransmits(f)<=0)  |(# machine:=f WITH [state:= CLOSING], %|
                                actions:=str, %|
                               timer_op:=start #) || % THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
| state(f)= STOPPING
  & retransmits(f)<=0    | (# machine:=f WITH [state:= STOPPED], %|
                             actions:=tlf, %|
                              timer_op:=none #) || % THIS IS TO- CASE
%-----------------------+--------------------------------------------||

| state(f)= STOPPING
  & NOT(retransmits(f)<=0) |(# machine:=f WITH [state:= STOPPING], %|
                               actions:=str, %|
                               timer_op:=start #)||  % THIS IS TO+ CASE
%-----------------------+--------------------------------------------||

|state(f)=REQSENT & (retransmits(f)<=0) | (# machine:= f WITH [state:=
                                             STOPPED], %|
                                             actions:=tlf, %|
                                timer_op:=none #) || % THIS IS TO- CASE
%-----------------------+--------------------------------------------||
|state(f)=REQSENT&NOT(retransmits(f)<=0)|  (# machine:=
                          fsm_sconfreq(f,TRUE)WITH[state:= REQSENT], %|
                                              actions:=scr, %|
                                timer_op:=none #)||  % THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD & (retransmits(f)<=0) |   (# machine:=f WITH [state:=
                                                STOPPED], %|
                                                actions:=tlf, %|
                                 timer_op:=none #) || %THIS IS TO- CASE
%-----------------------+--------------------------------------------||
|state(f)=ACKRCVD&NOT(retransmits(f)<=0)|  (#
                 machine:=fsm_sconfreq(f,TRUE)WITH[state:= REQSENT], %|
                              actions:=scr, %|
                              timer_op:=none #)|| %    THIS IS TO+ CASE
%-----------------------+--------------------------------------------||
|state(f)=ACKSENT & (retransmits(f)<=0) |  (# machine:=f WITH [state:=
                                              STOPPED], %|
```

```
                              actions:=tlf, %|
                              timer_op:=none #) || % THIS IS TO- CASE
%-----------------------+-------------------------------------------||
|state(f)=ACKSENT&NOT(retransmits(f)<=0)|(#
                machine:=fsm_sconfreq(f,TRUE)WITH[state:= ACKSENT], %|
                                actions:=scr, %|
                                timer_op:=start #)|| %THIS IS TO+ CASE
%-----------------------+-------------------------------------------||

                ENDTABLE




%fsm_rconfreq(f,id,inp,len)
%int code,reject_if_disagree
fsm_rconfreq(f:fsm,%{g:fsm|state(g)=CLOSED OR state(g)=CLOSING OR
%state(g)=STOPPING OR state(g)=OPENED OR state(g)=STOPPED},
id: int, inp:string, len:int):ProgReq =

LET code:posint = IF len /= 0 THEN CONFREJ ELSE CONFACK ENDIF IN

              TABLE
%-----------------------+-------------------------------------------||
|(state(f)=INITIAL or state(f)=STARTING) & (code=CONFACK) | (#
machine:=f WITH [state:=ACKSENT], %|
                                 actions:= sca, %|
                                 timer_op:=none #) ||  %RCR+ CASE
%-----------------------+-------------------------------------------||
|(state(f)=INITIAL or state(f)=STARTING) & NOT(code=CONFACK) | (#
machine:=f WITH [state:=REQSENT], %|
                                 actions:= scn, %|
                                 timer_op:=none #) ||  %RCR- CASE
%-----------------------+-------------------------------------------||
|state(f)=CLOSED        | (# machine:= f WITH [state:=CLOSED], %|
                            actions:=sta, %|
                            timer_op:=none #)       ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPED & (code=CONFACK)    |(# machine:=
fsm_sconfreq(f,FALSE)WITH[state:=ACKSENT],%|
                                 actions:= irc o scr o sca, %|
                                 timer_op:=start #) || %RCR+ CASE
%-----------------------+-------------------------------------------||
|state(f)=STOPPED & NOT(code=CONFACK) |(# machine:=
fsm_sconfreq(f,FALSE) WITH[state:=REQSENT],   %|
                                 actions:= irc o scr o scn, %|
                                 timer_op:=start #) || %RCR- CASE
%-----------------------+-------------------------------------------||
|state(f)=CLOSING       | (# machine:= f WITH [state:=CLOSING], %|
                            actions:=none, %|
                            timer_op:=none #)             ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPING      | (# machine:= f WITH [state:=STOPPING], %|
                            actions:=none, %|
                            timer_op:=none #)             ||
%-----------------------+-------------------------------------------||
|state(f)=REQSENT & (code=CONFACK) | (# machine:=f WITH
```

```
[state:=ACKSENT], %|
                                  actions:= sca, %|
                                  timer_op:=none #) ||  %RCR+ CASE
%-----------------------+---------------------------------------------||
|state(f)=REQSENT & NOT(code=CONFACK) | (# machine:=f WITH
[state:=REQSENT], %|
                                  actions:= scn, %|
                                  timer_op:=none #) ||  %RCR-CASE
%-----------------------+---------------------------------------------||
|state(f)=OPENED & code=CONFACK | (# machine:=fsm_sconfreq(f,FALSE)
WITH [state:=ACKSENT], %|
                                  actions:= tld o scr o sca, %|
                                  timer_op:=start #) ||  %RCR+ CASE
%-----------------------+---------------------------------------------||
|state(f)=OPENED & NOT(code=CONFACK)| (# machine:=fsm_sconfreq(f,FALSE)
WITH [state:=REQSENT], %|
                                  actions:= tld o scr o scn, %|
                                  timer_op:=start #) ||  %RCR- CASE
%-----------------------+---------------------------------------------||
|state(f)=ACKRCVD & (code=CONFACK)|(# machine:= f WITH [state:=
OPENED],%|
                                       actions:= sca o tlu, %|
                                       timer_op:=stop #)  || %RCR+ CASE
%-----------------------+---------------------------------------------||
|state(f)=ACKRCVD & NOT(code=CONFACK) |
                                (# machine:= f,       %|
                                   actions:= scn, %|
                                   timer_op:=stop #)  || %RCR- CASE
%-----------------------+---------------------------------------------||
|state(f)=ACKSENT & (code=CONFACK) | (# machine:=f WITH
[state:=ACKSENT], %|
                                  actions:= sca, %|
                                  timer_op:=none #) ||  %RCR+ CASE
%-----------------------+---------------------------------------------||
|state(f)=ACKSENT & NOT(code=CONFACK) | (# machine:=f WITH
[state:=REQSENT], %|
                                  actions:= scn, %|
                                  timer_op:=none #) ||  %RCR-CASE
%-----------------------+---------------------------------------------||
                  ENDTABLE
```

```
%fsm_rconfack(f,id,inp,len)
fsm_rconfack(f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR state(g)=
CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR state(g)=OPENED}, id: int, inp:string,
len:int):ProgReq  =


               TABLE
%-----------------------+-------------------------------------------||
%f->seenack=1
%-----------------------+-------------------------------------------||
|state(f)=CLOSED       |(# machine:= f WITH  [state:=CLOSED], %|
                            actions:= sta, %|
                            timer_op:=none #)  ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPED       |(# machine:= f WITH [state:=STOPPED], %|
                            actions:=sta, %|
                            timer_op:=none #)  ||
%-----------------------+-------------------------------------------||
|state(f)=CLOSING      | (# machine:=f,                 %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=STOPPING     | (# machine:=f,                 %|
                            actions:=none,              %|
                            timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=REQSENT      | (# machine:=f WITH [state:=ACKRCVD], %|
                             actions:= irc, %|
                             timer_op:=none #) ||
%-----------------------+-------------------------------------------||
|state(f)=ACKRCVD      |(# machine:=
                         fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                            actions:=scr, %|
                            timer_op:=start #) ||
%-----------------------+-------------------------------------------||
|state(f)= ACKSENT      |(# machine:= f WITH [state:=OPENED],
                             actions:=irc o tlu, %|
                             timer_op:=stop #) ||
%-----------------------+-------------------------------------------||
|state(f)=OPENED        |(# machine:=
                         fsm_sconfreq(f,FALSE)WITH [state:=REQSENT], %|
                            actions:=tld o scr, %|
                            timer_op:=start #)||
%-----------------------+-------------------------------------------||

               ENDTABLE
```

```
%fsm_rconfnackrej(f,code,id,inp,len)
%int ret;
fsm_rconfnackrej(f:fsm, code:int, id: int, inp:string, len:int):ProgReq
=


               TABLE
%----------------------+---------------------------------------------||
%f->seenack=1
%----------------------+---------------------------------------------||
|state(f)=CLOSED        |(# machine:= f WITH [state:=CLOSED], %|
                            actions:=sta, %|
                            timer_op:=none #) ||
%----------------------+---------------------------------------------||
|state(f)=STOPPED        |(# machine:= f WITH [state:=STOPPED], %|
                            actions:=sta, %|
                            timer_op:=none #) ||
%----------------------+---------------------------------------------||
|state(f)=CLOSING      | (# machine:=f,                %|
                           actions:=none,              %|
                           timer_op:=none #) ||
%----------------------+---------------------------------------------||
|state(f)=STOPPING     | (# machine:=f,                %|
                           actions:=none,              %|
                           timer_op:=none #) ||
%----------------------+---------------------------------------------||
%|state(f)=REQSENT & (ret<0)                 | UNTIMEOUT(fsm_timeout,f)
||
%                                             f WITH [state:=STOPPED]
%||
% WE DO NOT HAVE THIS CASE IN THE SPECIFICATION RFC TABLE
%----------------------+---------------------------------------------||
| state(f)=REQSENT %& NOT(ret<0)
            |(#machine:=fsm_sconfreq(f,FALSE) WITH [state:=REQSENT], %|
                             actions:= irc o scr, %|
                             timer_op:=start #)||
%----------------------+---------------------------------------------||
%|state(f)=ACKSENT & (ret<0)                 | UNTIMEOUT(fsm_timeout,f)
||
%                                             f WITH [state:=STOPPED]
%||
% WE DO NOT HAVE THIS CASE IN THE SPECIFICATION RFC TABLE
%----------------------+---------------------------------------------||
| state(f)=ACKSENT %& NOT(ret<0)
                       |(# machine:=fsm_sconfreq(f,FALSE)
                                   WITH [state:=ACKSENT], %|
                                   actions:= irc o scr, %|
                                   timer_op:=start #) ||
%----------------------+---------------------------------------------||
|state(f)=ACKRCVD                        |(#
machine:=fsm_sconfreq(f,FALSE)WITH
[state:=REQSENT], %|
                               actions:= scr, %|
                               timer_op:=start #) ||
%----------------------+---------------------------------------------||
|state(f)=OPENED
            | (# machine:= fsm_sconfreq(f,FALSE)WITH [state:=REQSENT],
```

```
%|
                            actions:= tld o scr, %|
                            timer_op:=start #) ||
%----------------------+---------------------------------------------||
| ELSE          | (# machine:= f, actions:= none, timer_op:=none #)  ||
%----------------------+---------------------------------------------||
              ENDTABLE
```

```
%fsm_rtermreq(f,id,p,len)
% NOTE: They seem to be missing FSM_DEBUG call in code for
%       cases that are not defined by RFC.
%
fsm_rtermreq(f:fsm, id:int, p:string, len:int):ProgReq =
               TABLE
%----------------------+---------------------------------------------||
|state(f)=ACKRCVD      |(#machine:= f WITH [state:=REQSENT], %|
                            actions:= sta, %|
                            timer_op:=none #) ||
%----------------------+---------------------------------------------||
|state(f)= ACKSENT     | (#machine:= f WITH [state:=REQSENT], %|
                             actions:= sta, %|
                             timer_op:=none #)  ||
%----------------------+---------------------------------------------||
|state(f)=OPENED         |(#machine:=f WITH [state:=STOPPING], %|
                                         actions:= tld o zrc o sta, %|
                                         timer_op:=start #)||
%----------------------+---------------------------------------------||
|ELSE                  |(#machine:=f,                            %|
                               actions:= sta,                    %|
                               timer_op:= none #)                ||
```

```
% NOTE: They seem to be missing FSM_DEBUG call in code for
%       cases that are not defined by RFC.
%       same as in RTR(fsm_rtermreq)event.
fsm_rtermack(f:fsm): ProgReq =
       TABLE
%----------------------+---------------------------------------------||
|state(f)=CLOSING      | (# machine:= f WITH  [state:= CLOSED], %|
                                actions:= tlf, %|
                                timer_op:=stop #) ||
%----------------------+---------------------------------------------||
|state(f)=STOPPING     | (# machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=stop #)||
%----------------------+---------------------------------------------||
|state(f)=ACKRCVD       | (#machine:=f WITH [state:= REQSENT], %|
                             actions:= none, %|
                             timer_op:=none #)  ||
```

```
%----------------------+---------------------------------------------||
|state(f)=OPENED        |
            (#machine:= fsm_sconfreq(f,FALSE), %|
                           actions:= tld o scr, %|
                           timer_op:=start #) ||
% HERE IS MISTAKE BECAUSE IMPLEMENTERS DID NOT UPDATE A NEW STATE (IT
SHOULD BE REQSENT ACCORDING TO RFC DOCUMENT!
%----------------------+---------------------------------------------||
|ELSE                  |(#machine:=f , %|
                             actions:= none, %|
                             timer_op:=none #)  ||

             ENDTABLE



fsm_protreject(f:{g:fsm|state(g)=CLOSING OR state(g)=CLOSED OR
state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT
OR state(g)=STOPPED OR state(g)=OPENED}):ProgReq =

         TABLE
%----------------------+---------------------------------------------||

|state(f)=CLOSING      |(#machine:=f WITH[state:=CLOSED], %|
                           actions:= tlf, %|
                           timer_op:=stop #) ||
%----------------------+---------------------------------------------||
|state(f)=CLOSED       |(#machine:=f WITH [state:= CLOSED], %|
                           actions:= tlf, %|
                           timer_op:=none #)   ||

%----------------------+---------------------------------------------||

|state(f)=STOPPING     | (#machine:=f WITH [state:= STOPPED], %|
                            actions:= tlf, %|
                            timer_op:=stop #) ||

%----------------------+---------------------------------------------||
|state(f)=REQSENT      |  (#machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=stop #) ||

%----------------------+---------------------------------------------||

|state(f)=ACKRCVD      |  (#machine:=f WITH [state:= STOPPED], %|
                             actions:= tlf, %|
                             timer_op:=stop #) ||

%----------------------+---------------------------------------------||

|state(f)=ACKSENT      | (#machine:=f WITH [state:= STOPPED], %|
                            actions:= tlf, %|
                            timer_op:=stop #) ||
%----------------------+---------------------------------------------||
```

```
|state(f)=STOPPED      | (#machine:=f WITH [state:= STOPPED], %|
                          actions:= tlf, %|
                          timer_op:=none #)||

%----------------------+-------------------------------------------||
|state(f)=OPENED        | (#machine:=f WITH [state:= STOPPING], %|
                           actions:= tld o irc o str, %|
                           timer_op:=start #) ||
%----------------------+-------------------------------------------||
              ENDTABLE



  END pppCode
```

$$$pppCode.prf (PROOFS)

```
(|pppCode| (|fsm_lowerup_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerup_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerup_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerup_TCC4| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_lowerup_TCC5| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC6| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC7| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC8| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_lowerdown_TCC9| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_lowerdown_TCC10| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_open_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC6| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC7| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC8| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC9| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC10| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC11| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_open_TCC12| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_open_TCC13| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_close_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC5| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_close_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
 (|fsm_close_TCC7| "" (COND-COVERAGE-TCC) NIL NIL)
 (|fsm_timeout_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_TCC3| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_TCC4| "" (SUBTYPE-TCC) NIL NIL)
 (|fsm_timeout_TCC5| "" (SUBTYPE-TCC) NIL NIL)
```

```
(|fsm_timeout_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_timeout_TCC7| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_timeout_TCC8| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_timeout_TCC9| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_timeout_TCC10| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_timeout_TCC11| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_timeout_TCC12| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rconfreq_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC6| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfreq_TCC7| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rconfreq_TCC8| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC9| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC10| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfreq_TCC11| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfreq_TCC12| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rconfreq_TCC13| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfreq_TCC14| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rconfreq_TCC15| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfreq_TCC16| "" (COND-COVERAGE-TCC) NIL NIL)
(|fsm_rconfack_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfack_TCC7| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rconfnackrej_TCC7| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_rtermreq_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermreq_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermreq_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_rtermack_TCC3| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_protreject_TCC1| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC2| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC3| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC4| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC5| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC6| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC7| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC8| "" (SUBTYPE-TCC) NIL NIL)
(|fsm_protreject_TCC9| "" (COND-DISJOINT-TCC) NIL NIL)
(|fsm_protreject_TCC10| "" (COND-COVERAGE-TCC) NIL NIL))
```

4.
$$$pppInspection.pvs

```
pppInspection  % [ parameters ]
               : THEORY

  BEGIN

  % ASSUMING
   % assuming declarations
  % ENDASSUMING

  IMPORTING pppSpec, pppCode
  f:VAR fsm
  flag:VAR int


  Up:PROPOSITION pppSpec.fsm_lowerup = pppCode.fsm_lowerup

  Down:PROPOSITION pppSpec.fsm_lowerdown = pppCode.fsm_lowerdown

  Open:PROPOSITION pppSpec.fsm_open = pppCode.fsm_open

  Close:PROPOSITION pppSpec.fsm_close = pppCode.fsm_close

  Timeout:PROPOSITION pppCode.fsm_timeout =
        lambda (f:{g:fsm|state(g)=CLOSING OR state(g)=STOPPING OR
state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT}):
   IF retransmits(f)<=0 THEN pppSpec.fsm_timeout_minus(f)
   ELSE pppSpec.fsm_timeout_plus(f) ENDIF

 id:    VAR int
 inp, p: VAR string
 len:   VAR int
 code:  VAR int

  RCR:PROPOSITION
        (lambda (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED
               OR state(g)=CLOSING OR state(g)=STOPPING OR
               state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT
               OR state(g)=OPENED}):
pppCode.fsm_rconfreq(f, id, inp, len)) =
        (lambda (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED
               OR state(g)=CLOSING OR state(g)=STOPPING OR
               state(g)=REQSENT OR state(g)=ACKRCVD OR state(g)=ACKSENT
               OR state(g)=OPENED}):
   IF len=0 THEN pppSpec.fsm_rconfreq_plus(f)
   ELSE pppSpec.fsm_rconfreq_minus(f) ENDIF)



  RCA:PROPOSITION pppSpec.fsm_rconfack =
lambda (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR state(g)=
CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR state(g)=OPENED}):
pppCode.fsm_rconfack(f, id, inp, len)
```

```
  RCN:PROPOSITION
pppSpec.fsm_rconfnackrej=
lambda (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKSENT OR
state(g)=ACKRCVD OR state(g)=OPENED}): fsm_rconfnackrej(f, code, id,
inp,
len)

  RTR:PROPOSITION
pppSpec.fsm_rtermreq=
lambda (f:{g:fsm|state(g)=CLOSED OR state(g)=STOPPED OR state(g)=
CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR state(g)=ACKRCVD OR
state(g)=ACKSENT OR state(g)=OPENED}):fsm_rtermreq(f, id, p, len)

  RTA:PROPOSITION
(lambda (f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR state(g)=OPENED}):
pppSpec.fsm_rtermack(f)) =
(LAMBDA (f:{g:fsm| state(g)=CLOSED OR state(g)=STOPPED OR
state(g)=CLOSING OR state(g)=STOPPING OR state(g)=REQSENT OR
state(g)=ACKRCVD OR state(g)=ACKSENT OR
state(g)=OPENED}):pppCode.fsm_rtermack(f))

  RXJ:PROPOSITION
pppSpec.fsm_protreject = pppCode.fsm_protreject

% Check fsm_lowerup program in the case when flags are not set.
  Up_No_Flags:PROPOSITION
        NOT IsSet(flags(f),OPT_SILENT)
     => pppSpec.fsm_lowerup(f) = pppCode.fsm_lowerup(f)

% Check fsm_open program in the case when flags are not set.
  Open_No_Flags:PROPOSITION
        NOT IsSet(flags(f),OPT_SILENT) & NOT IsSet(flags(f),OPT_RESTART)
     => pppSpec.fsm_open(f) = pppCode.fsm_open(f)

  END pppInspection
```

$$$**pppInspection.prf (PROOFS)**

```
(|pppInspection|
 (|Up| "" (APPLY-EXTENSIONALITY)
  (("" (HIDE 2)
    (("" (GRIND) (("1" (POSTPONE) NIL NIL) ("2" (POSTPONE) NIL NIL))
NIL))
    NIL))
  NIL)
 (|Down| "" (APPLY-EXTENSIONALITY :HIDE? TRUE) (("" (GRIND) NIL NIL))
NIL)
 (|Open| "" (APPLY-EXTENSIONALITY :HIDE? TRUE)
  (("" (GRIND)
    (("1" (POSTPONE) NIL NIL) ("2" (POSTPONE) NIL NIL)
     ("3" (POSTPONE) NIL NIL) ("4" (POSTPONE) NIL NIL)
     ("5" (POSTPONE) NIL NIL) ("6" (POSTPONE) NIL NIL)
     ("7" (POSTPONE) NIL NIL) ("8" (POSTPONE) NIL NIL)
```

```
      ("9" (POSTPONE) NIL NIL))
     NIL))
   NIL)
  (|Close| "" (APPLY-EXTENSIONALITY :HIDE? TRUE) (("" (GRIND) NIL NIL))
NIL)
 (|Timeout_TCC1| "" (SUBTYPE-TCC) NIL NIL)
 (|Timeout_TCC2| "" (SUBTYPE-TCC) NIL NIL)
 (|Timeout| "" (APPLY-EXTENSIONALITY :HIDE? TRUE)
  (("1" (GRIND)
    (("1" (TYPEPRED "x!1") (("1" (GRIND) NIL NIL)) NIL)
     ("2" (TYPEPRED "x!1") (("2" (GRIND) NIL NIL)) NIL))
    NIL)
   ("2" (GRIND) NIL NIL) ("3" (GRIND) NIL NIL))
  NIL)
 (RCR "" (SKOLEM-TYPEPRED)
  (("" (APPLY-EXTENSIONALITY :HIDE? TRUE)
    (("" (TYPEPRED "x!1")
      (("" (LIFT-IF)
        (("" (SPLIT 1)
          (("1" (FLATTEN) (("1" (GRIND) NIL NIL)) NIL)
           ("2" (FLATTEN)
            (("2" (GRIND)
              (("2" (TYPEPRED "x!1")
                (("2" (GRIND) (("2" (POSTPONE) NIL NIL)) NIL)) NIL))
              NIL))
            NIL))
          NIL))
        NIL))
      NIL))
    NIL))
  NIL)
 (RCA "" (SKOLEM!)
  (("" (APPLY-EXTENSIONALITY :HIDE? TRUE) (("" (GRIND) NIL NIL)) NIL))
NIL)
 (RCN "" (SKOLEM!)
  (("" (APPLY-EXTENSIONALITY :HIDE? TRUE)
    (("" (EXPAND "restrict")
      (("" (GRIND) (("" (TYPEPRED "x!1") (("" (GRIND) NIL NIL)) NIL))
NIL))
      NIL))
    NIL))
  NIL)
 (RTR "" (SKOLEM!)
  (("" (APPLY-EXTENSIONALITY :HIDE? TRUE)
    (("" (TYPEPRED "x!1")
      (("" (GRIND)
        (("1" (DECOMPOSE-EQUALITY) NIL NIL) ("2" (DECOMPOSE-EQUALITY)
NIL NIL)
         ("3" (DECOMPOSE-EQUALITY) NIL NIL) ("4" (DECOMPOSE-EQUALITY)
NIL NIL)
         ("5" (DECOMPOSE-EQUALITY) NIL NIL))
        NIL))
      NIL))
    NIL))
  NIL)
 (RTA "" (APPLY-EXTENSIONALITY :HIDE? TRUE)
  (("" (EXPAND "restrict")
```

```
    ((""  (TYPEPRED "x!1") (("" (GRIND) ((""  (POSTPONE) NIL NIL)) NIL))
NIL))
    NIL))
   NIL)
  (RXJ_TCC1 "" (SUBTYPE-TCC) NIL NIL)
  (RXJ "" (APPLY-EXTENSIONALITY :HIDE? TRUE)
   (("1" (GRIND) NIL NIL) ("2" (GRIND) NIL NIL)) NIL)
  (|Up_No_Flags| "" (SKOLEM!) ((""  (GRIND) NIL NIL)) NIL)
  (|Open_No_Flags| "" (GRIND) NIL NIL))
```

APPENDIX B: Displays for the inspection of the PPP FSM Code

IMPORTANT NOTE 1:

In almost all cases the variables being referred to are fields of the datastructure f. Variables appearing in the left column all refer to fields `pf f` unless somehow indicated. Within the tables themselves, it then becomes a problem in cases such as Display 14 when there is a formal parameter "`id`" and `f->id`.
This is resolved with a convention using different fonts (e.g. bold for the formal parameter id).

IMPORTANT NOTE 2: Here are the names of the C code function that implements the actions. i.e. action scr for example is implemented by the function `fsm_sconfreq` and `fsm_data`.

Here is the list of the mapping from these C functions to actions would be very handy at the start of the Displays appendix. Also, in section 6.4 where you describe the two examples in detail, be sure to point out the actions being implemented by the C function calls in the code. You could then refer people to the displays appendix for the complete table.

% In the PPP code, the function fsm_sdata(f, code, id, inp, len) is a general function that is used to send data, in particular the various requests and acknowlegements that are required by the protocol actions, e.g:

```
% fsm_sdata(f, TERMREQ, f->reqid = ++ f->id,
%         (u_char *) f->term_reason, f->term_reason_len);

% implements the requirements action:

%       str = Send-Terminate-Request

% Similarly action scr,sca,scn,sta,str are implemented by fsm_sdata.
```

1. **str(Send-Terminate-Request)** action is implemented by function

```
fsm_sdata(f, TERMREQ, f->reqid = ++f->id,
              (u_char *) f->term_reason, f->term_reason_len);
```

2. **sta(Send-Terminate-Ack)**action is implemented by function
```
      fsm_sdata(f, TERMACK, id, NULL, 0);
```

3. **scn(Send-Configure-Nak/Rej)**action action is implemented by

```
      fsm_sdata(f, CODEREJ, ++f->id, inpacket, len + HEADERLEN);
```

4. **scr(Send a Configure-Request)**action is implemented by functions

```
fsm_sconfreq(f, 0);
fsm_sdata(f, CONFREQ, f->reqid, outp, cilen)
```

5. **sca (Send-Configure-Ack)**action is implemented by function

```
fsm_sdata(f, CONFACK, id, inp, len);
```

6. **tld (This-Layer-Down)**action is implemented by

```
if( f->callbacks->down )
    (*f->callbacks->down)(f);
```

7. **tlf (This-Layer-Finished)**action is implemented by

```
if( f->callbacks->finished )
    (*f->callbacks->finished)(f);
```

8. **tls (This-Layer-Started)action** is implemented by

```
if( f->callbacks->starting )
    (*f->callbacks->starting)(f);
```

9. **tlu (This-Layer-Up)**action is implemented by

```
if (f->callbacks->up)
      (*f->callbacks->up)(f);
```

10. **zrc=Send-Configure-Ack** action is implemented by function

```
UNTIMEOUT(fsm_timeout, f);
```

11. **irc (Initialize-Restart-Counter)**action is implemented by function

```
TIMEOUT(fsm_timeout, f, f->timeouttime);
```

Display 1

Specification

void fsm_init (fsm *f)	int state,int flags, u_char id, int timeouttime, int maxconfretramsmits, int maxtermtransmits,int term_reason_len
Ro (,) = ((state' = INITIAL) AND (flags' = 0) AND (id' = 0) AND (timeouttime' = DEFTIMEOUT) AND	
	(maxconfreqtransmits' = DEFMAXCONFREQS) AND (maxtermtransmits' = DEFMAXTERMREQS)
	AND (term_reason_len' = 0))

Program

```
/*
 * fsm_init - Initialize fsm.
 *
 * Initialize fsm state.
 */
void
fsm_init(f)
    fsm *f;
{
    f->state = INITIAL;
    f->flags = 0;
    f->id = 0;
    f->timeouttime = DEFTIMEOUT;
    f->maxconfreqtransmits = DEFMAXCONFREQS;
    f->maxtermtransmits = DEFMAXTERMREQS;
    f->term_reason_len = 0;
}
```

Specification of Invoked programs: Empty

End of Display 1

Display 2

Specification

d fsm_lowerup (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME										
	' state = INITIAL	' state = STARTING ∧		' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
		flags&OPT_SILENT	¬ (flags&OPT_SILENT)								
state' =	CLOSED	STOPPED	REQSENT								
callbacks' =	none	none	addci								
timer' =	none	none	start								
flags' =	' flags	OPT_SILENT	' flags								
code' =	' code	' code	CONFREQ								
reqid' =	' reqid	' reqid	id +1								
timeouttime' =	' timeouttime	' timeouttime	DEFTIMEOUT								
retransmits' =	' retransmits	' retransmits	maxconfreqtransmits								
protoname' =	' PROTONAME	' PROTONAME	' PROTONAME	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "
data' =	' data	' data	outp								
datalen' =	' datalen	' datalen	cilen								

Program

```
/*
 * fsm_lowerup - The lower layer is up
 */
void
fsm_lowerup(f)
    fsm *f;
{
    switch( f->state ){
    case INITIAL:
        f->state = CLOSED;
        break;

    case STARTING:
        if( f->flags & OPT_SILENT )
            f->state = STOPPED;
        else {
            /* Send an initial configure-request */
            fsm_sconfreq(f, 0);
            f->state = REQSENT;
        }
        break;

    default:
        FSMDEBUG(("%s: Up event in state %d!", PROTO_NAME(f), f>state));
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

FSMDEBUG(PROTO_NAME, state)		
R2 (,) = (PROTO_NAME = "MISTAKE")		

On Display 17

End of Display 2

Display 3

Specification

void fsm_lowerdown (fsm * f)	int state,int flags,u_char id,uchar reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTONAME									
	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			INITIAL	STARTING	INITIAL	STARTING	STARTING	STARTING	STARTING	STARTING
callbacks' =			none	starting	finished	finished	finished	finished	finished	down
timer' =			none	none	stop	stop	stop	stop	stop	none
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			' code	' code	' code	' code	' code	' code	' code	' code
reqid' =			' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			' data	' data	' data	' data	' data	' data	' data	' data
datalen' =			' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen

Program

```
/*
 * fsm_lowerdown - The lower layer is down.
 *
 */
void
fsm_lowerdown(f)
    fsm *f;
{
    switch( f->state ){
    case CLOSED:
        f->state = INITIAL;
        break;

    case STOPPED:
        f->state = STARTING;
        if( f->callbacks->starting )
            (*f->callbacks->starting)(f);
        break;

    case CLOSING:
        f->state = INITIAL;
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        break;

    case STOPPING:
    case REQSENT:
    case ACKRCVD:
    case ACKSENT:
        f->state = STARTING;
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        break;

    case OPENED:
        if( f->callbacks->down )
            (*f->callbacks->down)(f);
        f->state = STARTING;
        break;
```

```
    default:
        FSMDEBUG(("%s: Down event in state %d!", PROTO_NAME(f), f-
>state));
    }
}
```

Specification of Invoked programs

UNTIMEOUT (fsm_timeout, f)					
R1 (,) = (timer = stop)					

On Display 16

FSMDEBUG(PROTO_NAME, state)					
R2 (,) = (PROTO_NAME = "MISTAKE")					

On Display 17

End of Display 3

Display 4

Specification

void fsm_open (fsm * f)

int state, int flags, u_char id, uchar reqid, int timeouttime, int maxconfreqtransmits, int retransmits, int code, u_char callbacks, u_char data, int datalen, u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED ∧		' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
			flags&OPT_SILENT	¬ (flags&OPT_SILENT)							
state' =	STARTING	STARTING	STOPPED	REQSENT	STOPPED	CLOSING	STOPPING	REQSENT	ACKRCVD	ACKSENT	OPENED
callbacks' =	starting	none	none	addci	none	none	none	none	none	none	none
timer' =	none	none	none	start	none	none	none	none	none	none	none
flags' =	' flags	' flags	OPT_SILENT	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =	' code	' code	' code	CONFREQ	' code	' code	' code	' code	' code	' code	' code
reqid' =	' reqid	' reqid	' reqid	id	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid
timeouttime' =	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime
retransmits' =	' retransmits	' retransmits	' retransmits	maxconfreqtransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits
PROTO_NAME' =	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =	' data	' data	' data	outp	' data	' data	' data	' data	' data	' data	' data
datalen' =	' datalen	' datalen	' datalen	cilen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen

Program

```
/*
 * fsm_open - Link is allowed to come up.
 */
void
fsm_open(f)
    fsm *f;
{
    switch( f->state ){
    case INITIAL:
        f->state = STARTING;
        if( f->callbacks->starting )
            (*f->callbacks->starting)(f);
        break;

    case CLOSED:
        if( f->flags & OPT_SILENT )
            f->state = STOPPED;
        else {
            /* Send an initial configure-request */
            fsm_sconfreq(f, 0);
            f->state = REQSENT;
        }
        break;

    case CLOSING:
        f->state = STOPPING;
        /* fall through */
    case STOPPED:
    case OPENED:
        if( f->flags & OPT_RESTART ){
            fsm_lowerdown(f);
            fsm_lowerup(f);
        }
        break;
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

d fsm_lowerup (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME											
	' state = INITIAL	' state = STARTING ∧		' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
		flags&OPT_SILENT	¬ (flags&OPT_SILENT)								
state' =	CLOSED	STOPPED	REQSENT								
callbacks' =	none	none	addci								
timer' =	none	none	start								
flags' =	' flags	OPT_SILENT	' flags								
code' =	' code	' code	CONFREQ								
reqid' =	' reqid	' reqid	id +1								
timeouttime' =	' timeouttime	' timeouttime	DEFTIMEOUT								
retransmits' =	' retransmits	' retransmits	maxconfreqtransmits								
protoname' =	' PROTONAME	' PROTONAME	' PROTONAME	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "	" MISTAKE "
data' =	' data	' data	outp								
datalen' =	' datalen	' datalen	cilen								

On Display 2

void fsm_lowerdown (fsm * f)	int state,int flags,u_char id,uchar reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTONAME										
	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			INITIAL	STARTING	INITIAL	STARTING	STARTING	STARTING	STARTING	STARTING
callbacks' =			none	starting	finished	finished	finished	finished	finished	down
timer' =			none	none	stop	stop	stop	stop	stop	none
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			' code	' code	' code	' code	' code	' code	' code	' code
reqid' =			' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			' data	' data	' data	' data	' data	' data	' data	' data
datalen' =			' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen

On Display 3

End of Display 4

Display 5

Specification

void fsm_close (fsm * f, char * reason)) | int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =	INITIAL	INITIAL	CLOSED	CLOSED	CLOSING	CLOSING	CLOSING	CLOSING	CLOSING	CLOSING
callbacks' =	none	none	none	none	none	none	none	none	none	down
timer' =	none	none	none	none	none	none	start	start	start	start
flags' =	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =	' code	' code	' code	' code	' code	' code	TERMREQ	TERMREQ	TERMREQ	TERMREQ
reqid' =	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	id+1	id+1	id+1	id+1
timeouttime' =	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime
retransmits' =	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits
PROTO_NAME' =	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =	' data	' data	' data	' data	' data	' data	term_reason	term_reason	term_reason	term_reason
datalen' =	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	term_reason_len	term_reason_len	term_reason_len	term_reason_len

Program

```
/*
 * fsm_close - Start closing connection.
 *
 * Cancel timeouts and either initiate close or possibly go directly to
 * the CLOSED state.
 */
void
fsm_close(f, reason)
    fsm *f;
    char *reason;
{
    f->term_reason = reason;
    switch( f->state ){
    case STARTING:
        f->state = INITIAL;
        break;
    case STOPPED:
        f->state = CLOSED;
        break;
    case STOPPING:
        f->state = CLOSING;
        break;

    case REQSENT:
    case ACKRCVD:
    case ACKSENT:
    case OPENED:
        if( f->state != OPENED )
            UNTIMEOUT(fsm_timeout, f); /* Cancel timeout */
        else if( f->callbacks->down )
            (*f->callbacks->down)(f);  /* Inform upper layers we're down
*/

        /* Init restart counter, send Terminate-Request */
        f->retransmits = f->maxtermtransmits;
        fsm_sdata(f, TERMREQ, f->reqid = ++f->id,
                  (u_char *) f->term_reason, f->term_reason_len);
        TIMEOUT(fsm_timeout, f, f->timeouttime);
        --f->retransmits;

        f->state = CLOSING;
```

```
            break;
        }
}
```

Specification of Invoked programs

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)					
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK	
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK	
id' =	reqid	reqid+1	' id	' id	
data' =	outp	term_reason	NULL	inp	
datalen' =	cilen	term_reason_len	0	len	
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf	
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	

On Display 14

TIMEOUT (fsm_timeout, f , timeouttime)
R1 (,) = (timer = start) AND (timeouttime' = DEFTIMEOUT)

On Display 15

UNTIMEOUT (fsm_timeout, f)
R1 (,) = (timer = stop)

On Display 16

End of Display 5

Display 6

Specification

fsm_timeout (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME														
	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING ∧		' state = STOPPING ∧		' state = REQSENT ∧		' state = ACKRCVD ∧		' state = ACKSENT ∧		' state = OPENED
					retransmits<=0	¬ (retransmits<= 0)	retransmits<=0	¬ (retransmits<= 0)	retransmits<=0	¬ (retransmits<= 0)	retransmits<=0	¬ (retransmits<= 0)	retransmits<=0	¬ (retransmits<= 0)	
state' =					CLOSED	CLOSING	STOPPED	STOPPING	STOPPED	REQSENT	STOPPED	REQSENT	STOPPED	ACKSENT	
callbacks' =					finished	none	finished	none	finished	addci	finished	addci	finished	addci	
timer' =					none	start	none	start	none	start	none	start	none	start	
flags' =					' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	
code' =					' code	TERMREQ	' code	TERMREQ	' code	CONFREQ	' code	CONFREQ	' code	CONFREQ	
reqid' =					' reqid	id +1	' reqid	id +1	' reqid	id	' reqid	id	' reqid	id	
timeouttime' =					' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	
retransmits' =					retransmits<=0	retransmits > 0	retransmits<=0	retransmits > 0	retransmits<=0	maxconfretransmits	retransmits<=0	maxconfretransmits	retransmits<=0	maxconfretransmits	
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	"MISTAKE"
data' =					' data	term_reason	' data	term_reason	' data	outp	' data	outp	' data	outp	
datalen' =					' datalen	term_reason_len	' datalen	term_reason_len	' datalen	cilen	' datalen	cilen	' datalen	cilen	

Program

```
/*
 * fsm_timeout - Timeout expired.
 */
static void
fsm_timeout(arg)
    void *arg;
{
    fsm *f = (fsm *) arg;

    switch (f->state) {
    case CLOSING:
    case STOPPING:
        if( f->retransmits <= 0 ){
            /*
             * We've waited for an ack long enough.  Peer probably heard
us.
             */
            f->state = (f->state == CLOSING)? CLOSED: STOPPED;
            if( f->callbacks->finished )
                (*f->callbacks->finished)(f);
        } else {
            /* Send Terminate-Request */
            fsm_sdata(f, TERMREQ, f->reqid = ++f->id,
                      (u_char *) f->term_reason, f->term_reason_len);
            TIMEOUT(fsm_timeout, f, f->timeouttime);
            --f->retransmits;
        }
        break;

    case REQSENT:
    case ACKRCVD:
    case ACKSENT:
        if (f->retransmits <= 0) {
            warn("%s: timeout sending Config-Requests\n",
PROTO_NAME(f));
            f->state = STOPPED;
            if( (f->flags & OPT_PASSIVE) == 0 && f->callbacks->finished
)
                (*f->callbacks->finished)(f);

        } else {
```

```
            /* Retransmit the configure-request */
            if (f->callbacks->retransmit)
                (*f->callbacks->retransmit)(f);
            fsm_sconfreq(f, 1);                /* Re-send Configure-
Request */
            if( f->state == ACKRCVD )
                f->state = REQSENT;
        }
        break;

    default:
        FSMDEBUG(("%s: Timeout event in state %d!", PROTO_NAME(f), f-
>state));
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)					
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK	
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK	
id' =	reqid	reqid+1	' id	' id	
data' =	outp	term_reason	NULL	inp	
datalen' =	cilen	term_reason_len	0	len	
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf	
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	

On Display 14

UNTIMEOUT (fsm_timeout, f)		
R1 (,) = (timer = stop)		

On Display 16

FSMDEBUG(PROTO_NAME, state)		
R2 (,) = (PROTO_NAME = "MISTAKE")		

On Display 17

End of Display 6

Display 7

Specification

fsm_rconfreq (fsm * f, u_char id, u_char * inp, int len) | int state, int flags, u_char id, u_char reqid, int timeouttime, int maxconfreqtransmits, int retransmits, int code, u_char callbacks, u_char data, int datalen, u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED ∧		' state = CLOSING	' state = STOPPING
				code=CONFACK	¬ (code=CONFACK)		
state' =			CLOSED	ACKSENT	REQSENT	CLOSING	STOPPING
callbacks' =			none	addci	addci	none	none
timer' =			none	start	start	none	none
flags' =			' flags	' flags	' flags	' flags	' flags
code' =			TERMACK	CONFACK	CONFREQ	' code	' code
reqid' =			id	id	id	' reqid	' reqid
timeouttime' =			' timeouttime	DEFTIMEOUT	DEFTIMEOUT	' timeouttime	' timeouttime
retransmits' =			' retransmits	maxconfreqtransmits	maxconfreqtransmits	' retransmits	' retransmits
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			NULL	inp	inp	' data	' data
datalen' =			0	0	len	' datalen	' datalen

	' state = REQSENT ∧		' state = ACKRCVD ∧		' state = ACKSENT ∧		' state = OPENED ∧	
	code=CONFACK	¬ (code=CONFACK)	code=CONFACK	¬ (code=CONFACK)	code=CONFACK	¬ (code=CONFACK)	code=CONFACK	¬ (code=CONFACK)
state' =	ACKSENT	REQSENT	OPENED	REQSENT	ACKSENT	REQSENT	ACKSENT	REQSENT
callbacks' =	none	none	up	none	none	none	addci	addci
timer' =	none	none	stop	none	none	none	start	start
flags' =	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =	CONFACK	CONFREQ	CONFACK	CONFREQ	CONFACK	CONFREQ	CONFACK	CONFREQ
reqid' =	id	id	id	id	id	id	id	id
timeouttime' =	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	maxconfreqtransmits	maxconfreqtransmits
PROTO_NAME' =	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =	inp	inp	inp	inp	inp	inp	inp	inp
datalen' =	0	len	0	len	0	len	0	len

Program

```
/*
 * fsm_rconfreq - Receive Configure-Request.
 */
static void
fsm_rconfreq(f, id, inp, len)
    fsm *f;
    u_char id;
    u_char *inp;
    int len;
{
    int code, reject_if_disagree;

    switch( f->state ){
    case CLOSED:
        /* Go away, we're closed */
        fsm_sdata(f, TERMACK, id, NULL, 0);
        return;
    case CLOSING:
    case STOPPING:
        return;

    case OPENED:
        /* Go down and restart negotiation */
        if( f->callbacks->down )
            (*f->callbacks->down)(f);  /* Inform upper layers */
        fsm_sconfreq(f, 0);            /* Send initial Configure-Request
*/
        break;

    case STOPPED:
        /* Negotiation started by our peer */
        fsm_sconfreq(f, 0);            /* Send initial Configure-Request
*/
        f->state = REQSENT;
        break;
    }

    /*
     * Pass the requested configuration options
     * to protocol-specific code for checking.
     */
    if (f->callbacks->reqci){          /* Check CI */
```

```
        reject_if_disagree = (f->nakloops >= f->maxnakloops);
        code = (*f->callbacks->reqci)(f, inp, &len, reject_if_disagree);
    } else if (len)
        code = CONFREJ;                 /* Reject all CI */
    else
        code = CONFACK;

    /* send the Ack, Nak or Rej to the peer */
    fsm_sdata(f, code, id, inp, len);

    if (code == CONFACK) {
        if (f->state == ACKRCVD) {
            UNTIMEOUT(fsm_timeout, f); /* Cancel timeout */
            f->state = OPENED;
            if (f->callbacks->up)
                (*f->callbacks->up)(f);        /* Inform upper layers */
        } else
            f->state = ACKSENT;
        f->nakloops = 0;

    } else {
        /* we sent CONFACK or CONFREJ */
        if (f->state != ACKRCVD)
            f->state = REQSENT;
        if( code == CONFNAK )
            ++f->nakloops;
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)					
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK	
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK	
id' =	reqid	reqid+1	' id	' id	
data' =	outp	term_reason	NULL	inp	
datalen' =	cilen	term_reason_len	0	len	
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf	
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	

On Display 14

UNTIMEOUT (fsm_timeout, f)					
R1 (,) = (timer = stop)					

On Display 16

End of Display 7

Display 8

Specification

fsm_rconfack (fsm * f, int **id**, u_char * **inp**, int **len**) | int state,int flags,uchar id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			CLOSED	STOPPED	CLOSING	STOPPING	ACKRCVD	REQSENT	OPENED	REQSENT
callbacks' =			none	none	none	none	none	addci	up	addci
timer' =			none	none	none	none	none	start	stop	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			TERMACK	TERMACK	' code	' code	' code	CONFREQ	' code	CONFREQ
reqid' =			id	id	' reqid	' reqid	' reqid	id	' reqid	id
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			NULL	NULL	' data	' data	' data	outp	' data	outp
datalen' =			0	0	' datalen	' datalen	' datalen	cilen	' datalen	cilen

Program

```
/*
 * fsm_rconfack - Receive Configure-Ack.
 */
static void
fsm_rconfack(f, id, inp, len)
    fsm *f;
    int id;
    u_char *inp;
    int len;
{


    f->seen_ack = 1;

    switch (f->state) {
    case CLOSED:
    case STOPPED:
        fsm_sdata(f, TERMACK, id, NULL, 0);
        break;

    case REQSENT:
        f->state = ACKRCVD;
        f->retransmits = f->maxconfreqtransmits;
        break;

    case ACKRCVD:
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        fsm_sconfreq(f, 0);
        f->state = REQSENT;
        break;

    case ACKSENT:
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        f->state = OPENED;
        f->retransmits = f->maxconfreqtransmits;
        if (f->callbacks->up)
            (*f->callbacks->up)(f);    /* Inform upper layers */
        break;

    case OPENED:
        /* Go down and restart negotiation */
        if (f->callbacks->down)
```

```
            (*f->callbacks->down)(f);  /* Inform upper layers */
        fsm_sconfreq(f, 0);            /* Send initial Configure-Request
*/
        f->state = REQSENT;
        break;
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)				
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK
id' =	reqid	reqid+1	' id	' id
data' =	outp	term_reason	NULL	inp
datalen' =	cilen	term_reason_len	0	len
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN

On Display 14

UNTIMEOUT (fsm_timeout, f)
R1 (,) = (timer = stop)

On Display 16

End of Display 8

Display 9

Specification

fsm_rconfnakrej (fsm * f ; int code ; int **id** ; u_char * **inp**; int **len**) | int state,int flags,u_char id,uchar reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT ∧		' state = ACKRCVD	' state = ACKSENT ∧		' state = OPENED
							(ret < 0)	(ret > = 0)		(ret < 0)	(ret > = 0)	
state' =			CLOSED	STOPPED	CLOSING	STOPPING	STOPPED	REQSENT	REQSENT	STOPPED	ACKSENT	REQSENT
callbacks' =			none	none	none	none	none	addci	addci	none	addci	addci
timer' =			none	none	none	none	none	start	start	stop	start	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			TERMACK	TERMACK	' code	' code	' code	CONFREQ	CONFREQ	' code	CONFREQ	CONFREQ
reqid' =			id	id	' reqid	' reqid	' reqid	id	id	' reqid	id	id
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	maxconfretransmits	maxconfretransmits	' retransmits	maxconfretransmits	maxconfretransmits
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			NULL	NULL	' data	' data	' data	outp	outp	' data	outp	outp
datalen' =			0	0	' datalen	' datalen	' datalen	cilen	cilen	' datalen	cilen	cilen

Program

```
/*
 * fsm_rconfnakrej - Receive Configure-Nak or Configure-Reject.
 */
static void
fsm_rconfnakrej(f, code, id, inp, len)
    fsm *f;
    int code, id;
    u_char *inp;
    int len;
{

    int ret;

    f->seen_ack = 1;

    switch (f->state) {
    case CLOSED:
    case STOPPED:
        fsm_sdata(f, TERMACK, id, NULL, 0);
        break;

    case REQSENT:
    case ACKSENT:
               UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        if (ret < 0)
            f->state = STOPPED;                /* kludge for stopping CCP
*/
        else
            fsm_sconfreq(f, 0);                /* Send Configure-Request
*/
        break;

    case ACKRCVD:
               UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        fsm_sconfreq(f, 0);
        f->state = REQSENT;
        break;

    case OPENED:
        /* Go down and restart negotiation */
```

```
        if (f->callbacks->down)
            (*f->callbacks->down)(f);  /* Inform upper layers */
        fsm_sconfreq(f, 0);            /* Send initial Configure-Request
*/
        f->state = REQSENT;
        break;
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)		u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen					
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

UNTIMEOUT (fsm_timeout, f)					
R1 (,) = (timer = stop)					

On Display 16

End of Display 9

Display 10

Specification

void fsm_rtermreq (fsm * f ; int **id** ; u_char * **p** ; int **len** ;) | int state,int flags,uchar id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			CLOSED	STOPPED	CLOSING	STOPPING	REQSENT	REQSENT	REQSENT	STOPPING
callbacks' =			none	none	none	none	none	none	none	down
timer' =			none	none	none	none	none	none	none	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			TERMACK	TERMACK	TERMACK	TERMACK	TERMACK	TERMACK	TERMACK	TERMACK
reqid' =			id	id	id	id	id	id	id	id
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	0
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
datalen' =			0	0	0	0	0	0	0	0

Program

```
/*
 * fsm_rtermreq - Receive Terminate-Req.
 */
static void
fsm_rtermreq(f, id, p, len)
    fsm *f;
    int id;
    u_char *p;
    int len;
{
    switch (f->state) {
    case ACKRCVD:
    case ACKSENT:
        f->state = REQSENT;                    break;

    case OPENED:
        if (len > 0) {
            info("%s terminated by peer (%0.*v)", PROTO_NAME(f), len,
p);
        } else
            info("%s terminated by peer", PROTO_NAME(f));
        if (f->callbacks->down)
            (*f->callbacks->down)(f);  /* Inform upper layers */
        f->retransmits = 0;
        f->state = STOPPING;
        TIMEOUT(fsm_timeout, f, f->timeouttime);
        break;
    }

    fsm_sdata(f, TERMACK, id, NULL, 0);
}
```

Specification of Invoked programs

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)					
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK	
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK	
id' =	reqid	reqid+1	' id	' id	
data' =	outp	term_reason	NULL	inp	
datalen' =	cilen	term_reason_len	0	len	
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf	
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	

On Display 14

TIMEOUT (fsm_timeout, f , timeouttime)
R1 (,) = (timer = start) AND (timeouttime' = DEFTIMEOUT)

On Display 15

End of Display 10

Display 11

Specification

void fsm_rtermack (fsm * f)

int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			CLOSED	STOPPED	CLOSING	STOPPED	REQSENT	REQSENT	ACKSENT	REQSENT
callbacks' =			none	none	finished	finished	none	none	none	addci
timer' =			none	none	stop	stop	none	none	none	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			' code	' code	' code	' code	' code	' code	' code	CONFREQ
reqid' =			' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	id
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	maxconfreqtransmits
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			' data	' data	' data	' data	' data	' data	' data	outp
datalen' =			' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	cilen

Program

```
/*
 * fsm_rtermack - Receive Terminate-Ack.
 */
static void
fsm_rtermack(f)
    fsm *f;
{
    switch (f->state) {
    case CLOSING:
        UNTIMEOUT(fsm_timeout, f);
        f->state = CLOSED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;
    case STOPPING:
        UNTIMEOUT(fsm_timeout, f);
        f->state = STOPPED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;

    case ACKRCVD:
        f->state = REQSENT;
        break;

    case OPENED:
        if (f->callbacks->down)
            (*f->callbacks->down)(f);  /* Inform upper layers */
        fsm_sconfreq(f, 0);
        break;
    }
}
```

Specification of Invoked programs

fsm_sconfreq(f,retransmit)		u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen					
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

On Display 13

UNTIMEOUT (fsm_timeout, f)					
R1 (,) = (timer = stop)					

On Display 16

End of Display 11

Display 12

Specification

void fsm_protreject (fsm * f)

int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING	' state = STOPPING	' state = REQSENT	' state = ACKRCVD	' state = ACKSENT	' state = OPENED
state' =			CLOSED	STOPPED	CLOSED	STOPPED	STOPPED	STOPPED	STOPPED	REQSENT
callbacks' =			finished	finished	finished	finished	finished	finished	finished	down
timer' =			none	none	stop	stop	stop	stop	stop	start
flags' =			' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags
code' =			' code	' code	' code	' code	' code	' code	' code	TERMREQ
reqid' =			' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	' reqid	id + 1
timeouttime' =			' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	' timeouttime	DEFTIMEOUT
retransmits' =			' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	' retransmits	maxtermtransmits
PROTO_NAME' =	" MISTAKE "	" MISTAKE "	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME
data' =			' data	' data	' data	' data	' data	' data	' data	term_reason
datalen' =			' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	' datalen	term_reason_len

Program

```
* fsm_protreject
 *
 * Treat this as a catastrophic error (RXJ-).
 */
void
fsm_protreject(f)
    fsm *f;
{
    switch( f->state ){
    case CLOSING:
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        /* fall through */
    case CLOSED:
        f->state = CLOSED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;

    case STOPPING:
    case REQSENT:
    case ACKRCVD:
    case ACKSENT:
        UNTIMEOUT(fsm_timeout, f);     /* Cancel timeout */
        /* fall through */
    case STOPPED:
        f->state = STOPPED;
        if( f->callbacks->finished )
            (*f->callbacks->finished)(f);
        break;

    case OPENED:
        if( f->callbacks->down )
            (*f->callbacks->down)(f);

        /* Init restart counter, send Terminate-Request */
        f->retransmits = f->maxtermtransmits;
        fsm_sdata(f, TERMREQ, f->reqid = ++f->id,
                  (u_char *) f->term_reason, f->term_reason_len);
        TIMEOUT(fsm_timeout, f, f->timeouttime);
```

```
            --f->retransmits;

            f->state = STOPPING;
            break;

        default:
            FSMDEBUG(("%s: Protocol-reject event in state %d!",
                      PROTO_NAME(f), f->state));
        }
}
```

Specification of Invoked programs

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)				
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK
id' =	reqid	reqid+1	' id	' id
data' =	outp	term_reason	NULL	inp
datalen' =	cilen	term_reason_len	0	len
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN

On Display 14

TIMEOUT (fsm_timeout, f , timeouttime)
R1 (,) = (timer = start) AND (timeouttime' = DEFTIMEOUT)

On Display 15

UNTIMEOUT (fsm_timeout, f)
R1 (,) = (timer = stop)

On Display 16

FSMDEBUG(PROTO_NAME, state)
R2 (,) = (PROTO_NAME = "MISTAKE")

On Display 17

End of Display 12

Display 13

Specification

fsm_sconfreq(f,retransmit)	u_char callbacks,int code,u_char reqid,int timeouttime,int retransmits,u_char data,int datalen						
	state=STARTING	state = CLOSED	state =STOPPED	state = REQSENT	state = ACKRCVD	state=ACKSENT	state=OPENED
callbacks' =	addci	addci	addci	addci	addci	addci	addci
timer' =	start	start	start	start	start	start	start
code' =	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ	CONFREQ
reqid' =	id	id	id	id	id	id	id
timeouttime' =	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT	DEFTIMEOUT
retransmits' =	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits	maxconfretransmits
data' =	outp	outp	outp	outp	outp	outp	outp
datalen' =	cilen	cilen	cilen	cilen	cilen	cilen	cilen

From Display 2, 4, 6, 7, 8, 9, 11

Program

```
/*
 * fsm_sconfreq - Send a Configure-Request.
 */
static void
fsm_sconfreq(f, retransmit)
    fsm *f;
    int retransmit;
{
    u_char *outp;
    int cilen;

        if( !retransmit ){
        /* New request - reset retransmission counter, use new ID */
        f->retransmits = f->maxconfreqtransmits;
        f->reqid = ++f->id;
    }

    /*
     * Make up the request packet
     */
        if (f->callbacks->addci)
            (*f->callbacks->addci);

    /* send the request to our peer */
    fsm_sdata(f, CONFREQ, f->reqid, outp, cilen);

    /* start the retransmit timer */
    --f->retransmits;
    TIMEOUT(fsm_timeout, f, f->timeouttime);
}
```

Specification of Invoked programs

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)				
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK
id' =	reqid	reqid+1	' id	' id
data' =	outp	term_reason	NULL	inp
datalen' =	cilen	term_reason_len	0	len
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN

On Display 14

TIMEOUT (fsm_timeout, f , timeouttime)
R1 (,) = (timer = start) AND (timeouttime' = DEFTIMEOUT)

On Display 15

End of Display 13

Display 14

Specification

fsm_sdata (fsm*f; u_char **code, id**;u_char *data;int datalen)				
	' code= CONFREQ	' code= TERMREQ	' code= TERMACK	' code= CONFNAK
code' =	CONFREQ	TERMREQ	TERMACK	CONFNAK
id' =	reqid	reqid+1	' id	' id
data' =	outp	term_reason	NULL	inp
datalen' =	cilen	term_reason_len	0	len
outp' =	outpacket_buf	outpacket_buf	outpacket_buf	outpacket_buf
outlen' =	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN	datalen+HEADERLEN

From Display 6, 7, 8, 10, 12, 13

Program

```
/*
 * fsm_sdata - Send some data.
 *
 * Used for all packets sent to our peer by this module.
 */
void
fsm_sdata(f, code, id, data, datalen)
    fsm *f;
    u_char code, id;
    u_char *data;
    int datalen;
{
        /* Adjust length to be smaller than MTU */
    outp = outpacket_buf;

    outlen = datalen + HEADERLEN;

}
% NOTE: In the PPP code, the function fsm_sdata is a general function
%that
% is used to send data, in particular the various requests and
% acknowlegements that are required by the protocol actions, e.g:

% fsm_sdata(f, TERMREQ, f->reqid = ++ f->id,
%          (u_char *) f->term_reason, f->term_reason_len);

% implements the requirements action:

%       str = Send-Terminate-Request

% Similarly action scr,sca,scn,sta are implemented by fsm_sdata.
```

Specification of Invoked programs: Empty

End of Display 14

Display 15

Specification

TIMEOUT (fsm_timeout, f , timeouttime)
R1 (,) = (timer = start) AND (timeouttime' = DEFTIMEOUT)

Program

```
TIMEOUT(fsm_timeout, f, f->timeouttime);
{
   f->timeouttime->DEFTIMEOUT;
   f->timer->start;
   fsm_timeout(f);
}
```

Specification of Invoked programs

fsm_timeout (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING ∧		' state = STOPPING ∧		' state = REQSENT ∧		' state = ACKRCVD ∧		' state = ACKSENT ∧		' state = OPENED
					retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	
state' =					CLOSED	CLOSING	STOPPED	STOPPING	STOPPED	REQSENT	STOPPED	REQSENT	STOPPED	ACKSENT	
callbacks' =					finished	none	finished	none	finished	addci	finished	addci	finished	addci	
timer' =					none	start	none	start	none	start	none	start	none	start	
flags' =					' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	
code' =					' code	TERMREQ	' code	TERMREQ	' code	CONFREQ	' code	CONFREQ	' code	CONFREQ	
reqid' =					' reqid	id +1	' reqid	id +1	' reqid	id	' reqid	id	' reqid	id	
timeouttime' =					' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	
retransmits' =					retransmits <=0	retransmits > 0	retransmits <=0	retransmits > 0	retransmits <=0	maxconfretransmits	retransmits <=0	maxconfretransmits	retransmits <=0	maxconfretransmits	
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	"MISTAKE"
data' =					' data	term_reason	' data	term_reason	' data	outp	' data	outp	' data	outp	
datalen' =					' datalen	term_reason_len	' datalen	term_reason_len	' datalen	cilen	' datalen	cilen	' datalen	cilen	

On Display 6

End of Display 15

Display 16

Specification

UNTIMEOUT (fsm_timeout, f)
R1 (,) = (timer = stop)

Program

```
UNTIMEOUT(fsm_timeout, f);
{
   f->timer->stop;
   fsm_timeout(f);
}
```

Specification of Invoked programs

fsm_timeout (fsm * f)	int state,int flags,u_char id,u_char reqid,int timeouttime,int maxconfreqtransmits,int retransmits,int code,u_char callbacks,u_char data,int datalen,u_char PROTO_NAME

	' state = INITIAL	' state = STARTING	' state = CLOSED	' state = STOPPED	' state = CLOSING ∧		' state = STOPPING ∧		' state = REQSENT ∧		' state = ACKRCVD ∧		' state = ACKSENT ∧		' state = OPENED
					retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	retransmits <=0	¬ (retransmits <= 0)	
state' =					CLOSED	CLOSING	STOPPED	STOPPING	STOPPED	REQSENT	STOPPED	REQSENT	STOPPED	ACKSENT	
callbacks' =					finished	none	finished	none	finished	addci	finished	addci	finished	addci	
timer' =					none	start	none	start	none	start	none	start	none	start	
flags' =					' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	' flags	
code' =					' code	TERMREQ	' code	TERMREQ	' code	CONFREQ	' code	CONFREQ	' code	CONFREQ	
reqid' =					' reqid	id +1	' reqid	id +1	' reqid	id	' reqid	id	' reqid	id	
timeouttime' =					' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	' timeouttime	DEFTIMEOUT	
retransmits' =					retransmits <=0	retransmits > 0	retransmits <=0	retransmits > 0	retransmits <=0	maxconfretransmits	retransmits <=0	maxconfretransmits	retransmits <=0	maxconfretransmits	
PROTO_NAME' =	"MISTAKE"	"MISTAKE"	"MISTAKE"	"MISTAKE"	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	' PROTO_NAME	"MISTAKE"
data' =					' data	term_reason	' data	term_reason	' data	outp	' data	outp	' data	outp	
datalen' =					' datalen	term_reason_len	' datalen	term_reason_len	' datalen	cilen	' datalen	cilen	' datalen	cilen	

On Display 6

End of Display 16

Display 17

Specification

FSMDEBUG(PROTO_NAME, state)					
R2 (,) = (PROTO_NAME = "MISTAKE")					

Program

```
FSMDEBUG(("%s: Mistake in state %d!",PROTO_NAME(f), f->state));
```

Specification of Invoked Programs: Empty