AN ESTELLE.Z PARSER FOR A PROTOCOL TEST GENERATION ENVIRONMENT TESTGEN+

By

Rui Zhang

B. Eng. (Computer Science) Xi'an Jiaotong University

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1996

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Computer Science
The University of British Columbia
2366 Main Mall
Vancouver, Canada
V6T 1Z4

Date: Aug. 20, 1996

Abstract

Protocol testing is an indispensable constituent in protocol development. The test suite generation and selection processes are usually tedious and time-consuming. It is difficult to manually generate and select test suites without errors. Therefore, an automatic protocol test suite generation and selection environment is required. For this purpose, the protocol test suite generation and selection environment, TESTGEN+, was developed. TESTGEN+ includes the TESTGEN, TESTSEL and TESTVAL tools for test suite generation, selection, and validation, respectively.

In this thesis, a new front end of the TESTGEN tool, an Estelle.Z parser which is for an Estelle-like protocol specification language, is designed and implemented. A real world protocol, home agent – a major component of the Mobile IP protocol, is specified in Estelle.Z. The specification is fed into the TESTGEN tool to yield a test suite, which is then fed into the TESTSEL tool to get a subset of an efficient test suite with user satisfactory coverage and cost. In order to evaluate the correctness of the parser, the same protocol is specified in Estelle.Y+ASN.1 and fed through a different parser into TESTGEN and TESTSEL generating a test suite. Comparing the two test suites, results from the experiment indicate: a) the Estelle.Z parser works with the TESTGEN tool well; b) TESTGEN and TESTSEL are competent tools to generate and select test suites for protocol specifications; c) furthermore, there is room to improve the TESTGEN and the TESTSEL tools.

Table of Contents

List of Tables

List of Figures

Acknowledgment

First of all, I would like to express my sincerest appreciation to my supervisor, Dr. Son Vuong, for his constant encouragement and invaluable guidance throughout my research work, which made it possible for me to complete this thesis on time. His enthusiasm has made the past years an enjoyable and unforgettable period.

I would like to thank Dr. Norm Hutchinson for his helpful comments and careful reading of the final draft despite his very busy schedule.

I wish to thank Dr. ir. Rolf J. Velthuys and Mr. Binh Do for their helpful suggestions and productive discussions, and Ms. Andrea Warrington who read the thesis draft and made very good suggestions to improve the thesis.

I wish to acknowledge the Computer Science department at the University of British Columbia for providing an environment suitable for my research and for providing financial support. The financial support in the form of RA from the joint grant of Motorola and NSERC/IOR is also gratefully acknowledged.

I wish to thank my relatives and friends who gave me constant encouragement and moral support during my studies in Canada.

I wish to express my gratitude and appreciation to my husband Yidong and my daughter Jenny for their unceasing support and understanding. Without their patience and support, this thesis could not have been completed.

Chapter 1

Introduction

1.1 Thesis Motivation and Contributions

In general, a communication protocol is quite complex and takes a considerable effort to move from a standard protocol specification to an implementation on a real system. The protocol standard may lead to several different implementations. The testing of each protocol implementation for conformance to the specification of the protocol standard becomes critical. The complexity of protocols necessitates the use of automated tools to test the protocol implementation. For this purpose, the TESTGEN+ environment was developed to generate test suites based on the protocol specifications in the Estelle.Y+ASN.1 language. In TESTGEN+, there are three major modules: TESTGEN, TESTSEL and TESTVAL for test suite generation, test suite selection and test suite verification respectively. TESTGEN+ accepts Estelle.Y+ASN.1 specifications and yields test suites.

The Estelle language is one of the FDTs (Formal Description Techniques) based on an extended state transition model. A subset of Estelle defined as Estelle.Z is employed in TESTGEN+, because the complete Estelle language is too complex and the TESTGEN kernel does not support it.

In this thesis, the research objectives are:

- To design and develop the front end of the testing environment TESTGEN+ for

Estelle.Z.

- To demonstrate the usefulness of the tools, TESTGEN and TESTSEL, by applying it to a real world protocol and generating a test suite for the protocol.
- To prove the parser works correctly by applying the real world protocol specification to it.

The front end of the testing environment is the essential constituent of the whole test environment. It analyzes the protocol specification, which is the main information given by the user. In addition, it provides the internal representation of the protocol in which the external behaviors of a protocol are precisely and completely described. The internal representation of protocol must be accessible to the TESTGEN engine, which is the key process of the whole testing generation environment.

The Mobile IP protocol is a well-known real world protocol. In this thesis, it is employed to test the new parser and the TESTGEN+ tool. The home agent, a major component of Mobile IP, is specified in both Estelle.Z and Estelle.Y+ASN.1 languages. These specifications are fed into the TESTGEN+ tool to yield two sets of test suites. Results from the experiment indicate that the parser works with the TESTGEN engine. They also indicate that while the TESTGEN+ environment is an efficient tool, it can still be improved further.

1.2 Thesis Outline

The rest of this thesis is organized as follows. In chapter 2, an overview of the TESTGEN+ environment is introduced. An introduction to each main module in TESTGEN+ and the architecture of TESTGEN+ are presented.

In chapter 3, the design and implementation of the front end of TESTGEN are discussed. The definition and function of the Estelle.Z language are introduced, the object data structure of the internal representation of protocol is described, and then the implementation of the Estelle.Z parser is presented.

In chapter 4, a review of a real world protocol, home agent – a component of Mobile IP, is given first. The protocol is specified in Estelle.Y for the control component and specified in ASN.1 for the data component. Also the protocol specification in Estelle.Z is presented.

In chapter 5, The applications of TESTGEN+ to both the specifications given in chapter 4 are discussed, and the results are analyzed. At the end, the strong points and weak points of TESTGEN and TESTSEL are discussed.

In chapter 6, Some conclusions are drawn and future improvement work on the parser, TESTGEN and TESTSEL is discussed.

The appendices contain listings of files used in the thesis. The source codes for the specification in ASN.1 and Estelle.Y are shown in Appendix A and B respectively. The source code for the specification in Estelle.Z is shown in Appendix C. The complete home agent message parameters specification in ASN.1 is shown in Appendix D. The selected control sequences and the subset of selected test cases are shown in Appendix E. The test cases in TTCN (Tree and Tabular Combined Notation) form are shown in Appendix F.

Chapter 2

TESTGEN+ Environment

Introduction

Protocol testing is an essential phase in the protocol development process. To ensure that each protocol implementation is able to inter operate with other implementations correctly, conformance testing is applied to each implementation.

There are many well known test methods which have been applied to protocol conformance testing with varying degrees of success, such as U-, D-, W-methods [SD88, Gon70, Cho78]. Even though these methods have been improved and optimized [CVI89, VCI89], they still have a major shortcoming: they only address the control part of the protocols. The TESTGEN environment overcomes this weak point by combining control flow testing and data flow testing in test suite generation. This has been applied to some practical protocols successfully, such as the InRes, OSI class transport and LAPB protocols.

The protocol TEST suite Generation, selection, and validation ENvironment for conformance testing (**TESTGEN+**) was developed at the University of British Columbia [Vuo93]. The overall functional structure of TESTGEN+ is described in Figure 2.1. There are three major modules: TESTGEN, TESTSEL and TESTVAL for test suite generation, selection and validation respectively. *TESTGEN* accepts protocol specifications in a combination of Estelle.Y and ASN.1 from the user. The specifications are converted to an internal data structure , Protocol Data Structure (PDS), by the parsers.

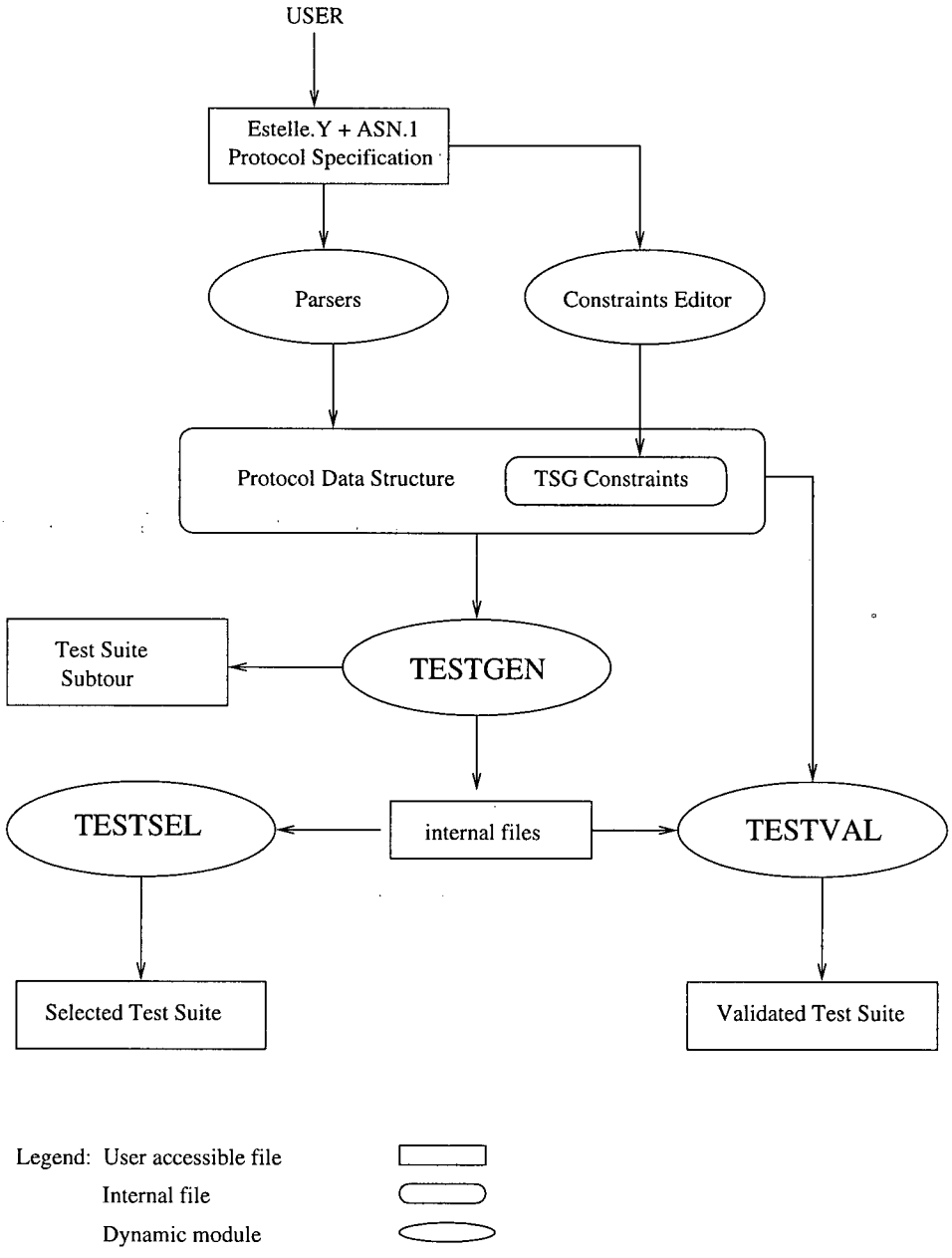


Figure 2.1: TESTGEN+ Environment

It also accepts a set of user-defined constraints (TSG constraints) through the constraints editor. The TESTGEN tool obtains the protocol knowledge and the TSG constraints from the PDS, generates a test suite, and outputs the test suite in the subtour form. The test suite can be fed into the *TESTSEL* tool to get a refined test suite which has the user acceptable cost and satisfied fault coverage. The test suite can also be fed into the *TESTVAL* tool to check the validation of the test suite with respect to the given specifications.

2.1 Test Suite Generation – TESTGEN

TESTGEN is a major module in TESTGEN+. The function of the TESTGEN is to generate test suites according to protocol specifications. This module accepts protocol specifications in ASN.1 (for data), protocol specifications in Estelle.Y (for control), and constraints, which can easily be edited by the user. It also generates test suite in subtour, and two internal files for TESTVAL and TESTSEL respectively. The overall structure of TESTGEN is depicted in Figure 2.2. Within the TESTGEN module, there are five modules: Estelle.Y parser, ASN.1 parser, constraints editor, test suite generation engine and output module. In addition, there is one internal data structure, the PDS which contains all information obtained from both specifications. The modules' functions are given as follows:

- Estelle.Y parser: This is a parser for the Estelle.Y language which is a modified subset of Estelle language, and is used to describe the control part of protocol specifications. The parser object is an internal data structure the PDS.
- ASN.1 parser: This is a parser for the ASN.1 language which is used to describe the data part of protocol specifications. The parser object is an internal data structure

which is part of the PDS.

- Constraints editor: The constraints editor module provides the user with an interactive interface for definition of the TESTGEN constraints. In TESTGEN, constraints are a set of boolean predicates that are used for the generation of each subtour. There are two types of constraints: (1) the maximum and the minimum usage conditions for all states, transitions, ISPs (Input Service Primitive), OSPs (Output Service Primitive), PDUs (Protocol Data Unit), constants, variables, and

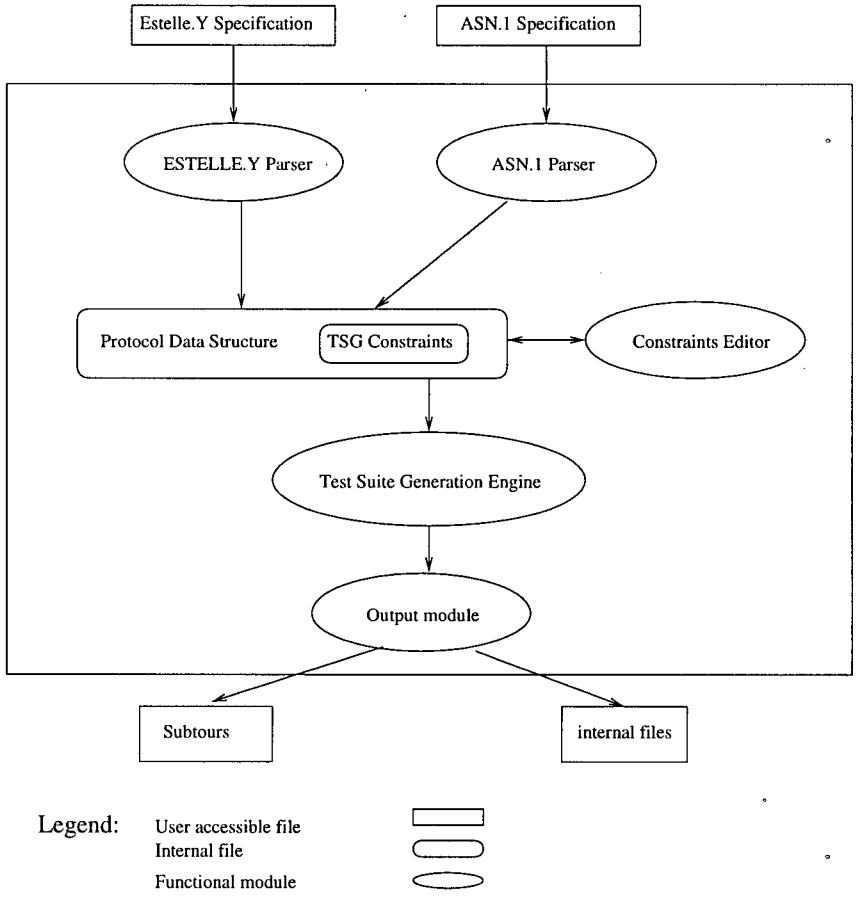


Figure 2.2: TESTGEN Structure

timers. (2) the parameter variations which include up to 10 values for each parameter of each ISP or PDU. The user can modify all of these constraints, save and restore the values, and set the default values through the editor.

- Test suite generation engine: This module generates test suites based on the given specifications and a set of constraints. The test suite is divided into test cases which are subtours in TESTGEN. The subtour identification algorithm performs an exhaustive backtracking depth first search algorithm. It operates over the behavior tree representation of the protocol. Each subtour starts and ends at the same state, the initial state, and satisfies all the constraints conditions. These constraints limit the length of each subtour. According to the protocol specification, only a finite number of the transitions can be applied in each state. The parameter variation constraints on the parameters of the ISP and PDU limit the number of different instances of the ISP and PDU in the transition. Thus, the length and the number of different subtours are limited if the backtracking algorithm is to finish successfully. Each subtour is recorded and saved into files through the output module.
- Output module: This module is to save the test cases from the test suite generation engine into files. The test cases are in subtour form, and two internal files used by the TESTVAL and the TESTSEL respectively.

2.2 Test Suite Selection – TESTSEL

The function of TESTSEL is to select test cases from the test suite output by TESTGEN according to the user's requirement, such as cost and coverage tolerance, i.e., the number of test cases, and the distance between the test cases. This tool can accept test cases generated by TESTGEN or other tools in proper format which TESTGEN can recognize,

and produce a subset of the test cases which satisfies the user.

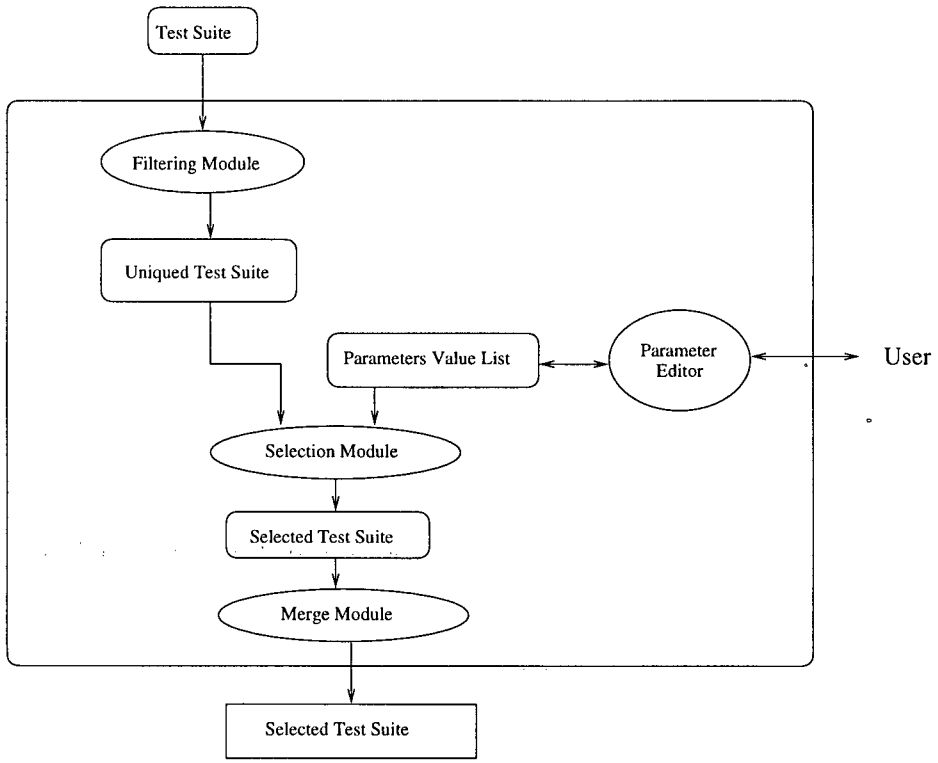


Figure 2.3: TESTSEL Structure

The overall structure of the TESTSEL is in Figure 2.3. Within the TESTSEL module, there are three functional modules: filtering module, selection module and merging module. Their functions are as follows:

- Filtering module: This module strips off the data flow information from the test cases, leaving the control flow information in the test cases as the output of filtering module. Complete test cases are composed of both control and data components. Test cases are filtered before being fed into the selection module because the selection module selects test cases based solely on the control component and not on both components.

- Selection module: This module selects test cases from the internal test cases which are the output of the filtering module and produces a subset of test cases by using the user defined constraints. These constraints are (1) the testing distance definition parameters p_k and r_k, (2) the test selection constraints including the maximum cost, the initial coverage tolerance ϵ and minimum coverage tolerance ϵ_{min} which is the test selection radius, (3) the radius scale factor for each pass. By using a multi-pass algorithm, this module selects test cases to maximize coverage subject to the cost constraint. It has been proven that all Cauchy sequences converge in the metric space and repeated application of the greedy algorithm with successively narrower test case radii produces such Cauchy sequences, thus, the greedy approach yields a set of test cases with general coverage of the test suite and guarantees convergence to the test suite. The test selection algorithm is guaranteed to yield a set of test cases which converge to the initial test suite as more test cases are selected. This guarantees that no specific test cases are overlooked. A detailed algorithm can be found in [MVC92] .

- Merging module: This module accepts the subset test cases from the output of the selection module and merges the control component and the data component to form the final test cases. The selection module only processes the control components of test cases, hence only control information in the output test cases is available. The data component of the test cases should be inserted to the control component to get the complete test cases for the user.

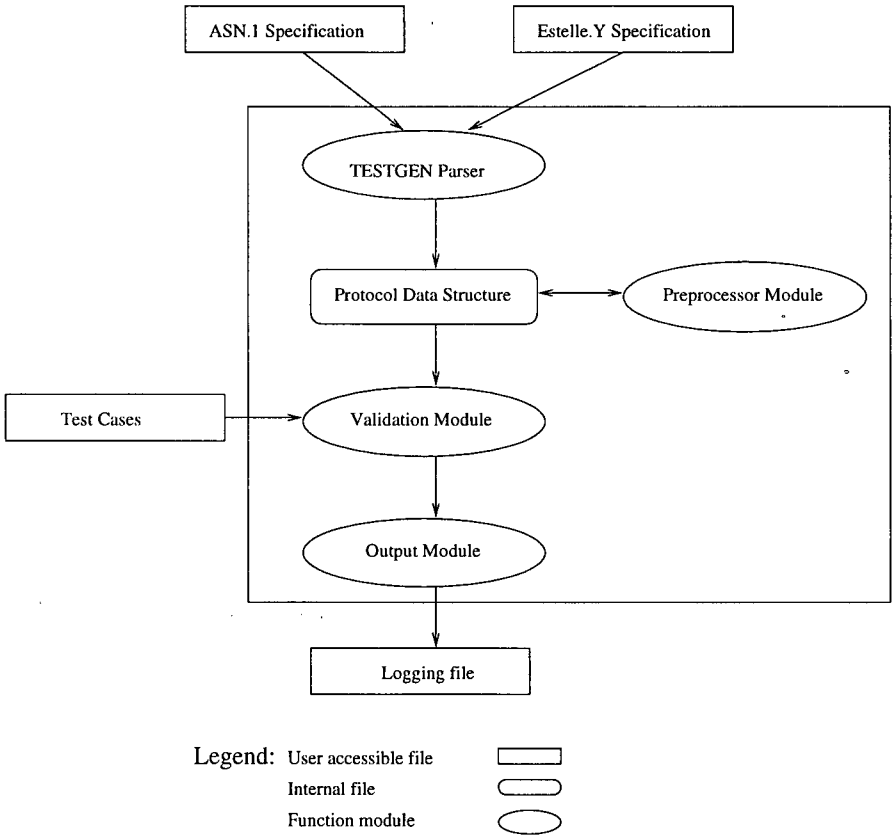


Figure 2.4: TESTVAL Structure

2.3 Test Suite Validation – TESTVAL

The function of TESTVAL is to check whether a given test suite is valid with respect to a given Estelle.Y and ASN.1 specification. Like TESTGEN, TESTSEL also accepts ASN.1 specification and Estelle.Y specification, and gives out a logging file which contains the position of an event which is in error or an indication the the test case is valid. The overall structure of TESTVAL is given in Figure 2.4. There are three functional modules: preprocessor module, validation module and output module. Their functions are as follows:

- Preprocessor module: To process the test suite validation quickly and efficiently, this module converts some types of transitions in the specification into a more suitable form for the validation module. It also translates the EFSM form of an Estelle.Y specification into a simple FSM form.
- Validation module: This module validates given test cases with respect to the PDS of the protocol specifications.
- Output module: This module logs the traces of states and transitions which satisfy the given test cases in a file. If a test case is valid according to the given formal specifications, a message - "test case valid" is in the logging file. If a test case is invalid, the position is logged in the file. By using the information in the logging file, errors in test cases may be located.

Chapter 3

Estelle.Z Parser for TESTGEN

3.1 Introduction

The TESTGEN tool accepts protocol specifications in Estelle.Y for the control component and specifications in ASN.1 for the data component of the protocol through two parsers. Hence, users need to learn both the ASN.1 and Estelle.Y languages. On the other hand, Estelle.Y is only a small subset of the Estelle language, and is not powerful enough to specify a real world protocol. Furthermore, the Estelle language is one of the most popular specification languages in the real world. (Estelle, SDL and LOTOS are three common specification languages.) Therefore, it is necessary to expand Estelle.Y+ASN.1 into a large subset of Estelle or even to the whole Estelle language. As a result of the TESTGEN kernel accepting only the functions in Estelle.Y and ASN.1, we have to define the Estelle.Z language which supports all functions in Estelle.Y+ASN.1, and can easily be expanded into the entire Estelle language.

The output of the TESTGEN tool is a set of subtours, which is easy to read for people, but not easily to be recognized by computers. Hence, the TTCN (Tree and Tabular Combined Notation) format is adapted for the TESTGEN output. Also a graphic user interface is implemented.

In this chapter, a description of the Estelle.Z language is presented briefly, followed by a discussion on the implementation of the Estelle.Z parser. At the end, the improvement

of the interface of the TESTGEN tool is discussed.

3.2 Estelle.Z Language

Estelle.Z is a formal language defined to specify protocols for automatic test suite generation. An Estelle.Z specification is a modified, single module Estelle specification enhanced by introducing explicit language support for timers. A subset of the Estelle language is defined as Estelle.Z. The design goal is (1) to be easily expanded to the whole Estelle language; (2) to get an exact or very similar object data structure the PDS so that the TESTGEN kernel can recognize the information in the PDS; (3) to follow the syntax of Estelle as closely as possible.

The advantage of Estelle.Z is (1) users can specify protocols in one language rather than two languages; (2) the syntax of Estelle.Z is more closer to Estelle than Estelle.Y; (3) the parser can be very easily expanded to complete Estelle language.

Syntax and Semantics

The syntax of Estelle.Z is defined in Backus-Naur Form (BNF) notation. An Estelle.Z specification is a single module definition. It contains three components: declaration, initialization and transition. Variables and constants used in the specification are defined in declaration component; the protocol state machine is initialized in the initialization component; the transitions of the protocol state machine are defined in the transition component.

Declaration The declaration component contains constant declarations, variable declarations, channel declarations, timer declarations and state declarations.

The syntax of variable and constant declarations is similar to that in Pascal.

Estelle.Z language supports five data types for variables and constants: integer, boolean, character string, enumeration and record. Input Service Primitives (ISPs), Output Service Primitives (OSPs) and Protocol Data Structures (PDUs) are defined in the channel declaration, these are necessary for the TESTGEN kernel. Timeout values are declared for timers. All states in the protocol machine are defined in the state declaration. Figure 3.1 is an example of the declaration component.

```
Specification HomeAgent ;
CONST   mipMHauthExtAuth = 100 ;
          AdLifetime = 18000 ;
        ...
VAR     PduMessage : ( RegRequest, DeRegRequest, RegReply, Datagram, Advertisement );
        RegRequest :  record
                          ipSourceAddr  :  integer ;
                          isDestAddr    :  integer ;
                          ...
                      end ;

        identification : integer;
        ...
CHANNEL SPchannel ( Net, Other ) ;
      by Net :
           Junk ;
      by Other :
           Junk ;
CHANNEL PDUchannel ( pdu,  another ) ;
      by pdu :
          RegRequest ;
          RegReply ;
          Datagram ;
TIMER  ad_timer       600 ;
STATE    REGISTERED, IDLE ;
...
```

Figure 3.1: Example of the Declaration Component

Initialization The initial state of the protocol machine is defined by the initialization. In addition, all the variables may be assigned their initial values in the initialization, or given default values if not explicitly initialized. Figure 3.2 is an example of the initialization.

Transition The transition component contains a set of state transitions which define the start states, the end states, and the input/output events. The clause group (including from, to, when, priority, and provided clauses) specify the present state and the next state of a transition, the sending and receiving of the SPs and PDUs, the priority, and the enabling predicate.

A group of Pascal statements specify the action function. Five Pascal statements are supported: assignment statement, if statement, while statement and compound statement and output statement. Moreover, a set of timer statements are supported to specify the operations on the timers.

If the enabling predicate is satisfied, an ISP or PDU is received, the protocol is in the right control state, and the transition has the highest priority among all transitions, then the transition is fired, and the protocol machine moves to the next state. Figure 3.3 is an example of a transition declaration.

A complete example of specification for the home agent is in Appendix C.

```
INITIALIZE
    TO    IDLE
    BEGIN
        identification := 0 ;
        careofAddr   := 0 ;
        . . .
    END ;
```

Figure 3.2: Example of the Initialization Component

```
TRANS

    FROM  IDLE
    TO    REGISTERED
    PROVIDED
            ( RegRequest.mipMHauthExtSPI = mipMHauthExtSPI )    AND
            (RegRequest.mipMHauthExtAuth  =  mipMHauthExtAuth) AND
            ( RegRequest.mipIdentification > identification )    AND
            ( RegRequest.ipDestAddr  =  mipHomeAgent )
      BEGIN
          RegReply.ipSourceAddr  =  mipHomeAgent ;
          RegReply.mipType  : =  3 ;
          RegReply.mipLifetime  :=  RegRequest.mipLifetime - 1 ;
          . . .
          IF  ( bindno  =  1 )
          THEN careofAddr  :=  RegRequest.mipCOA
          ELSE  careof Add1  :=  RegRequest.mipCOA ;
          . . .
          output   pdu.RegReply ;

      END;
```

Figure 3.3: Example of the Transition Component

3.3 Implementation of Estelle.Z Parser

3.3.1 Lexical Analysis of Estelle.Z

Lexical analysis for Estelle.Z is done by the lexical analyzer LEX. The lexical analyzer reads from the source program, and carves the source program into tokens. Each token can be treated as a single logical entity. Identifiers, keywords, constants, operators and punctuation symbols are typical tokens. The lexical analyzer returns to its caller a code for the token that it found. It also returns a token value if the token is not a reserved word, punctuation, or operator, such as an identifier, integer and string.

In a compiler implementation, lexical analysis and syntactic analysis are normally performed in the same pass. The lexical analyzer operates under the control of the parser. The parser asks the lexical analyzer for the next token whenever the parser needs one. Then the lexical analyzer returns the token code and an applicable token value to the parser.

In the lexical analyzer of Estelle.Z, all the reserved words, punctuation and predefine operators in Estelle are included, even though they are not utilized in Estelle.Z. Therefore, this lexical analyzer can be used without much modification when Estelle.Z is expanded to the whole Estelle language.

3.3.2 Syntax Analysis of Estelle.Z

The parser for Estelle.Z is generated by the syntactic analyzer tool YACC. The syntactic analysis checks that input tokens occurring according to the patterns that are permitted by the specification for the source language.

YACC input is produced based on the BNF rules of Estelle.Z. Figure 3.4 depicts the process of Estelle.Z parser generation. The object of the Estelle.Z parser is generation of

syntax trees which store the syntactical information of the statements and expressions in an Estelle.Z specification. The syntax trees are stored in data structure the PDS, from which the TESTGEN engine can obtain information to generate test suites.

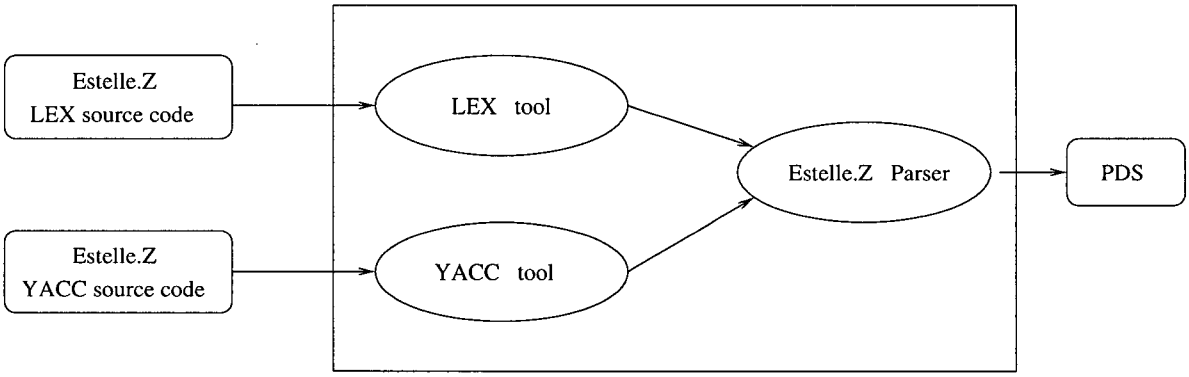


Figure 3.4: Estelle.Z Parser Generation

In a syntax tree, the statements are represented as tree nodes. For example, An assignment statement, variable-name := expression, is depicted in Figure 3.5(a). The variable-name can be a simple variable name or be a parameter at the left-hand side of the assignment statement. The expression can be a tree of any expression on the right-hand side of the assignment statement. Figure 3.5(b) shows an example of the syntax tree of an assignment statement and its right-hand side expression. Similarly, **if** statements, **while** statements, **compound** statements and **output** statements are converted into syntax trees stored in the PDS. In addition, expressions are represented by syntax trees.

Declaration

In the declaration section, variables, constants, channels, timers and states are declared. The parser converts the information into key, name, type and value (if applicable)

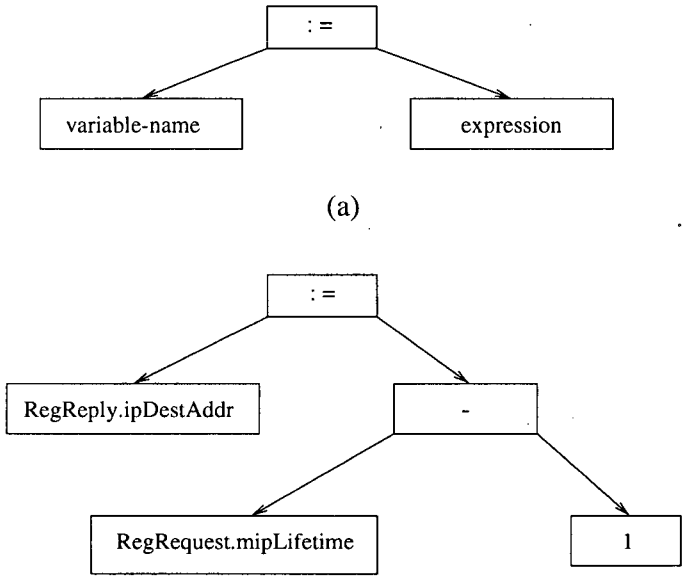


Figure 3.5: Syntax Tree of Assignment Statement

which are then stored in the PDS. For example, in the constant declaration,

$$mipMHauthExtAuth = 100$$

is stored as in Figure 3.6, where the key is an index into constants.

key = 0
name = mipMHauthExtAuth
type = INT_TYPE
val = 100

Figure 3.6: Structure of Constant Declaration

Variables, timers and states are processed in the same way as constants. The channel declaration varies slightly from the above. There are two types of declarations, SPchannel and PDUchannel. SPchannel declares the ISPs and OSPs used in the specification. PDUchannel declares the PDUs used in the specification. The information on SPs and PDUs is necessary for the TESTGEN engine. All of the information, such as the parameter's field name and type, is stored in the PDS. For example, the PDU RegRequest is stored as in Figure 3.7. The PDU type tree is a pointer pointing to the PDU's parameters including name, type and other features (if applicable).

Initialization

The parser sets default initial values for all variables, parameters and timers if they are not explicitly initialized. The default value is 0 for those of integer type, *false* for

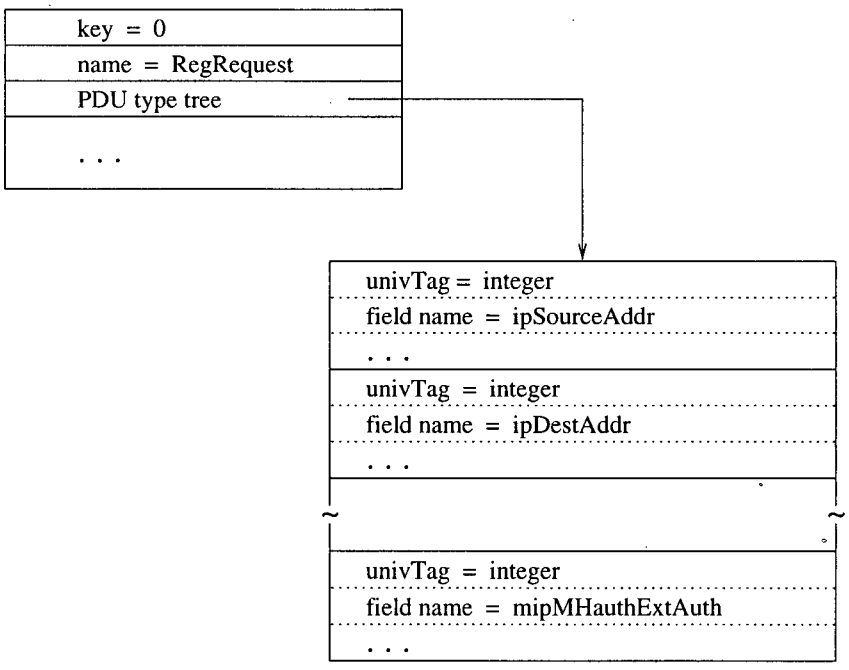


Figure 3.7: Syntax Tree of Channel Declaration (PDU)

those boolean type and the empty string for those of character string type. The initial state of protocol machine is assigned in this part. Otherwise the first state in the state declaration is assumed to be the initial state.

Transition

The transition consists of a group of transition definitions. Each transition definition is identified by a key, which is stored in the trans_key[] field of the starting state of that transition. The transition definition includes the from state, the to state, the input events, the enabling predicate, the priority and the action function which is a set of statements. This definition is converted into a syntax tree. Whenever the basic components such as variables, constants, parameters are referenced, their types and keys are stored. The leaf nodes of a syntax tree may be variables, constants, parameters and timer constructs.

Figure 3.8 shows the syntax tree of the transition in Figure 3.3. In this example, the transition is from the IDLE state to the REGISTERED state, the input event is RegRequest, the output event is RegReply, the enabling predicate is the boolean expression stored in epred field, and the action function consists of 12 statements, which are linked respectively to their syntax by the keys and types.

3.3.3 Semantic Analysis

Semantic analysis is used to determine the type of expressions, to check that arguments are of types that are legal for an application of an operator, and to convert the statements into syntax trees stored in the PDS. Semantic analysis is carried out during the syntax analysis phase.

3.3.4 Structure of the PDS

The internal object of the parser is the PDS, based on which the TESTGEN kernel can generate test suites automatically. The PDS is designed to represent the formal protocol specifications based on Estelle.Z formalism in a machine accessible form. The

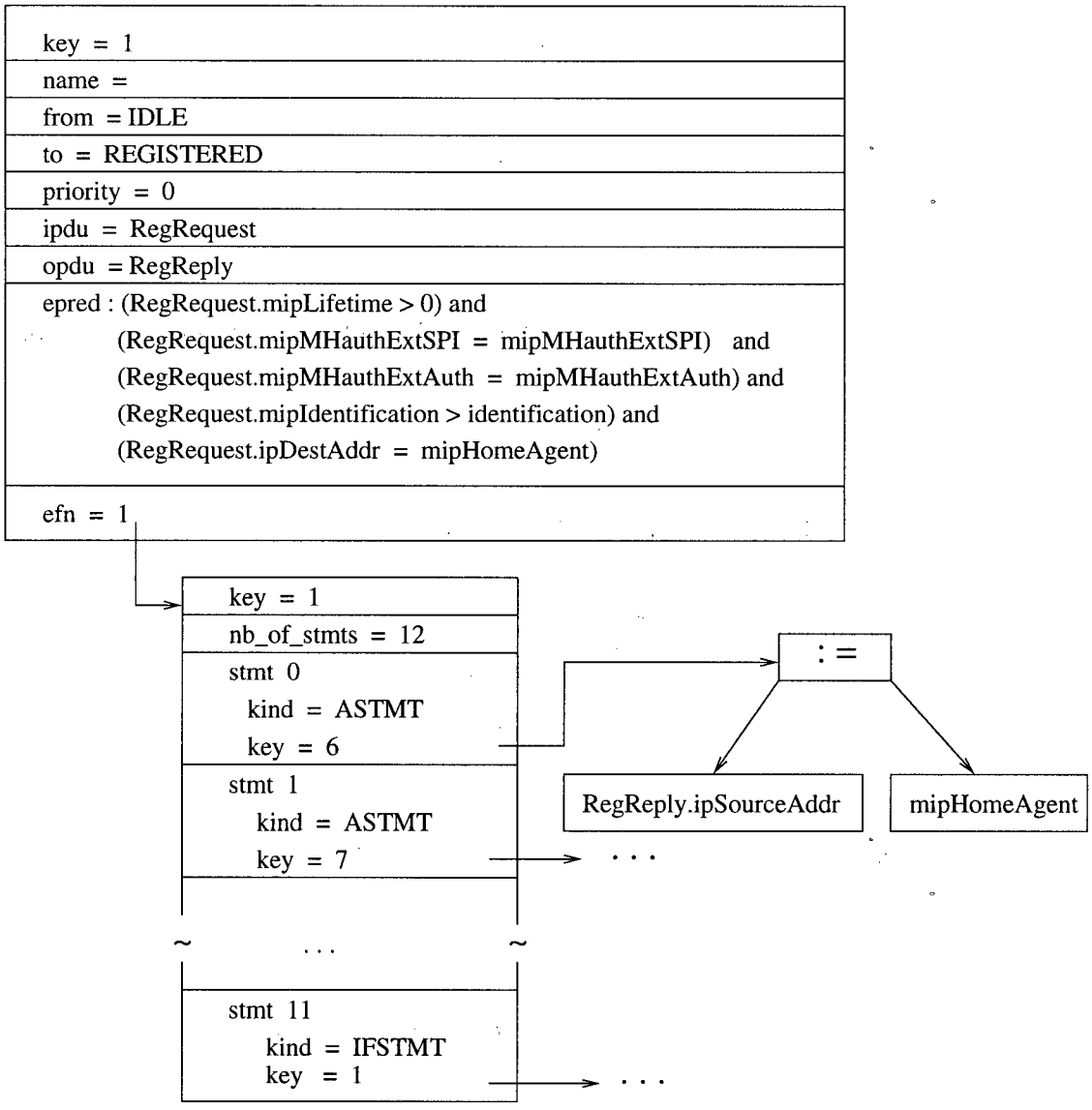


Figure 3.8: Syntax Tree of State Transition

protocol specification saved in the PDS will be directly used for the subtours identification process which is the key process of automatic test suite generation. Since TESTGEN is completely dependent on the data in the PDS, it is essential to ensure that the PDS correctly represents the information of the protocol specifications in order to generate test suites correctly. On the other hand, it is also important for information in the PDS to be easily accessed by TESTGEN. In addition, the PDS can be used as an internal medium between the test suite engine and the protocol specification, i.e., the PDS can be the internal object of any specification language parser with some modification if necessary, and the test suite generation engine can yield test suites based on the PDS as showed in Figure 3.9.

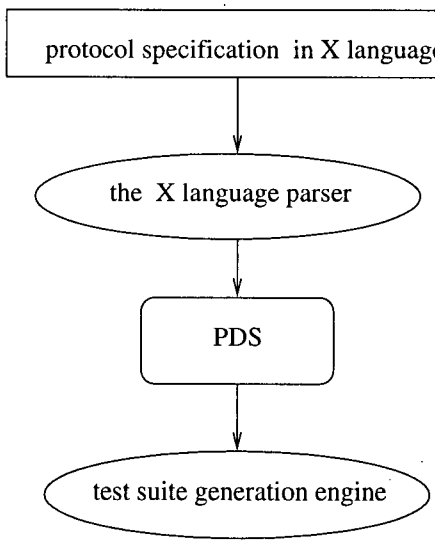


Figure 3.9: Environment of the Parser

The PDS is organized in a way shown in Figure 3.10.

Using the Estelle.Z language, a protocol is described in terms of several sets of components, such as states, variables, constants, channels, timers and transitions. Some protocol components are further described in terms of other sets of protocol components.

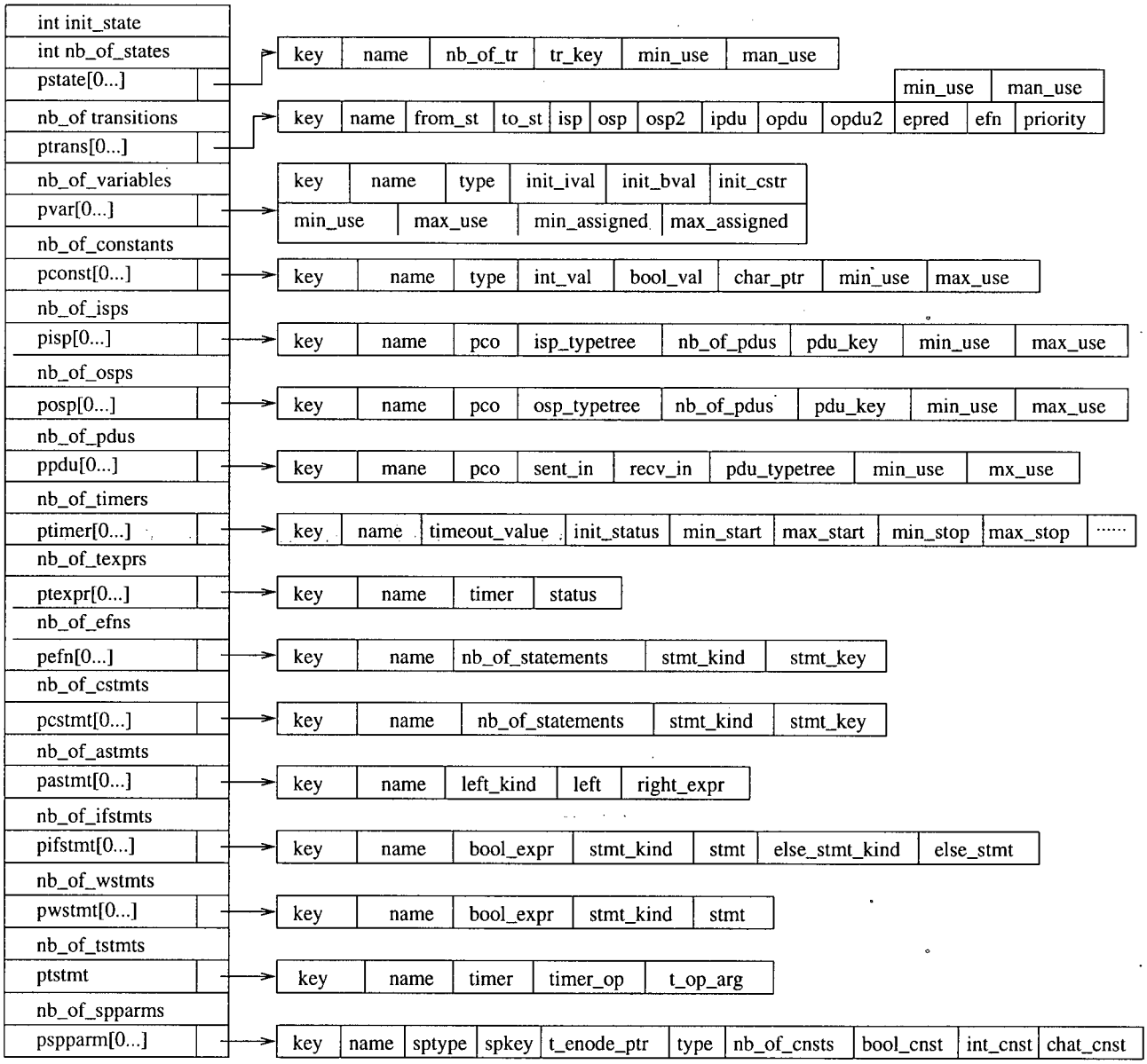


Figure 3.10: Protocol Data Structure

For example, transitions may be described by enabling predicates, action functions, statements, etc. Each set of protocol components is defined by a structure in the PDS. The definitions of the major structures: states, transitions, variables, constants and statements, are given briefly in the following sections.

State Structure definition

State is defined to describe the control states of a protocol. Five fields are used to represent a state as depicted in Figure 3.11a.

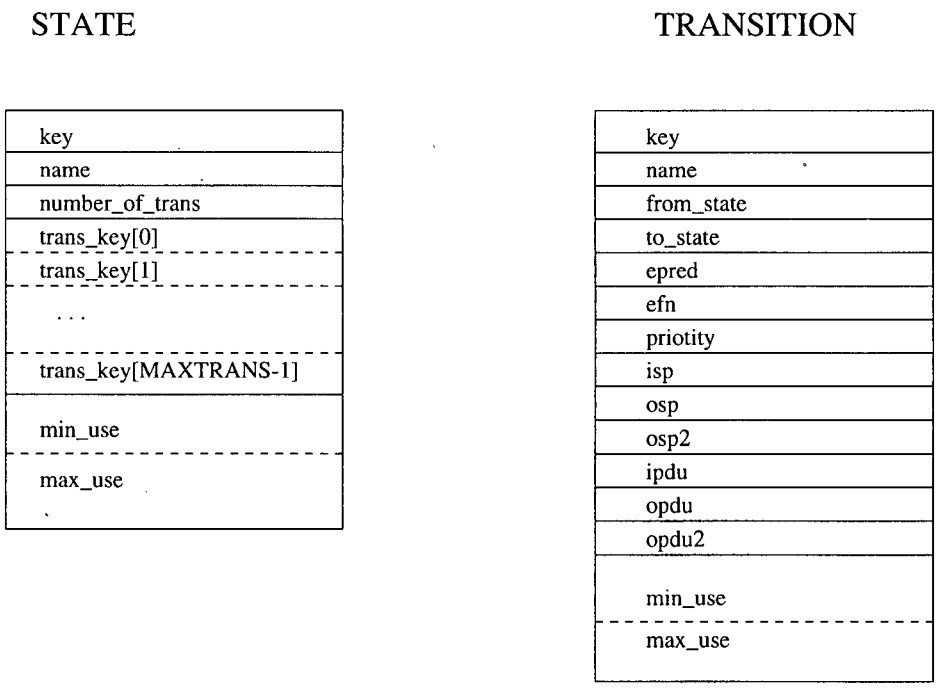


Figure 3.11: Data Structure of State in the PDS

- key is an index into the *pstate array*. The current state is the *key*th state in the

state declaration and may be referenced as *pstate[key]*.

- name is the state's name.
- number_of_trans indicates the total number of transitions possible from this state.
- trans_key is an array storing the keys to the transitions which are possible from this state.
- min_use, max_use indicate the limitations of the usage of the current state, i.e., passing the state at least min_use times and at most max_use times. These are not a part of the specification, but are TESTGEN constraints edited by the users.

All the transitions possible from the given state can be assessed consequently by looking up the *trans_key*[*key*] array. This is used for the subtours identification process in the test suite generation engine.

Transition Structure Definition

The *transition* structure defines the state transitions of the protocol machine. Fields are as shown in Figure 3.11b.

- key is the *key*th state transition in the transition declaration.
- name is the name of the transition if applicable.
- from_state, to_state indicate the beginning and ending states of the transition by the keys to the states.
- epred, afn are keys to the enabling predicate and the action function associated with this transition.
- priority indicates the priority of the transition if applicable.

- min_use, max_use are the limitations that the transition can be used at least min_use times and at most max_use times. These are not a part of the specification, but are TESTGEN constraints edited by the users.

The data structure of variables and constants consists of key, name, type and value fields. The value may be an integer, boolean or string stored in *int_val*, *bool_val* and *char_val* for constants, or stored in *init_ival*, *init_bval* and *init_cstr* for variables.

The statements consist of *if_statement*, *while_statement*, *compound_statement* and *assignment_statement*. The syntax trees of these statements are stored in the PDS as shown in Figure 3.10, and the meaning of the fields are plainly indicated by the field name.

3.3.5 Testing for the parser

The accuracy of the PDS is very crucial to the test suite generation engine since the test suite generation engine completely depends on the information in the PDS. Three verification mechanisms are used to verify the accuracy of a generated the PDS.

Printing the PDS

The PDS can be saved into a text file, and can also be displayed interactively on the screen by the user. The user can check all the information in the PDS by reading the text file or check a particular part, such as an expression or a statement, by displaying the expression or statement on screen through a set of printing functions.

Since there is a parser of Estelle.Y+ASN.1 which converts a protocol specification into a PDS, the two PDSs generated by the two parsers, the Estelle.Y+ANS.1 parser and the Estelle.Z parser, can be compared. One protocol can be specified in Estelle.Y+ASN.1, as well as in Estelle.Z. The two PDSs should be similar if the protocol specifications are

equivalent.

Consistency checking

In TESTGEN+, there is a set of consistency checking functions developed to check the consistency of the PDS after it has been generated. Theses functions can be used to check the PDS generated by the Estelle.Z parser.

Test suite coverage checking

By feeding the two specifications, one in Estelle.Y+ASN.1 and the other in Estelle.Z, for the same protocol into their respective parsers, the same test suites should be generated. The existence of numerous test cases in a test suite makes it difficult to compare the two test suites on an individual case by case basis. The test coverage tool is used to compare the coverage of the two test suites. If they are the same or very similar, then the PDS generated by the Estelle.Z parser is correct.

In Chapter 5, the home agent protocol is specified in both Estelle.Y+ASN.1 and Estelle.Z. The two specifications are applied to the TESTGEN engine and two test suites are generated. The two test suites are fed into the test coverage tool and the same results obtained. This will be discussed in detail in Chapter 5.

3.4 Improvement on the interface of TESTGEN

Output in TTCN.MP format

The output formats of TESTGEN are subtours for people to read, and the internal files for TESTVAL and TESTSEL. None of them is standard output format that can be used outside TESTGEN+. Therefore, it is necessary to provide a standard output for TESTGEN+ so that the TESTGEN+ tool can be widely used.

For this purpose, the TTCN.MP format, a standardized test notation, is provided. The Tree and Tabular Combined Notation (TTCN) was developed and standardized by the ISO (International Organization for Standardization) in 1990, and is used for precise specification of abstract test suites [ISO].

The TTCN consists of two types of notations: the tree notation, used in dynamic behavior descriptions to describe events which can occur as alternative responses to a previous event, and a tabular component, used to simplify the representation of all static elements. The TTCN is in two formats, a *graphical form* denoted TTCN.GR, and a *machine-processable form* denoted TTCN.MP. The TTCN.GR is human readable and understandable. The TTCN.MP is a packed format representation to allow more uniform and more efficient storage [Pro92].

Because TTCN is a standardized test notation and TTCN.MP is a packed representation and a machine processable format, the TTCN.MP is developed as a output module in TESTGEN+ . This module obtains information from TESTGEN and the PDS, organizes them in TTCN.MP format, and writes them to a file.

A test suite consists of four sections: overview, declaration, dynamic and constraints as shown in Figure 3.12. The overview section names the test suite and defines its context with respect to the protocol standard and test method. The declaration section provides the name, type definition, range and comments for all of the objects that are referenced in the test suite itself. The constraints section specifies particular data values for PDU fields or ASP parameters as used in a constrained test event in the dynamic behavior section. These constraints are different from the constraints in the TESTGEN tool. The dynamic section comprises a set of test cases, the main body of the abstract test suite including the test case library, the test step library and the default behaviors library.

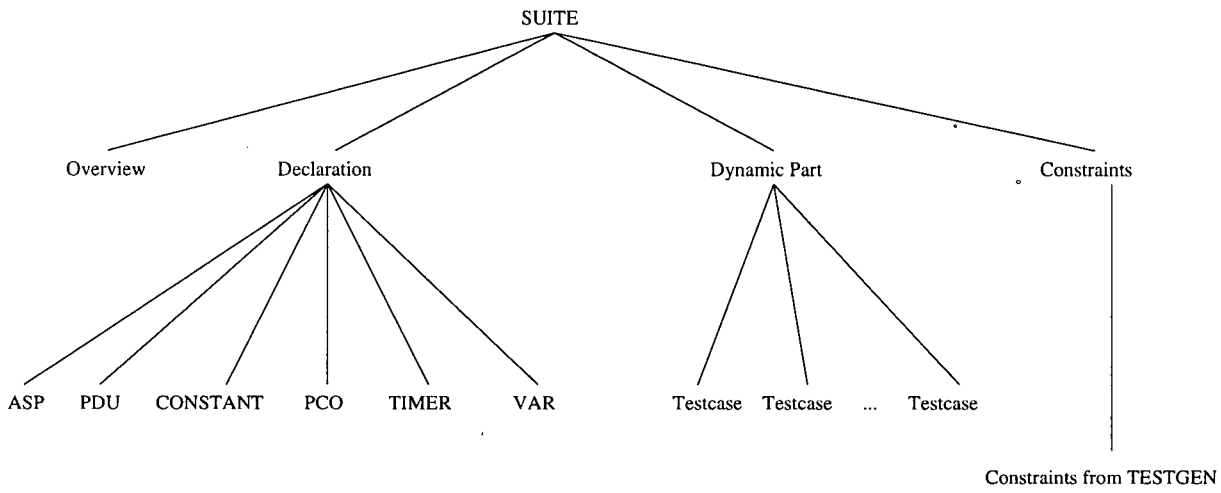


Figure 3.12: Architecture of TTCN Tree From TESTGEN

A collection of test cases for the home agent in TTCN.MP format is shown in Appendix F.

Graphical user interface

In order to give users a user friendly interface, a graphical user interface is provided based on the menu driven user interface. The graphical user interface is implemented on XView library under SunOS UNIX 4.1.3. Under this user interface, TESTGEN, TESTSEL and TESTVAL are integrated under TESTGEN+, so that the user can easily travel among the three tools simply by clicking on the mouse.

The function of the graphical user interface is the same as the menu driven user interface for TESTGEN, TESTSEL and TESTVAL. In addition, some items are added to enhance the TESTGEN menu, such as storing the constraint instances and parameters of PDUs, ISPs and OSPs, removing the subtours file when restarting the TESTGEN tool, choosing to display the default constraints, editing the subtours file and TTCN.MP file, etc.

3.5 Summary

In this chapter, the Estelle.Z parser has been presented. The Estelle.Z parser accepts the Estelle.Z specification language, and produces a representation of the protocol as a PDS. the object structure of The lexical, syntactic and semantic analyses of the parser have been discussed. In chapter 5, a real world protocol specification in Estelle.Z will be fed in the parser to test its accuracy.

Chapter 4

Specification of a Subset of the Mobile IP Protocol

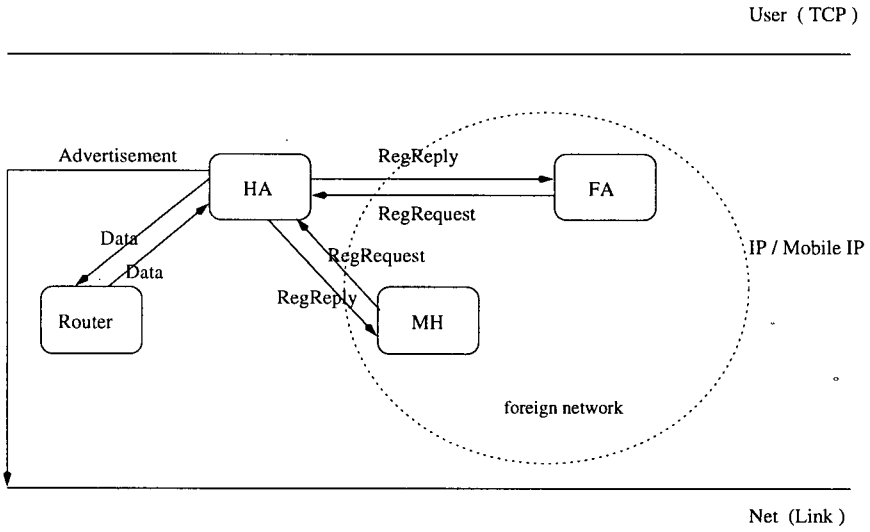
Introduction

The Mobile IP Protocol, also called IP Mobility Support, is an Internet-Draft submitted by the Mobile IP Working Group of the Internet Engineering Task Force (IETF), to route IP datagrams transparently in the Internet.

Version 15 of this protocol released in February, 1996 [Per96] is discussed in this thesis.

In the Mobile IP Protocol, each mobile node is identified by its home IP address, regardless of where the mobile node's current point of attachment to the Internet is. If a mobile node is away from its home network, then it is associated with a care-of address which provides information about its current point of attachment to the Internet. The mobile node then registers the care-of address with its home agent. The home agent sends datagrams headed for the mobile node through a tunnel to the care-of address. At the end of the tunnel, each datagram is either directly delivered to the mobile node or is delivered via a foreign agent.

In this chapter, the basic concepts and functions of the Mobile IP Protocol are described. One major element, the home agent, is discussed in detail. The formal specification of the home agent is presented.



Note: * HA : Home Agent
* FA : Foreign Agent
* MH : Mobile Host

Figure 4.1: Module of Mobile IP

4.1 Basics of the Protocol

There are three major modules in the mobile IP protocol: (1) Home Agent, (2) Foreign Agent and (3) Mobile Host or Mobile Node as shown in Figure 4.1.

The home agent is a router on the mobile node's home network. When a mobile node is away from its home, the home agent maintains information on its current location and tunnels datagrams to the mobile node.

The foreign agent is a router on the mobile node's foreign network. When a mobile node is registered with a foreign agent, the foreign agent provides routing services to the mobile node.

The mobile host is a host or a router which can change its point of attachment from one network to another. A mobile host may change its location without changing its IP

address and may continue to communicate with other Internet nodes by using this IP address.

In the following subsections, an overview of the Mobile IP protocol is given and the functions of the three modules are discussed.

4.1.1 Protocol Overview

Mobile IP protocol provides two types of services:

1. Agent Discovery: The home agent and the foreign agent advertise their services on the subnetworks where they are available. By listening to these advertisements, a newly arrived mobile node can locate a foreign agent if it is in a foreign network, or a home agent if it is in its home network.
2. Registration: When a mobile node is away from its home, it registers its care-of address with the home agent. If the mobile node can find a foreign agent in the current network, then the foreign agent's IP address will be the *care-of address* and the mobile node can register with its home agent through the foreign agent. Otherwise, the mobile node can get a *co-located care-of address*, a temporary or a long-term address used by the mobile node while visiting the foreign network, by some external assignment mechanism, and register this temporary address with its home agent directly.

The Mobile IP Protocol operates in the following manner:

- The foreign agent and the home agent advertise their presence by sending an Agent Advertisement message.

- A mobile node analyzes the Agent Advertisement messages it receives and assesses whether it is on its home network or on a foreign network. If the mobile node is on its home network, it operates without mobility service. If it is returning to its home network from a foreign network, the mobile node de-registers with its home agent by sending a Registration Request and receiving a Registration Reply. If the mobile node moves to a foreign network, it registers its new care-of address with its home agent by sending a Registration Request and receiving a Registration Reply either directly or through a foreign agent.
- Datagrams delivered to the mobile node's home address are intercepted and tunneled by its home agent to the mobile node's care-of address. They are then delivered to the mobile node by a foreign agent or tunneled to the mobile node's co-located care-of address, i.e., the mobile node.
- Datagrams sent by the mobile node are generally delivered to their destinations by using standard IP routing mechanisms, and not by using any mobility services.

4.1.2 Home Agent

The services provided by the Home Agent are:

- Agent Advertisement: The home agent advertises its services on a link by sending Agent Advertisement messages periodically. The mobile nodes use these advertisements to determine their current point of attachment to the Internet, i.e., whether they are on their home networks. An Agent Advertisement is an ICMP (Internet Control Message Protocol) Router Advertisement that has been extended to also carry a Mobility Agent Advertisement Extension.

- Registration: The home agent receives Registration Requests, processes them and sends Registration Replies to grant or deny the services requested. If service permission is granted, then the home agent will update its mobility binding list with the care-of address of the tunnel and send the registration reply with an appropriate message. If the registration request is denied, then the home agent will send the Registration Reply with a suitable error code. If the parameter life-time in the Registration Request is zero, then the home agent will de-register the current care-of address. By maitaining the binding list, the home agent can serve more than one mobile nodes.
- Routing: The home agent intercepts all the datagrams on the home network that are addressed to the mobile node while the mobile node is registered away from its home. If the IP Destination Address of the arriving datagrams is the same as the home address of the mobile node, then the home agent tunnels the datagrams to the mobile node's currently registered care-of address. If the mobile node has no current mobility bindings, i.e., it is at home, and the home agent is also a router handling common IP traffic, then the home agent will forward the datagram directly to the mobile node. If the mobile node requires, then the home agent will forward the broadcast messages to the mobile node.

4.1.3 Foreign Agent

The foreign agent supports the following services:

- Agent Advertisement: Similar to the home agent, the foreign agent advertises its services on a link by sending Agent Advertisement messages periodically to the local network. The foreign agent uses different parameters from that of the home

agent, including the foreign agent's care-of address for the mobile nodes. The care-of address usually is the IP address of the foreign agent.

- Registration: When a mobile node registers with its home agent via a foreign agent, the foreign agent only relays valid Registration Requests between the mobile node and the home agent and valid Registration Replies between the home agent and the mobile node. Each foreign agent maintains a visitors' entry list containing the information obtained from the mobile node's Registration Requests. When the foreign agent receives a valid Registration Reply from the home agent, hence indicating a successful Registration, the foreign agent modifies the visitors' entry list. By maintaining the visitors' entry list, the foreign agent can serve more than one visiting mobile nodes.
- Routing: The foreign agent receives datagrams sent to its advertised care-of address. If the datagrams are destined for its visiting mobile nodes, then the foreign agent forwards the datagrams to these mobile nodes, otherwise it discards them.

4.1.4 Mobile Host

A mobile host, also called a mobile node, has the following functions:

- Agent Discovery: After a mobile node receives an Agent Advertisement, it can determine whether it is on its home network or on a foreign network. If the mobile node returns to its home network from a foreign network, it de-registers with its home agent directly. If the mobile node moves to a foreign network, it registers with its home agent via a foreign agent, which is sending an Agent Advertisement with a care-of address. If the mobile node does not receive any Agent Advertisement for a certain amount of time, and if it can obtain a care-of address through a link-layer

protocol or other means, then the mobile node registers the care-of address with its home agent directly. Otherwise, the mobile node sends an Agent Solicitation to search for an agent.

- Registration: A mobile node initiates a registration whenever it detects a change in its network connectivity. When it is away from its home, the mobile node's Registration Request allows its home agent to create or modify a mobility binding for the mobile node. When it is at home, the mobile node's (de)Registration Request allows its home agent to delete any previous mobility binding(s) to it. A mobile node is able to operate without the support of mobility functions when it is at home.

  After sending a Registration Request, the mobile node will receive a Registration Reply, either from a home agent or a foreign agent. If the registration is accepted, then the mobile node configures its routing table accordingly. If the registration is rejected, then the mobile node modifies its Registration Request according to the error information on the Registration Reply, and tries again.

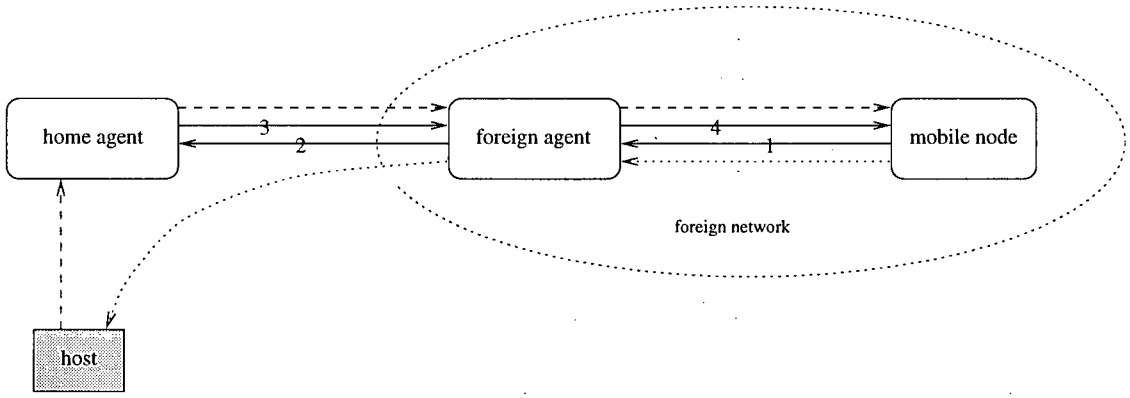
- Routing: When a mobile node is connected to its home network, it operates as a stationary host. When it is registered on a foreign network, the mobile node chooses a default router. If it is registered with a foreign agent, the foreign agent becomes the default router, otherwise, the local network router becomes the router.

4.1.5 Example

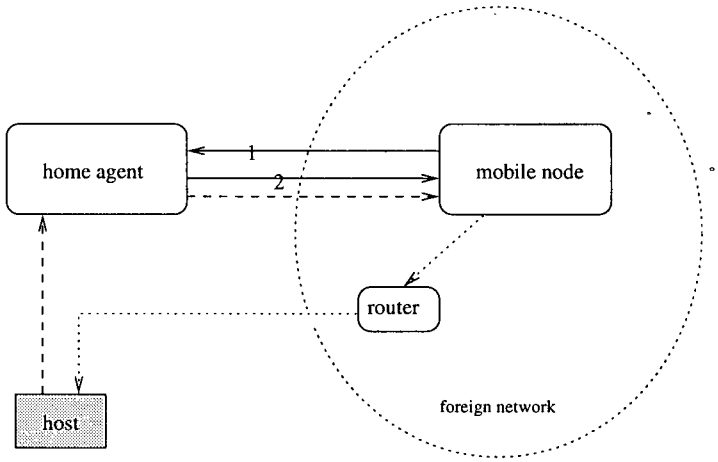
In this section, an example is given to show how the mobile IP protocol functions.

In Figure 4.2 (a), the mobile node moves into a foreign network and detects a foreign agent. The mobile node sends a Registration Request to the foreign agent (1), the foreign

agent relays the request to the mobile node's home agent(2). If the request is accepted, the home agent sends a Registration Reply to the foreign agent(3), and the foreign agent



(a)



(b)

Legend:
- Data sent from mobile node
- Data send to mobile node
- Registration Request/Reply

Figure 4.2: Example of mobile IP Operation

relays the request to the mobile node(4). If a message is sent to the mobile node, the home agent intercepts the message and tunnels it to the foreign agent. Subsequently, the foreign agent forwards the message to the mobile node after checking if the message is for the mobile node who is visiting the local network (dashed line). If a message is to be sent from the mobile node, it is first sent to the foreign agent and then to the destination host via normal IP routing methods (dotted line).

In Figure 4.2 (b), a mobile node moves into a foreign network where no foreign agent is presently implemented. The mobile node obtains a co-local care-of address, and registers with its home agent directly(1). If the request is accepted, the mobile node receives a Registration Reply from the home agent(2). If a message is to be sent to the mobile node, the home agent tunnels it to the mobile node (dashed line). If a message is to be sent from the mobile node, it is initially directed to the local router and then delivered to the host (dotted line).

The home agent is discussed more thoroughly in the next section.

4.2 Home Agent

The home agent is one of the most important elements in mobile IP protocol. It is necessary to discuss the messages received or sent by the home agent to fully understand its behaviors.

The home agent receives one kind of message, Registration Request, and sends two kinds of messages, Registration Reply and Agent Advertisement. The home agent also receives and forwards datagrams. These datagrams are IP datagrams, therefore they are not discussed in this thesis.

Table 4.1 shows the major fields in Registration Request:

Field	Name	Description	Required
1	Source Address	address from which the message is sent	√
	Destination Address	address of foreign agent or the home agent	√
2	Type	1	√
	Lifetime	the life time for this request	√
	Home address	IP address of the mobile node	√
	Home agent	IP address of the mobile node's home agent	√
	Care-of address	IP address of the end of tunnel	√
	Identification	constructed by mobile node	√
3	Type	32	√
	SPI	Security Parameter Index	√
	Authenticator	message authentication code	√
4	Type	33	
	SPI	Security Parameter Index	
	Authenticator	message authentication code	
5	Type	34	
	SPI	Security Parameter Index	
	Authenticator	message authentication code	

Table 4.1: Fields in Registration Request

There are five fields in this table: field one is the *IP field* which indicates the source and destination addresses of the message; field two is the *UDP field* which indicates information about the mobile node; field three is the *Mobile-Home Authentication Extension* which indicates the authentication between a mobile node and its home agent. These three fields are essential in the Registration Request message. Field four is the *Mobile-foreign Authentication Extension*, which indicates the authentication between a mobile node and a foreign agent in the foreign network, if applicable. Field five is the *Foreign-Home Authentication Extension*, which indicates the authentication between the mobile node's home agent and a foreign agent, if applicable. In some networks, there is no implementation of a foreign agent. Thus, there is no authentication related to the foreign agent. Hence the last two fields are optional.

Table 4.2 shows the major fields in Registration Reply:

Field	Name	Description	Required
1	Source Address	address from which the message is sent	√
	Destination Address	source address of Registration Request	√
2	Type	3	√
	Code	0/1 for success, others for error code	√
	Lifetime	the life time for this request	√
	Home address	IP address of the mobile node	√
	Home agent	IP address of the mobile node's home agent	√
	Care-of address	IP address of the end of tunnel	√
	Identification	based on the Identification of Registration Request	√
3	Type	32	√
	SPI	Security Parameter Index	√
	Authenticator	message authentication code	√
4	Type	33	
	SPI	Security Parameter Index	
	Authenticator	message authentication code	
5	Type	34	
	SPI	Security Parameter Index	
	Authenticator	message authentication code	

Table 4.2: Fields in Registration Reply

Similar to the fields in Registration Request, there are five fields in Registration Reply. The first two fields are related to *IP field* and *UDP field.* The difference between a Registration Request and a Registration Reply is the code field which indicates the registration result. The code is 0 or 1 if the registration is successful, otherwise, the code is a number (greater than 1) indicating an error message. The last three fields are similar to the fields in the Registration Request in that they indicate the authentication among the mobile node, the home agent and the foreign agent.

Table 4.3 shows the major fields in an Advertisement:

Field	Name	Description	Required
1	Destination Address	link layer destination address	√
2	TTL	time to live, set to 1	√
	Destination Address	IP address of local broadcast address	√
3	Lifetime	the life time for this Advertisement	√
	H	set to 1 if it is a home agent	√
	Type	16	√
	Sequence number	integer	√
	Registration lifetime	maximum of acceptable seconds	√

Table 4.3: Fields in Advertisement

There are three fields in Agent Advertisement: *link-layer field*, to indicate the destination; *IP fields*, to indicate Time To Live of the message; and *Mobility Agent Advertisement Extension*, to indicate that an ICMP Router Advertisement message is also an Agent Advertisement being sent by a mobility agent for additional information about the mobility service that the agent supports.

The IP Mobility Support documentation[Per96] gives more detailed information on each message communicated between the mobile node, the home agent and the foreign

agent.

4.3 The formal Specifications of the Home Agent

From the above sections, we have acquired more knowledge about the home agent. Based on this knowledge, we can discuss the formal specification of the home agent, which is used by the test generation tool (TESTGEN) to generate test cases. To simplify the specification, one mobile node is assumed. The TESTGEN tool accepts specification in ASN.1 for the data part and specification in Estelle.Y for the control part, or specification in Estelle.Z for both parts.

Data Part in ASN.1

The corresponding protocol data units (PDUs) received or sent by the home agent and their basic parameters are as follows:

- RegRequest : Registration Request. To simplify the specifications, only 11 major parameters are presented as follows:

```
  Name              Type          Description / Meaning
 =======           ======       ========================

        -- IP fields
ipSourceAddr        INTEGER     addr. from which the message is sent
ipDestAddr          INTEGER     addr. of foreign agent / home agent
        --  Mobile IP fields
mipType             INTEGER     set to 1
mipS                INTEGER     1: simultaneous bindings
mipLifetime         INTEGER     time remaining before expired
```

```
mipHomeAddr          INTEGER     IP addr. of mobile node
mipHomeAgent         INTEGER     IP addr. of mobile node's home agent
mipCOA               INTEGER     IP addr. of the end of the tunnel
mipIdentification    INTEGER     number constructed by the mobile node
              -- Mobile-Home Authentication Extension
mipMHauthExtSPI      INTEGER      Security Parameter Index (4 Bytes)
mipMHauthExtAuth     INTEGER      Authenticator, keyed MD5
```

- RegReply : Registration Reply. To simplify the specifications, only 10 parameters are presented as follows:

```
 Name                Type          Description / Meaning
 ======              ======        ======================
        -- IP fields
ipSourceAddr         INTEGER     addr. from which the message is sent
ipDestAddr           INTEGER     addr. of foreign/home agent
        --  Mobile IP fields
mipType              INTEGER     set to 3
mipCode              INTEGER     the result of registration request
mipLifetime          INTEGER     time remaining before expired
mipHomeAddr          INTEGER     IP addr. of mobile node
mipHomeAgent         INTEGER     IP addr. of mobile node's home agent
mipIdentification    INTEGER     number constructed by the mobile node
              -- Mobile-Home Authentication Extension
mipMHauthExtSPI      INTEGER     Security Parameter Index (4 Bytes)
mipMHauthExtAuth     INTEGER      Authenticator, keyed MD5
```

- Advertisement : Agent Advertisement. To simplify the specifications, only 6 parameters are presented as follows:

```
    Name              Type        Description / Meaning
   ======            ======      ======================

         -- IP fields
ipDestAddr          INTEGER      multicast addr. or
                                     limited broadcast addr.
ipTTL               INTEGER      IP field, must be set to 1
         -- ICMP fields
icmpLifetime        INTEGER      life time of valid advertisement
                                     interval = 1/3 lifetime
icmpCode            INTEGER      0: as a router too
         -- Mobility Agent Advertisement Extension
extRegLifetime      INTEGER     longest time acceptable of registration.
extH                INTEGER (0..1)  1: this is a home agent
```

Appendix A contains the complete ASN.1 specification for the simplified home agent. Appendix D contains the ASN.1 specification for all the parameters in all PDUs.

Control Part in Estelle.Y and Estelle.Z

The Extended Finite State Machine (EFSM) of the home agent is shown in Figure 4.3.

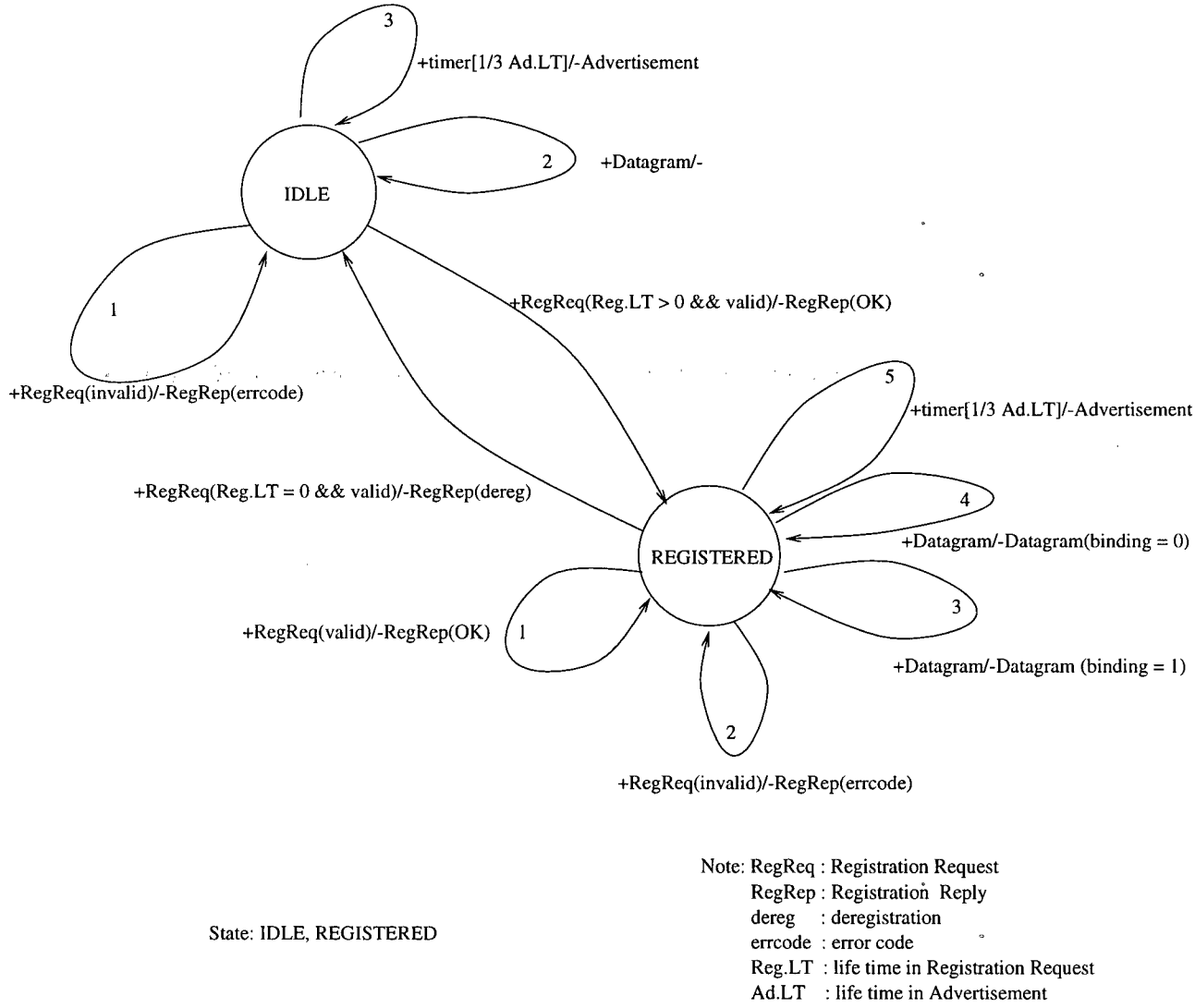


Figure 4.3: EFSM of the Home Agent

The EFSM has 2 states, IDLE and REGISTERED, which are defined as follows:

1. IDLE: Initial state. Athe the beginning, the home agent is awaiting the Registration Request from the foreign agent, the mobile node, or the datagrams from networks, while the home agent sends Agent Advertisement periodically.

   When the home agent receives a Registration Request, it checks its validity. If it is invalid, it sends a Registration Reply with a related error code and it remains at a IDLE state; if it is valid, it sends a Registration Reply with a successful code and transfers to a REGISTERED state; if datagrams are received, the home agent discards them as the mobile node has not been registered yet.

2. REGISTERED: When a Registration Request is accepted, the home agent moves to this state, which means that it provides mobility service to the mobile node from now on. If datagrams are received while in this state, the home agent tunnels them to the care-of address of the mobile node; if another Registration Request is received, and if the home agent supports simultaneous bindings, then the home agent registers the mobile node with another care-of address; if a de-Registration Request is received, then the home agent checks its validity, if it is valid, then it de-registers for the mobile node. At the same time, the home agent sends Agent Advertisement periodically.

When receiving PDUs or internal events, such as timeouts, the home agent may change its state or remain in its current state, and send corresponding PDUs.

State transmissions are described as follows:

- from IDLE to IDLE:

  1. receive Registration Request (invalid), reply with error code

2. receive Datagrams for non-registered mobile node, discard
3. send Agent Advertisement periodically (time out)

- from IDLE to REGISTERED
  - receive Registration Request (valid), reply with 0/1 code
- from REGISTERED to IDLE
  - receive Registration Request (Lifetime = 0, valid), reply with 0/1 code
- from REGISTERED to REGISTERED

1. receive Registration Request (valid), reply with corresponding code
2. receive Registration Request (invalid), reply with corresponding error code
3. receive Datagrams for mobile node (if it is in a foreign network), tunnel the Datagram
4. receive Datagrams for mobile node (it is in its home network), deliver the Datagram
5. send Agent Advertisement periodically (time out)

The complete specification in Estelle.Y is given in Appendix B.

The complete specification in Estelle.Z is given in Appendix C.

Chapter 5

Applying Mobile IP to TESTGEN

Introduction

In chapter 3, the new parser was described. In chapter 4, the specifications of the Mobile IP protocol were presented. In this chapter, the specifications of the Mobile IP are fed into the TESTGEN tool to yield test cases in order to determine whether the new parser functions correctly. Hence, we can compare the two sets of test cases yielded from the two parsers. Since the number of test cases is too large, and there is extreme data variation in the test cases, we fed those test cases into the TESTSEL tool to test their coverage and other features. In this chapter, the quality of test cases generated by TESTGEN is discussed, followed by a discussion on the coverage of test cases. Comments on the two parsers are summarized at the end of this chapter.

5.1 The Test Suite Generation

When a protocol is implemented, we need to test if it has been implemented consistently with its specification and if it can cooperate with existing networks smoothly, i.e., we need to apply conformance testing to the implementation. TESTGEN is one tool that allows the generation of various test cases to check the reaction of the implementations. In this section, the algorithm of TESTGEN is given succinctly [LV93], followed by the constraints chosen for the testing of the Mobile IP protocol.

5.1.1 The Algorithm of TESTGEN

Before we introduce the algorithm of TESTGEN, let's first define two terminologies that will be used later.

1. An extended transition system (**ETS**) is a quadruple $ETS = (Q, E, \rightarrow, q_{init})$.

   - $Q = States \times Variables \times Constants \times Timers$, denoting the set of states of the ETS.
   - $E = ISP \cup OSP \cup PDU$, denoting the set of events of the ETS.
   - $\rightarrow \subseteq Q \times E \times Q$, denoting the transition relation for the ETS.
   - $q_{init} \in Q$, denoting the initial state of the ETS.

2. **Subtour** (X, S_i, S_0, C) is the set of all executable paths (subtours) from an initial state S_i to a final state S_0, where S_0 is the initial state of a finite state machine X with constraint C. In TESTGEN, $S_i = S_0$, because the test control sequences begin and end at the same initial state S_0 (is usually *idle*). The constraint C is a set of feasible paths, including restrictions on the iteration number of while loops and transition loops, etc..

TESTGEN generates subtours (test cases) based on a set of constraints and the information in the PDS which has been established by the parsers according to a given specification. The subtours generation algorithm is an exhaustive depth first search over the protocol behavior tree. If a test branch of this tree fulfills all the defined constraints, then a subtour is identified with this test branch. The algorithm is given briefly as follows:

```
Generate_subtour( ETS, C_min, C_max, q, u, subtour)
{
    find transition t in the ETS that can be fired at state q;
    for each t  {
        if the event of t is an ISP or IPDU (Input PDU) {
            for all ISP or IPDU parameter value combination v ∈ CV  {
                if t is enabled by v  {
                    q_next ← apply action function to state q and event (IPDU or ISP);
                    u_next ← u+ the elements usage of action function of t
                    subtour_next ← subtour + (t, v);
                    if C_min ≤ u_next ≤ C_max and q_next = q_init
                        { output the subtour to the file; return; }
                    if u_next ≤ C_max
                        Generate_subtour ( ETS, C_min, C_max, q_next ,u_next, subtour_next);
                }
            }
        };

        if the event of t is a PROVIDED clause and no WHEN clause {
                if t is enabled  {
                    q_next ← apply action function to state q;
                    u_next ← u+ the the elements usage of action function of t
                    subtour_next ← subtour + (t, v);
                    if C_min ≤ u_next ≤ C_max  and  q_next = q_init
```

```
                    { output the subtour to the file; return; }
                if u_next ≤ C_max
                Generate_subtour ( ETS, C_min, C_max, q_next ,u_next, subtour_next);
            }
        }
    }
}
```

where C_{max} is a vector of all the maximal constraints, C_{min} is a vector of all the minimal constraints, q is a state, u is a vector of the number of current usages for all parameters, CV is a set of all allowed parameter value combinations.

Using the algorithm above, subtours that satisfy the constraints can be generated. The algorithm generateds finite subtours, because of the restriction of the constraints.

5.1.2 Constraints of TESTGEN

There are two types of constraints in TESTGEN. One type sets the maximum and minimum usage for all elements, including states, ISPs, OSPs, PDUs, constants, variables, and timers. The other type is the parameter variations which define a set of values for all ISPs and PDU's parameters. In addition, there is a set of default constraint values set by TESTGEN. The default values for all maximum and minimum usage of all elements are 1 and 0 respectively, and for parameter variations are 0 and 99 respectively. With the default constraint values, 2053 test cases are generated from the home agent specification. The 2053 test cases come from 3 test control sequences with data variation: (1) IDLE to IDLE with PDU Advertisement; (2) IDLE to IDLE with PDU Datagram and no output; (3) IDLE to IDLE with no meaningful PDU RegRequest and RegReply.

The reason is that the maximum usages of all parameters are set to 1 and the minimum usages of all parameters are set to 0, the values of PDU parameters are set to 0 and 99. Such values are not meaningful in the home agent protocol.

We modify the constraints value in the following way: state REGISTERED is passed at least once and at most 3 times; state IDLE is passed at least once and at most twice. Hence the mobile node can apply for registration, get the service of data transmission and advertisements from home agent, and apply for de-registration. The values of PDU parameters are set according to the meaning of the parameters and the testing purpose. It is necessary to test the validation of the Registration Request according to specification, so both the correct and incorrect parameter values for the Registration Request's parameters are supplied, including mipMHauthExtSPI and mipMHauthExtAuth, etc.. All the ISPs and OSPs' parameters are set to 0, because these ISPs and OSPs are not utilized in our testing, but they are required by the grammar of the specification. The major min. and max. constraint values and the major parameter variation constraints of PDUs in the Mobile IP protocol are given as follows:

1. State

   IDLE (1/2)

   REGISTERED (1/3)

2. Transitions

   For all transitions (0/99)

3. ISP

   For all ISPs (0/0)

4. OSP

   For all OSPs (0/0)

The following are the major parameter variation constraints for PDUs:

1. RegRequest

   ipSourceAddr (8285)

   ipDestAddr (821045, 82832)

   mipType (1)

   mipS (1)

   mipLifetime (10)

   mipHomeAddr (821012)

   mipHomeAgent (821045)

   mipCOA (821020)

   mipIdentification (5, 20)

   mipMHauthExtSPI (200, 250)

   mipMHauthExtAuth (100, 150)

2. DeRegRequest

   ipSourceAddr (821012)

   ipDestAddr (821045)

   mipType (1)

   mipS (1)

   mipHomeAddr (821012)

   mipHomeAgent (821045)

   mipCOA (0)

   mipIdentification (50)

   mipMHauthExtSPI (200)

mipMHauthExtAuth (100)

3. Advertisement

   For all parameters (1)

4. RegReply

   For all parameters (1)

All the parameters of Advertisement and RegReply are assigned in the protocol specification, so the values in the constraint part are initialized to 1. This does not affect the test case generation.

With the above constraint values, there are 641 test cases generated by TESTGEN which come from 4 test control sequences with data variation. The 4 typical test control sequences and the typical test cases in TTCN.MP form are given in Appendix F.

If IDLE (1/2) is changed to IDLE (1/3), there will be 27802 test cases. These test cases are to be used in the TESTSEL later.

5.1.3 Comments on TESTGEN

With the experience of utilizing TESTGEN on Mobile IP, my experience and comments on TESTGEN are as follows:

Advantages:

1. By using formal specification, Estelle.Y and ASN.1 or Estelle.Z, the ambiguities in the specification written in English can be detected earlier.
2. It is convenient for the user to have various choices through TESTGEN menus to select different parameter constraints according to the testing purposes. Also, it is

easy to repeat the operation until the user gets a satisfactory result.

3. TESTGEN combines both the control component and the data component, therefore it can yield many different instances for test cases. In the real testing, there may be all kinds of messages fed into the implementation. For example, in the home agent implementation, a RegRequest with different parameters may be received. From TESTGEN the instances of such RegRequest can be generated.

Problems in TESTGEN:

1. Assignment statement: Sometimes the assignment statements in the specification are not executed correctly. For example, all the parameters of RegReply PDU are assigned in the specification according to the test purpose, but the subtours generated by the TESTGEN reveal values which are not assigned in the specification, such as -1, 99 and so on. These parameter values of RegReply render the subtours meaningless. For the Advertisement PDU, the parameters are correctly assigned as the values in the specification.

2. Provided statement: Sometimes the provided statements are executed incorrectly. For example, when conditions are satisfied, the FSM should transform from state REGISTERED to state IDLE, however it does not. Hence, it is necessary to add one more PDU deRegRequest into the specification. When conditions are satisfied, the deRegRequest PDU is sent out and the FSM is transformed from a REGISTERED state to a IDLE state. This is not convenient for specification.

3. Expression: Sometimes the expressions are processed incorrectly. After changing the formula, the expressions can be computed correctly. For instance, changing $not\,(a > b)$ into $a \leq b$ enables TESTGEN to run more smoothly.

4. Single module: Because TESTGEN can only process a single module, the test for the input/output message from/to the module can not be done. In this thesis, we only assume that the home agent receives a RegRequest PDU, and sends out an Advertisement PDU, a RegReply PDU and a Datagram PDU. Any tests on the communications between the home agent and mobile node can not be done. In the real test, it is necessary to test the communications among the home agent, the mobile node and the foreign agent.

5. Function of the subset: At present, TESTGEN can only support a very small subset of the Estelle language, which makes TESTGEN tool very limited, Hence, only a small protocol can be specified and fed into the TESTGEN tool. In the real world, the protocols, such as the Mobile IP Protocol, are much bigger and more complex than that which can be specified in Estelle.Y and ASN.1. It would be more practical if TESTGEN can support more features, such as communication between two entities, parallelism between two module instances, dynamic structure, more statements and variable types, and etc.

6. Constraints: Some functions of constraints are duplicated. For example, if the maximum usage of REGISTERED state is increased, and the maximum usage of RegRequest and Datagram are not changed, no more test cases will generated. Because only when some events, such as RegRequest and Datagram, happen, can the protocol state machine pass the state REGISTERED one more time. The usage of state and PDU/ISP are related.

Future study:

1. More examples are needed to test every aspect of TESTGEN to check if it works

perfectly according to its design.

2. The TESTGEN tool can be expanded into several aspects: multiple modules, more data types, more statements and subroutines.

5.2 The Coverage Of The Test Suite

TESTGEN yields a large number of test cases, ranging from a few thousand to tens of thousands. These test cases are combinations of both the control sequences and data variations. All of these test cases come from a few control sequences. In these control sequences, which have more coverage than the others? When there are numerous control sequences, we must address this question. The TESTSEL tool tries to answer this question. In this section, the algorithm of TESTSEL is given, followed by the results of applying selection to the test suite in the above section. The coverage will be discussed in later section.

5.2.1 The algorithm of the test selection tool

All the test cases form a test space. In this space, some test cases are very close, i.e., the distance between them is short. However, some test cases are distant. If a set of test cases is to be selected, which will be in the final test case set? Figure 5.1 shows circle B is bigger than circle A, even there are both five test cases in each circle.

We would say test case set B has more coverage than test case set A, since the distance between B and the furthest test case to B is smaller than the distance between A and the furthest test case to A. B and A may have the same cost, since both B and A have five test cases and they may spend same steps to find these five test cases respectively. The formal algorithms of calculating the distance between test cases, the cost, and the

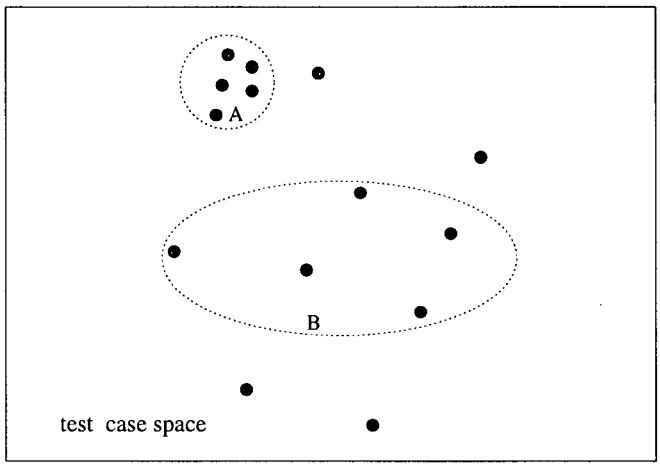


Figure 5.1: Test Case Space

coverage are in [MVC92].

The algorithm starts with an empty set of the selected test cases TC, and the current cost (Cur_Cost) 0. While the tolerance (i.e., the distance between test cases) ϵ is greater than ϵ_{min} for each test control sequence (tc_0) in the test suite TS, the minimum distance between tc_0 and any test control sequence tc in TS is calculated. If the minimum distance is greater than the current tolerance ϵ , then the tc_0 is added to the test cases TC. Once all the test control sequences in the TS have been tested, the test selection tolerance ϵ is shrunk by multipling the user defined scale factor (Scale), and reiterating the algorithm until the minimum tolerance (ϵ_{min}) is reached or the maximum cost (MaxCost) is reached.

The brief test selection algorithm is given as follows:

1. initialization: TC $\leftarrow$ 0; Cur_Cost $\leftarrow$ 0;
2. loop: **while** $\epsilon \geq \epsilon_{min}$ {

reset TS;

while (there are unprocessed test cases in TS) {

```
            tc0 ← pick up a test case in TS randomly;
            distance ← min_{tc∈TC} { all dt(tc0, tc) };
            if ( distance > ε )
              if ( Cur_Cost + Cost(tc0) < MaxCost ) {
                TC ← TC ∪ { tc0 };
                Cur_Cost ← Cost(tc0) + Cur_Cost ;
              }
            else break from the inter while loop;
            Mark the tc0 as processed
        }
        ε ← ε × Scale ;
    }
```

where **dt** is a function to compute the distance between the two test control sequences and **Cost** is a function to compute the cost for the test control sequence. A detailed algorithm is in [MVC92].

5.2.2 The results of the TESTSEL

The results of running the TESTSEL tool are given in Table 5.1, including the information on test cases (TC), control sequences (CS), epsilon (Eps), min. epsilon (m_Eps), max. acceptable cost (MCost), scale (Scale), real cost (RCost), coverage (Coverage), selected control sequence (SCS) and selected test cases (STC). X denotes no result in this place.

Four groups of test selection results are given in the table. In the first group there are 641 test cases generated by TESTGEN. These 641 test cases are composed from 4 test control sequences with data variation. In the fourth group, there are 27802 test cases, which consist of 62 test control sequences with data variation. From Table 5.1, the following information can be observed, and some strong and weak points can be observed:

- In the first group, if the parameter cost is set to be 20 and the scale to be 0.3, then the test selection finishes with 5 real costs, 2 control sequences, 129 test cases and 76% for the test coverage. If the parameter cost is set to be 20, and the scale to be 0.1, then the test selection finishes with 5 real costs, 1 control sequence, 128 test

Group	TC	CS	Eps	m_Eps	MCost	Scale	RCost	Coverage	SCS	STC
	641	4	1.8	0.3	20	0.3	5	0.76	2	129
	641	4	2.0	0.1	20	0.5	5	1.00	4	296
1	641	4	1.8	0.3	20	0.1	5	0.51	1	128
	641	4	1.0	0.05	20	0.5	0	1.00	4	296
	641	4	1.8	0.3	5	0.3	0	-49.0	0	0
	641	4	1.8	0.3	10	0.3	5	0.76	2	129
	529	7	1.8	0.3	20	0.5	6	0.88	3	257
	529	7	1.5	0.1	20	0.2	12	0.88	3	257
2	529	7	1.0	0.01	30	0.2	12	1.00	7	521
	529	7	1.0	0.05	20	0.4	12	1.00	7	521
	529	7	1.0	0.1	20	0.4	12	0.94	5	265
	17425	13	1.8	0.3	20	0.5	6	0.88	3	X
	17425	13	1.5	0.1	40	0.2	12	0.88	3	X
3	17425	13	1.0	0.01	50	0.5	12	0.99	9	X
	17425	13	1.0	0.05	40	0.3	24	0.98	7	X
	17425	13	1.0	0.01	40	0.1	36	1.00	13	X
	27802	62	1.8	0.3	20	0.5	16	0.75	5	X
4	27802	62	1.5	0.1	40	0.2	33	0.88	7	X
	27802	62	1.0	0.01	50	0.5	48	0.94	24	X
	27802	62	1.0	0.01	70	0.5	37	0.99	39	X

Table 5.1: Results of the TESTSEL for Specification in Estelle.Y+ASN.1

cases and 51% for the test coverage. If the epsilon and min. epsilon are changed as well, the test selection finishes with 5 real cost, 4 control sequences, 296 test cases and the test coverage is 100%.

- The strong points are:

  1. The user can choose any parameter value to satisfy their requirement. If the bigger epsilon, smaller min. epsilon and smaller scale are chosen, then more control sequences are selected and more coverage is obtained.
  2. It is an efficient tool for a big group of test cases, because the user can select more efficient test cases with an acceptable cost and a satisfactory coverage from the whole collection of test cases.

- The weak points are:

  1. Real cost: The cost is the sum of the number of steps for selecting control sequences according to the parameters given by the user.
     - From the data in group one, we observed that almost all the test costs are 5, regardless of how many control sequences are selected and how much the coverage is. This is unrealistic in the real world. It will invariably cost more when more control sequences are selected and more coverage is obtained.
     - In another test case selection, we even have 4 control sequences selected with no cost.
     - In one test case selection from the fourth group, 24 control sequences are selected with the cost at 48 and the coverage at 94%. In another test case

selection with the same parameters except with the max. cost parameter at 70, 39 control sequences are selected with cost at 37 and coverage at 99%. This is unrealistic in the real world. From the algorithm above, the cost should be higher than 48 because 15 more control sequences are selected and 5% more coverage is reached.

2. Selection: In the selected control sequences, sometimes there are two similar control sequences, i.e., one control sequence is selected and put into the file twice. For example:

```
Phase = 5, TC Id. = 0
 (- Advertisement -, 1)  (RegRequest RegReply -, 1)
 (- Advertisement -, 1)  (Datagram Datagram -, 1)
 (DeRegRequest RegReply -, 1) (RegRequest RegReply -, 1)

Phase = 0, TC Id. = 0
 (- Advertisement -, 1)  (RegRequest RegReply -, 1)
 (- Advertisement -, 1)  (Datagram Datagram -, 1)
 (RegRequest RegReply -, 1) (DeRegRequest RegReply -, 1)
```

   In phase 0, test case 0 is selected and put into the file; In phase 5, test case 0 is selected and put into the file again. That means one more control sequence and more duplicative test cases are collected later. From the algorithm above, there should only be one test case 0 in the test cases set to TC.

3. Coverage: When 4 test control sequences are selected from 4 control sequences, the coverage is 100%. This is reasonable. When no control sequence is selected,

the coverage should be 0 instead of -49.00. The formula used with no control sequence is $1 - 100.0/2.0 = -49.0$. This is incorrect.

4. Merging: From 641 test cases, 4 control sequences are discovered. After merging these 4 control sequences with data variation, there should be 641 test cases. Because the merge module reproduces the selected test cases in the original form, i.e., all of the original instances generated by TESTGEN should be included. But from the TESTSEL, only 296 test cases are reproduced. We lose more than half the original test cases. The same happens in the second group.

5. Big group test cases: TESTSEL is more useful for selecting more efficient control sequences from a large number of control sequences generated from the TESTGEN tool. In the first group, the number of control sequences is 4, which is a bit small. Hence we have other groups with more control sequences. The merging module does not work for the third and the fourth group. (The error report is " Wrong result in id_file and select_file", but both id_file and select_file are internal files in the test selection tool.) So the complete selected test cases can not be obtained. This defeats the function of TESTSEL.

5.3 Results from the Estelle.Z parser

The Estelle.Z specification language is defined, and its parser is implemented and linked with the TESTGEN kernel. To test whether the parser functions correctly with the TESTGEN kernel, the specifications of the home agent in both Estelle.Z and Estelle.Y+ASN.1 are fed into the TESTGEN tool. If two exact test suites or two very similar test suites are generated by TESTGEN, then the parser of Estelle.Z is working

correctly.

From Figure 5.2, we can observe: if we try to get exact or very similar test suites, then we need to have exact or very similar data structure the PDS for the same protocol specifications, because the TESTGEN kernel generates test suites completely based on all information in the PDS. All data in the PDS are stored in the internal file pds.file. There are two PDS files, the PDS1 and the PDS2, for two home agent specifications, a specification in Estelle.Y+ASN.1 and a specification in Estelle.Z. After comparing the PDS1 and the PDS2, we observed the PDS2 is the same as the PDS1. Besides, with the same constraints of the TESTGEN tool, 641 test cases are generated for both home agent specifications. After fed into the TESTSEL, two sets of 641 test cases result in the same coverage and the same cost. The same results are developed for the three other groups. From Table 5.2 and Table 5.1, we can conclude the the parser works well.

5.4 Summary

This chapter has discussed the algorithms of the TESTGEN tool and the TESTSEL tool, the two test suites from the TESTGEN tool for home agent specification in Estelle.Y+ASN.1 and in Estelle.Z, and the quality of the two test suites.

The results of the TESTSEL has demonstrated that the parser of Estelle.Z can be used to be the front part of the TESTGEN tool, and the parser simplifies the protocol specifications. In addition, there is still further work to make the TESTGEN+ environment more practical and more efficient

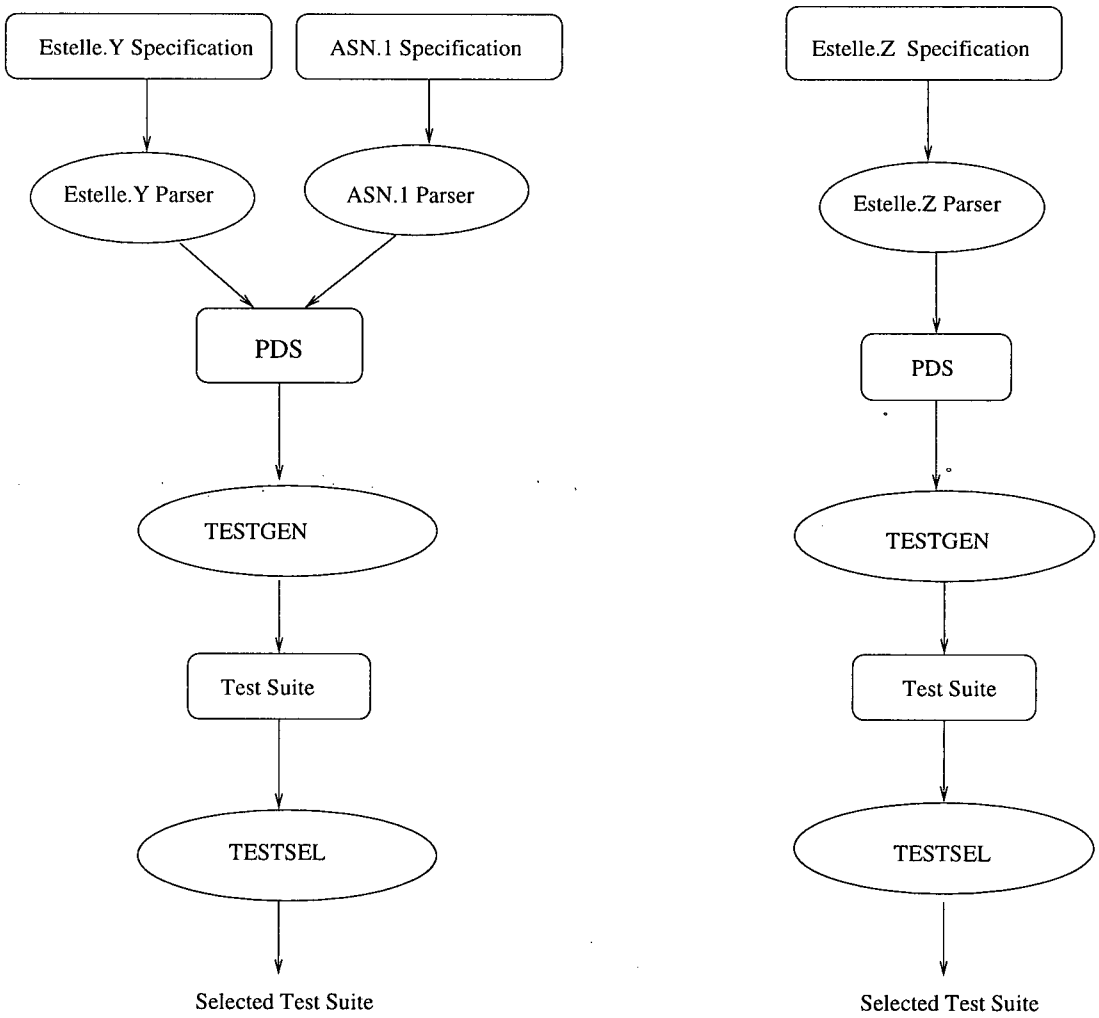


Figure 5.2: Two Parsers

Group	TC	CS	Eps	m_Eps	MCost	Scale	RCost	Coverage	SCS	STC
	641	4	1.8	0.3	20	0.3	5	0.76	2	129
	641	4	2.0	0.1	20	0.5	5	1.00	4	296
1	641	4	1.8	0.3	20	0.1	5	0.51	1	128
	641	4	1.0	0.05	20	0.5	0	1.00	4	296
	641	4	1.8	0.3	5	0.3	0	-49.0	0	0
	641	4	1.8	0.3	10	0.3	5	0.76	2	129
	529	7	1.8	0.3	20	0.5	6	0.88	3	257
	529	7	1.5	0.1	20	0.2	12	0.88	3	257
2	529	7	1.0	0.01	30	0.2	12	1.00	7	521
	529	7	1.0	0.05	20	0.4	12	1.00	7	521
	529	7	1.0	0.1	20	0.4	12	0.94	5	265
	17425	13	1.8	0.3	20	0.5	6	0.88	3	X
	17425	13	1.5	0.1	40	0.2	12	0.88	3	X
3	17425	13	1.0	0.01	50	0.5	12	0.99	9	X
	17425	13	1.0	0.05	40	0.3	24	0.98	7	X
	17425	13	1.0	0.01	40	0.1	36	1.00	13	X
	27802	62	1.8	0.3	20	0.5	16	0.75	5	X
4	27802	62	1.5	0.1	40	0.2	33	0.88	7	X
	27802	62	1.0	0.01	50	0.5	48	0.94	24	X
	27802	62	1.0	0.01	70	0.5	37	0.99	39	X

Table 5.2: Results of the TESTSEL for Specification in Estelle.Z

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, a front end has been developed for the protocol specification language, Estelle.Z, which is based on Estelle ISO/IEC ISO 9074:1989(E). This front end performs the syntactic and semantic analysis of an Estelle.Z specification, and produces an annotated parse tree stored in the PDS as the output. It supports the main structures of the Estelle language. The new parser can accept the declarations of specifications in Estelle.Z instead of in ASN.1 language, and can be easily expanded to complete Estelle language.

A real world protocol, home agent – a component of Mobile IP, was used to test the parser. First, the protocol was specified in Estelle.Y+ASN.1. Through its parser, the PDS was developed. A test suite was generated by the TESTGEN tool, then the test suite was fed into the TESTSEL tool, and a set of satisfactory test cases was developed. Also a set of constraints for TESTGEN and a set of parameters for TESTSEL were obtained. Second, the protocol was specified in Estelle.Z. Through the Estelle.Z parser, the same PDS was developed as in the Estelle.Y+ASN.1 parser. A test suite was generated by TESTGEN, and the test suite was fed into TESTSEL, the same set of selected test suite was successfully obtained, with the same set of constraints for TESTGEN and the same set of parameters for the TESTSEL tool. By comparing the two sets of data obtained

through the two parsers, we observed that the Estelle.Z parser works correctly.

The framework established in this thesis is quite powerful. The syntax of Estelle.Z can be expanded to the whole Estelle language easily. The tree structure is general and will be augmented to support all of Estelle without any difficulty. To add more information in the tree structure, users simply augment the node structure.

By analyzing the results, TESTGEN and TESTSEL, the thesis discussed the advantage and limitation of TESTGEN and TESTSEL.

6.2 Future Work

A number of features should be implemented or improved to enhance the function of the test suite generation environment TESTGEN+ listed as the following:

Parser

The following needs to be done for the parser module in the future:

1. At present, the parser has been tested successfully by a real protocol specification, the home agent specification. Due to the time constraints, we only fed one real protocol specification to test the parser. It would be better to have had more real protocol specifications to test all aspects of the parser.

2. A number of features should be implemented to enhance the parser. Such as adding more data types, more statements, expanding the PDS to accommodate more information on the Estelle language, expanding the Estelle.Z to whole Estelle language to specify more complex real world protocols. This work needs to be done in conjunction with extension to the TESTGEN engine.

3. Furthermore, the parser for SDL and LOTOS can be established to make the TESTGEN+ environment more widely used.

TESTGEN

The following needs to be done for the TESTGEN module in the future:

1. Expand the TESTGEN engine to support multiple modules. In the real world protocols, there are more than one module. For instance, in Mobile IP, there are three modules: home agent, foreign agent and mobile host. To test the implementation of the whole mobile IP protocol, the test suite for the whole Mobile IP, including the home agent, the foreign agent and the mobile host, should be developed. In addition, with only one module, it is impossible to test the cooperation among the modules.

2. Expand the TESTGEN engine to process more information in the PDS. As the parser processes the whole Estelle, the TESTGEN engine should be able to process all the information in the PDS to generate a test suite. Without the support of the TESTGEN engine, even if the parser were expanded to whole Estelle, the TESTGEN tool still does not make any more sense than before. Therefore, the parser and the TESTGEN engine should be expanded at the same time.

3. Resolve the problems in the TESTGEN engine. The problems, such as the problems in the processing of assignment statements, need to be resolved. These problems prevent the TESTGEN engine from generating meaningful test suites.

TESTSEL

The following needs to be done for the TESTSEL module in the future:

1. Output. The selected test cases are in an internal file. They are neither in subtours

which are readable by people, nor in TTCN form which is standard test suite notation. It is necessary to modify the output into TTCN form to have TESTSEL widely used.

2. Merging. Usually real protocols are huge, and the test cases are numerous, it is thus necessary to improve the merging module to process large test suites.

3. Cost. The cost is one of the indices to evaluate the quality of the selected test suite. From my practical experience on the home agent, the cost calculation does not correctly reflect the related cost for the test suite. Therefore it is necessary to improve this algorithm.

Bibliography

[Cho78] T. S. Chow. Testing Software Design Modeled by Finite State Machines . *IEEE Transactions on Software Engineering*, 4(3):178–187, March 1978.

[CVI89] W. Y. L. Chan, S. T. Vuong, and M. R. Ito. An Improved Protocol Test Generation Procedure Based on UIOs. In *Proc. of the ACM SIGCOMM '89*, September 1989.

[Gon70] G. Gonenc. A Method for the Design of Fault Detection Experiments. *IEEE Transactions on Computers*, 19(6):551–558, June 1970.

[ISO] ISO DIS/9646-3, OSI Conformance Testing Methodology and Framework, Part 3: The Tree and Tabular Combined Notation, march 1990. and interim working documents throughout 1990, particularly TTCN Extensions, ISO/IEC JTC 1/SC 21 N5077.

[LV93] Sangho Lee and Son T. Vuong. TESTGEN+: An environment for protocol test suite generation, selection and validation, July 1993.

[MVC92] M. McAllister, S. T. Vuong, and J. Curgus. Automated Test Case Selection Based on Test Coverage Metrics. In *Proc. of the IWPTS V*, Montreal, Canada, September 1992.

[Per96] Charles Perkins. IP mobility support, draft-ietf-mobileip-protocol-15.txt, Feb 1996.

[Pro92] R.L. Probert. TTCN: the international notation for specifying tests of communications. *Computer Networks and ISDN Systems*, 23:417–438, 1992.

[SD88] K. K. Sabnani and A. T. Dahbura. A Protocol Test Generation Procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, September 1988.

[VCI89] S. T. Vuong, W. Y. L. Chan, and M. R. Ito. The UIOv-Method for Protocol Test Sequence Generation. In *Proc. of the IWPTS II*, Berlin, Germany, October 1989.

Appendix A

ASN.1 specification of the simplified Home Agent in Mobile IP

```
HomeAgent  DEFINITIONS ::=

BEGIN

PduMessage ::= CHOICE
       {
           RegRequest,
           DeRegRequest,
           RegReply,
           Datagram,
           Datagrambak,
           Advertisement
       }

DeRegRequest ::= SEQUENCE
    {
            -- IP fields
     ipSourceAddr       INTEGER,  -- addr. from which the msg is sent
     ipDestAddr         INTEGER,  -- addr. of foreign agent / home agent
```

```
        -- Mobile IP fields
    mipType             INTEGER,  -- set to 1
    mipS                INTEGER,  -- 1: simultaneous bindings
    mipHomeAddr         INTEGER,  -- IP addr. of mobile node
    mipHomeAgent        INTEGER, -- IP addr. of mobile node's home agent
    mipCOA              INTEGER,  -- IP addr of the end of the tunnel
    mipIdentification   INTEGER, -- number constructed by mobile node

          -- Mobile-Home Authentication Extension
    mipMHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
    mipMHauthExtAuth    INTEGER   -- Authenticator, keyed MD5
    }

RegRequest ::= SEQUENCE
   {
        -- IP fields
    ipSourceAddr        INTEGER,  -- addr. from which the msg is sent
    ipDestAddr          INTEGER,  -- addr. of foreign agent / home agent

        -- Mobile IP fields
    mipType             INTEGER,  -- set to 1
    mipS                INTEGER,  -- 1: simultaneous bindings
    mipLifetime         INTEGER,  -- time remaining before expired
```

```
    mipHomeAddr         INTEGER,  -- IP addr. of mobile node
    mipHomeAgent        INTEGER,  -- IP addr. of mobile node's home agent
    mipCOA              INTEGER,  -- IP addr of the end of the tunnel
    mipIdentification   INTEGER,  -- number constructed by  mobile node

            -- Mobile-Home Authentication Extension
    mipMHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
    mipMHauthExtAuth    INTEGER   -- Authenticator, keyed MD5
    }

RegReply  ::= SEQUENCE
    {
            -- IP fields
    ipSourceAddr        INTEGER,  -- addr. from which the msg is sent
    ipDestAddr          INTEGER,  -- addr. of foreign/home agent

            --  Mobile IP fields
    mipType             INTEGER,  -- set to 3
    mipCode             INTEGER,  -- the result of registration request
    mipLifetime         INTEGER,  -- time remaining before expired
    mipHomeAddr         INTEGER,  -- IP addr. of mobile node
    mipHomeAgent        INTEGER,  -- IP addr. of mobile node's home agent
    mipIdentification   INTEGER,  -- number constructed by  mobile node
```

```
            -- Mobile-Home Authentication Extension
     mipMHauthExtSPI    INTEGER,  -- Security Parameter Index (4 Bytes)
     mipMHauthExtAuth   INTEGER   -- Authenticator, keyed MD5
    }

Advertisement ::= SEQUENCE
    {
         -- IP fields
     ipDestAddr         INTEGER,  -- milticase addr. or
                                  -- limited broadcase addr.
     ipTTL              INTEGER,  --  IP field, must be set to 1

         -- ICMP fields
     icmpLifetime       INTEGER,  -- life time of valid advertisemt
                                  -- interval = 1/3 lifetime
     icmpCode           INTEGER,  -- 0: as a router too

         -- Mobility Afent Advertisement Extension
     extRegLifetime     INTEGER,  -- longest time acceptable for reg.
     extH               INTEGER (0..1)  -- 1: this is a home agent
     }
Datagram ::= SEQUENCE
        {
           ipDestAddr               INTEGER,
```

```
            data                        INTEGER
        }
Datagrambak ::= SEQUENCE
        {
            ipDestAddr                  INTEGER,
            data                        INTEGER
        }
User ::= CHOICE
       {
          Junki
       }
Net ::=  CHOICE
       {
          Junko
}
Junki ::= SEQUENCE
       {
          dummy                       INTEGER
       }
Junko ::= SEQUENCE
       {
         dummy                        INTEGER
       }
END
```

Appendix B

Estelle.Y specification of the simplified Home Agent in Mobile IP

```
specification home_agent ;

CONST
        mipMHauthExtAuth  = 100 : int;
        mipMHauthExtSPI   = 200 : int;
        mipNodeHomeAddr   = 821012 : int;
        mipHomeAgent      = 821045 : int;
        mipS              = 1   : int;
        localNetAddr      = 82832 : int;
        router            = 1   : int;
        AdLifetime         = 18000 : int;
        RegLargestLifetime = 10 : int;

VAR
         identification : int;
         careOfAddr     : int;
         careOfAddr1    : int;
         bindno         : int;
```

```
ISP
        Junki    User;

OSP
        Junko    Net;

PDU
        RegRequest      recv_in   Junki;
DeRegRequest    recv_in   Junki;
        RegReply        recv_in   Junki;
        Datagram        recv_in   Junki;
        Datagrambak     recv_in   Junki;
        Advertisement   recv_in   Junki;

TIMER
        ad_timer   600;

STATE
        REGISTERED, IDLE ;

INITIALIZATION
     TO IDLE
     BEGIN
            bindno          := 0;
```

```
            identification := 0;
            careOfAddr     := 0;
    END;

TRANS

FROM IDLE
    TO IDLE
     provided timeout(ad_timer)
     output Advertisement
     begin
       Advertisement.ipDestAddr := localNetAddr;
       Advertisement.ipTTL  := 1;
       Advertisement.icmpLifetime := AdLifetime ;
       Advertisement.icmpCode := 1;
       Advertisement.extRegLifetime := RegLargestLifetime;
       Advertisement.extH  := 1;
     end;

FROM IDLE
    TO REGISTERED
       when RegRequest
       provided
         (RegRequest.mipLifetime > 0 )
```

```
        AND
        (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
        AND
        (RegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
        AND
        (RegRequest.mipIdentification > identification )
        AND
        (RegRequest.ipDestAddr = mipHomeAgent )
      output RegReply
      begin
         RegReply.ipSourceAddr := mipHomeAgent;
         RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
RegReply.mipType      := 3;
         RegReply.mipCode      := mipS;
         RegReply.mipLifetime  := RegRequest.mipLifetime-1;
         RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
         RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
         RegReply.mipIdentification:=RegRequest.mipIdentification+1;
         RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
         RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
         identification := RegRequest.mipIdentification +1 ;
         if (mipS = 0)
         then
             begin
```

```
            bindno := bindno + 1;
              if (bindno = 1)
              then  careOfAddr := RegRequest.mipCOA
              else
                careOfAddr1 := RegRequest.mipCOA;
        end
        else begin
            careOfAddr := RegRequest.mipCOA;
            bindno := 1;
        end;
    end;

FROM IDLE
    TO IDLE
      when Datagram;

FROM IDLE
    TO IDLE
      when RegRequest
       provided
            RegRequest.ipDestAddr = localNetAddr
       output RegReply
       begin
          RegReply.ipSourceAddr := mipHomeAgent;
```

```
        RegReply.ipDestAddr    := RegRequest.ipSourceAddr;
        RegReply.mipType       := 3;
        RegReply.mipLifetime   := RegRequest.mipLifetime-1;
        RegReply.mipHomeAddr   := RegRequest.mipHomeAddr;
        RegReply.mipHomeAgent  := RegRequest.mipHomeAgent;
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipCode       := 136;
        RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
       identification := RegRequest.mipIdentification+1;
    end;

FROM REGISTERED
  TO IDLE
    when DeRegRequest
    provided
      (DeRegRequest.mipMHauthExtSPI = mipMHauthExtSPI
      AND
      (DeRegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
      AND
      (DeRegRequest.ipDestAddr = mipHomeAgent )

    output RegReply
  begin
```

```
RegReply.ipSourceAddr := mipHomeAgent;
RegReply.ipDestAddr   := DeRegRequest.ipSourceAddr;
RegReply.mipType      := 3;
RegReply.mipCode      := 0;
RegReply.mipLifetime  := 0;
RegReply.mipHomeAddr  := DeRegRequest.mipHomeAddr;
RegReply.mipHomeAgent := DeRegRequest.mipHomeAgent;
RegReply.mipIdentification:=DeRegRequest.mipIdentification +1;
identification        := DeRegRequest.mipIdentification;
bindno := bindno - 1;
RegReply.mipMHauthExtSPI :=DeRegRequest.mipMHauthExtSPI;
RegReply.mipMHauthExtAuth:=DeRegRequest.mipMHauthExtAuth;

if (DeRegRequest.mipCOA = mipNodeHomeAddr)
then
   begin
            careOfAddr := 0;
            careOfAddr1:= 0;
   end
   else
   begin
           if ( careOfAddr =DeRegRequest.mipCOA)
           then   careOfAddr := 0;
           if ( careOfAddr1 =DeRegRequest.mipCOA )
```

```
                    then   careOfAddr1 := 0;
            end;
        end;

FROM REGISTERED
     TO REGISTERED

     provided timeout(ad_timer)
     output Advertisement
     begin
       Advertisement.ipDestAddr := localNetAddr;
       Advertisement.ipTTL  := 1;
       Advertisement.icmpLifetime := AdLifetime ;
       Advertisement.icmpCode := 1;
       Advertisement.extRegLifetime := RegLargestLifetime;
       Advertisement.extH  := 1;
     end;

FROM REGISTERED
    TO REGISTERED

       when Datagram
       provided ( bindno <> 0 )
       output Datagram
```

```
        begin
                Datagram.ipDestAddr := careOfAddr;
                Datagram.data := Datagram.data;
        end;

FROM REGISTERED
    TO REGISTERED

        when Datagram
        provided ( bindno = 0 )
              AND (Datagram.ipDestAddr = mipNodeHomeAddr)
        output Datagram
        begin
              if ( router = 0 )
              then
                begin
                 Datagram.ipDestAddr := 0;
                 Datagram.data := 0;
                end
              else
                Datagram.ipDestAddr := mipNodeHomeAddr;
        end;

FROM REGISTERED
```

```
    TO REGISTERED

       when Datagram
       provided (Datagram.ipDestAddr = mipNodeHomeAddr )
                AND  ( bindno = 1)
       output Datagram
       begin
            Datagram.ipDestAddr := careOfAddr ;
       end;

FROM REGISTERED
    TO REGISTERED

       when Datagram
       provided (Datagram.ipDestAddr = mipNodeHomeAddr )
                AND ( bindno = 2)
       output Datagram, Datagrambak
          begin
               Datagram.ipDestAddr := careOfAddr ;
               Datagrambak.ipDestAddr := careOfAddr1;
               Datagrambak.data := Datagram.data;
          end;

FROM REGISTERED
```

```
    TO REGISTERED

       when RegRequest
        provided
             (RegRequest.mipMHauthExtSPI <> mipMHauthExtSPI)
             OR
             (RegRequest.mipMHauthExtAuth  <>  mipMHauthExtAuth)
        output RegReply
        begin
          RegReply.ipSourceAddr := RegRequest.ipDestAddr;
          RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
          RegReply.mipType      := 3;
          RegReply.mipCode      := 131;
          RegReply.mipLifetime  := RegRequest.mipLifetime-1;
          RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
          RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
          RegReply.mipIdentification:=RegRequest.mipIdentification+1;
          RegReply.mipMHauthExtSPI := 0;
          RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
          identification := RegRequest.mipIdentification+1;
         end;

FROM IDLE
    TO IDLE
```

```
        when RegRequest
        provided RegRequest.mipLifetime < 0 ;

FROM REGISTERED
    TO REGISTERED
        when RegRequest
        provided RegRequest.mipLifetime < 0 ;

FROM IDLE
    TO IDLE

         when RegRequest
         provided
              (RegRequest.mipIdentification < identification )
         output RegReply
          begin
             RegReply.mipCode       := 133;
             RegReply.mipLifetime   := RegRequest.mipLifetime-1;
             RegReply.mipIdentification:=RegRequest.mipIdentification+1;
             RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
             RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
             identification := RegRequest.mipIdentification+1;
          end;
```

```
FROM REGISTERED
    TO REGISTERED

        when RegRequest
        provided
            (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
            AND
            (RegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
            AND
            (RegRequest.mipIdentification > identification )
            AND
            (RegRequest.mipLifetime > 0 )
            AND
            ( mipS = 0 ) AND
            ( bindno = 1 )
output RegReply
         begin
           RegReply.ipSourceAddr := RegRequest.ipDestAddr;
           RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
           RegReply.mipType      := 3;
           RegReply.mipCode      := mipS;
           RegReply.mipLifetime  := RegRequest.mipLifetime-1;
           RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
           RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
```

```
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
        identification := RegRequest.mipIdentification+1;
        bindno := bindno + 1;
        careOfAddr1 := RegRequest.mipCOA;
      end;

FROM REGISTERED
    TO REGISTERED

      when RegRequest
      provided
            (RegRequest.mipIdentification < identification )

      output RegReply
        begin
          RegReply.ipSourceAddr := RegRequest.ipDestAddr;
          RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
          RegReply.mipType      := 3;
          RegReply.mipCode      := 133;
          RegReply.mipLifetime  := RegRequest.mipLifetime-1;
          RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
          RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
```

```
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
        identification := RegRequest.mipIdentification+1;
      end;

FROM IDLE
    TO IDLE

      when RegRequest
      provided
          not (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
          OR
          not (RegRequest.mipMHauthExtAuth  = mipMHauthExtAuth)
      output RegReply
      begin
        RegReply.ipSourceAddr := RegRequest.ipDestAddr;
        RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
        RegReply.mipType      := 3;
        RegReply.mipCode      := 131;
        RegReply.mipLifetime  := RegRequest.mipLifetime-1;
        RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
        RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
```

```
        RegReply.mipMHauthExtSPI := 0;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
        identification := RegRequest.mipIdentification+1;
      end;

end.
```

Appendix C

Estelle.Z specification of the simplified Home Agent in Mobile IP

```
specification home_agent ;

const
        mipMHauthExtAuth  = 100 ;
        mipMHauthExtSPI   = 200 ;
        mipNodeHomeAddr   = 821012;
        mipHomeAgent      = 821045 ;
        mipS              = 1    ;
        localNetAddr      = 82832 ;
        router            = 1    ;
        AdLifetime         = 18000 ;
        RegLargestLifetime = 10 ;

var

  PduMessage : ( RegRequest, DeRegRequest, RegReply, Datagram,
       Datagrambak, Advertisement );

        DeRegRequest :
```

```
        record
                ipSourceAddr      :     integer;
                ipDestAddr        :     integer;
                mipType           :     integer;
                mipS              :     integer;
                mipHomeAddr       :     integer;
                mipHomeAgent      :     integer;
                mipCOA            :     integer;
                mipIdentification :     integer;
                mipMHauthExtSPI   :     integer;
                mipMHauthExtAuth  :     integer;
        end;

RegRequest :
        record
                ipSourceAddr      :     integer;
                ipDestAddr        :     integer;
                mipType           :     integer;
                mipS              :     integer;
                mipLifetime       :     integer;
                mipHomeAddr       :     integer;
                mipHomeAgent      :     integer;
                mipCOA            :     integer;
                mipIdentification :     integer;
```

```
                mipMHauthExtSPI   :       integer;
                mipMHauthExtAuth  :       integer;
        end;

RegReply :
        record
                ipSourceAddr      :       integer;
                ipDestAddr        :       integer;
                mipType           :       integer;
                mipCode           :       integer;
                mipLifetime       :       integer;
                mipHomeAddr       :       integer;
                mipHomeAgent      :       integer;
                mipIdentification :       integer;
                mipMHauthExtSPI   :       integer;
                mipMHauthExtAuth  :       integer;
        end;

Advertisement :
        record
                ipDestAddr        :       integer;
                ipTTL             :       integer;
                icmpLifetime      :       integer;
                icmpCode          :       integer;
```

```
                extRegLifetime    :    integer;
                extH              :    integer;
        end;

Datagram :
        record
                ipDestAddr        :    integer;
                data              :    integer;
        end;

Datagrambak :
        record
                ipDestAddr        :    integer;
                data              :    integer;
        end;

User : ( Junki );

Junki : record
                dummy             :    integer;
        end;

Net : ( Junko );
```

```
        Junko : record
                        dummy           :       integer;
                end;

         identification : integer;
         careOfAddr     : integer;
         careOfAddr1    : integer;
         bindno         : integer;

channel SPchannel(other, Net);
        by  other:
            Junki;

        by  Net :
            Junko;

channel PDUchannel  ( pdu, other );
        by pdu:
           RegRequest ;
           DeRegRequest ;
           RegReply ;
           Datagram ;
           Datagrambak ;
           Advertisement ;
```

```
TIMER
          ad_timer     600;

state
        REGISTERED, IDLE ;

        { initialization part }

initialize   to   IDLE
     begin
             bindno         := 0;
             identification := 0;
             careOfAddr     := 0

     end;

trans

FROM IDLE                                                { 1 }
    TO IDLE

        provided timeout(ad_timer)
       begin
```

```
        Advertisement.ipDestAddr := localNetAddr;
        Advertisement.ipTTL  := 1;
        Advertisement.icmpLifetime := AdLifetime ;
        Advertisement.icmpCode := 1;
        Advertisement.extRegLifetime := RegLargestLifetime;
        Advertisement.extH := 1;
        output pdu.Advertisement;
    end;

FROM IDLE                                                    { 2 }
   TO REGISTERED

      when pdu.RegRequest
      provided
             (RegRequest.mipLifetime > 0 )
             and
             (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
             and
             (RegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
             and
             (RegRequest.mipIdentification > identification )
             and
             (RegRequest.ipDestAddr = mipHomeAgent )
      begin
```

```
RegReply.ipSourceAddr := mipHomeAgent;
RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
RegReply.mipType      := 3;
RegReply.mipCode      := mipS;
RegReply.mipLifetime  := RegRequest.mipLifetime-1;
RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
RegReply.mipIdentification:=RegRequest.mipIdentification+1;
RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
identification := RegRequest.mipIdentification +1 ;
if (mipS = 0)
then
  begin
    bindno := bindno + 1;
    if (bindno = 1)
    then  careOfAddr := RegRequest.mipCOA
    else
      careOfAddr1 := RegRequest.mipCOA;
  end
  else begin
        careOfAddr := RegRequest.mipCOA;
        bindno := 1;
      end;
```

```
        output pdu.RegReply;
      end;

FROM IDLE                                                    { 3 }
    TO IDLE
      when pdu.Datagram
      begin    end;

FROM IDLE                                                    { 4 }
    TO IDLE
      when pdu.RegRequest
      provided
            ( RegRequest.ipDestAddr = localNetAddr)
      begin
        RegReply.ipSourceAddr := mipHomeAgent;
        RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
        RegReply.mipType      := 3;
        RegReply.mipLifetime  := RegRequest.mipLifetime-1;
        RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
        RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipCode      := 136;
        RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
```

```
        identification := RegRequest.mipIdentification+1;
        output pdu.RegReply;
      end;

FROM REGISTERED                                              { 5 }
    TO IDLE

      when pdu.DeRegRequest
      provided
                (DeRegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
                and
                (DeRegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
                and
                (DeRegRequest.ipDestAddr = mipHomeAgent )
     begin
       RegReply.ipSourceAddr := mipHomeAgent;
       RegReply.ipDestAddr   := DeRegRequest.ipSourceAddr;
       RegReply.mipType      := 3;
       RegReply.mipCode      := 0;
       RegReply.mipLifetime  := 0;
       RegReply.mipHomeAddr  := DeRegRequest.mipHomeAddr;
       RegReply.mipHomeAgent := DeRegRequest.mipHomeAgent;
       RegReply.mipIdentification:=DeRegRequest.mipIdentification +1;
       identification        := DeRegRequest.mipIdentification;
```

```
        bindno := bindno - 1;
        RegReply.mipMHauthExtSPI :=DeRegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=DeRegRequest.mipMHauthExtAuth;
        if (DeRegRequest.mipCOA = mipNodeHomeAddr)
        then
          begin
              careOfAddr := 0;
              careOfAddr1:= 0;
          end
        else
           begin
               if ( careOfAddr = DeRegRequest.mipCOA)
               then   careOfAddr := 0;
                   if ( careOfAddr1 = DeRegRequest.mipCOA )
                   then   careOfAddr1 := 0;
           end;
        output pdu.RegReply;
    end;

FROM REGISTERED                                              { 6 }
     TO REGISTERED

     provided timeout(ad_timer)
     begin
```

```
        Advertisement.ipDestAddr := localNetAddr;
        Advertisement.ipTTL  := 1;
        Advertisement.icmpLifetime := AdLifetime ;
        Advertisement.icmpCode := 1;
        Advertisement.extRegLifetime := RegLargestLifetime;
        Advertisement.extH  := 1;
        output pdu.Advertisement;
      end;

FROM REGISTERED                                          { 7 }
    TO REGISTERED

        when pdu.Datagram
        provided ( bindno <> 0 )
        begin
                Datagram.ipDestAddr := careOfAddr;
                Datagram.data := Datagram.data;
                output pdu.Datagram;
        end;

FROM REGISTERED                                          { 8 }
    TO REGISTERED

        when pdu.Datagram
```

```
        provided ( bindno = 0 )
              AND (Datagram.ipDestAddr = mipNodeHomeAddr)
        begin
              if ( router = 0 )
              then
                begin
                 Datagram.ipDestAddr := 0;
                 Datagram.data := 0;
                end
              else
                Datagram.ipDestAddr := mipNodeHomeAddr;
                output pdu.Datagram;
        end;

FROM REGISTERED                                              { 9 }
    TO REGISTERED

        when pdu.Datagram
        provided (Datagram.ipDestAddr = mipNodeHomeAddr )
                 AND  ( bindno = 1)
        begin
             Datagram.ipDestAddr := careOfAddr ;
             output pdu.Datagram;
        end;
```

```
FROM REGISTERED                                                { 10 }
    TO REGISTERED

       when pdu.Datagram
       provided (Datagram.ipDestAddr = mipNodeHomeAddr )
                AND ( bindno = 2)
          begin
               Datagram.ipDestAddr := careOfAddr ;
               Datagrambak.ipDestAddr := careOfAddr1;
               Datagrambak.data := Datagram.data;
               output pdu.Datagram;
       output pdu.Datagrambak;
          end;

FROM REGISTERED                                                { 11 }
    TO REGISTERED

       when pdu.RegRequest
       provided
             (RegRequest.mipMHauthExtSPI <> mipMHauthExtSPI)
             OR
             (RegRequest.mipMHauthExtAuth  <>  mipMHauthExtAuth)
      begin
```

```
        RegReply.ipSourceAddr := RegRequest.ipDestAddr
        RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
        RegReply.mipType      := 3;
        RegReply.mipCode      := 131;
        RegReply.mipLifetime  := RegRequest.mipLifetime-1;
        RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
        RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipMHauthExtSPI := 0;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
        identification := RegRequest.mipIdentification+1;
        output pdu.RegReply;
      end;

FROM IDLE                                                    { 12 }
    TO IDLE
       when pdu.RegRequest
       provided RegRequest.mipLifetime < 0
       begin    end;

FROM REGISTERED                                              { 13 }
    TO REGISTERED
       when pdu.RegRequest
       provided RegRequest.mipLifetime < 0
```

```
    begin   end;

FROM IDLE                                                    { 14 }
    TO IDLE

      when pdu.RegRequest
      provided
           (RegRequest.mipIdentification < identification )
      begin
        RegReply.mipCode      := 133333;
        RegReply.mipLifetime  := RegRequest.mipLifetime-1;
        RegReply.mipIdentification:=RegRequest.mipIdentification+1;
        RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
        RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
        identification := RegRequest.mipIdentification+1;
        output pdu.RegReply;
      end;

FROM REGISTERED                                              { 15 }
    TO REGISTERED

      when pdu.RegRequest
       provided
            (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
```

```
        AND
        (RegRequest.mipMHauthExtAuth = mipMHauthExtAuth)
        AND
        (RegRequest.mipIdentification > identification )
        AND
        (RegRequest.mipLifetime > 0 )
        AND
        ( mipS = 0 )
        AND ( bindno = 1 )
    begin
      RegReply.ipSourceAddr := RegRequest.ipDestAddr;
      RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
      RegReply.mipType      := 3;
      RegReply.mipCode      := mipS;
      RegReply.mipLifetime  := RegRequest.mipLifetime-1;
      RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
      RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
      RegReply.mipIdentification:=RegRequest.mipIdentification+1;
      RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
      RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
      identification := RegRequest.mipIdentification+1;
      bindno := bindno + 1;
      careOfAddr1 := RegRequest.mipCOA;
      output pdu.RegReply
```

```
        end;

FROM REGISTERED                                                { 16 }
    TO REGISTERED

       when pdu.RegRequest
       provided
             (RegRequest.mipIdentification < identification )
       begin
         RegReply.ipSourceAddr := RegRequest.ipDestAddr;
         RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
         RegReply.mipType      := 3;
         RegReply.mipCode      := 133;
         RegReply.mipLifetime  := RegRequest.mipLifetime-1;
         RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
         RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
         RegReply.mipIdentification:=RegRequest.mipIdentification+1;
         RegReply.mipMHauthExtSPI :=RegRequest.mipMHauthExtSPI;
         RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
         identification := RegRequest.mipIdentification+1;
         output pdu.RegReply;
       end;

FROM IDLE                                                      { 17 }
```

```
    TO IDLE

        when pdu.RegRequest
        provided
             not (RegRequest.mipMHauthExtSPI = mipMHauthExtSPI)
             or
             not (RegRequest.mipMHauthExtAuth  = mipMHauthExtAuth)
        begin
          RegReply.ipSourceAddr := RegRequest.ipDestAddr;
          RegReply.ipDestAddr   := RegRequest.ipSourceAddr;
          RegReply.mipType      := 3;
          RegReply.mipCode      := 131;
          RegReply.mipLifetime  := RegRequest.mipLifetime-1;
          RegReply.mipHomeAddr  := RegRequest.mipHomeAddr;
          RegReply.mipHomeAgent := RegRequest.mipHomeAgent;
          RegReply.mipIdentification:=RegRequest.mipIdentification+1;
          RegReply.mipMHauthExtSPI := 0;
          RegReply.mipMHauthExtAuth:=RegRequest.mipMHauthExtAuth;
          identification := RegRequest.mipIdentification+1;
          output pdu.RegReply;
        end;
end.
```

Appendix D

ASN.1 specification of message parameters for the complete Home Agent

Home Agent processes five types of messages: Registration Request, DeRegistration Request, Registration Reply, Advertisement and datagram. We specify these messages in ASN.1 to describe their whole parameters.

```
HomeAgent  DEFINITIONS ::=

BEGIN

PduMessage ::= CHOICE
       {
           RegRequest,
           DeRegRequest,
           RegReply,
           Datagram,
           Advertisement
       }


RegRequest ::= SEQUENCE
```

```
{
      -- IP fields

ipSourceAddr      INTEGER,  -- addr. from which the msg is sent
ipDestAddr        INTEGER,  -- addr. of foreign/home agent

      -- UDP fields

udpSourcePort     INTEGER,  -- variable
udpDestPort       INTEGER,  -- constant 434

      --  Mobile IP fields

mipType           INTEGER,  -- set to 1 (registration request)
mipS              INTEGER,  -- 1: simultaneous bindings
mipB              INTEGER,  -- 1: broadcase datagrams required
mipD              INTEGER,  -- 1: decapsulation by mobile node
mipM              INTEGER,  -- 1: minimal encapsulation
mipG              INTEGER,  -- 1: GRE encapsulation
mipV              INTEGER,  -- 1: Van Jacobson header compression
mipRSV            INTEGER,  -- reserved bits, sent as 0
mipLifetime       INTEGER,  -- time remaining before expired
mipHomeAddr       INTEGER,  -- IP addr. of mobile node
mipHomeAgent      INTEGER, --IP addr. of mobile node's home agent
```

```
mipCOA              INTEGER,  -- IP addr of the end of the tunnel
mipIdentification INTEGER,--number constructed by the mobile node

        -- Mobile-Home Authentication Extension

mipMHauthExtType    INTEGER,  -- 32
mipMHauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMHauthExtAuth    INTEGER,  -- Authenticator, keyed MD5

        -- Mobile-Foreign Authentication Extension,  optional

mipMFauthExtType    INTEGER,  -- 33
mipMFauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMFauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMFauthExtauth    INTEGER,  -- authenticator, keyed MD5

        -- Foreign-Home Authentication Extension,  optional

mipFHauthExtType    INTEGER,  -- 34
mipFHauthExtLength  INTEGER,  -- 4 + length of authenticator
mipFHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipFHauthExtAuth    INTEGER   -- Authenticator, keyed MD5
 }
```

```
DeRegRequest ::= SEQUENCE
      {
            -- IP fields

      ipSourceAddr      INTEGER,  -- addr. from which the msg is sent
      ipDestAddr        INTEGER,  -- addr. of foreign/home agent

            -- UDP fields

      udpSourcePort     INTEGER,  -- variable
      udpDestPort       INTEGER,  -- constant 434

            --  Mobile IP fields

      mipType           INTEGER,  -- set to 1 (registration request)
      mipS              INTEGER,  -- 1: simultaneous bindings
      mipB              INTEGER,  -- 1: broadcase datagrams required
      mipD              INTEGER,  -- 1: decapsulation by mobile node
      mipM              INTEGER,  -- 1: minimal encapsulation
      mipG              INTEGER,  -- 1: GRE encapsulation
      mipV              INTEGER,  -- 1: Van Jacobson header compression
      mipRSV            INTEGER,  -- reserved bits, sent as 0
```

```
mipHomeAddr        INTEGER,  -- IP addr. of mobile node
mipHomeAgent       INTEGER, --IP addr. of mobile node's home agent
mipCOA             INTEGER,  -- IP addr of the end of the tunnel
mipIdentification INTEGER,--number constructed by the mobile node

        -- Mobile-Home Authentication Extension

mipMHauthExtType    INTEGER,  -- 32
mipMHauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMHauthExtAuth    INTEGER,  -- Authenticator, keyed MD5

        -- Mobile-Foreign Authentication Extension,  optional

mipMFauthExtType    INTEGER,  -- 33
mipMFauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMFauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMFauthExtauth    INTEGER,  -- authenticator, keyed MD5

        -- Foreign-Home Authentication Extension,  optional

mipFHauthExtType    INTEGER,  -- 34
mipFHauthExtLength  INTEGER,  -- 4 + length of authenticator
mipFHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
```

```
      mipFHauthExtAuth     INTEGER    -- Authenticator, keyed MD5
       }



RegReply  ::= SEQUENCE

      {
            -- IP fields

      ipSourceAddr         INTEGER,  -- addr. from which the msg is sent
      ipDestAddr           INTEGER,  -- addr. of foreign/home agent

            -- UDP fields

      udpSourcePort        INTEGER,  -- variable
      udpDestPort          INTEGER,  -- constant 434

            --  Mobile IP fields

      mipType              INTEGER,  -- set to 3 (registration request)
```

```
mipCode             INTEGER,  -- the result of registration request
mipLifetime         INTEGER,  -- time remaining before expired
mipHomeAddr         INTEGER,  -- IP addr. of mobile node
mipHomeAgent        INTEGER, -- IP addr. of mobile node's home agent
mipIdentification   INTEGER, --number constructed by the mobile node

        -- Mobile-Home Authentication Extension

mipMHauthExtType    INTEGER,  -- 32
mipMHauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMHauthExtAuth    INTEGER,  -- Authenticator, keyed MD5

        -- Mobile-Foreign Authentication Extension,  optional

mipMFauthExtType    INTEGER,  -- 33
mipMFauthExtLength  INTEGER,  -- 4 + length of authenticator
mipMFauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
mipMFauthExtauth    INTEGER,  -- authenticator, keyed MD5

        -- Foreign-Home Authentication Extension,  optional

mipFHauthExtType    INTEGER,  -- 34
mipFHauthExtLength  INTEGER,  -- 4 + length of authenticator
```

```
      mipFHauthExtSPI     INTEGER,  -- Security Parameter Index (4 Bytes)
      mipFHauthExtAuth    INTEGER   -- Authenticator, keyed MD5
       }


Advertisement ::= SEQUENCE
      {
          -- Link-Layer fields

      linkDestAddr        INTEGER,  -- link layer dest. addr.

          -- IP fields

      ipTTL               INTEGER,  --  IP field, must be set to 1
      ipDestAddr          INTEGER,  -- milticase addr. or
                                        -- limited broadcase addr.
          -- ICMP fields

      icmpCode            INTEGER,  -- 0: as a router too
                                        -- 1: not handle common traffic
      icmpLifetime        INTEGER,  -- life time of valid advertisemt
                                        -- interval = 1/3 lifetime
      icmpRouterAddr      INTEGER,  --
      icmpNumAddrs        INTEGER,  -- may be set to 0
```

```
    -- Mobility Afent Advertisement Extension

extType             INTEGER,  -- 16
extLength           INTEGER,  -- 6 + 4 * N
                              -- N = No. Of Care of Addresses
extSeqNum           INTEGER,  -- 0 .. 0xffff - 256
extRegLifetime      INTEGER,  -- longest time acceptable
extR                INTEGER,  -- 1: for FA, Reg. required
extB                INTEGER,  -- 1: busy
extH                INTEGER,  -- 1: this is a home agent
extF                INTEGER,  -- 1: this is a foreign agent
extM                INTEGER,  -- 1: minimal encapsulation
extG                INTEGER,  -- 1: GRE encapsulation
extV                INTEGER,  -- 1:Van Jacobson header compression
extReserved         INTEGER,  -- reserved, 0
extCOA              INTEGER   -- for foreign agent, Care Of Addr.

    -- Prefix-Lengths Extension, optional

PLextType           INTEGER,  -- 19
PLextLength         INTEGER,  -- length
PLextPrelen         INTEGER,  -- Prefix Length(s)
```

```
            -- One-byte Padding Extension, optional

      OBType              INTEGER   -- 0
      }


Datagram ::= SEQUENCE
        {     -- there is not datagram format in this document
              -- here only necessary/related  fields are declared

           ipDestAddr     INTEGER,
           data           INTEGER
        }
END
```

Appendix E

Selected Control Sequences and Test Cases

There are 529 test cases generated by TESTGEN, in which there are 7 control sequences. We feed the 7 control sequences into TESTSEL, then we got 5 control sequences with the coverage 94.5% and the cost 12. The 7 control sequences and the 5 control sequences are as the follows:

1.

- Advertisement - RegRequest RegReply - - Advertisement - Datagram Datagram - RegRequest RegReply - DeRegRequest RegReply -

2.

RegRequest RegReply - - Advertisement - Datagram Datagram - RegRequest RegReply - DeRegRequest RegReply - - Advertisement -

3.

RegRequest RegReply - - Advertisement - RegRequest RegReply - Datagram Datagram - DeRegRequest RegReply - - Advertisement -

4

RegRequest RegReply - Datagram Datagram - - Advertisement - RegRequest RegReply - DeRegRequest RegReply - - Advertisement -

5.

RegRequest RegReply - Datagram Datagram - RegRequest RegReply - - Advertisement - DeRegRequest RegReply - - Advertisement -

```
6.
 RegRequest  RegReply  -  RegRequest  RegReply  -  -  Advertisement  -
 Datagram  Datagram  -  DeRegRequest  RegReply  -  -  Advertisement  -
7.
 RegRequest  RegReply  -  RegRequest RegReply  - Datagram  Datagram  -
 -  Advertisement  -  DeRegRequest  RegReply  -  -  Advertisement  -
```

The parameters used in TESTSEL are:

Eps = 1.000000, Min_Eps = 0.100000, Scale = 0.4, MaxCost = 20

The selected control sequences are:

```
Phase = 0, TC Id. = 0
(- Advertisement -, 1)(RegRequest RegReply -, 1)(- Advertisement -, 1)
(Datagram Datagram -, 1)  (RegRequest RegReply -, 1)
(DeRegRequest RegReply -, 1)

Phase = 1, TC Id. = 2
(RegRequest RegReply -, 1)(- Advertisement -, 1)
(RegRequest RegReply -, 1)  (Datagram Datagram -, 1)
(DeRegRequest RegReply -, 1)(- Advertisement -, 1)

Phase = 1, TC Id. = 36
(RegRequest RegReply -, 2)  (Datagram Datagram -, 1)
(- Advertisement -, 1)  (DeRegRequest RegReply -, 1)
```

```
(- Advertisement -, 1)

Phase = 2, TC Id. = 1
(RegRequest RegReply -, 1)  (- Advertisement -, 1)
(Datagram Datagram -, 1)  (RegRequest RegReply -, 1)
(DeRegRequest RegReply -, 1)(- Advertisement -, 1)

Phase = 2, TC Id. = 19
(RegRequest RegReply -, 1)(Datagram Datagram -, 1)
(RegRequest RegReply -, 1) (- Advertisement -, 1)
(DeRegRequest RegReply -, 1)  (- Advertisement -, 1)

Coverage = 0.945312  with Cost : 12
```

```
**   0

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 200 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   1

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 200 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
```

```
**   2

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 200 150 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   3

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 250 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
```

```
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   4

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 250 150 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   5

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 20 200 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
```

```
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   6

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 20 200 150 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   7

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 20 250 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
```

```
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   8

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 20 250 150 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -

DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   9

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
```

```
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   10

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 /
RegReply 821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 150 /
RegReply 821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 /
RegReply 821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -

**   11

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - -
- - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 250 100 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - -
```

```
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   12

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 250 150 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   13

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 20 200 100 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
```

```
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   14

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 20 200 150 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   15

RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 20 250 100 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
```

```
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -

**   16
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
RegRequest 8285 821045 1 1 10 821012 821045 821020 20 250 150 / RegReply
821045 8285 3 133 9 821012 821045 6 200 100 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
**   17
RegRequest 8285 821045 1 1 10 821012 821045 821020 5 200 100 / RegReply
821045 8285 3 1 9 821012 821045 6 200 100 / - - - - - - - -
RegRequest 8285 82832 1 1 10 821012 821045 821020 5 200 100 / RegReply
82832 99 3 131 98 99 99 21 0 150 / - - - - - - - -
Datagram 1 1 - - - - / Datagram 821020 1 - - - - / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
DeRegRequest 821012 821045 1 1 821012 821045 0 50 200 100 / RegReply
821045 821012 3 0 0 821012 821045 51 200 100 / - - - - - - - -
- - - - - - - - /Advertisement 82832 1 18000 1 10 1 / - - - - - - - -
```

Appendix F

4 Typical Control Sequences and Typical test cases in TTCN.MP

1. Datagram - - RegRequest RegReply - - Advertisement -
   RegRequest RegReply - DeRegRequest RegReply -
2. RegRequest RegReply - - Advertisement - Datagram Datagram -
   DeRegRequest RegReply - RegRequest RegReply -
3. RegRequest RegReply - Datagram Datagram - -
   Advertisement - DeRegRequest RegReply - RegRequest RegReply -
4. RegRequest RegReply - RegRequest RegReply - -
   Advertisement - DeRegRequest RegReply - Datagram - -

Test Cases in TTCN.MP:

```
$Suite
  $SuiteId   suite0001
  $DeclarationsPart
      $Begin_TS_ConstDcls
       $TS_ConstDcl
        $TS_ConstId  mipMHauthExtAuth
        $TS_ConstType  INTEGER
        $TS_ConstValue  100
       $End_TS_ConstDcl
```

```
$TS_ConstDcl
 $TS_ConstId  mipMHauthExtSPI
 $TS_ConstType  INTEGER
 $TS_ConstValue  200
$End_TS_ConstDcl
...
   $Begin_ASN1_PDU_TypeDef
     $PDU_Id  DeRegRequest
     $PCO_TypeId
     $Comment
     $ASN1_TypeDefinition
       SEQUENCE
       {
       ipSourceAddr  INTEGER,
       ipDestAddr  INTEGER,
       mipType  INTEGER,
       mipS  INTEGER,
       mipHomeAddr  INTEGER,
       mipHomeAgent  INTEGER,
       mipCOA  INTEGER,
       mipIdentification  INTEGER,
       mipMHauthExtSPI  INTEGER,
       mipMHauthExtAuth  INTEGER
       }
```

```
      $End_ASN1_TypeDefinition
    $End_ASN1_PDU_TypeDef
...
$Begin_ASN1_PDU_Constraint
  $ConsId  RegRequest_62
  $PDU_Id  RegRequest
  $DerivPath
  $ASN1_ConsValue
    {
    ipSourceAddr   8285 ,
    ipDestAddr   821045 ,
    mipType   1 ,
    mipS   1 ,
    mipLifetime   10 ,
    mipHomeAddr   821012 ,
    mipHomeAgent   821045 ,
    mipCOA   821020 ,
    mipIdentification   5 ,
    mipMHauthExtSPI   200 ,
    mipMHauthExtAuth   100
    }
$End_ASN1_PDU_Constraint
...
$Begin_TestCase
```

```
$TestCaseId test30
$TestPurpose test_test30
$BehaviourDescription
  $End_BehaviourDescription
   $BehaviourLine
    $Label
    $Line [4] ! RegReply
    $Cref  RegReply_215
    $Verdict
    $Comment
            {
             ipSourceAddr   821045 ,
             ipDestAddr   821012 ,
             mipType   3 ,
             mipCode   0 ,
             mipLifetime   0 ,
             mipHomeAddr   821012 ,
             mipHomeAgent   821045 ,
             mipIdentification   51 ,
             mipMHauthExtSPI   200 ,
             mipMHauthExtAuth   100 ,
            }
   $End_BehaviourLine
   $BehaviourLine
```

```
            $Label
            $Line [5] ? RegRequest
            $Cref  RegRequest_216
            $Verdict
            $Comment
                    {
                     ipSourceAddr   8285
                     ipDestAddr   82832
                     mipType   1
                     mipS   1
                     mipLifetime   10
                     mipHomeAddr   821012
                     mipHomeAgent   821045
                     mipCOA   821020
                     mipIdentification   20
                     mipMHauthExtSPI   250
                     mipMHauthExtAuth   100 }
          $End_BehaviourLine
          ...
       $End_BehaviourDescription
     $End_TestCase
   $End_TestCases
 $End_DynamicPart
$End_Suite
```