ABSTRACT

PROTOCOL CONFORMANCE TEST GENERATION USING CIRCULAR UIO WITH OVERLAPPING

by

Sesharao Patchipala

The purpose of the protocol conformance testing is to ensure that protocol implementations are consistent with their specifications. After the U-method was introduced, several test methods based on the Unique Input/Output (UIO) sequences which were the main concept in the U-method have been proposed, namely, the RCP-method, MUIO-method, MUIO with overlapping method, B-method, C-method. A good test sequence must be short and have wide fault coverage. By comparing the test sequences generated by the above test methods based on experimental results, reveals that the test sequences by the MUIO with overlapping method are the worst in quality but their lengths are the shortest in general. In this paper, code is implemented for UIO sequences and a new method, the C-UIO with overlapping method which combines the merits of both methods, is presented and compared with other methods based on the experimental results.

PROTOCOL CONFORMANCE TEST GENERATION USING CIRCULAR UIO WITH OVERLAPPING

by

Sesharao Patchipala

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Department of Computer & Information Science

January, 1993

APPROVAL PAGE

Protocol Conformance Test Generation Using Circular UIO with Overlapping

Sesharao Patchipala

Dr. Daniel Yuh Chao, Thesis Adviser
Assistant Professor of Computer and Information Science,
NJIT

David Wang
Assistant Professor of Computer and Information Science,
NJIT

Dung Dao Chuan
Assistant Professor of Computer and Information Science,
NJIT

BIOGRAPHICAL SKETCH

Author: Sesharao Patchipala

Degree: Master of Science in Computer and Information Science

Date: January, 1993

Date of Birth:

Place of Birth

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1993
- Bachelor of Engineering in Mechanical Engineering, Osmania University, Hyderabad, India, 1990

Major: Computer Science

ACKNOWLEDGEMENT

I would like to thank my Thesis Advisor, Dr. Daniel Yuh Chao for his help and guidance in completing this thesis. His constructive criticism coupled with the time he spent every week on the thesis helped me a great deal in doing this thesis. The group meetings and the presentations held every week helped to improve my knowledge in the field of Communication Protocols and X Windows.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

A communication protocol is a set of rules which define all possible interactions among the communicating entities. A protocol standard, in general, can lead to different implementations which necessitates the needs for conformance testing of an implementation to its standard. A protocol specification (the control portion) is typically modeled as a finite state machine (FSM). An FSM is represented by a directed graph called *transition diagram.* A directed edge labeled x,y in the FSM represents a state transition which outputs the symbol y if and only if the input symbol is x. Its starting and ending points respresent the starting and the ending states of the transition respectively Figure 1.

A common approach to conformance testing is to apply a sequence of inputs to the *Implementation Under Test* (IUT) and observe if the output sequence is the desired output sequence. The sequence of input and desired output pairs is called a *test sequence.* However, the general conformance testing problem is an unsolvable problem. In other words, there is no algorithm which can generate a test sequence for a given FSM such that any faulty IUT would be identified by this sequence. Despite the difficult nature of the problem, people use heuristic methods to generate test sequences which would catch errors in the IUT with high probability. Among them, the U-method is a milestone which greatly shortens the test lengths. Based on the U-method, the RCP-method, MUIO-method, MUIO with overlapping method, B-method, C-method have recently been proposed. As we know, the length and quality (fault coverage) of a test sequence are the two main factors which affect the applications of these testing techniques. The experimental results show that the C-method produces test sequences with the best quality, while the MUIO with overlapping method generates the poorest sequences. However, the overlapping technique greatly shortens test sequences. It is interesting to know if we can integrate the overlapping technique into the C-method to reduce the lengths of test sequences, while maintaining the high quality.

In this paper, we introduce a new protocol generation method called the C-UIO *with overlapping method.* A general procedure for generating test sequences is presented. Our experimental results show that the fault coverage of the test sequences by the new method is competitive with that by the C-method and the lengths are shortened by about 10% - 30% compared with the original C-method.

The rest of this paper is organized as follows. Chapter 2 reviews two test generation methods and introduces some related notations and terminologies. Chapter 3 presents the C-UIO with overlapping method. Chapter 4 gives an introduction to X Window system and introduces some related notations and terminologies. Chapter 5 describes about an experiment on the test sequences. Chapter 6 gives detailed analysis of the experimental results. Chapter 7 concludes this paper.

CHAPTER 2

Overview of the Test Methods

2.1 Preliminaries

An n-state FSM can be represented by a labeled directed graph with n vertices, G = (V,E), which is called transition diagram. The graph is always assumed to be strongly connected and minimal. An example of an FSM is shown in Figure 1, where state 1 is the initial state.

All test generation methods follow a common scheme which is to test each edge in an FSM one by one in a systematic manner. Thus, a test sequence consists of many subsequences, one for each edge. Such a subsequence called an *edge-sequence* consists of three parts. The first part is to lead the IUT to the starting state, the second part is the input/output label on this edge, and the third part is used to verify the ending state. A protocol test sequence is the concatenation of all the edge-sequences.

A UIO *sequence of a state* is an input/output behavior which cannot be observed except that it is applied to this state. Because of the uniqueness, a UIO sequence of a state can be used to verify this state. For example, b,e a,b is a UIO sequence for state 1 in Figure 1. A UIO sequence is also called an F-UIO (*forward UIO*) sequence for distinguishing from a B-UIO (*backward UIO*) sequence discussed later. A state may have more than one shortest UIO sequence as shown in Table 1, where the complete set of shortest UIO sequences for the FSM in Figure 1 is given.

A *B-UIO sequence* for a state is an input/output behavior which can be observed only if the corresponding state transitions end up at this state. B-UIO sequences for the FSM in Figure 1 are shown in Table 2.

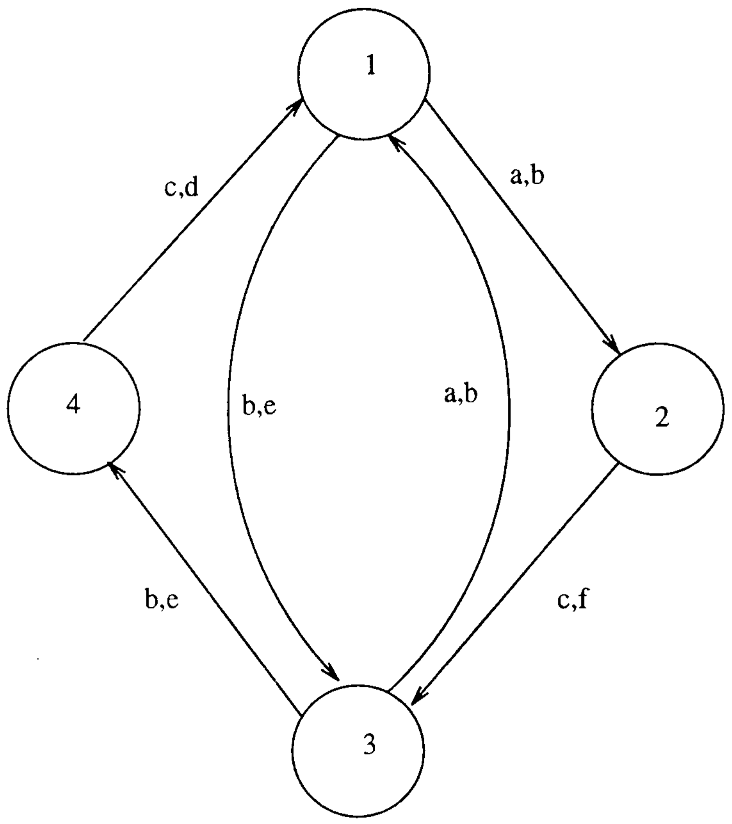

Figure 1. A transition diagram for an FSM.

Table 1. The complete set of shortest F-UIO sequences for the FSM in Figure 1.

state	F-UIO Sequence	Tail State
1	b,e a,b	1
1	a,b c,f	3
1	b,e b,e	4
2	c,f	3
3	b,e c,d	1
3	a,b a,b	2
3	a,b b,e	3
4	c,d	1

Table 2. B-UIO sequences for the FSM in Figure 1.

state	B-UIO Sequence	Starting State
1	c,d	4
2	a,b a,b	3
3	c,f	2
4	b,e b,e	1

A *circular UIO (C-UIO) sequence* for a state is an F-UIO sequence for this state followed by a B-UIO sequence for the same state. The C-UIO sequences for the FSM in Figure 1 are shown in Table 3. Because the starting and the ending states are the same for a C-UIO sequence, it will go back to the same state after an application to this state.

Note that a state will be verified twice each time when a C-UIO for this state is applied while it is verified only once by an F-UIO sequence in other methods.

Table 3. The F-UIO, B-UIO, and C-UIO sequences for the FSM in Figure 1.

State	F-UIO Sequence	B-UIO sequence	C-UIO sequence
1	b,e b,e	c,d	b,e b,e c,d
2	c,f	a,b a,b	c,f a,b a,b
3	a,b a,b	c,f	a,b a,b c,f
4	c,d	b,e b,e	c,d b,e b,e

2.2 The MUIO with/without Overlapping Method

The general concept of using UIO sequence is illustrated in Figure 2, where edge from v_i to v_j is a state transition and the UIO_j sequence is used to verify the ending state v_j.

The multiple UIO (MUIO) method allows each edge to choose one F-UIO sequence from several choices such that the length of the overall test sequence wil be minimized. It is observed that some edge sequences are completely contained in one (or a sequence) of other test sequences. The basic idea of MUIO with overlapping method is to select an F-UIO sequence for each edge in G such that maximum overlap occurs among the resulting edge sequences.

Taking the FSM in Figure 1 as an example, Figure 3 shows the six edges together with their F-UIOs and the overlapping patterns.

After eliminating the overlapped parts, we obtain the following test sequence: b,e b,e c,d a,b c,f a,b b,e a,b which contains only 8 pairs.

2.3 The C-method

Based on C-UIO sequences, the C-method is defined as the following test generation procudure:

Step 1. Compute the complete set S of the shortest F-UIO sequence for each state *v*, using the algorithm in the U-method.

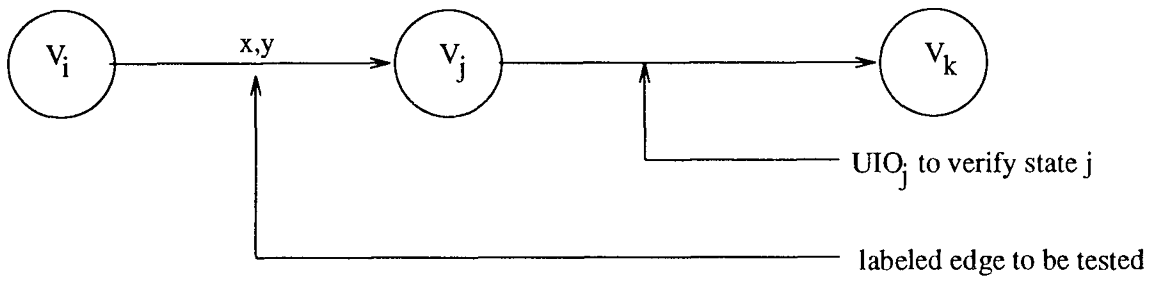

Figure 2. General usage of a UIO sequence.

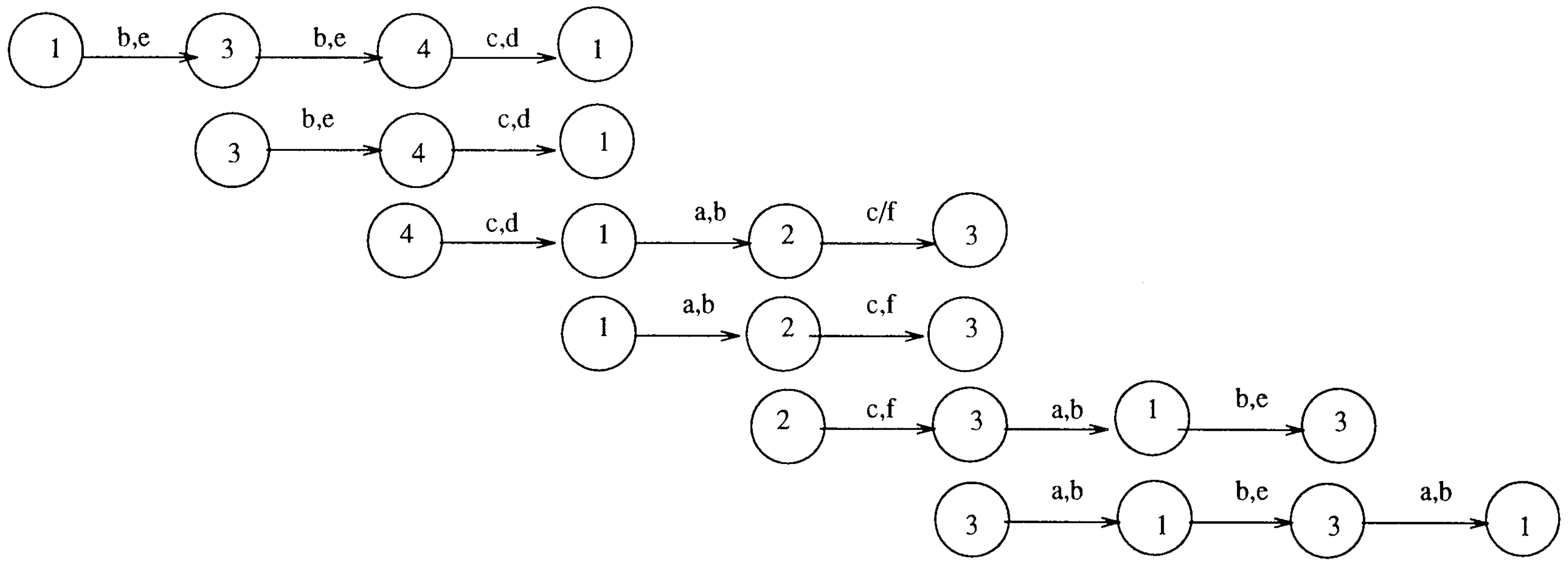

Figure 3. An illustraion of the overlapped edge sequences for the FSM in Figure 1.

Step 2. For each shortest F-UIO sequence in S which begins at state *v* and ends at state *x*, compute the shortest B-UIO sequence from *x* to *v*, using the algorithm in the B-method, to obtain a C-UIO sequence for the state *v*.

Step 3. Choose the shortest among all the C-UIO sequences for state *v*.

Step 4. Compute a *Chinese Postman tour* (CP-tour) for the FSM. A Chinese Postman tour is a tour on the graph which passes each edge at least once, minimizing the total length of the tour.

Step 5. Insert a corresponding C-UIO sequence after each state transition along the CP-tour.

For example, a CP-tour for the FSM in Figure 1 is b,e a,b a,b c,f b,e c,d, and the test sequence by the C-method is b,e C3 a,b C1 a,b C2 c,f C3 b,e C4 c,d C1 , where C_i $(1 < C_i < 4)$ represents the C-UIO sequence for state *i*. This sequence contains 24 pairs: b,e a,b a,b c,f a,b b,e b,e c,d a,b c,f a,b a,b c,f a,b a,b c,f b,e c,d b,e b,e c,d b,e b,e c,d.

CHAPTER 3

The C-UIO with Overlapping Method

Faced with several protocol conformance test generation methods, which one generates the test sequence with the highest quality? conducted an experiment to evaluate several protocol conformace test methods. Several different FSMs were used. The experimental results show that the C-method generates the test sequences with the best fault coverage but the length of the test sequences are longer than the multiple UIO (with or without overlapping) methods. The MUIO with overlapping method generates the test sequences with the poorest quality, but the length of the test sequences is the shortest among the test sequences generated by the six methods. This gives us a hint: can we pick up the advantages from both test methods and obtain short test sequences with higher quality?

A new method, called the *circular UIO (C-UIO) with overlapping method*, tries to extract the merits and get rid of the drawbacks in these two methods. The C-UIO with overlapping method insists on using circular UIO sequence bacause it is the core idea of the C-method. A C-UIO sequence verifies the corresponding state twice with its F-UIO and B-UIO sequences, while other methods only verify each state once. That is the reason why test sequences generated by the C-method achieve the highest fault coverage than others. The C-UIO with overlapping method uses single C-UIO sequence for each edge.

The C-UIO with overlapping method uses the basic idea of overlapping technique but different construction procedure. We develop a transition sequence called *overlapped transition sequence* (**OTS**). A compound edge is defined as a state transition together with its C-UIO sequence. All compound edges in an FSM are marked as **UNVERIFIED** initially. The C-UIO with overlapping method first chooses one of the compound edges of the initial state as the initial transition sequence and marks it as **VERIFIED**. Then, in each subsequent iteration, it checks all remaining **UNVERIFIED** compound edges and attempts to find one which has the maximum overlapping with the current **OTS**.

The following procedure shows detailed steps of this method:

Step 1. Compute the set S of C-UIO sequences, one for each state.

Step 2. Each edge is appended with its C-UIO sequence to form a compound edge. Each compound edge is marked UNVERIFIED. Suppose there are m compound edges in the FSM.

Step 3. Pick one of compound edges starting from the initial state as the initial transition sequence (OTS) and mark it VERIFIED.

Step 4. For number_edge := 1 to m-1 do

Step 5. Search among the UNVERIFIED compound edges and try to find one which has the maximum match between the rear part of the OTS and the front part of the compound edge. Two cases will happen:

- If the maximum match is found, mark the corresponding compound edge VERIFIED and append the unmatched part to the OTS.
- If no match is found, extend the OTS to a starting state of a UNVERIFIED compound edge whose distance from the OTS is the shortest. Then append this compound edge to the OTS and mark it VERIFIED.

Step 6. Go back to the Step 4 if there is any UNVERIFIED compound edge.

Taking the FSM in Figure 1 as an example, the compound edges are shown in Table 4. First, we pick one of the compound edges of the initial state, b,e a,b a,b c,f as initial *transition sequence.*

Then, we compare the rest five **UNVERIFIED** compound edges and try to find the one which has the maximum match with the transition sequence. Here, by match, we mean that not only should the input/output pairs be identical but so are the states. Compound edge a,b c,f a,b a,b has been chosen because it has two edges which match the transition sequence. Next, the compound edge c,f a,b a,b c,f has been chosen. The compound edge has three edges which match the transition sequence.

Table 4. The compound edges for the FSM in Figure 1.

Edge Set	Compound edge E	Starting State	Tail State
a,b (1,2)	a,b c,f a,b a,b	1	2
b,e (1,3)	b,e a,b a,b c,f	1	3
c,f (2,3)	c,f a,b a,b c,f	2	3
a,b (3,1)	a,b b,e b,e c,d	3	1
b,e (3,4)	b,e c,d b,e b,e	3	4
c,d (4,1)	c,d b,e b,e c,d	4	1

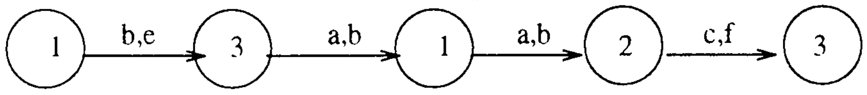

Figure 4. The initial transition sequence.

Now we could not find any compound edges which matches the rear part of the transition sequence. We append the compound edge b,e c,d b,e b,e to the OTS because the starting state of this compound edge (state 3) happens to be the tail of the OTS.

Finally, We get the test sequence b,e a,b a,b c,f a,b a,b c,f b,e c,d b,e b,e c,d a,b c,f a,b b,e b,e c,d which contains 18 pairs.

The length of the test sequence generated by the C-UIO with overlapping method is shorter compared with 24 pairs by the C-method in this example. It is slightly longer than the test sequence generated by the multiple UIO method (16 steps).

CHAPTER 4
X WINDOW ENVIRONMENT

4.1 INTRODUCTION

The X Window System is an industry-standard software system that allows programmers to develop portable graphical user interfaces. One of the most important features of X is its unique device-independent architecture. X allows programs to display windows containing test and graphics on any hardware that supports the X protocol without modifying, recompiling, or relinking the application. It is also network transparent, thus allowing software to reside on one network workstation and create screens on another network workstation that might have been made by a different manufacturer. The device independence, network transparency, along with X's position as an industry standard, allows X-based applications to function in a heterogeneous environment consisting of mainframes, workstations, and personal computers. To utilize these advantages to the maximum, the graphical user interface is developed using X windows.

The X Window System was designed as a distributed, network transparent, device independent, multi-tasking windowing and graphics system. It permits the display of multiple applications on the same screen, and allows multiple windows to be used by a single application. It also supports overlapping and hidden windows, text with soft fonts, and two dimensional graphics drawing. X window achieves some device independence by splitting the job of drawing windows into two parts using the familiar client-server model.

In X, a client-server term refers to a single process, known as server, which is responsible for all input and output devices, and a client, which is an application that makes use of the facilities provided by the server. The server creates and manipulates windows on the screen, produces text and graphics, and handles input devices such as the keyboard and mouse. The server provides a portable layer between all applications and the display hardware. The X server typically runs on a workstation or personal computer with a graphics display or on a dedicated X terminal that implements all or part of server via a network connection using a byte-stream protocol. X supports many network protocols, including TCP/IP, DECnet, Chaos, OSI protocols. Multiple clients can connect to a single server concurrently, and an individual client can connect to multiple servers.

4.2 Componets of the X Window System

4.2.1 X Client

The client further consists of several layers of software. At the lowest level are routines used to provide a functional interface to the X Window protocol. These routines are referred to as the X library or Xlib. The layer above Xlib provides routines that manage higher level user interface objects called Widgets. This layer is referred to as the toolkit layer. Above this layer exists the application layer.

4.2.2 X Server

The primary function of the X server is to act as an intermediary between users and X applications. The server performs the job of collecting input such as mouse movements, pointer device input, or keystrokes from a keyboard and passes this information to appropriate X client(s). It also handles output from X clients that are destined for video display or other clients (as in inter-client communications).

A single workstation may have several screens driven by the same server, or a single computer might run more than one server with different graphics terminals attached. More likely, each workstation will have its own X server. Since a single X server can service requests from many client applications, the screen might have several windows containing the output from different programs. The client programs might be running on the server machine or on several others on the network. Equally, programs running on the local machine or workstation may open windows on other workstations.

The X server's primary job is to share scarce resources among the client applications that request them. The two principal resources are processor time, for drawing and test manipulation, and screen space. An intermediary program, the window manager allots screen space. The server is responsible for scheduling work performed on behalf of the client programs, for memory management, and for such subsidiary processes as maintaining communication links with each client.

From the structural point of view, an X server consists of a device-independent layer that receives and translates client request messages in the X protocol format, an operating system-dependent layer that interfaces to a particular operating system, and a device-dependent layer that is a collection of device drivers for the specific hardware supported.

4.2.3 X protocol

X protocol requests are variable length data packets that begin with an 8-bit op code that identifies the type of request, followed by a 16-bit field specifying length, and one or more bytes of additional data. The added data might be numeric parameters or coordinates, text strings to write, or raw bit-map data in scan line order.

4.2.4 Xlib Layer

Xlib contains a library of graphics and windowing functions. Xlib contains about 300 routines that map to X protocol requests or provide utility functions. What Xlib actually does is convert the C language function calls to the X protocol request that implements the given function, such as XPutImage to draw an image. The functions include creating, destroying, moving and sizing windows; drawing lines and polygons; setting background patterns; and tracking the mouse. Xlib also allows you to access windows in a variety of ways, including overlapping and simultaneous output to multiple windows. It supports multiple fonts, common raster operations, line drawing, and both color and monochrome applications. The Xlib functions are available in C, Pascal, FORTRAN, modula-2, and Ada.

4.3 Xt Intrinsics Layer

Xt Intrinsics layer resides above the Xlib layer. It is made up of sets of prefabricated routines built out of Xlib routines, and is responsible for the management of higher level interface objects called widgets. Such widgets provide a specific user interface service and posses a distinctive visual appearance and a well defined response to user input. The Xt Intrinsics supports many different widget sets. The Xt Instrisics provides functions that allow an application to create, modify, and destroy widgets, and a framework that allows the programmer to combine these components to provide a complete user interface.

In a sense, the Xt Intrinsic layer is a part of the X Toolkit layer. The X Toolkit layer contains a set of predefined widgets. The Xt Intrinsic layer is considered as the low level application interface responsible for the widget management. The X Toolkit layer is considered as the high level application interface containing widget sets.

4.4 X Toolkit Layer

Motif provides widgets for user-interface elements including labels, pushbuttons, menus, dialog boxes, scroll bars, and text entry or display areas. There are also other widgets, so that the application does not have to worry about details of widget placement when the application is moved or resized. Motif supports independent development of new or extended widgets. All these widgets can be used either independently or in combination to create complex applications. A widget operates independently of the application, except through prearranged interactions.

Motif user-interface is created using Motif Xm library and the Intrinsics Xt library. Xt provides functions for creating and setting resources on widgets and Xm provides the widgets themselves, plus an array of utility and convenience functions for creating groups of widgets that are used collectively as a single type of user-interface element. For example, the Menu Bar is not implemented as one particular widget but as a collection of smaller widgets put together by one convenience function.

Writing Motif Widget Programs involves nine steps in sequence:

1. Include the required header files
2. Initialize Xt Intrinsics
3. Add additional toplevel windows
4. Setup argument list for widget
5. Create widget
6. Add callback routines
7. Realize widgets and loop
8. Link relevant libraries
9. Create defaults file

Steps 4-6 should be repeated for each widget.

4.5 Application Layer

Application layer contains the application which utilizes the widget sets available in the lower layer. The application can also create, modify, or destroy widgets. The application can also construct their own custom widgets using the sets of predefined routines available in the Xt Intrinsic layer.

4.6 Window Manager

Window manager is a special client that manages the positions and sizes of the main windows of the applications on a server's display. The window manager is just another client, but by convention it is given special responsibility to mediate competing demands for the physical resources of the display, including screen space, color resources, and the keyboard. The window manager allows the user to move windows around on the screen, resize them, and usually start new applications. The window manager also defines much of the visible behavior of the window system, such as whether windows are allowed to overlap or are forced to tile (side by side), or whether the keyboard focus simply follows the pointer from one window to the other, or whether the user must click a pointer button in a window to change the keyboard focus.

4.7 TERMINOLOGY

4.7.1 Windows

A window is a rectangular area on a workstation's video screen. They let the user view output, and they give application a rationale way of managing the use of space on the screen. Whenever an X application generates visible graphical output, it must specify a particular window to receive the output. Windows may overlap or windows may nest inside other windows.

4.7.2 Display

A dispaly is a set of one or more screens that are driven by a single X server. Before a client can communicate with the X server it must open a connection to the server.

4.7.3 Screen

A screen is a single hardware output device.

4.7.4 Event

An X event is a data structure sent by the server that describes something that just happened that may be of interest to the application. There are two major categories of events: user input and window system side effects. For example, the user presses a keyboard key or clicking a mouse button generates an event; a window being moved on the screen also generates events-possibly in other applications as well, if the movement changes the visible portions of their windows. It is the server's job to distribute events to the various windows on the screen.

4.7.5 Widget

A Widget is a basic object in a toolkit. A widget includes both code and data, and can therefore serve as an input or output object. Widgets consist of an X Window along with some procedures that operate on the window. Examples of widgets include pushbuttons, scrollbars, menus, and dialog boxes.

4.7.6 Gadget

Gadgets are identical to widgets, except that they have no windows of their own. The main objective of providing a set of gadgets is to improve the performance, both in execution time and data space. A gadget must display text or graphics in the window provided by its parent, and must also rely on its parent for input. Because reducing the number of windows in an application reduces the number of server requests, using gadgets can make an application much more efficient.

4.7.7 Callback

A callback is an application function registered with a widget by the application using either of the calls XtAddCallback or XtAddCallbacks or through an argument list. A widget declares one or more callbaks lists as resources; applications add functions to these lists in order to link widgets to application code.

CHAPTER 5

An experiment on the Test Sequences

The ability of a test sequence to decide whether a protocol implementation conforms to its specification solely relies upon the range of faults that it can capture. A common method to evaluate the fault coverage of a test sequence is to apply this sequence to IUTs with various faults to see how many such IUTs can be caught by this sequence.

We conducted an experiment to evaluate the MUIO with overlapping method, C-method and C-UIO with overlapping methods. Several different FSMs were used. For each FSM, the faulty IUTs were randomly generated by performing one or more following operations on the FSM. (Those IUTs which did not define an FSM were identified and discarded.)

- Randomly change the ending state(s) of one or more edges.
- Randomly change the starting state(s) of one or more edges.
- Randomly change the output(s) of one or more edges.
- Randomly add edge(s) between states and randomly assign the input/output labels too.

CHAPTER 6

Analysis of the Experimental Results

The experimental results show that the test sequences generated by the C-method or C-UIO with overlapping method achieve almost the same fault coverage.

On the other hand, the overlapping technique does shorten the test sequences. In general, the length of a test sequence by the C-UIO with overlapping method is shorter by 10% to 30% compared with that by the original C-method. The percentage of reduction depends on how complex the FSM is.

The complexity of the algorithm used by the test generation method should also be concerned. The algorithm used by the C-UIO with overlapping method is little bit complicated than the one used by the C-method but much simpler than the one used by the MUIO with overlapping method. It is not difficult to develop a program which will automatically generate test sequences by using the C-UIO with overlapping method.

CHAPTER 7

Conclusion

In this paper, we have implemented the C code to find the F-UIO sequence, B-UIO sequence, and C-UIO sequences. A Graphical User Interface (GUI) is given using Motif under X-Windows environment. We have shown a new protocol test generation method, the C-UIO with overlapping method. As can be seen from the experimental results that the test sequences generated by the C-UIO with overlapping is as good as the test sequences by the C-method which provide the highest fault coverage among the existing methods. The lengths of the test sequences have been reduced about 10% to 30% compared with those by the C-method.

APPENDIX

```
#include <stdio.h>
#include <string.h>
#define  YESA  1
#define  NOA   0
struct hblock{
                int   startpt;
          char  cir;
                struct hblock *next;
                struct dblock *down;
                };
struct dblock{
               int   endpt;
               char  input;
               char  output;
               struct dblock *down;
          };
struct input{
              char   inp;
              struct input *next;
         };
struct stateA{
              int edge;
              int status;
        char bluff;
              struct stateA *next;
              };
struct nodeA{
            char input;
            char flag;
            struct nodeA *parent;
            struct class *left;
            struct nodeA *child;
            struct nodeA *next;
            };
struct class{
             struct member *sub;
             struct class *next;
             };
struct member{
              int start;
              int end;
              char out;
              struct member *next;
              };
typedef struct hblock HNODE;
typedef struct dblock DNODE;
typedef struct stateA STATE;
typedef struct input INPUT;
STATE *statelist;
STATE *rstatelist;
INPUT *inputlist;
INPUT *rinputlist;
```

```
char   fr;
int    z=0;
int    over();
HNODE *hptr;
HNODE *hihi;
HNODE *riri;
HNODE *rhptr;
DNODE *hope();
DNODE *finals;
void   deep();
void   prints();
void   creats();
void   update();
void   final();
void   circular();
void   separate();
void   get_input();
void   convert();
void   put_output();
char   check_end();
struct member *find();
struct nodeA  *subclass();
struct class *class_init();
struct nodeA  *create_sequence();
#include "petri-net.h"
HNODE *create_hnode() {
        HNODE *temp;
        temp=((HNODE *)malloc(sizeof(HNODE)));
      if(!temp)printf("INSUFFICENT MEMORY\n");
        temp->startpt=0;
        temp->next=NULL;
        temp->cir='n';
        temp->down=NULL;
        return temp;
        }
DNODE *create_dnode() {
        DNODE *temp1;
        temp1=((DNODE *)malloc(sizeof(DNODE)));
    if(!temp1)printf("INSUFFICENT MEMORY\n");
        temp1->endpt=0;
        temp1->input=' ';
        temp1->output=' ';
        temp1->down=NULL;
        return temp1;
        }
STATE *creat_state() {
        STATE *temp2;
        temp2=(STATE *)malloc(sizeof(STATE));
    if(!temp2)printf("INSUFFICENT MEMORY\n");
        temp2->edge=0;
    temp2->next=NULL;
        temp2->status=NOA;
        temp2->bluff='n';
      return temp2;
```

```
        }
INPUT  *creat_input() {
           INPUT *temp3;
           temp3=(INPUT *)malloc(sizeof(INPUT));
     if(!temp3)printf("INSUFFICENT MEMORY\n");
           temp3->inp='\0';
       temp3->next=NULL;
       return temp3;
           }
struct nodeA *create_node() {
          struct nodeA *temp;
          temp=(struct nodeA *)malloc(sizeof(struct nodeA));
    if(!temp)printf("INSUFFICENT MEMORY\n");
          temp->parent=NULL;
          temp->input='\0';
          temp->flag='n';
          temp->left=NULL;
          temp->child=NULL;
          temp->next=NULL;
          return temp;
    };
struct class *create_class() {
           struct class  *temp;
           temp=(struct class *)malloc(sizeof(struct class));
     if(!temp)printf("INSUFFICENT MEMORY\n");
           temp->sub=NULL;
           temp->next=NULL;
```

```
            return temp;
            };
struct member *create_member() {
            struct member  *temp;
            temp=(struct member *)malloc(sizeof(struct member));
      if(!temp)printf("INSUFFICENT MEMORY\n");
            temp->start=0;
            temp->end=0;
            temp->out='\0';
            temp->next=NULL;
            return temp;
            };
extern char *seshfilename;
void search();
struct nodeA *create_sequence(pptr,fr)
struct nodeA *pptr;
char   fr;
{
           struct nodeA *curnode;
           INPUT *curinput;
        if(fr == 'f')
              curinput=inputlist;
        else
              curinput=rinputlist;
              while(curinput != NULL){
                if(pptr->child==NULL){
                   pptr->child=create_node();
                pptr->child->input=curinput->inp;
                pptr->child->parent=pptr;
              }
              else{
                curnode=pptr->child;
                while(curnode->next != NULL)
                  curnode=curnode->next;
                        curnode->next=create_node();
                   curnode=curnode->next;
                        curnode->input=curinput->inp;
                       curnode->parent=pptr;
                 }
        curinput=curinput->next;
    }
  return(pptr);
 }
void separate(classptr)
struct class *classptr;
{
struct class *cur;
struct class *end;
struct class *head;
struct class *active,*extra;
struct member *mhead;
struct member *mcur;
struct member *destiny;
int    flag;
int   DONE;
if(classptr){
 if(classptr->sub){
  flag=-1;
  head=classptr;
  end=classptr;
    while(end->next != NULL)
          end=end->next;
    cur=end;
  while(flag<1 ){
    if(head==end)
       flag++;
       mhead=head->sub;
       mcur=mhead->next;
    while(mcur != NULL){
      if(mhead->out != mcur->out){
        active=cur;
        DONE=0;
        while(active->next != NULL && !DONE){
        destiny=active->next->sub;
        if(mcur->out==destiny->out){
           while(destiny->next != NULL)
                         destiny=destiny->next;
               destiny->next=mcur;
               mhead->next=mcur->next;
               mcur=mcur->next;
        destiny->next->next=NULL;
```

```
   DONE=1;
   }
      else
     active=active->next;
  }
  if(!DONE){
   active->next=create_class();
   active->next->sub=mcur;
   mhead->next=mcur->next;
   mcur=mcur->next;
   active->next->sub->next=NULL;
   }
  }
 else{
        mcur=mcur->next;
        mhead=mhead->next;
    }
 }
 while(cur->next != NULL)
         cur=cur->next;
 if(head->next != NULL){
   if(head->next->sub==NULL)
     {
      while(head->next && head->next->sub == NULL){
      extra=head->next;
      head->next=head->next->next;
      extra->next=NULL;
```

```
            free(extra);
            if(head->next != NULL)
              head=head->next;
              }
             }
          else
             head=head->next;
          }
       if(flag==0)
             flag++;
     }
    }
   }
 }
struct class *class_init(fr)
char   fr;
{
struct class  *class_ptr;
struct member *cur;
struct member *temp;
STATE  *curstate;
class_ptr=create_class();
if( fr == 'f')
   curstate=statelist;
else
   curstate=rstatelist;
while(curstate != NULL){
      temp=create_member();
      temp->start=curstate->edge;
      temp->end=curstate->edge;
      temp->out='\0';
      if(class_ptr->sub==NULL)
        class_ptr->sub=temp;
      else{
        cur=class_ptr->sub;
        while(cur->next != NULL)
          cur=cur->next;
          cur->next=temp;
          temp=NULL;
        }
      curstate=curstate->next;
    }
   return class_ptr;
  }
  struct member *find(memptr,take,fr)
  struct member *memptr;
  char   take;
  char   fr;
  {
    HNODE *curhnode;
    DNODE *curdnode;
    struct member *temp;
    if(fr == 'f')
     curhnode=hptr;
  else
   curhnode=rhptr;
  while(curhnode->startpt != memptr->end)
      curhnode=curhnode->next;
  curdnode=curhnode->down;
  while(curdnode->down != NULL && curdnode->input != take)
      curdnode=curdnode->down;
  temp=NULL;
  if(curdnode->input == take) {
    temp=create_member();
    temp->start=memptr->start;
    temp->end=curdnode->endpt;
    temp->out=curdnode->output;
    }
   return temp;
}
struct nodeA *subclass(nodeptr)
struct nodeA *nodeptr;
{
  struct nodeA *parent;
  struct class *sclass,*tclass;
  struct member *cmember,*tmember,*csmember;
  parent=nodeptr->parent;
  if(parent != NULL){
  sclass=parent->left;
  while(sclass != NULL){
   if(nodeptr->left==NULL){
      nodeptr->left=create_class();
      tclass=nodeptr->left;
      }
    else{
        tclass->next=create_class();
        tclass=tclass->next;
        }
  csmember=sclass->sub;
  while(csmember != NULL){
  tmember=find(csmember,nodeptr->input,fr);
  if(tmember){
    if(tclass->sub==NULL){
       tclass->sub=tmember;
       cmember=tmember;
       }
    else{
        cmember->next=tmember;
        cmember=cmember->next;
        }
    }
   csmember=csmember->next;
  }
  sclass=sclass->next;
}
while(nodeptr->left != NULL && nodeptr->left->sub == NULL){
   tclass=nodeptr->left;
   nodeptr->left=nodeptr->left->next;
  free(tclass);}
 if(nodeptr->left) {
  tclass=nodeptr->left;
  sclass=tclass->next;
  while(sclass){
   if(sclass->sub == NULL){
     tclass->next=tclass->next->next;
     sclass->next=NULL;
     free(sclass);
     }
   tclass=tclass->next;
   if(tclass)
     sclass=tclass->next;
   else
     sclass=NULL;
   }
  }
 }
 return nodeptr;
}
int over(fr)
char fr;
{
 int done;
 STATE *cur;
 done=1;
 if(fr == 'f')
```

```
   cur=statelist;
  else
   cur=rstatelist;
  while(cur != NULL && done){
     if(cur->status==NOA)
       done=NOA;
     cur=cur->next;
     }
 return done;
}
void update(classptr,curnode,fr)
struct class *classptr;
struct nodeA  *curnode;
char   fr;
{
  struct class *curclass;
  STATE  *curstate;
  curclass=classptr;
  while(curclass != NULL){
      if(curclass->sub->next == NULL){
        if(fr == 'f')
             curstate=statelist;
        else
             curstate=rstatelist;
           while(curstate->edge!=curclass->sub->start)
               curstate=curstate->next;
        curstate->status=YESA;
        search(curclass->sub->start,curnode,fr); }
    curclass=curclass->next;
    }
}
void creats(cnode)
struct nodeA *cnode;
{
  if(cnode){
     if(!(cnode->child) && cnode->left && cnode->flag =='n')
          cnode=create_sequence(cnode,fr);
     else if(cnode->child)
             creats(cnode->child);
      if(cnode->next)
         creats(cnode->next);
   }
}
void deep(dnode)
struct nodeA *dnode;
{
  if(dnode){
     if(!dnode->child && !dnode->left && dnode->flag =='n'){
             dnode=subclass(dnode);
         if(dnode->left) {
           separate(dnode->left);
              update(dnode->left,dnode,fr);
           dnode->flag=check_end(dnode->left);
           }
          }
   else if (dnode->child)
          deep(dnode->child);
   if(dnode->next != NULL)
     deep(dnode->next);
     }
}
struct nodeA *resultA(fnode)
struct nodeA *fnode;
{
   while(!over(fr)){
          creats(fnode);
          deep(fnode);
       final(fr);
       }
  return fnode;
}
void search(status,snode,fr)
struct nodeA *snode;
int    status;
char   fr;
{
   struct class *class2;
   struct member *curmember;
   int    found=0;
   STATE *curstate;
 if(snode){
  class2=snode->left;
  search(status,snode->parent,fr);
  while(class2 != NULL && !found){
      curmember=class2->sub;
  while(curmember != NULL && !found) {
  if(curmember->start==status){
  if(fr == 'f')
   curstate=statelist;
  else
   curstate=rstatelist;
   while(curstate->edge != status)
     curstate=curstate->next;
   if(curstate->bluff != 'y'){
     if(fr=='f')
  put_output(curmember->start,curmember->end,snode>input,
    curmember->out);
    else
    put_output(curmember->start,curmember->end,snode
>input,curmember->out);
     }
   found=1;
   }
 curmember=curmember->next;
  }
 class2=class2->next;
 }
}
```

```
}
char check_end(classptr)
struct class *classptr;
{
  struct class *temp;
  char flag='y';
  if(classptr) {
    temp=classptr;
    while(temp != NULL && flag=='y') {
       if(temp->sub->next != NULL)
         flag='n';
       temp=temp->next;
    }
  }
  return flag;
 }
void final(fr)
char fr;
{
 STATE *cur;
 if(fr == 'f' )
   cur=statelist;
else
   cur=rstatelist;
 while(cur != NULL){
   if(cur->status == YESA)
     cur->bluff='y';
  cur =cur->next;
 }
}
void get_input()
{
        int  start,end;
        char  j1,j2,j3,j4,j5,j6,str[80];
        char inpt,string,outpt;
        HNODE *hcur;
        DNODE *dcur,*temp;
   STATE *curstate;
   INPUT *curinput;
        HNODE *rhcur;
        DNODE *rdcur;
   STATE *rcurstate;
   INPUT *rcurinput;
        FILE  *fp,*ofp,*tfp;
   fp=fopen(seshfilename,"r");
        if(!fp){
          printf("can't open %s file\n",seshfilename);
          exit(1);
          }
     while(!feof(fp)){
     fscanf(fp,"%s",str);
     if(!feof(fp)){
sscanf(str,"%c%c%d%c%d%c%c%c%c%c",&j1,&j2,&start,&j3,&end,
&j4,&j5,&inpt,&j6,&outpt);
```

```
  }
  if(!feof(fp)){
  if(hptr==NULL){
       hptr=create_hnode();
       hptr->startpt=start;
       }
hcur=hptr;
     while(hcur->next != NULL && hcur->startpt != start)
         hcur=hcur->next;
    if(hcur->startpt != start){
              hcur->next = create_hnode();
              hcur = hcur->next;
              hcur->startpt = start;
          }
     temp=create_dnode();
     temp->endpt=end;
     temp->input=inpt;
     temp->output=outpt;
if(hcur->down==NULL){
        hcur->down=temp;
        temp=NULL;
        }
     else
     {
        dcur=hcur->down;
        while(dcur->down!=NULL)
      dcur=dcur->down;
```

```
        dcur->down=temp;
      temp=NULL;
      }
    if(inputlist==NULL){
        inputlist=creat_input();
        inputlist->inp=inpt;
        }
curinput=inputlist;
      while(curinput->next != NULL && curinput->inp != inpt)
          curinput=curinput->next;
      if(curinput->inp != inpt){
               curinput->next = creat_input();
               curinput = curinput->next;
               curinput->inp = inpt;
          }
    if(rhptr==NULL){
        rhptr=create_hnode();
        rhptr->startpt=end;
        }
rhcur=rhptr;
      while(rhcur->next != NULL && rhcur->startpt != end)
          rhcur=rhcur->next;
      if(rhcur->startpt != end){
               rhcur->next = create_hnode();
               rhcur = rhcur->next;
               rhcur->startpt = end;
          }
        temp=create_dnode();
        temp->endpt=start;
        temp->input=outpt;
        temp->output=inpt;
    if(rhcur->down==NULL){
            rhcur->down=temp;
            temp=NULL;
            }
        else
        {
            rdcur=rhcur->down;
            while(rdcur->down!=NULL)
          rdcur=rdcur->down;
            rdcur->down=temp;
        temp=NULL;
        }
      if(rinputlist==NULL){
            rinputlist=creat_input();
            rinputlist->inp=outpt;
            }
    rcurinput=rinputlist;
         while(rcurinput->next!=NULL && rcurinput->inp !=outpt)
             rcurinput=rcurinput->next;
             if(rcurinput->inp != outpt){
                  rcurinput->next = creat_input();
                  rcurinput = rcurinput->next;
```

```
                rcurinput->inp = outpt;
              }
          }
     } /* while feof */
     statelist=creat_state();
          curstate=statelist;
          hcur=hptr;
          while(hcur != NULL){
               curstate->edge=hcur->startpt;
               hcur=hcur->next;
            if(hcur != NULL){
                 curstate->next=creat_state();
                 curstate=curstate->next;
                 }
        }
          rstatelist=creat_state();
          rcurstate=rstatelist;
          rhcur=rhptr;
          while(rhcur != NULL){
               rcurstate->edge=rhcur->startpt;
               rhcur=rhcur->next;
           if(rhcur != NULL){
                 rcurstate->next=creat_state();
                 rcurstate=rcurstate->next;
                 }
       }
}
```

```
void convert(data)
graphics_data  *data;
{
        int   i,start,end;
        char   j1,j2,j3,j4,j5,j6,str[80],test[50];
        char  inpt,string,outpt;
        char           *rs,*rs1;
        HNODE *hcur;
        DNODE *dcur,*temp;
    STATE *curstate;
    INPUT *curinput;
        HNODE *rhcur;
        DNODE *rdcur;
    STATE *rcurstate;
    INPUT *rcurinput;
        FILE  *fp,*ofp,*tfp;
    for (i=0; i<data->next_pos; i++){
    if(data->buffer[i].name[0]=='L' ||
        data->buffer[i].name[0]=='A')
    {
     strcpy(test,data->buffer[i].name);
     rs=test;
     strcpy(rs,strtok(rs+2,"P"));
     start=atoi(rs);
     strcpy(test,data->buffer[i].name);
     rs=test;
     strcpy(rs,strtok(strstr(rs+2,"P"),"@"));
```

```
 end = atoi(rs+1);

 strcpy(test,data->buffer[i].name);

 strcpy(test,strstr(test, "$"));

 rs=test+1;

 strtok(rs, ",");

 inpt=rs[0];

 strcpy(test,data->buffer[i].name);

 strcpy(test,strstr(test, ","));

 rs=test+1;

 outpt=rs[0];


    if(hptr==NULL){

      hptr=create_hnode();

      hptr->startpt=start;

      }

hcur=hptr;

    while(hcur->next != NULL && hcur->startpt != start)

        hcur=hcur->next;

    if(hcur->startpt != start){

             hcur->next = create_hnode();

             hcur = hcur->next;

             hcur->startpt = start;

        }

    temp=create_dnode();

    temp->endpt=end;

    temp->input=inpt;

    temp->output=outpt;
```

```
if(hcur->down==NULL){
        hcur->down=temp;
        temp=NULL;
        }
     else
     {
        dcur=hcur->down;
        while(dcur->down!=NULL)
   dcur=dcur->down;
        dcur->down=temp;
     temp=NULL;
     }
   if(inputlist==NULL){
        inputlist=creat_input();
        inputlist->inp=inpt;
        }
curinput=inputlist;
      while(curinput->next != NULL && curinput->inp != inpt)
          curinput=curinput->next;
     if(curinput->inp != inpt){
               curinput->next = creat_input();
               curinput = curinput->next;
               curinput->inp = inpt;
           }
   if(rhptr==NULL){
        rhptr=create_hnode();
        rhptr->startpt=end;
```

```
              }
rhcur=rhptr;
          while(rhcur->next != NULL && rhcur->startpt != end)
               rhcur=rhcur->next;
         if(rhcur->startpt != end){
                    rhcur->next = create_hnode();
                    rhcur = rhcur->next;
                    rhcur->startpt = end;
               }
          temp=create_dnode();
          temp->endpt=start;
          temp->input=outpt;
          temp->output=inpt;
    if(rhcur->down==NULL){
             rhcur->down=temp;
             temp=NULL;
             }
          else
         {
             rdcur=rhcur->down;
             while(rdcur->down!=NULL)
       rdcur=rdcur->down;
             rdcur->down=temp;
         temp=NULL;
         }
       if(rinputlist==NULL){
             rinputlist=creat_input();
         rinputlist->inp=outpt;}
rcurinput=rinputlist;
      while(rcurinput->next != NULL &&
     rcurinput->inp != outpt)
           rcurinput=rcurinput->next;
           if(rcurinput->inp != outpt){
                 rcurinput->next = creat_input();
                 rcurinput = rcurinput->next;
                 rcurinput->inp = outpt;
           }
   }
} /* while feof */
statelist=creat_state();
      curstate=statelist;
      hcur=hptr;
      while(hcur != NULL){
           curstate->edge=hcur->startpt;
           hcur=hcur->next;
        if(hcur != NULL){
             curstate->next=creat_state();
             curstate=curstate->next;
             }
    }
      rstatelist=creat_state();
      rcurstate=rstatelist;
      rhcur=rhptr;
      while(rhcur != NULL){
              rcurstate->edge=rhcur->startpt;
              rhcur=rhcur->next;
            if(rhcur != NULL){
               rcurstate->next=creat_state();
               rcurstate=rcurstate->next;
               }
      }
}
void put_output(start,end,inpt,outpt)
int   start,end;
char  inpt,outpt;
{
     HNODE *hcur;
     DNODE *dcur,*temp;
     if(fr == 'f'){
      if (inpt=='\0'){
            if(hihi==NULL){
             hihi=create_hnode();
             hihi->startpt=start;
          hcur=hihi;
             }
      else{
          hcur=hihi;
          while(hcur->next != NULL )
               hcur=hcur->next;
           hcur->next = create_hnode();
           hcur = hcur->next;
```

```
        hcur->startpt = start;
            }
   }
    else if(inpt != '\0')
      {
        hcur=hihi;
        while(hcur->next != NULL)
          {
            if(hcur->startpt == start)
                tail=hcur;
            hcur=hcur->next;
          }
        if(hcur->startpt == start)
            tail=hcur;
        temp=create_dnode();
        temp->endpt=end;
        temp->input=inpt;
        temp->output=outpt;
if(tail->down==NULL){
            tail->down=temp;
            temp=NULL;
            }
        else
       {
           dcur=tail->down;
           while(dcur->down!=NULL)
         dcur=dcur->down;
        dcur->down=temp;
      temp=NULL;
      }
    }
  }
  else if(fr == 'r'){
   if(inpt=='\0'){
       if(riri==NULL){
         riri=create_hnode();
         riri->startpt=start;
      hcur=riri;
         }
   else{
       hcur=riri;
       while(hcur->next != NULL )
            hcur=hcur->next;
        hcur->next = create_hnode();
        hcur = hcur->next;
        hcur->startpt = start;
            }
    }
    else if(inpt != '\0')
      {
       hcur=riri;
       while(hcur->next != NULL)
         {
          if(hcur->startpt == start)
```

```
                    tail=hcur;
                 hcur=hcur->next;
               }
              if(hcur->startpt == start)
                 tail=hcur;
              temp=create_dnode();
              temp->endpt=end;
              temp->input=outpt;
              temp->output=inpt;
        if(tail->down==NULL){
                 tail->down=temp;
                 temp=NULL;
                  }
              else
              {
                 dcur=tail->down;
                 while(dcur->down!=NULL)
           dcur=dcur->down;
                 dcur->down=temp;
              temp=NULL;
              }
           }
         }
} /* while feof */
void prints(hptr,choice)
HNODE *hptr;
char  choice;
```

```
{
 HNODE *hcur;
 HNODE *tail,*temp;
 DNODE *tcur;
 DNODE *dcur;
 FILE  *fpt;
   hcur=hptr;
   system("rm result.dat");
   fpt=fopen("result.dat","w");
     while(hptr != NULL){
   if(choice == 'c'){
    if(hptr->cir =='y'){
         fprintf(fpt,"%d (",hptr->startpt);
         dcur=hptr->down;
         while(dcur != NULL) {
    if(dcur->down != NULL)
      fprintf(fpt," %c,%c ", dcur->input,dcur->output);
    else
  fprintf(fpt," %c,%c ) \n", dcur->input,dcur->output);
              dcur=dcur->down;
               }
    }
  }
   else{
         fprintf(fpt,"%d (",hptr->startpt);
         dcur=hptr->down;
         while(dcur != NULL) {
```

```
        if(dcur->down != NULL)
          fprintf(fpt," %c,%c ", dcur->input,dcur->output);
        else
        fprintf(fpt," %c,%c ) %d\n", dcur->input,
                        dcur->output,dcur->endpt);
                        dcur=dcur->down;
                        }
      }
            hptr=hptr->next;
          }
      fclose(fpt);
}
void circular()
{
 int ffirst,flast,gotit;
 int bfirst,blast;
 HNODE *fhcur,*bhcur;
 DNODE *fdcur,*bdcur;
 fhcur=hihi;
 while(fhcur != NULL)
 {
  ffirst=fhcur->startpt;
  flast='\0';
  fdcur=fhcur->down;
  gotit=0;
  if(fdcur){
  while(fdcur->down != NULL)
 fdcur=fdcur->down;
flast=fdcur->endpt;
}
 bhcur=riri;
 while(bhcur != NULL && !gotit)
 {
  if(bhcur->startpt == ffirst)
   {
     bdcur=bhcur->down;
     if(bdcur){
     while(bdcur->down != NULL)
       bdcur=bdcur->down;
       if(bdcur->endpt == flast && ffirst!=flast)
       {
        fdcur->down=hope(bhcur->down);
        gotit=1;
        finals=NULL;
        }
      }
    }
  bhcur=bhcur->next;
  }
if(gotit)
fhcur->cir='y';
fhcur=fhcur->next;
}
}
```

```
DNODE *hope(dnodeptr)
DNODE *dnodeptr;
{
 DNODE *rhead;
 if(dnodeptr){
  if(dnodeptr->down)
    finals=hope(dnodeptr->down);
   {
    if(finals==NULL){
    finals=create_dnode();
    finals->endpt=dnodeptr->endpt;
    finals->input=dnodeptr->input;
    finals->output=dnodeptr->output;
    finals->down=NULL;
    }
    else{
    rhead=finals;
    while(rhead->down != NULL)
      rhead=rhead->down;
    rhead->down= create_dnode();
    rhead=rhead->down;
    rhead->endpt=dnodeptr->endpt;
    rhead->input=dnodeptr->input;
    rhead->output=dnodeptr->output;
    rhead->down=NULL;
    }
 }
}
return finals;
}
UIO_fun( answer,data)
graphics_data  *data;
char   answer;
{
struct nodeA    *first;
struct nodeA    *second;
system("rm result.dat");
          convert(data);
          switch ( answer) {
            case 'F' :
                    fr='f';
                    first = create_node();
                    first->left = class_init(fr);
                    first=resultA(first);
              prints(hihi,'f');
                    break;
        case 'B' :
                    fr='r';
                    second = create_node();
                    second->left = class_init(fr);
                    second=resultA(second);
              prints(riri,'r');
                    break;
        case 'C' :
                fr='f';
                first = create_node();
                first->left = class_init(fr);
                   first=resultA(first);
                fr='r';
                second = create_node();
                second->left = class_init(fr);
                second=resultA(second);
            circular();
            prints(hihi,'c');
                 break;
       }
}
#include <Xm/ScrollBar.h>
#include <Xm/Text.h>
#include <Xm/PushB.h>
#include <Xm/Label.h>
#include <Xm/DialogS.h>
#include <sys/stat.h>
#include "petri-net.h"
#include "place.bitmap"
#include "trans.bitmap"
#include "h_trans.bitmap"
#include "arrow.bitmap"
#include "extplace.bitmap"
#define PLACE data->Petri.place_t
```

```
#define TRANS data->Petri.trans_t
#define plce_arr PLACE
#define trns_arr TRANS
#define stg2 data2->Stg
#define SCALE_MAX 10
#define SCALE_MAX1 100
#define SCALE_FMAX 1000
void input_uio();
void forward_uio();
void backward_uio();
void circular_uio();
int sesh_flag = 0;
char *seshfilename;
extern UIO_fun();
Widget
GettopShell( w)
Widget w;
{
        while ( w && !XtIsWMShell( w))
                w = XtParent( w);
      return( w);
}
void
exitd( w, d)
Widget w,d ;
{
        XtDestroyWidget( d);
```

```
}
Widget
create_popup( w)
Widget w;
{
        Widget          txt, dialog, form, pb;
        Arg     xargs[10];
        int     n;
    dialog = XtVaCreatePopupShell ( "dialog",
                        topLevelShellWidgetClass, GettopShell( w),
                        XmNdeleteResponse, XmDESTROY,
                        NULL );
        form = XtVaCreateManagedWidget( "form",
                        xmFormWidgetClass, dialog,
                        XmNwidth, 500,
                        XmNheight, 500, NULL );
        n = 0;
        XtSetArg( xargs[n], XmNwidth, 500 ); n++;
        XtSetArg( xargs[n], XmNheight, 400 ); n++;
        XtSetArg( xargs[n], XmNscrollHorizontal, False); n++;
        XtSetArg( xargs[n], XmNblinkRate, 0); n++;
        XtSetArg( xargs[n], XmNeditable, False); n++;
        XtSetArg( xargs[n],XmNcursorPositionVisible,False);n++;
    txt = XmCreateScrolledText( form, "text", xargs, n);
        XtVaSetValues( XtParent( txt),
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, XtParent( txt),
                XmNtopAttachment, XmATTACH_WIDGET,
                XmNtopWidget, XtParent( txt),
                XmNrightAttachment, XmATTACH_WIDGET,
                XmNrightWidget, XtParent( txt),
                NULL );
  XtManageChild ( txt);
      pb = XtVaCreateManagedWidget( "pb",
                    xmPushButtonWidgetClass, form,
                    XmNlabelString, XmStringCreateSimple ("OK"),
                    XmNwidth, 80,
                    XmNheight, 40,
                    XmNx, 210,
                    XmNy, 430,
                    NULL );
      XtAddCallback ( pb, XmNactivateCallback, exitd, dialog);
  XtPopup( dialog, XtGrabNone);
      return txt;
}
void
readfile( fs, client_dat, cbs)
Widget fs;
XtPointer client_dat;
XmFileSelectionBoxCallbackStruct *cbs;
{
XmStringGetLtoR( cbs->value, XmSTRING_DEFAULT_CHARSET, &seshfilename);
if ( !*seshfilename )
                printf("Error: Select the input file\n");
```

```
        else {
                sesh_flag = 100;
                printf("Selected Input File = %s\n",seshfilename);
        }
        XtUnmanageChild( fs);
}
void input_uio(w,data,client_data)
Widget   w;
graphics_data *data;
caddr_t client_data;
{
Widget dialog;
dialog = XmCreateFileSelectionDialog( w, "filesb", NULL, 0);
XtAddCallback( dialog, XmNcancelCallback,
XtUnmanageChild, NULL );
XtAddCallback( dialog, XmNokCallback, readfile, NULL );
XtManageChild( dialog);
}
display_result( text_w, outputfilename)
Widget text_w;
char outputfilename[];
{
        struct stat statb;
        char *text;
        FILE *fptr;
    stat( outputfilename, &statb);
    fptr = fopen( outputfilename, "r");
```

```
   if (!(text = XtMalloc((unsigned)(statb.st_size+1)))){
                printf("can't alloc enought space for %s", outputfilename);
        XtFree( outputfilename);
        return;
        }
   if (!fread(text,sizeof(char),statb.st_size+1,fptr))
        printf("Warning: may not have read entire file!\n");
   text[statb.st_size]=0;
        XmTextSetString( text_w, text);
   XtFree( text);
        XtFree( outputfilename);
}
void forward_uio(w,data,client_data)
Widget   w;
graphics_data *data;
caddr_t client_data;
   {
        Widget text;
      sesh_flag = 100;
        if ( sesh_flag == 100) {
           UIO_fun( 'F',data);
           text = create_popup( w);
           display_result( text, "result.dat");
           printf("forward code \n");
           sesh_flag = 0;
           }
        else
```

```
                printf( "ERROR: Select the input file and try forward\n");
   }
void backward_uio(w,data,client_data)
Widget   w;
graphics_data *data;
caddr_t client_data;
{
         Widget text;
    sesh_flag = 100;
         if ( sesh_flag == 100) {
            UIO_fun( 'B',data);
            text = create_popup( w);
            display_result( text, "result.dat");
            printf("backward code  \n");
            sesh_flag = 0;
            }
         else
                printf( "ERROR: Select the input file and try backward\n");
}
void circular_uio(w,data,client_data)
Widget   w;
graphics_data *data;
caddr_t client_data;
   {
         Widget text;
    sesh_flag = 100;
         if ( sesh_flag == 100) {
            UIO_fun( 'C',data);
            text = create_popup( w);
            display_result( text, "result.dat");
            printf("Circular code \n");
            sesh_flag = 0;
            }
        else
           printf( "ERROR: Select the input file and try Circular UIO\n");
}
```

REFERENCES

Chan, W., and S.T. Vuong. 1989. "An Improved Protocol Test Generation Procedure Based on UIOs." First Edition. Vol. 19, 4:283-292.

Holzmann, Gerard J. 1987. "Design and Validation of Computer Protocols." *AT&T Bell Laboratories*. Princeton Hall Software Series. 300- 340.

Sabnani, K., and A. Dahbura. 1974. "A Protocol Test Generation Procedure." *Computer Networks and ISDN Systems,* volume 15. 4: 285-297.

Yang, B. 1983. "Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping." 2:118-125.