An Estelle Interpreter for Incremental Protocol Verification*

Chung-Ming Huang, Jenq-Muh Hsu, Huei-Yang Lai, Jao-Chiang Pong and Duen-Tay Huang

Laborary of Computer-Aided Protocol Engineering
Institue of Information Engineering
National Cheng Kung University
Tainan, Taiwan 70101
email: huangcm@locust.iie.ncku.edu.tw

Abstract

Formal Description Techniques (FDTs) provide formal and abstract ways to specify what protocols have to do and what the features of the protocols need. Protocol verification is a process for detecting logical errors, such as deadlock, unspecified receptions, and channel overflow errors, in communication protocols. Estelle is a FDT defined by the International Organization for Standardization (ISO) for protocol specifications. In this paper, we present an incremental protocol verification technique and an Estelle interpreter which accepts Estelle protocol specifications for incremental protocol verification.

1 Introduction

Over the past years a lot of efforts have been made to promote the use of Formal Description Techniques (FDTs) to support different phases of developing complex protocols. ISO's Estelle is an FDT based on the extended state transition model, e.g., the Extended Communication Finite State Machine (ECFSM) model [9]. Estelle can formally specify distributed and/or concurrent information processing system.

Protocol verification is a process for detecting logical errors, such as deadlock, unspecified receptions, and channel overflow errors, in communication protocols. Global state reachability analysis is one of the most straightforward and easily automated ways for analyzing state-transition-based communication protocols. Although global state reachability analysis is easily automated, global state reachability analysis suffers from the state explosion problem. In order to relieve the state explosion problem, many Communication Finite State Machine (CFSM) -based reduction techniques have been proposed in the past decade. However, these CFSM-based reduction techniques cannot be directly applied to Estelle-based protocol verification systems, because Estelle is ECFSM-based. Chu and Liu's reduction technique, which is based on the dead and live variables concept [2], is one of the few ECFSM-based ones.

In the traditional protocol verification approach, global states reachability analysis must be re-explored from the initial state when logical errors are detected and protocols are modified accordingly. All of the global states need to be re-explored again even if only one or few transitions are modified. Incremental protocol verification tries to avoid re-exploring all global states after protocol modification [5]. To apply the incremental protocol verification to Estelle-specified protocols, we propose an ECFSM-based incremental protocol verification technique in this paper. To further reduce the number of global states to be verified, the dead and live concept is also incorporated in the new technique. Based on the ECFSM-based incremental protocol verification technique, we are currently developing an Estelle interpreter for incremental protocol verification.

The rest of this paper is organized as follows. In Section 2, an overview of Estelle and the restrictions in our system are given, and the main idea of Chu and Liu's reduction technique is introduced. In Section 3, the ECFSM-based incremental protocol verification technique is described. In Section 4, the Estelle interpreter is presented. In Section 5, the usage of the Estelle interpreter and an example of incremental protocol verification are given. In Section 6, we have the conclusion remarks.

2 Overview

In this section, we will introduce main features of Estelle, the restrictions in our Estelle interpreter, and the main idea of Chu and Liu's reduction technique [2].

2.1 Estelle

Estelle is an FDT based on the extended state transition model, e.g., the ECFSM model. Using the module structures, production-rule-like control structures and Pascal-based statements, Estelle provides the ability to formally specify communication protocols.

*The research was supported by the National Science Council of the Republic of China under the grant NSC 81-0408-E-006-568

0-8186-3670-X/93 $03.00

Our Estelle interpreter is for protocol verification, thus, some features are restricted to have fully automatic execution of verification and simplify the execution process. These restrictions are similar to those in [1, 4] and are described in the following paragraphs.

As it is mentioned in [3, 4], global synchronization for system process modules may lead to undesired overspecification. Hence, the systemprocess and process modules are removed. Only systemactivity and activity attributes are supported to have the execution of interleaving semantics [1].

Dynamic features are not allowed, i.e, module instances and connections cannot be created dynamically. In other words, a static configuration similar to that in [1] is supported. Since the system do not deal with timed verification, the "delay" clause is removed.

In order to have fully automatic execution, incomplete definitions of functions/procedures, "...", and "any" types are not allowed.

2.2 Overview of Chu and Liu's Reduction Technique

In the ECFSM model, each protocol entity consists of a Communication Finite State Machine (CFSM) and a set of context variables. Chu and Liu's reduction technique requires the data flow analysis on all CFSMs describing protocol entities to obtain the information called ***dead variable sets*** [2].

A context variable is accessed in two ways : ***reference*** and ***assignment***. A context variable is said to be : (1) ***referenced***, when it is accessed in a predicate, in the right-hand side of an infix assignment operator, e.g., ':=' operator in Pascal, or in the parameter list of a send event; (2) ***assigned***, when it is accessed in the left-hand side of an infix assignment operator or in the parameter list of a receive event. Assume x is a context variable of entity e and s is a state of the ECFSM of entity e. Then x is said to be **dead** at s if, starting from state s, x will not be referenced in all possible future execution paths of entity e, or the next possible accesses to x in all possible future execution paths of entity e are all assignments. Two global state are considered as equivalent when their corresponding element values are identical to each other, except for the values taken on by the context variables that are dead. Only one state of a set of equivalent states needs to be checked during global state reachability analysis.

3 The ECFSM-based Incremental Protocol Verification Technique

In this section, some definitions used in the incremental verification technique and the logical errors which can be detected are introduced first. Then, the new incremental protocol verification technique which is applicable to the ECFSM model is present.

3.1 Definitions and Logical Errors

For convenience, some definitions are introduced as follows: (1) Spontaneous transition: the transition whose condition part has no input event, i.e., the condition part can only contain a predicate and the action part has an output event. (2) When transition: the transition whose condition part has an input event and/or a predicate. (3) Spontaneous state: the state whose outgoing transitions are all spontaneous transitions. (4) When state: the state whose outgoing transitions are all when transitions. (5) Mixed state: the state whose outgoing transitions consist of spontaneous and when transitions.

In our ECFSM-based global state rechability analysis, some logical errors can be detected. These logical errors are defined as follows (assume n is the number of protocol entities):

- A global state contains a ***deadlock error*** if the followings conditions are satisfied: (1) All of the communicating channels $Q_{I->J}$, where I=1..n, J=1..n, I≠J, are empty. (2) All of the current state S_I of entity I, I=1..n, have no spontaneous transition or some have spontaneous transition but the associated predicates are false. (3) There is a state S_I of entity E_I such that S_I is not a terminal state, when terminal states are defined in the protocol.

- A global state contains an ***unspecified reception error*** if the following conditions are satisfied: (1) If there is an I, such that all of the head messages of the communicating channel $Q_{J->I}$, J=1..n, J≠I are unspecified in the current state S_I of entity E_I or some of the head messages of communicating channel $Q_{J->I}$, J=1..n, J≠I are specified but their associated predicates are false. (2) The current state S_I of entity E_I has no spontaneous outgoing transition or has some spontaneous outgoing transitions but their associated predicates are false.

- A global state contains a ***transmitted lock error*** if the following conditions are satisfied: For an entity I, I=1..n, (1) all of the outgoing transitions are spontaneous transitions, (2) all of the associated predicates are false.

- A global state contains a ***channel overflow error*** if the number of messages in the communicating channel is greater than the channel size.

3.2 The Incremental Protocol Verification Algorithm

Incremental protocol verification is divided into two parts: adding new transitions and deleting old transitions. Adding new transitions can eliminate some logical errors. For example, adding a send transition can make a global state with a deadlock error to be error-free. Additionally, adding new transitions may also generate new global states from the global states which contain the head states of the added transitions. Therefore, adding new transitions may have a side effect of generating new errors.

Deleting transitions can eliminate logical errors too. Deleting transitions will remove those global states and the associated logical errors which are generated

by executing the deleted transitions. However, deleting transitions may also have a side effect of generating new errors. For example, if a global state's communicating queues are all empty and all of its executable transitions are part of the deleted transitions, then it may have a deadlock error after the modification. Therefore, the parent states of the deleted states need to be checked.

Figure 1 formalizes the global state reachability analysis with the concept of dead and live variables for n-entity protocols. There are three phases in the global state reachability analysis. Phase one evaluates dead and live variables in each of the n CFSMs of the protocol to be verified. Phase two initializes the initial global state. For convenience, symbol S represents the state of a CFSM, $\overline{S}$ represents the state and the variables in a CFSM. Phase three checks logical correctness, calls the incremental verification process when an error is detected, explores executable transitions, and so on. $S_i - e \rightarrow S_j$ represents transition e is exexuted at state S_i and state S_j is generated after the execution of e.

Figure 2 formalizes the incremental verification. There are seven phases in the incremental verification. Phase one adds and deletes transitions. Phase two re-evaluates the dead and live variables of each state after adding and deleting transitions.

Phase three calculates those global states which are originally regarded as the same but become different due to their dead variables becoming live. Step two of procedure ***Dead-tol-Live*** first checks whether these affected global states become other states' duplicates or not. Phase four calculates those global states which become the same due to their live variables becoming dead. Phase five inspects the logical correctness of affected states and deletes those states which need to be removed for the deleting case. Phase six selects those global states which become re-extendable after the new transitions have been added. After these adjustments, phase seven returns to the normal global state reachability analysis.

4 The Incremental Estelle Interpreter

In the section, we will briefly describe an incremental Estelle interpreter machine and the corresponding functionalities.

4.1 An Interpreter Machine

The interpreter performs its tasks in four parts: (1) The first part consists of a ***lexical analyzer*** whose job is to scan the input and convert sequences of characters into tokens. (2) In the second part, a parser reads tokens and assembles them into language constructs. For instance, the constructs of the Estelle language describe how keywords, identifiers, and expressions can be translated to tree structures. (3) The third part, which deals with actions upon the input, is done by a tracer based on the tree structures. (4) The fourth part is the incremental processing which performs the

Algorithm G-S-R-A:

1. Evaluate live and dead variables of each state in $CFSM_i, i = 1..n$.
2. Set $S_i, i = 1..n$, to initial values and set $q_{i \rightarrow j}, i = 1..n, j = 1..n, i \neq j$, to be empty, add global state $(S_1, ..., S_n, q_{1 \rightarrow 2}, ..., q_{n \rightarrow n-1})$ to EXTENDABLE.
3. **while** EXTENDABLE is not empty **do**

   Remove a global state GS from EXTENDABLE,

   if GS does not have any executable transition

   then

   case

   GS is not a terminal state:

   mark GS as an erroneous state according to the error type and add GS to GLOBALSTATE.

   if the designer wants to modify the protocol at this point

   then Incremental-Processing. {Figure 2.}

   GS is a terminal state:

   mark GS as a terminal state and add GS to GLOBALSTATE.

   endcase

   else

   while executable transitions from GS is not empty **do**

   select one executable transition e which has not been selected and generate GS' such that $GS - e \rightarrow GS'$.

   if there is a GS_i in EXTENDABLE or GLOBALSTATE which is unique or the first occurrence such that GS_i is equal to GS' except for the values taken on by dead variables

   then

   mark GS_i as the first occurrence if it is originally unique,

   mark GS' as the duplicated state of GS_i and add GS' to GLOBALSTATE.

   else

   mark GS' as a unique state and add GS' to EXTENDABLE.

   endif

   endwhile

   endif

   endwhile

Figure 1: Gloabl state reachability analysis based on the dead and live variable analysis.

Algorithm Incremental-Processing:

1. Modify the protocol: add and/or delete transitions.
2. Evaluate live and dead variables of each state in the modified $CFSM_i, i = 1..n$.
3. **Dead-to-Live.** {Dead-to-Live is in Figure 5.}
4. **Live-to-Dead.** {Live-to-Dead is in Figure 3.}
5. **Delete-Process.** {Delete-Process is in Figure 6.}
6. **Add-Process.** {Add-Process is in Figure 7.}
7. go to phase 3 of algorithm **G-S-R-A**.

Figure 2: The incremental verification.

modifications of specifications, i.e., adding/deleting transitions in the Estelle specifications.

4.2 Functionalities of the Estelle Interpreter

In the subsection, we will present the major parts of the Estelle interpreter.

(1) ECFSM Table Generator :

Each specification module is translated to an ECFSM table. The ECFSM table can be obtained from the "INIT" statement in the Estelle specification but not the declaration part, because the entity declared may be never used in the later. The ECFSM table is built as a link list, in which each node is a large structure. The table structure is shown in Figure 8.

(2) Module Body Generator :

The body structure contains body name, initial state, a state pointer which points to the state list, a variable pointer which points to the variable list, and a transition pointer which points to the transition list. The states declared in the specification can be recorded in the state list of the body structure.

The state structure node of the state list contains the identifier, state name, live variable pointer which points to the live variable list, and the next state pointer which points to the next state. The module body and state structures are shown in Figure 9

The transitions is recorded as a list in which each node contains (1) WHEN part, which contains interaction point name, message name, parameter list pointer. (2) PROVIDED part, which contains provided tree pointer, and (3) ACTION part, which contains action list pointer which points to the action list. The node of the transition list is shown in Figure 10.

(3) Queue Generator :

The channel declaration part can be translated to a channel table. The communication queues in the global state structure can be generated from the "CONNECT" statements in the Estelle specification and by the role definition channel table.

(4) Dead and Live Variable Generator :

Dead and live variables of each state can be obtained based on the PROVIDED tree and the action list of the transition part in the body structure. The process of seeking dead and live variables is divided into two stages. In the first stage, we traverse all transitions to decide each variable's first access type, assignment or reference, for each transition. In the second stage,

Procedure Live-to-Dead:

1. **for** each state S in $CFSM_i, i = 1..n$, **do**

   if there is a variable V $\in$ S such that V is changed from live to dead

   then add S to live-to-dead-i.

   endfor

2. **for** each GS which is marked as unique or the first occurrence in GLOBALSTATE or EXTENDABLE **do**

   if there is a $GS.S_i$, i = 1..n, in live-to-dead-i and there is no GS_j, j = 1..n, in dead-to-live-j

   then

   if there is a GS' in GLOBALSTATE or EXTENDABLE which is marked as unique or the first occurrence such that GS' is equal to GS except the values taken on by corresponding dead variables

   then

   mark GS' as the first occurrence if it is originally unique,

   mark GS as a duplicated state of GS',

   modify each duplicated state of GS as a duplicated state of GS',

   for each child(GS) **do Clear(child(GS),** ε) {in Figure 4.}

   endfor

   endif

   endif

   endfor

Figure 3: Live to dead analysis.

Procedure Clear(GS, T):

1. remove GS from GLOBALSTATE or EXTENDABLE.
2. **if** GS is marked as the first occurrence

   then

   promote one of the duplicated states GS_i of GS which is not a descendant of GS and which is not generated by executing transition T as the first occurrence,

   add GS_i to EXTENDABLE.

   else

   for each child(GS) **do Clear(child(GS), T).**

   endif

Figure 4: The procedure used for removing descendant global state sequences.

Procedure Dead-to-Live:

1. **for** each state S in $CFSM_i, i = 1..n$, **do**

   if there is a variable V $\in$ S such that V is changed from dead to live

   then add S to dead-to-live-i.

   endfor

2. **for** each GS which is marked as unique or the first occurrence in GLOBALSTATE **do**

   if there is a $GS.S_i$ in dead-to-live-i, i = 1..n

   then

   if there is a GS' in GLOBALSTATE or EXTENDABLE which is marked as unique or the first occurrence such that GS is equal to GS' except the values taken on by the new dead variables

   then

   mark GS' as the first occurrence if GS' is originally unique,

   mark GS as a duplicated state of GS',

   for each child(GS) **do**

   Clear(child(GS), ε) {in Figure 4.}

   endfor

   endif

   for each state GS" that is a duplicate of GS **do**

   if there is a $GS''.S_i \neq GS.S_i, i = 1..n$, which results from the dead-to-live change

   then

   if there is a GS"' in GLOBALSTATE or EXTENDABLE which is marked as unique or the first occurrence such that GS"' is equal to GS" except the values taken on by the new dead variables

   then mark GS" as the duplicated state of GS"'.

   else add GS" to EXTENDABLE.

   endif

   else

   if GS becomes a duplicate state of GS'

   then mark GS" as a duplicate state of GS'

   endif

   endfor

   endif

   endfor

Figure 5: Dead to live analysis.

Procedure Delete-Process:

for each deleted transition e **do**

for each GS in GLOBALSTATE or EXTENDABLE which is generated by executing transition e **do**

if parent(GS) becomes an erroneous state

then mark parent(GS) as an erroneous state according to the error type,

remove GS from GLOBALSTATE or EXTENDABLE and delete the associated errors.

case GS

A unique state:

for each child(GS) **do**

Clear(child(GS), e).

endfor

The first occurrence:

promote one of the duplicated state GS_i of GS which is not GS's descendant state and is not generated by executing the deleted transition T as the first occurrence and add GS_i to EXTENDABLE.

for each child(GS) **do**

Clear(child(GS), e).

endfor

The second occurrence: do nothing

endcase

endfor

endfor

Figure 6: The process of deleting transitions.

Procedure Add-Process:

for each global state GS which is unique or first occurrence in GLOBALSTATE **do**

if $GS.S_i$ is equal to the head state of a new transition T_i of entity i

then

delete the corresponding errors associated with GS which will not occur after the new transition is added and add GS to EXTENDABLE.

endif

endfor

Figure 7: The process of adding transitions.

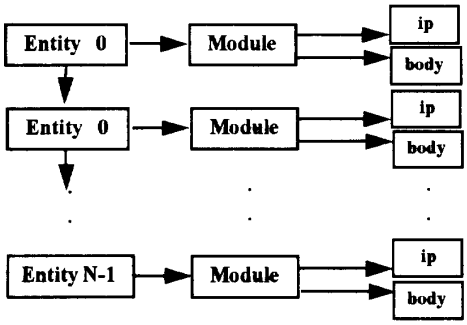


Figure 8: The structure of the ECFSM table.

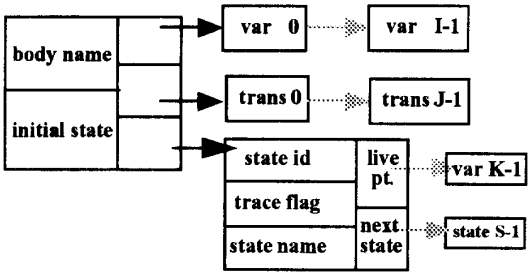


Figure 9: The structure of the module body and state.

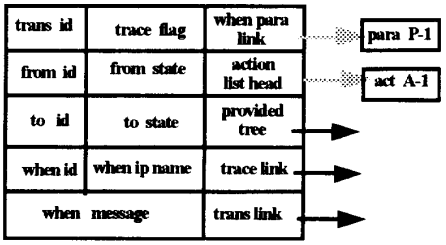


Figure 10: The structure of the transition list.

each ECFSM graph is traced to decide the dead variables for each state in each entity. The progress of traverse is Breadth First Search (BFS) in the corresponding ECFSM graph.
(5) Global State Initialization :
In the ECFSM model, a state vector which contains state identifier and context variables is used to record the state of each module (entity).

4.3 Incremental Protocol Verification for the ESTELLE specification

In order to apply the incremental verification for the protocols specified in Estelle, the modification of protocols is also specified in Estelle for designers' convenience. The major issue of the incremental Estelle translation is that it only needs to translate and interpret the modified part. That is, an incremental translation and interpreting is needed at the run time. The translation and interpreting of the added transition part is as follows: (1) Deciding to which entity and which ***module body*** the added transitions belong. (2) Translating and interpreting the added transitions to the corresponding transition structures. Therefore, The Estelle interpreter should have the ability of translating and interpreting the added transition block alone. To resolve the major difficultly, i.e., the incremental translation and interpreting, we have a minor change of the production rules of the Estelle's Backus-Naur Formalism(BNF) [10]. The minor change is in the Specification-rule of Estelle's BNF. Figure 11 shows the original format and the modified format. In the interpreter, a guarded flag is used to detect which path, i.e., "SPECIFICATION IDENTIFIER..." in Figure 11-(a) or the incremental path, "TRANS TransitionGroups"in Figure 11-(b) should be selected. (3) If the added transitions have no syntax errors after translating and interpreting, then the interpreter will link the new transitions into the transition link of the corresponding ***module body***.

When designers want to delete transitions, they will decide to which entity and ***module body*** the deleted transitions belong. The system will display all of the transitions and the associated transition IDs. After the designers' selection, the system will remove the deleted transitions.

After adding/deleting transitions, the incremental verification process described in the previous section is invoked to continue the verification.

5 Usage and Example

In this section, we will introduce our Incremental Protocol Verification System (IPVS), and gives an example to show the usage.

Currently, IPVS has been developed on a SUN SPARC workstation. A graphical user interface (GUI) based on the OPEN LOOK X window system is provided in IPVS.

The following functions are available for protocol designers in IPVS: (1) loading an existing text file of an Estelle-specified protocol or edit an Estelle-specified protocol from the file editor provided by

Specification ::= SPECIFICATION IDENTIFIER
SystemClass ';' DefaultOptions
TimeOptions BodyDefinition END '.' ;

(a) The original production rule in [10]

Specification ::= SPECIFICATION IDENTIFIER
SystemClass ';' DefaultOptions
TimeOptions BodyDefinition END '.' ;
|
TRANS TransitionGroups ;

(b) The modified production rule

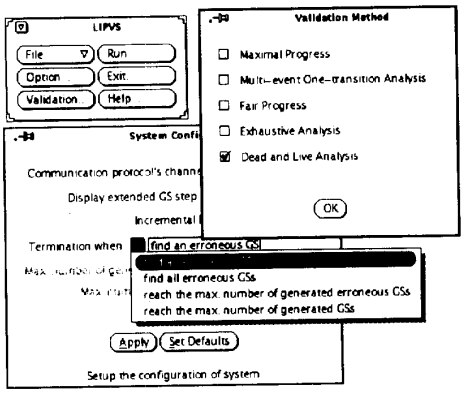
Figure 11: The modified Estelle's BNF.

IPVS, and preseting various maxiumum channel capacity. (2) Selecting incremental protocol verification optionally and selecting termination condition: find an erroneous global state, find all erroneous global states, reach the maxiumum number of generated global states, or reach the maxiumum number of generated erroneous global states. (3) Selecting a verification method from five methods provided by IPVS. (Figure 12-(a)) (4) Adding/deleting transitions, saving the modification to the Estelle-specified protocol and displaying the total number of explored global states. (Figure 12-(b)) (5) Displaying the global state sequence or transition sequence of a selected erroneous global state and printing function is available to print some formated reports, e.g., explored global states report, erroneous global states report, etc., to the laser printer. (Figure 12-(c))

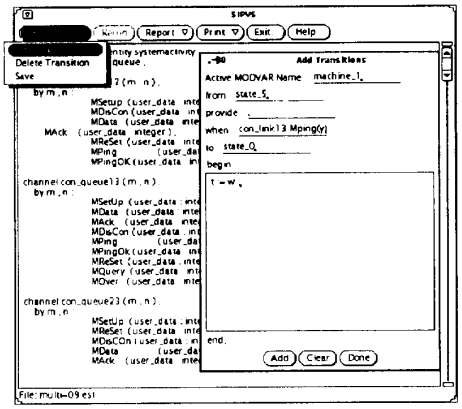
Figure 13 shows a 3-entity protocol specified in the ECFSM model, where 'm?' represents receiving a message from entity m, and 'n!' represents sending a message to entity n. In Figure 13, variables s, t, v, x, z of entity 1, variables s, t, x, z of entity 2, and variables t, x, z of entity 3 are live, and the others are dead. There is an unspecified reception error in the protocol. To eliminate the error, the protocol is modified as follows: adding "$\frac{3?MPing(y)}{t:=w}$" to entity 1, from state 5 to state 0, and adding "$\frac{1?MPing(y)}{t:=1-t}$" to entity 3, from state 4 to state 0. After modification, the attribute of variable w of entity 1 is changed from dead to live. As a result, some global states need to be re-explored. Using dead and live variables analysis, the numbers of global states of the original and modified protocols are 5233 and 5300, respectively, in which 67 states are newly generated. Without using incremental processing, all 5300 should be explored. There will be 12792 and 12879 global states respectively if the exhaustive method is used.

6 Conclusion

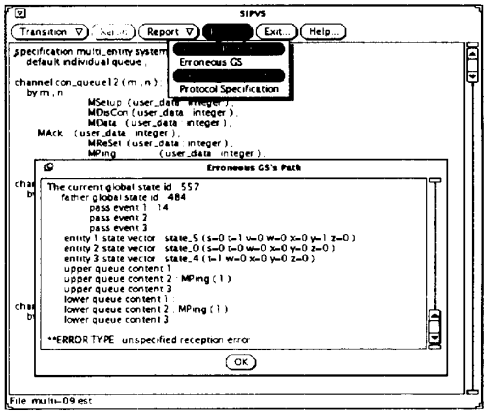
In this paper, we have proposed an ECFSM-based incremental protocol verification algorithm which also



(a)



(b)



(c)

Figure 12: Examples of using IPVS.

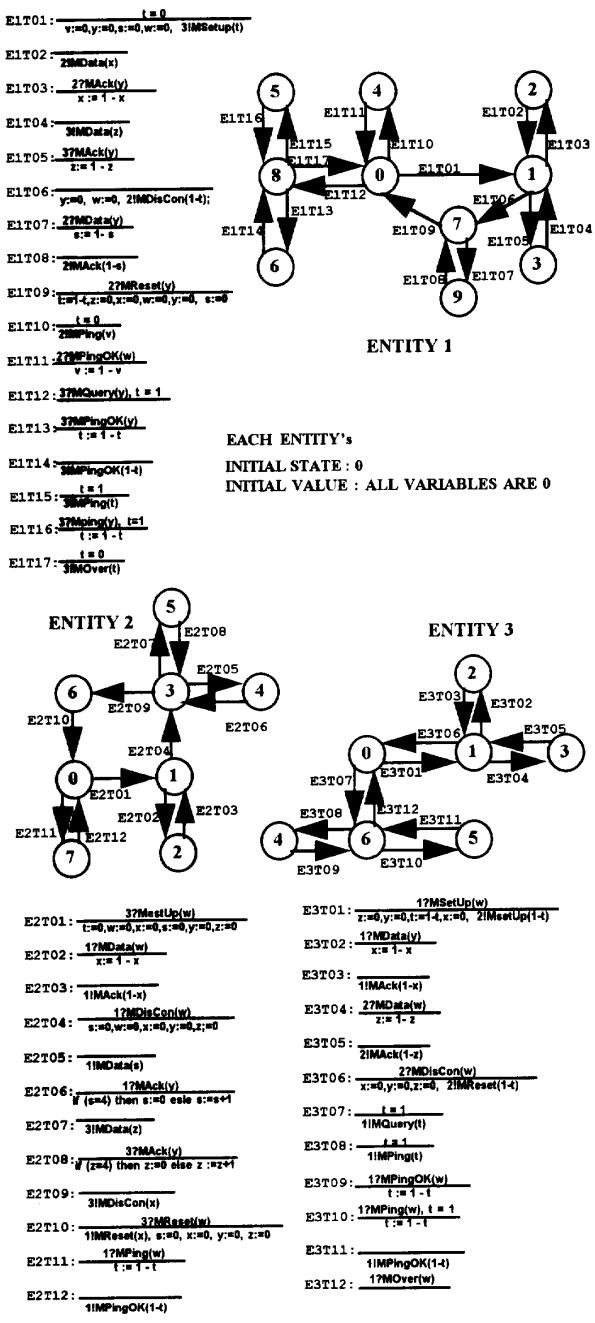


Figure 13: A 3-entity protocol specified in the ECFSM model.

incorporates Chu and Liu's ECFSM-based reduction technique. The corresponding incremental Estelle interpreter is also presented. To further reduce the number of gloabl state to be explored, some reduction technique, e.g., ECFSM-based fair progress state exploration [7], ECFSM-based maximal progress state exploration [8], multi-event-one-transition ECFSM-based reduction technique [6], are incorporated into our Estelle-based incremental protocol verification system for designers' choice.

Currently, our Estelle interpreter and the associated verification system cannot verify time-based protocols. The future work is to enhance the Estelle interpreter to be able to verify those protocols which have time requirements.

References

[1] B. Algayres, V. Coelho, L Doldi, H. Garavel, Y. Ljeune and C. Rodriguez, "VESAR: a pragmatic approach to formal specification and verification," ***Computer Networks and ISDN System (25)***, pp. 779-790, 1993.

[2] P. Y. Chu and M. T. Liu, "Global State Graph Reduction Techniques for Protocol Validation in the EFSM Model," ***Proc. of IEEE Phoenix Conf. on Computers and Communications***, pp. 371-377, 1989.

[3] J. P. Courtiat, "Estelle*: a Powerful Dialect of Estelle for OSI Protocol Description", Proc. IFIP WG6.1 8th International Symposiumon ***Protocol Specification, Testing, and Verification , VIII, 7-10, North-Holland***, pp. 171-186, 1988.

[4] J. P. Courtiat and Pierre de Saqui-Sannes, "ESTIM: an Intergrated Environment for the simulation and Verification of OSI Protocols specified in Estelle*", ***Computer Networks and ISDN System (25)***, pp. 83-98, 1993.

[5] C. M. Huang, Y. I. Chang, and M. T. Liu, "A Computer-Aided Protocol Design by Production System Approach," ***IEEE Journal on Selected Areas in Communications***, VOL. 8, NO. 9, pp. 1748-1762, 1990.

[6] H. Y. Lai, "A Reduced Incremental Protocol Verification System for N-Entity Estelle-Specified Protocols", ***Master thesis (in Chinese)***, Institute of Information Engineering, National Cheng Kung University, Taiwan R.O.C., 1992.

[7] J. C. Pong, "An Integrated Approach for Estelle-based Protocol Verification", ***Master thesis (in Chinese)***, Institute of Information Engineering, National Cheng Kung University, Taiwan R.O.C., 1992.

[8] J. M. Hsu, "A Computer Aided Incremental Protocol Verification System", ***Master thesis (in Chinese)***, Institute of Information Engineering, National Cheng Kung University, Taiwan R.O.C., 1992.

[9] ISO - Information Processing Systems - Open Systems Interconnection, "ESTELLE - A Formal Description Technique Based on Extended State Transition Model," ***DIS. 9074***, 1987.

[10] National bureau of Standard, "User Guide for the NBS Prototype Compiler for Estelle", Report NO. ICST/SNA-87/3, 1987.