Test Generation for a Protocol Specified in SDL with Complex Loops by Event-based EFSM Modeling

Tae-Hyong Kim†

Kumoh National Institute of Technology, Gumi, Korea

Summary

Test case generation for a network protocol by extended finite state machine (EFSM) based modeling is a well-known technique in formal methods in conformance testing. An input output based EFSM (IOEFSM) is a popular model for a protocol specified in the specification and description language (SDL) because an SDL process diagram is based on an input-driven EFSM. However, as an SDL specification may have a very complicated part such as complex nested loops, IOEFSM modeling may not be appropriate to represent such a complex SDL specification. This paper proposes a test generation method for a protocol specified in SDL using an event-based EFSM (EEFSM) for more exact modeling. It also shows the relations between an IOEFSM and an EEFSM, and their inter-conversion methods. Empirical results with a real network protocol, the service specific connection oriented protocol (SSCOP) showed the efficacy of the proposed method.

Key words:

Test generation, *SDL*, *EFSM Modeling*, *Loop testing*

1. Introduction

Lately as the user requirements for telecommunication services have been getting diverse and composite, communication protocols are getting more various and complex. Thus how to develop a reliable protocol is a major issue in the protocol engineering and conformance testing methodology and framework was standardized to verify whether a protocol implementation conforms to the standard and the specification of that protocol [1]. The development of formal description techniques for clear and precise specification of communication protocols has enabled formal methods in protocol design, verification, and testing. The specification and description language (SDL) is the most popular formal description technique which is widely used in design of various distributed systems owing to its graphical notations [2]. Formal methods in conformance testing with SDL have been interested in automatic test generation of a protocol specification for a long time. Model based test generation is a general method that makes a model for a protocol and analyzes that model to derive test cases. A finite states machine (FSM), an extended finite state machine (EFSM), and a labeled transition system (LTS) are well-known models. Since a process diagram in SDL for behavioral description is based on an EFSM model, an EFSM has been mainly used for modeling and test case generation of a communication protocol specified in SDL [3-6].

Interestingly, most of the existing EFSM-based test generation methods use input/output-based EFSM (IOEFSM)'s where the label of a transition has an I/O pair. An input to a protocol and its response, (an) output(s), of that protocol normally represent a unit behavior of a protocol in an IOEFSM. Since a transition in an SDL process diagram surely starts with an input and may have (an) output(s), it seems natural to model an SDL process specification into an IOEFSM. However, an SDL process diagram has high flexibility to specify protocol behaviors, e.g., nested loops inside a transition, algorithmic notations in a task, save symbols, and etc. If you want to model such a complicated specification into an IOEFSM model, you may have to transform that specification to a simple equivalent one [7], or remove the complicated parts by assuming some simplicity rules on SDL specifications [3].

Actually, an event-based EFSM (EEFSM) may be a more appropriate model for an SDL specification because a transition of an EEFSM, whose label has one event, e.g., an input or an output, is able to model an SDL transition behavior more exactly than an IOEFSM's transition. An EEFSM is, however, somewhat troublesome to handle due to its flexibility in the definition of events. There has been little work of modeling and test generation with EEFSM's. This paper proposes a model-based test generation method for an SDL specification especially with complex loops using an EEFSM model. It also examines the properties of an EEFSM as a protocol model and the relation between an EEFSM and an IOEFSM.

This paper is organized as follows. Section 2 defines an EEFSM and its properties. The relations between an IOEFSM and an EEFSM with their inter-conversions are also described in that section. Then, the proposed test generation method with EEFSM modeling of an SDL specification is explained in section 3. Section 4 shows

Manuscript received March 5, 2008
Manuscript revised March 20, 2008

empirical results by applying the proposed method to a real network protocol. Finally, conclusions are drawn in section 5.

2. Event-Based EFSM and its Properties

This section describes the basic definitions and properties of an EEFSM, and its relation to an IOEFSM.

2.1 Basic definitions and properties

As the specification model of a protocol, an input-output based EFSM (IOEFSM) and an event-based EFSM (EEFSM) are defined respectively as follows.

Definition 1. An IOEFSM M is the 6-tuple (S, s_0, I, O, $\tilde{v}$, T) where S is the finite set of logical states, $s_0(\in S)$ is the initial state, I and O are the finite sets of input and output declarations respectively such that $i(\tilde{p})\in I$ and $o(\tilde{v},\tilde{p})\in O$, where $\tilde{p}$ is the finite set of input parameters, $\tilde{v}$ is the finite set of variables, T is the finite set of transitions, where the label of a transition $t(\in T)$ is denoted by the 5-tuple (s_s, s_f, $i(\tilde{p})$, $P(\tilde{v},\tilde{p})$, $A(\tilde{v},\tilde{p},O)$) in which s_s and s_f are the start and the final state of t respectively, and $P(\tilde{v},\tilde{p})$ and $A(\tilde{v},\tilde{p},O)$) are the predicate and the action of t respectively.

Definition 2. An EEFSM N is the 5-tuple (S, s_0, $\tilde{v}$, Σ, T) where S is the finite set of logical states, $s_0(\in S)$ is the initial state, $\tilde{v}$ is the finite set of variables, Σ is the finite set of events the element of which $\varepsilon(\in\Sigma)$ is denoted by $\sigma e(\tilde{p})$ or $C(\tilde{v},\tilde{p})$, where $\sigma\in\{?,!\}$, e is a message name, $\tilde{p}$ is the finite set of event parameters, and C is a logical equation constructed with $\tilde{v}$ and $\tilde{p}$, and T is the finite set of transitions, where the label of a transition $t(\in T)$ is denoted by the 4-tuple (s_s, s_f, ε, $A(\tilde{v},\tilde{p})$) in which s_s and s_f are the start and the final state of t respectively, and $A(\tilde{v},\tilde{p})$) is the action of t.

Note that an event in an EEFSM can be a logical equation as well as an input or output triggering as different in the definitions of others [8]. Since SDL, our target specification language, allows nested loops in a transition, we included conditions for behavioral choice in event set to model and handle such a transition more exactly. A transition t=(s_s, s_f, ε, $A(\tilde{v},\tilde{p})$) is called a *condition transition*, an *input transition*, or an *output transition* when ε is $C(\tilde{v},\tilde{p})$, $?e(\tilde{p})$, or $!e(\tilde{p})$, respectively.

A major difference between an IOEFSM and an EEFSM is about state transitions. While state transitions of an IOEFSM depend on the global state and triggered inputs, those of an EEFSM are up to events instead of inputs. This difference varies the properties, modeling and test generation of an EEFSM against those of an IOEFSM. In general, if an IOEFSM model is used, test generation takes the following assumptions on that model for simplicity: an IOEFSM is minimized, deterministic, strongly connected, and completely specified [9]. Those assumptions, however, should be redefined in an EEFSM due to their differences. For example, the complete specification assumption on an IOEFSM is that for each possible global state and for each (parametered) input, there is a firable transition. In an EEFSM, however, there may be some events that cannot happen at a certain logical state. We define those assumptions for an EEFSM model generated from an SDL specification as follows.

Definition 3. An EEFSM N is *minimized* if for every pair of its logical states (s_i, s_j), there no event sequence x ($\in\varepsilon^*$) that can happen at both s_i and s_j.

Definition 4. An EEFSM N is *deterministic* if for each logical state $s(\in S)$ and for each possible event $\varepsilon(\in\Sigma)$ at s, there is at most one transition defined in N.

Definition 5. An EEFSM N is *strongly connected* if for every pair of its logical states (s_i, s_j), there is a transition path going from s_i to s_j.

As for the completeness of an EEFSM, it is important what kinds of events are allowed to be fetched at a logical state, which is closely related to the operation of an EEFSM. Fig.1 shows two different approaches to EFSM modeling. Intrinsic view modeling is a general one where inputs or events are fetched by the machine when needed by means of local channel. SDL process modeling is a typical intrinsic view modeling. In an SDL system, each process has a FIFO queue for buffering input signals. The foremost buffered signal is consumed by a process in general when that process finishes its action and stops logically at the next state. An SDL transition means a unit behavior initiated by only an input and no inputs are allowed to be consumed during executing a transition.

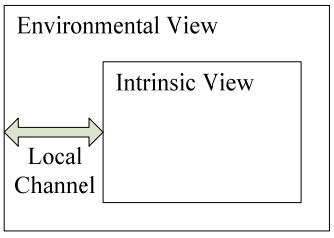

Fig. 1 The concept of EFSM modeling

By the way, environmental view modeling is usually used for description of the whole system as seen by an external tester. This modeling depends on local channel design and channel access mechanism of the system. Triggering of every input sequence by the environment can be represented without being interrupted by other types of

events. Reachability trees are often constructed with this approach. If we apply the concept of IOEFSM's completeness to an EEFSM, a completely specified EEFSM may be unbounded. Therefore, we focus an EEFSM which was constructed only by intrinsic view modeling approach in this paper.

When we construct an EEFSM model from an SDL specification, only a part of logical states of that EEFSM can be mapped to their corresponding SDL states. Each of the other logical states just indicates a certain middle points on a transition of that SDL specification. From this consideration we can classify logical states of an EEFSM constructed from an SDL specification as follows. According to that classification, the completeness of such an EEFSM is defined.

Definition 6. A logical state s of an EEFSM $N_{SDL}=(S, s_0, \tilde{v}, \Sigma, T)$ constructed from an SDL specification is an *input state* if there exist outgoing input transitions from s; otherwise, s is a *transient state*.

Definition 7. An EEFSM N_{SDL} constructed from an SDL specification is *completely specified* if for each input state $s_I \in S$ and for each input event $\varepsilon_I \in \Sigma$, there is a transition defined in N_{SDL}.

2.2 Relations between an IOEFSM and an EEFSM

Now we show how to transform an IOEFSM to its equivalent EEFSM and vice versa from the examination of the relations between an IOEFSM and an EEFSM. Actually many communication protocols have been modeled into IOEFSM's and a lot of test generation methods use an IOEFSM as an analysis model. As we seen the definitions and discussions of those machines in subsection 2.1, the relations between logical states of two machines is a key to that machine transformation problem. A logical state of an IOEFSM can correspond to a series of logical states of an EEFSM which is composed of one or more logical states where foremost logical state is an input state.

First, the conversion from an IOEFSM to an equivalent EEFSM requires splitting of logical states of an IOEFSM. A logical state of an IOEFSM with an outgoing transition whose label has an output or a valid predicate should be split according to possible events of an EEFSM. We can obtain an initial form of EEFSM by this splitting states and reconstructing transitions. The final EEFSM is obtained by the minimization of that machine. In the minimization of an EFSM, variable domain at a state must be considered to check if two logical states are identical.

Some notation and functions are introduced to explain two machine transformation algorithms. Let Δ and Λ denote the domains constructed from all control variables in $\tilde{v}$ and all parameters in $\tilde{p}$ of the input events, respectively. The functions $R_\Delta(\cdot)$: $P(\tilde{v},\tilde{p}) \rightarrow \wp(\Delta)$ and $R_\Lambda(\cdot)$: $P(\tilde{v},\tilde{p}) \rightarrow \wp(\Lambda)$ transform a predicate to the subdomains of Δ and Λ that satisfy that predicate respectively, where $\wp(\cdot)$ is the powerset operator. Their inverse functions $R_\Delta^{-1}(\cdot)$: $\wp(\Delta) \rightarrow P(\tilde{v},\tilde{p})$ and $R_\Lambda^{-1}(\cdot)$: $\wp(\Lambda) \rightarrow P(\tilde{v},\tilde{p})$ generate the predicates that determine the input subdomains of Δ and Λ, respectively. The subset of Δ allowed at a state s is called the *domain* of the state s and denoted by $d(s)$. $s_s(\cdot)$:$T \rightarrow S$ and $s_f(\cdot)$:$T \rightarrow S$ are the starting state and final state functions of a transition respectively. Algorithm 1 shows in detail how to transform an IOEFSM to an EEFSM that is equivalent to that IOEFSM. Functions *out* and *act* of an action extract the first output and the part of action before that output from that action respectively, and *rem* of a pair of an action and an output extracts the following part of that action after that output.

Algorithm 1. Conversion of an IOEFSM to an EEFSM
- Inputs: IOEFSM $M = (S_M, s_0, I, O, \tilde{v}, T_M)$
- Output: EEFSM $N = (S_N, s_0, \tilde{v}, \Sigma, T_N)$

Step.1: Initialize variables as follows:
$S_N \leftarrow S_M$, $\Sigma \leftarrow \varnothing$, $T_N \leftarrow \varnothing$, $d(S_N) \leftarrow \Delta$;

Step.2: $i \leftarrow 0$; /* initial splitting */
for each transition $t_M (\in T_M) = (s_s, s_f, i(\tilde{p}), P(\tilde{v},\tilde{p}), A(\tilde{v},\tilde{p},O))$
  $e_n \leftarrow ?i(\tilde{p})$, $\Sigma \leftarrow \Sigma \cup \{e_n\}$; /* e_n: the next event */
  $d_n \leftarrow \Delta$; /* e_n: domain of the next state */
  if $P(\tilde{v},\tilde{p}) \neq \varnothing$ then $S_N \leftarrow S_N \cup \{s_i\}$, $t_N \leftarrow (s_s, s_i, e_n, \varnothing)$;
    $T_N \leftarrow T_N \cup \{ t_N \}$, $i \leftarrow i+1$, $e_n \leftarrow P(\tilde{v},\tilde{p})$;
    $\Sigma \leftarrow \Sigma \cup \{e_n\}$, $d_n \leftarrow R_\Delta(P(\tilde{v},\tilde{p}))$;
  endif
  if $A(\tilde{v},\tilde{p},\varnothing)$ then
    $t_N \leftarrow (s_s, s_f, e_n, A(\tilde{v},\tilde{p}))$, $T_N \leftarrow T_N \cup \{ t_N \}$, $d(s_s) \leftarrow d_n$;
  else $j \leftarrow 0$, $A_i(\tilde{v},\tilde{p},O) \leftarrow A(\tilde{v},\tilde{p},O)$;
    while not $A_i(\tilde{v},\tilde{p},\varnothing)$
      $A_{i,j}(\tilde{v},\tilde{p}) \leftarrow act(A_i(\tilde{v},\tilde{p},O))$, $o_j(\tilde{p}) \leftarrow out(A_i(\tilde{v},\tilde{p},O))$;
      $S_N \leftarrow S_N \cup \{s_i\}$, $t_N \leftarrow (s_{i-1}, s_i, e_n, A_{i,j}(\tilde{v},\tilde{p}))$;
      $T_N \leftarrow T_N \cup \{ t_N \}$, $d(s_{i-1}) \leftarrow d_n$, $i \leftarrow i+1$;
      $e_n \leftarrow !o_j(\tilde{p})$, $\Sigma \leftarrow \Sigma \cup \{e_n\}$;
      $A_i(\tilde{v},\tilde{p},O) \leftarrow rem(A_i(\tilde{v},\tilde{p},O), o_j(\tilde{p}))$, $j \leftarrow j+1$;
    endwhile
    $t_N \leftarrow (s_i, s_f, e_n, act(A_i(\tilde{v},\tilde{p},O))$;
    $T_N \leftarrow T_N \cup \{ t_N \}$, $d(s_i) \leftarrow d_n$;
  endif
endfor

Step.3: /* minimization */
for each state $s_i \in S_N$
  for each transition pair t_j, $t_k \in T_N$ such that
    $t_j = (s_a, s_i, \varepsilon_a, A_a(\tilde{v},\tilde{p}))$ and $t_k = (s_b, s_i, \varepsilon_b, A_b(\tilde{v},\tilde{p}))$
    if $\varepsilon_a = \varepsilon_b$ and $A_a(\tilde{v},\tilde{p}) = A_b(\tilde{v},\tilde{p})$ then
      if $d(s_a) = d(s_b)$ then
        $T_N \leftarrow T_N - \{t_k\}$, $S_N \leftarrow S_N - \{s_b\}$;
      endif
    endif

```
   endfor
 endfor
```
Step.4: return $N=(S_N, s_0, \tilde{v}, \Sigma, T_N)$;

By the way, for the conversion of an EEFSM to an equivalent IOEFSM, we must put some restrictions on the target EEFSM because a general EEFSM has high flexibility in modeling. First, the target EEFSM must be an intrinsic view model. Some environmental view model EEFSM's do not have their equivalent IOEFSM's because their various channel management systems cannot be considered in IOEFSM's. Second, a condition transition must follow an input or other condition transition with null action only in the target EEFSM. Otherwise, constructing an equivalent IOEFSM requires complex semantic analysis of that EEFSM including domain-based state partition, and some IOEFSM should have spontaneous transitions with no inputs. Algorithm 2 shows the conversion of an EEFSM to an IOEFSM which is equivalent to that EEFSM. There is no minimization step in algorithm 2. If an EEFSM is minimized, the converted IOEFSM is also minimized with the above restrictions on an EEFSM.

Algorithm 2. Conversion of an EEFSM to an IOEFSM
• Inputs: EEFSM $N = (S_N, s_0, \tilde{v}, \Sigma, T_N)$
• Output: IOEFSM $M = (S_M, s_0, I, O, \tilde{v}, T_M)$
Step.1: Initialize variables as follows:
$S_M \leftarrow \varnothing$, $I \leftarrow \varnothing$, $O \leftarrow \varnothing$, $T_M \leftarrow \varnothing$;
Step.2: /* transition construction */
for each input transition $t_{N,i}(\in T_N)=(s_{s,i}, s_{f,i}, ?i_i(\tilde{p}), A_i(\tilde{v},\tilde{p}))$
  $S_M \leftarrow S_M \cup \{ s_{s,i} \}$, $I \leftarrow I \cup \{ i_i(\tilde{p}) \}$;
  $s_{s,M} \leftarrow s_{s,i}$, $s_{f,M} \leftarrow s_{f,i}$, $i_M \leftarrow i_i(\tilde{p})$, $P_M \leftarrow null$, $A_M \leftarrow A_i(\tilde{v}, \tilde{p})$;
  $e_p \leftarrow$ input; /* e_p : previous event, input/output/cond */
  SearchTr(N, $s_{f,M}$);
endfor
Step.3: return $M=(S_M, s_0, I, O, \tilde{v}, T_M)$;

```
function SearchTr(EEFSM N, state s)
  for each transition t_{N,j}(∈T_N)=(s_{s,j}, s_{f,j}, ε_j, A_j(ṽ,p̃))
      such that s_{s,j} = s
    if ε_j = ?i_j(p̃) then
      if s_{s,j} ∈ S_M then
        return;
      else
        T_M ← T_M ∪{ (s_{s,M}, s_{f,M}, i_M, P_M, A_M) };
        S_M ← S_M ∪{ s_{s,j} }, I ← I ∪{ i_j(p̃) };
        s_{s,M} ← s_{s,j}, s_{f,M} ← s_{f,j}, i_M ← i_j(p̃);
        P_M ← ∅, A_M ← A_j(ṽ,p̃), e_p ← input;
        SearchTr(N, s_{f,M});
      endif
    else if ε_j = !o_j(p̃) then
      A_M ← A_M @ !o_j(p̃) @ A_j(ṽ,p̃), e_p ← output;
          /* @: action concatenation operator */
    else if ε_j = C_j(ṽ,p̃) then
      if not((e_p=input or cond) and A_M=null))
        return EFSM_Err; /* EEFSM restriction error */
      P_M ← P_M and C_j(ṽ,p̃), A_M ← A_M @ A_j(ṽ,p̃);
      e_p ← cond;
    endif
  endfor
endfunction
```

We show a simple machine conversion example with Responder module of Inres protocol which is widely used as a test protocol in protocol engineering [10]. Fig. 2 shows the target IOEFSM model of Responder which has 3 states and 8 transitions except transitions for making complete specification. The transformed EEFSM is shown in Fig.3 which has 12 states and 17 transitions.

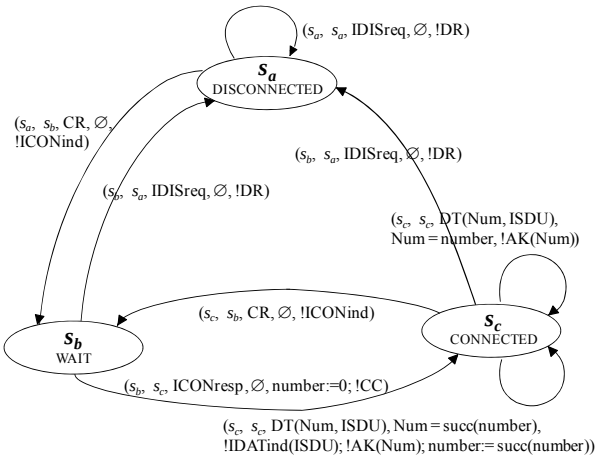

Fig. 2 An IOEFSM model of Inres Responder protocol

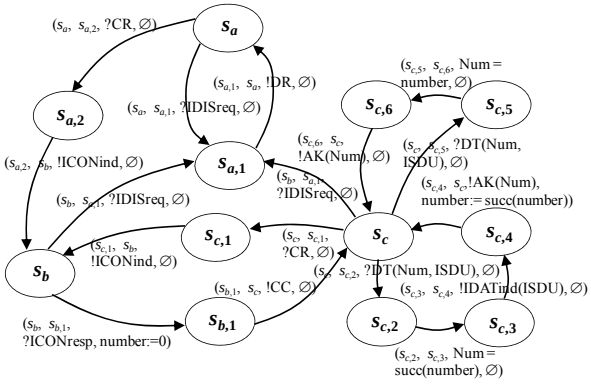

Fig. 3 The transformed EEFSM of Inres Responder protocol

3. Test Generation with an EEFSM Modeling

3.1 An EEFSM modeling of SDL specifications

SDL has a variety of syntax to describe complicated behaviors of a network protocol in a simple and clear

manner. Although SDL is a formal description language, it has some syntax to describe nondeterministic or random behavior such as spontaneous transitions with 'NONE' input or 'ANY' condition. Therefore, such complicated or troublesome syntax is not supported fully by SDL tools.

In this paper, we also put some restrictions on an SDL specification of a protocol. First, we do not allow nondeterminism in an SDL model. This restriction is reasonable for normal network protocols. Second, we avoid so-called shorthand syntax including saves or enabling conditions which can be rewritten with simple and basic syntax [11]. An SDL specification having some parts designed with such shorthand syntax is rewritten for easier test case generation. An SDL process diagram is assumed to be composed of the following basic symbols after rewriting: states, inputs, conditions, tasks, and outputs. Lastly, we assume that an SDL specification is well-designed: free of deadlocks, livelocks, and infinite loops with unbounded global control state space [7].

Even if such a simple process diagram, its EFSM modeling for test generation may be straightforward due to complex control flows with loops. Fig. 4 shows a part of a simple SDL process diagram whose control flows have some loops. As the exact control flows including the traversing number of loops depend on the global state, in other words, the previous path from the initial state, it may be very difficult to simply model that part as a normal IOEFSM. This paper uses an EEFSM as a target model for test generation to cope with that problem. The EEFSM modeling rules for an SDL process diagram are as follows.

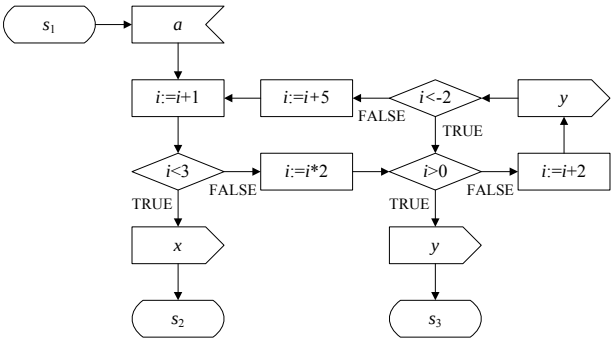

Fig. 4 A part of SDL design with complex loops

– Each state symbol is changed by a logical state.
– A logical state is inserted just before each condition and output symbol.
– A control flow branch created by a (series of) condition symbol(s) and the following (series of) task symbol(s) is converted to a condition transition.
– An input/output condition symbol and the following (series of) task symbol(s) is converted to an input/output transition.

Fig. 5 shows the EEFSM converted from the SDL process diagram in Fig.4. That EEFSM model shows the control flows of the original SDL diagram identically, which indicates that an EEFSM is appropriate for modeling an SDL specification. Any complex control flows of an SDL specification with the above restrictions can be represented by an EEFSM model.

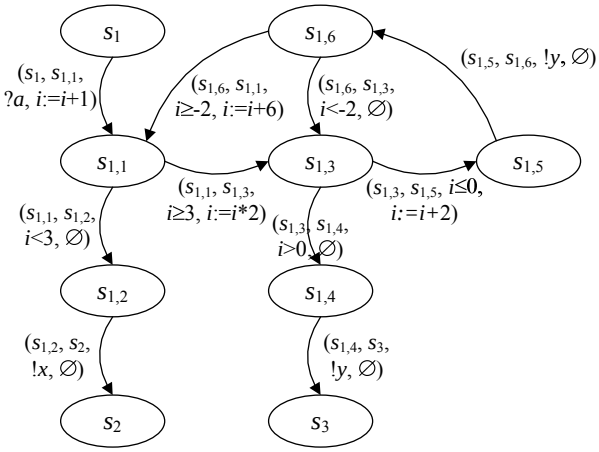

Fig. 5 The EEFSM converted from an SDL design in Fig.4

3.2 Test generation from an EEFSM model

Test case generation of an (E)FSM model for conformance testing has been studied in many literatures [3-6]. They tried to derive optimized test cases with high fault coverage and low cost, and a lot of test generation methods were developed [12]. Those test generation methods are mostly based on an IO(E)FSM model for easily checking the current logical state of the target system under test with outputs. Actually test generation for an E(E)FSM model will not be fundamentally different from that for an IO(E)FSM. But the existing methods may have to be adjusted according to the properties of an E(E)FSM.

A most serious issue of test generation for an EFSM with data flows is to generate executable test cases. To make a test case executable in an EFSM is to decide a suitable preamble path and values of input parameters for that test cases. Semantic or domain analysis is usually used for that purpose. However, some methods may not be appropriate to an EEFSM model. For example, the expansion of an EEFSM based domain partitioning of each state [3] is not a suitable method because it requires all transitions of an EFSM to be able to have predicates. In this paper, we use a modified backward tracking with symbolic evaluation and linear programming to find an executable path under the assumption that all conditions and actions can be represented as (in)equalities and assignments with linear equations respectively. Details of that method will be shown in the proposed test generation algorithm.

Conformance testing of an (E)FSM M with n states, a machine identification experiment, requires to generate a checking sequence of M that distinguishes M from all other machines with n sates [9]. Checking experiment can be accomplished by distinguishing, characterizing, or indentifying sequences. In this paper, we try to use unique input output (UIO) sequences in order to decrease the length of test sequences [9]. As for the test coverage, both state identification and transition identification must be considered in test generation for the full fault coverage of control flows [12]. Those checking experiment and state/transition identification are based on inputs and outputs of an IO(E)FSM. A transition of an EEFSM, however, does not have either an input or an output, or even anything. Therefore, state and transition identification with UIO sequences are modified for an EEFSM in this paper as follows.

First, UIO sequences for checking experiment are generated not for all logical states but only input states in an EEFSM. UIO sequences composed of unconditional transitions only are preferred to be used in state and transition identifications due to their better applicability. A condition transition with a condition event $C(\tilde{v},\tilde{p})$ is called a conditional transition if is $\tilde{v}$ not null and $\tilde{p}$ is null; otherwise an unconditional transition. Second, test cases for state identification are generated only for all input states in an EEFSM. Third, test cases for transition identification are generated only for transitions terminating at input states in an EEFSM. For the other transitions, test cases only for transition tour coverage are coverage in our method. For higher fault coverage, data flow test cases satisfying condition coverage or all-uses criterion may be added [13]. Our test generation method is shown in algorithm 3. We assume that there is a reliable reset input ri in a protocol system.

Algorithm 3. Test generation for an EEFSM
- Inputs: EEFSM $N = (S, s_0, \tilde{v}, \Sigma, T)$
- Output: the set of test cases TS_N

Step.1: (symbolic evaluation of variables) Starting from each input state, for every possible control flows to next input state, action and condition parts are rewritten as follows. If (1) a variable v_1 is defined in a transition t_1: $v_1 \leftarrow f_1(\tilde{v},\tilde{p})$, (2) v_1 is used to define other variable v_2, $v_2 \leftarrow f_2(\tilde{v},\tilde{p})$, or is used in a condition event $C_2(\tilde{v},\tilde{p})$, in a following transition t_2, and (3) there is no other definition of v_1 and v_2 on any transition path from t_1 to t_2, then the definition of v_2 and the condition of $C_2(\tilde{v},\tilde{p})$ in t_2 are rewritten as $v_2 \leftarrow f_2 \bullet f_1(\tilde{v},\tilde{p})$ and $C_2(f_1(\tilde{v},\tilde{p}),\tilde{p})$, respectively.

Step.2: (preambles generation) For each input state $s_i \in S$, decide an executable transition path starting from the initial state s_0 and terminating at s_i, which is called a preamble to s_i (denoted by $pr(s_i)$). When deciding $pr(s_i)$, an unconditional preamble is preferred which is composed of unconditional transitions only. If there is no unconditional preamble, a set of several conditional preambles is generated, which is denoted by $Pr(s_i) = \{ pr_j(s_i) | pr_j(s_i)$ is a possible conditional preamble to $s_i \}$.

Step.3: (UIO sequences generation) For each input state $s_i \in S$, decide a minimal-length UIO sequence of s_i, denoted by $uio(s_i)$. When deciding $uio(s_i)$, an unconditional UIO sequence is preferred which is constructed from unconditional transitions only. If there is no unconditional UIO sequence, a set of several conditional UIO sequence is generated, which is denoted by $Uio(s_i) = \{ uio_j(s_i) | uio_j(s_i)$ is a possible conditional UIO sequence of $s_i \}$.

Step.4: (test generation for state identification) For each input state $s_i \in S$, $i=1,\ldots,m$, construct a test case for state identification as follows: $tcs_i \leftarrow ri \ @ \ pr_{(1)}(s_1) \ @ \ \mathrm{uio}_{(1)}(s_1) \ \ldots \ ri \ @ \ pr_{(m)}(s_m) \ @ \ \mathrm{uio}_{(m)}(s_m)$, where @ is the string concatenation operator and $pr_{(j)}(s_j)$ and $\mathrm{uio}_{(j)}(s_j)$, $j=1,\ldots,m$, are an unconditional or conditional but executable preamble to s_j and UIO sequence of s_j in tcs_i. If there exist parameters at some input events in tcs_i, decide the values of those parameters by linear programming. Then, perform $TS_N \leftarrow \{ tcs_i \mid i=1,\ldots,m \}$.

Step.5: (transition construction for transition identification) For each input state pair (s_i, s_j), let $T_{i,j} \subset T$ denote the set of all transitions constructing every path starting from s_i and terminating at s_j without visiting any input state. For each input state pair (s_i, s_j), construct a set of transition sequences denoted by $\Im_{i,j}$ as follows. (1) At state s_j, select all transitions $t_1,\ldots,t_n$ which terminate at s_j, perform $\tau_{i,j} \leftarrow \{t_1,\ldots,t_n\}$, and $T_{i,j} \leftarrow T_{i,j} - \{t_1,\ldots,t_n\}$. (2) Select a transition sequence ts_k whose starting state is s_k. Find all transitions t_k, $k=1,\ldots,m$, terminating at s_k such that $d(s_k) \cap d(s_s(ts_k)) \neq \varnothing$, and replace ts_k by $t_k @ ts_k$, $k = 1, \ldots, m$ in $\tau_{i,j}$, and perform $T_{i,j} \leftarrow T_{i,j} - \{t_1,\ldots,t_k\}$. If there is no such transitions, perform $\tau_{i,j} \leftarrow \tau_{i,j} - \{ts_k\}$. If for a transition t_k, $s_s(t_k)$ is an input state, perform $\Im_{i,j} \leftarrow \Im_{i,j} \cup \{ t_k @ ts_k \}$ and $\tau_{i,j} \leftarrow \tau_{i,j} - \{ t_k @ ts_k \}$. (3) Iterate the subprocess (2) until $T_{i,j} \neq \varnothing$.

Step.6: (test generation for transition identification) For every input state pair (s_i, s_j), and for each transition sequence $ts_k \in \Im_{i,j}$, construct a test case for transition identification as follows: $tct_{i,j,k} \leftarrow ri \ @ \ pr_{(i,k)}(s_i) \ @ \ ts_k \ @ \ \mathrm{uio}_{(j)}(s_j)$, where $pr_{(i,k)}(s_j)$ and $\mathrm{uio}_{(j)}(s_j)$ are an executable unconditional or conditional preamble to s_i and UIO sequence of s_j in $tct_{i,j,k}$. If there exist parameters at some input events in $tct_{i,j,k}$, decide the values of those parameters by linear programming. Then, perform $TS_N \leftarrow TS_N \cup \{ tct_{i,j,k} \mid$ for every possible i, j, and $k \}$.

4. Empirical Results

In order to show the efficacy of the proposed test generation method, we applied it to the SSCOP (Service

Specific Connection Oriented Protocol) used in the B-ISDN AAL(ATM Adaptation Layer). This protocol has been often used for test generation because its SDL specification is shown in ITU-T Recommendation Q.2110 [14]. That SDL specification of the SSCOP is very complicated. The SSCOP process diagram is composed of 43 pages and has 10 states, 149 transitions, and 98 subtransitions constructing branches or loops. Fig.6 shows a complicated part of the SSCOP process diagram which has several complex loops. Test generation for such a part seems to be not easy with IOEFSM modeling. We modeled the SSCOP process diagram into an EEFSM. Fig.7 shows the EEFSM model of the part of the SSCOP in Fig.6, where only an event is shown as the label of a transition for simplicity.

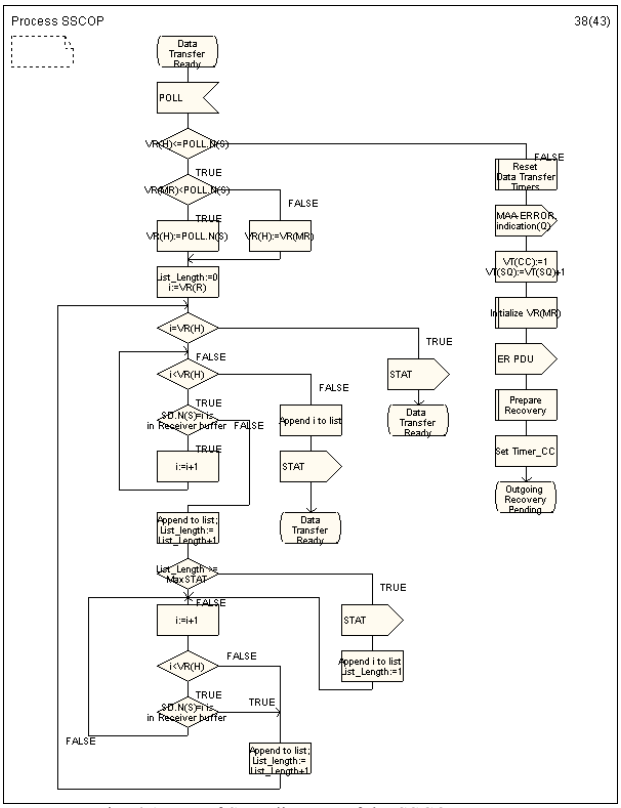

Fig. 6 A part of SDL diagram of the SSCOP process

After the modeling of the SSCOP, we generated test cases for the SSCOP according to the proposed test generation algorithm shown in algorithm 3. First, symbolic evaluation of variables was performed to simply actions and conditions. For example, the condition event of t_7 in Fig.7 was able to be rewritten by $VR(R)=N(S)$. As $N(S)$ is an input parameter, this condition transition can be an unconditional transition owing to this symbolic evaluation.

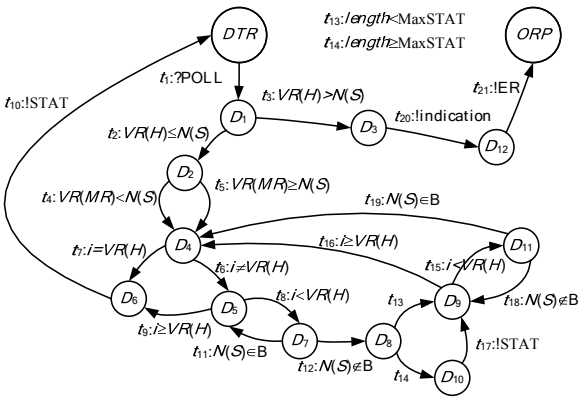

Fig. 7 The simplified EEFSM model of the SDL specification in Fig.6

Then, a preamble to each input state is generated. For example, an unconditional preamble for an input state *DTR* in Fig.6, $pr(DTR)$, was generated: $pr(DTR)$ = ?BGREJ !* ?END !* ?AA-ESTABLISH.request !BGN ? BGAK !AA-ESTABLISH.confirm. But some more preambles were generated for transition identification. Next, an UIO sequence of each input state is derived. Among 10 input states, 8 input states have unconditional UIO sequences and the remaining input states have only conditional ones. The state *DTR* has an unconditional UIO sequence: $uio(DTR)$=?Timer_POLL !POLL. With those UIO sequences, test cases for state and transition identification were generated. For transition identification, transition sequences were generated between two input states. The transition sequences generated for the part of Fig.6 are as follows, where transition concatenation operator is omitted for simplicity.

$ts_1=t_1t_3t_{20}t_{21}$,
$ts_2=t_1t_2t_4t_6t_{10}$,
$ts_3=t_1t_2t_5t_7t_9t_{10}$,
$ts_4=t_1t_2t_4t_7t_8t_{11}t_9t_{10}$,
$ts_5=t_1t_2t_5t_7t_8t_{12}t_{13}t_{16}t_6t_{10}$,
$ts_6=t_1t_2t_5t_7t_8t_{12}t_{13}t_{15}t_{19}t_7t_8t_{11}t_9t_{10}$,
$ts_7=t_1t_2t_5t_7t_8t_{12}t_{13}t_{15}t_{18}t_{16}t_6t_{10}$,
$ts_8=t_1t_2t_5(t_7t_8t_{12}t_{13}t_{15}t_{19})^* t_7t_8t_{12}t_{14}\ t_{17}t_{16}t_6t_{10}$,

Finally, we generated 10 test cases for state identification and 200 test cases for transition identification. For example, test cases for transition identification for the part in Fig.6 are as follows.

$tct_{DTR,ORP,1}= ri@pr(DTR)@ts_1@uio(ORP)$
$tct_{DTR,DTR,1}= ri@pr(DTR)@ts_2@uio(DTR)$
$tct_{DTR,DTR,2}= ri@pr_2(DTR)@ts_3@uio(DTR)$
$tct_{DTR,DTR,3}= ri@pr_3(DTR)@ts_4@uio(DTR)$
$tct_{DTR,DTR,4}= ri@pr_4(DTR)@ts_5@uio(DTR)$
$tct_{DTR,DTR,5}= ri@pr_5(DTR)@ts_6@uio(DTR)$
$tct_{DTR,DTR,6}= ri@pr_6(DTR)@ts_7@uio(DTR)$
$tct_{DTR,DTR,7}= ri@pr_7\ (DTR)@ts_8@uio(DTR)$

5. Conclusions

The complexity of a real network protocol is a most serious obstacle to formal methods in protocol testing. Since an SDL specification of a protocol is normally designed without consideration of testability, it may often have a complicated part such as nested loops. This paper proposed a test generation method for a protocol specified in SDL with such complicated parts by EEFSM modeling, because popular IOEFSM modeling may reduce the preciseness of such a complex SDL specification. Actually, EEFSM modeling has been often used in passive testing where both an input and an output are handled identically as a monitored event. However, conditions for transition execution has seldom used as an event in usual EEFSM modeling. We introduced a condition event to describe complex loops more exactly in this paper and showed the efficacy of this method with empirical results of a real complex network protocol.

The following issues are considered as future work of this study. First, we plan to develop a test generation method with higher fault coverage for transition identification with an EEFSM. Systematical preamble decision using state domain analysis of an EEFSM is another interesting issue to generate test cases for transition identification. Finally, test generation for data flow test with EEFSM modeling also deserves to be studied for complete testing.

Acknowledgment

This paper was supported by Research Fund, Kumoh National Institute of Technology.

References

[1] ISO, "OSI Conformance Testing Methodology and Framework", IS-9646, 1991
[2] ITU, "Specification and Description Language", ITU-T Recommendation Z.100, 2000
[3] Hasan Ural, "Formal methods for test sequence generation", Computer Communications, Vol.15, No.5, 1992, pp. 311-325.
[4] R.Miller, S. Paul, "Generating conformance test sequences for combined control and data flow of communication protocols", Protocol Specification, Testing, and Verification ’92, Chapman & Hall, 1992, pp.1-15.
[5] S.T.Chanson, Z. Zinsong, "A unified approach to protocol test sequence generation", IEEE INFOCOM ’93, San Francisco, CA, USA, 1993, pp.106-114.
[6] T.-H. Kim, I.-S. Hwang, M.-S. Jang, J.-Y. Lee, "Test case generation of a communication protocol by an adaptive state exploration", Computer Communication, Vol.24, 2001, pp.1242-1255.
[7] Hierons, R. M., Kim, T.-H., and Ural, H., "On the Testability of SDL Specifications", Computer Networks, VOl.44, 2004, pp.681-700.
[8] Tar-Hyong Kim, "A Passive Testing Technique with Minimized On-line Processing for Fault Management of Network Protocols", International Journal of Computer Science and Network Security, Vol.7, No.3, 2007, pp.7-14.
[9] David Lee, Mihalis Yannakakis, "Testing Finite-State Machines: State Identification and Verification", IEEE Trans. on Computers, Vol.43, No.3, 1994, pp.306-320.
[10] D. Hogrefe, "OSI formal specification case study: The INRES protocol and service", TR IAM-91-012, University of Berne, Institute of Computer Science and Applied Mathematics, 1991.
[11] G. Luo, A. Das, G. V. Bochmann, "Software Testing Based on SDL Specifications with Save", IEEE Trans. on Software Engineering, Vol.20, No.1, 1994, pp.72 – 87.
[12] Ricardo Anido, Ana Cavalli, "Guaranting full fault coverage for UIO-based testing methods", IWPTS ’95, Evry, France, 1995.
[13] Phyllis G. Frankl, Elaine J. Weyuker, "An Analytical Comparison of the Fault-Detecting Ability of Data Flow Testing Techniques", ICSE ’93, Baltimore, Maryland, USA, 1993, pp.415-424.
[14] ITU, B-ISDN ATM Adaptation Layer – Service Specific Connection Oriented Protocol (SSCOP), ITU-T Recommendation Q.2110, 1994.

Tae-Hyong Kim received the B.S. and M.S. degrees, from Yonsei University in 1992 and 1995, respectively, and a Ph.D. degree in electrical and electronic engineering from the same university in 2001. He was a postdoctoral fellow at the School of Information Technology and Engineering (SITE) at the University of Ottawa from 2001 to 2002. He is currently an assistant professor in the School of Computer and Software Engineering (SCSE) at the Kumoh National Institute of Technology (KIT) in Korea. His current research interests include software and protocol specification, verification and testing techniques, communication protocols, and next generation mobile networks. He is currently a member of the SDL Forum Society.