Verification of Extended Finite State Machines From Control Perspective

Peng Wang Xiang-Yun Wang Kai-Yuan Cai

Abstract

This brief report discusses verification problem of extended Finite state machines in the background of computer science. The problem is related to the traditional supervisory control theory (SCT) of discrete event systems, but the difference exists in using variables to reduce the computational complexity and implement the control. In particular we will use the variables and predicates to realize the supervior in SCT theory. With the variables and predicates the new model functions like extended finite state machines (EFSM), which is more practical in the engineering field, especially in computer programs.

1. Introduction.

Many discrete event systems are safety-critical, and logical correctness is a crucial property of these systems. In the past few decades formal methods were widely studied and used in verification of logic systems, especially for computer softwares. This paper focuses on the verification of discrete event systems modeled by extended finite-state machines (EFSMs). This model typically simulates flow graphs of computer programs, and is conditionally specified to model communication of several processes [BADR01], where each process is modeled by an EFSM, and the entire system is composed of multiple EFSMs, or known as CEFSM, namely Communicating Extended Finite State Machines. This model is applied to formal method and widely used in verification of protocol. This provides one of the backgrounds of our research work.

Another background of this paper refers to the supervisory control theory of discrete event systems [RW87, W04], which is firstly proposed by W. M. Wonham and P. J. Ramadge in 1980s, and nowadays it has formed a theoretical framework, named by RW framework [RW87]. In RW framework the discrete event system is modeled by finite state machine, where the verification of large-scale systems remains a challenge due to the state-space explosion problem. Verification must take all possible variable values into account, which can result in a large state space. This paper will try to discuss the connection between EFSM and control theory of DES. Especially, the update function and the predicate together act as a supervisor to control where program flow goes. Here the update function acts as the automaton of the supervisor, and the predicate in EFSM plays the role of state feedback map. This framework provides a novel way to review EFSM from control perspective.

The third background of our work is the timed hybrid systems. Time is considered as a special variable

that significantly impacts the response of the system. In a broad sense such a system is a mixture of discrete and continuous components, such as a digital controller that controls the continuous environment. Timer is one of the variables that change based on certain physcial laws in the environment, and it is either discrete or continuous within the system, but not fully controllable within the system modeled. As a result the system is hybrid in nature, and specification and verification of timed hybrid systems become another study topic we try to explore in this article.

To intuitively introduced the model of EFSMs, an Illustrative Example of EFSM is presented as below and it is also represented by a flowchart and computer program written in C Language.

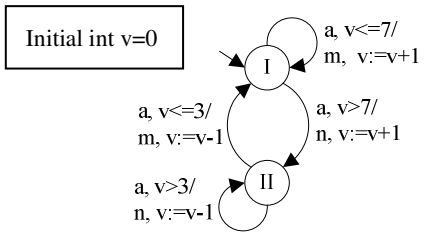

Figure 1 An Example of EFSM.

Here the states of EFSM are *I* or *II*, and the initial state is *I*. The input event is *a*, and it is the input signal from users or other systems. The output event is *m* and *n*, and they are messages output to the screen or to other systems. The variable *v* increases from 0 to 9 initially, and then decrease to 2, and then *v* oscillates from 2 and 9 again. Thus, the above EFSM can also be considered as a signal generator that output the signal *v* in oscillation.

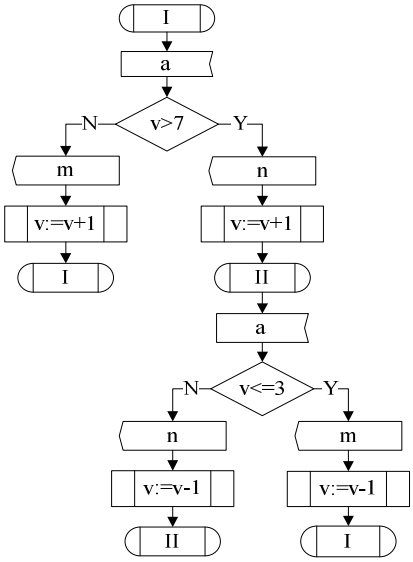

Figure 2. Flow Graph of EFSM

Then we transform the EFSM given in Figure 1 to the SDL flow graph in Figure 2, and the programming code can be further derived from of the SDL flow graph as follows. In this paper the programming code is given by C/C++. The code is structured based on the mathematical model presented in the next subsection.

```
typedef enum {I, II} state;
typedef enum {a} input;
typedef enum {m,n} output;

static state s; static input i; static output o; static int v;
void Initialization()
{ s=I; v=0; }

void Transition (input i)
{ switch(s)
      { case I:
            switch(i)
            { case a:   if(v>7) {s=II; o=n; v=v+1;} else {s=I; o=m; v=v+1;} break;
            } break;
       case II:
            switch(i)
            { case a: if(v<=3) {s=I; o=m; v=v-1;} else {s=II; o=n; v=v-1;} break;
            } break;
      }
}
```

2. About Definition of EFSM

An extended finite state machine is commonly structured by a seven-tuple

$$\text{EFSM} = (S, I, O, V, T, AP, L)$$

where S, I, O represent the state set, input event set and output event set, respectively. V is the variable set, composed of several variables, namely $V=\{v_1, v_2, \ldots v_n\}$. T is the state transition, and the uncontrolled state transition is defined by $T \subseteq S \times I \times S \times O$. AP is a set of atomic propositions. $L : S \rightarrow 2^{AP}$ is a labeling function. For example, a proposition is that s is a final state, and L will label this proposition with the state s to be either true or false. For a transition $t \in T$, it is denoted by

$$t = (s_t{}^{src}, s_t{}^{dest}, i_t, o_t, P_t, A_t)$$

where $s_t{}^{src} \in S$ and $s_t{}^{dest} \in S$ represent the source state and destination state of the transition, respectively. $i_t \in I$ is an input event and $o_t \in I$ is an output event. $P_t(V)$ is the predicate of the transition, which is defined with respect to the variables in V. $A_t(V)$ is the update function, which updates the values of the variables. By the definition as above, static structure of EFSM is illustrated. To make the system dynamic, there are two other issues to be mentioned:

(1) The initial condition of EFSM. The initial condition includes the initial state $s_0 \in S$ and the initial values of the variables, namely $V_0 = \{v_{10}, v_{20} \cdots v_{n0}\}$.

(2) The dynamic rule of EFSM. This rule regulates how the system evolves. For example, the transition is

given by $t = (s_t^{src}, s_t^{dest}, i_t, o_t, P_t, A_t)$, the rule is explained as follows: when event i_t is imported to state $s_t^{src} \in S$, if the current value of V makes $P_t(V)$ true, then the transition is enabled, the current state turns to $s_t^{dest} \in S$, and event o_t is output, and the value of V is updated by $A_t(V)$; otherwise the transition is disabled, the current state and the value of V remain unchanged. In other words, the predicates and variables function like the controller of the system, and they control how the state evolves [YG05, WC06]. The above structure can be linked with the framework established by Ramdage and Wonham in 1987 [RW87, W04].

Based on the aforementioned explanation, an example of EFSM is presented in Figure 1, where the state set is S = {I, II}, the input event set is I = {a} and the output event set is O = {m, n}. The variable set is given by V= {v}, and $dom(v) = \{0,1,2\cdots 9\}$. The predicates on variable v are given in Figure 2, namely v>7, v<=7, v>3 and v<=3. The update functions on variable v consist of v:=v+1 and v:=v-1. The initial state of EFSM is I, and the initial value of v is $v_0 = 0$. Then the EFSM evolves by the dynamic rule as mentioned above, and this EFSM is deterministic.

With respect to the EFSM model as introduced above, we have the following remarks.

(a) Sometimes the labeling fuction is defined by S $\rightarrow$ {0,1}, and it is an example that classifies the state into two disjoint subset, namely ordinary states and marker states [W04]. This implies that one proposition is given, and when the output is *1*, it means the state satisfies a given proposition, and if the output is *0,* the state does not. A more general case is that states are labeled with a set of proposition, and certain logic system can be applied such as temporal logic.

(b) Accordion to theory of automata and formal language, it is reasonable to use languages to identify the behavior of the EFSM. By the similar manners as introduced in Mealy Automata, we also prefigure the input and output languages as well as the combined language to depict the evolvement behavior of the EFSM model. Here we do not give the formal definition of languages for EFSMs, but corresponding concepts about the languages will be well explained in the following section. Here we also note that the update of variables is not described by the language, and thus the language does not provide the full information about system behaviour. In a sense the variables play the role of controller of the state transitions and it thus controls how the language is finally formulated.

(c) In the sense of mathematics the transition function is usually supposed to be a partial function defined on the input event set. That is, the transition function is defined on a subset of the event set, and some events are not involved in certain transitions. However, in computer programs the transition function considers all of the possible input events. If an input event is not explicitly addressed on the transition, it is assumed a self-loop is labeled with the event. As a result, the transition function is a total function in most practical problems in computer science.

(d) In certain areas such as real-time systems, the time is a special variable to be considered, and it should be especially modeled such as variable TIMER [BDL04], and thus it is possible to specify time-related logic to construct different system behaviour. Below is an example of Time EFSM, illustrating a simple logic of switching on or off a lamp. Here <Press> is the signal that users press the button of the lamp. In computer engineering TIMER is always a discrete variable, but in the mathematical sense we can simply assume TIMER is a continuous variable. Very importantly, it is noted that TIMER could be reset to be zero in the example,

but it also increases naturally with the time in the realistic world. This implies that the variables in the EFSM is only partly controllable because they are also governed by certain physical laws or follow dynamics of other systems. In this paper we call such physical laws or other systems as the environment of the EFSM, and we will discuss this issue in detail in the later sections.

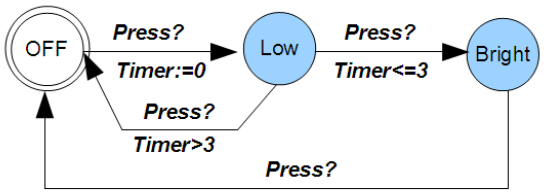

Figure 3. An Example of EFSM with Timer

In fact when we construct a logic system, it is actually important to identify the environment, or identify the boundary of the system and its environment. Even sometimes the environment is not directly shown in the model, it is actually taken into account implicitly. For example it is assumed in Figure 3 that TIMER increases with the natural time in the realistic world, and it is the physical law that TIMER must follow. This issue will be discussed in specific in the later sections.

3. About Deterministic EFSM and Empty Event

In a mathematical sense non-determinism in finite state machines (FSM) means uncertainty of state transitions, and it occurs when a state transition is labeled with empty events or several transitions at the same state are labeled with the same input events. However, the non-determinism in extended finite state machines is a more complicated issue because variables are captured in the model.

In mathematics empty event can be incorporated in the input event set of FSM, and the resulting FSM is non-deterministic. In a similar manner, certain transitions in EFSM occur without any event imported, and this is a common definition in computer science, especially in the field of model checking and formal method. Take Figure 4 for example, the transition happens if the variable meets the requirements, and no event is input. This situation is similar to labeling the transition with an empty event. However, such empty events in EFSM does not necessarily imply non-deterministic transition because the transition is still controlled by the variables.

It is noted that an EFSM can be non-deterministic if the current value of V enables more than one transition with the same input event. An important issue here is the variables, and they are the preconditions of the state transitions. However, from the perspective of computer programming an EFSM for a computing model is deterministic because the flow chart and the programming code cannot run ambiguously, but be compiled or interpreted to in a deterministic way (a computer program may get a random output by generating a random number and branching based on the random number, but not by ambiguous branches). Hence, this paper only discusses the deterministic EFSM where only one transition is enabled with the one input event in the lifetime of the system evolvement.

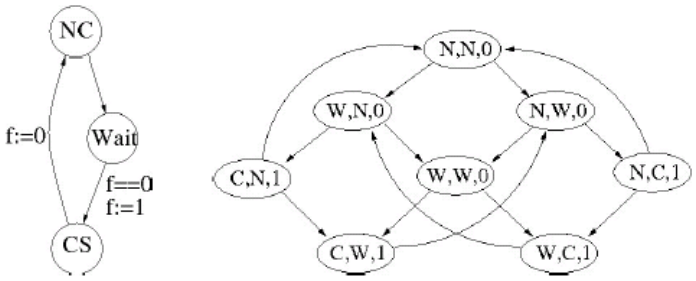

Figure 4. Example of Mutual Exclusion Protocol

In supervisory control theory if the output event set includes empty event, the system could be a partially observable system because control of a system is implemented by using partial information provided by output events. In a similar way the empty event can also be included in the output event set of EFSM, and empty event in the output event set means that nothing is output [WC12]. Or simply the output label in the transition arc is omitted. This is especially useful to model interaction of several EFSM models where output of one EFSM is going to another EFSM as an input event.

3. Supervisory Control Theory and EFSMs

In this section the EFSM model is compared with the typical model in the supervisory control theory (SCT) introduced by Ramdage and Wonham in1987 [RW87], and we will try to establish a link between EFSM and SCT, especially view EFSM from the perspective of SCT.

In the original framework in [RW87], a discrete event system is characterized by an finite state machine (FSM), or as called the plant in the field of control theory. The FSM is event-driven, and its state transition occurs when an input event (an input signal) is imported. The state transition is controlled by another FSM named by supervisor. The supervisor synchronizes with the plant, enabling or disabling transitions in the plant. A transition occurs if it is enabled by the supervisor, otherwise the state transition cannot occur. The general framework is illustrated as below.

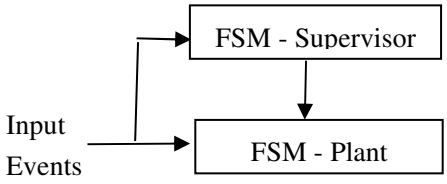

Figure 5. Supervisory Control Theory of Discrete Event Systems

In EFSMs the state transition is controlled by variables and predicates. The variables, predicates as well as update function together correspond to the supervisor in RW framework, and it controls the transitions of Plant FSM. The detailed work is presented in [YG05, WC06].

For example, the above state transition is controlled by variable *v*. When event <*a*> occurs, whether the state is updated or not depends on the current value. The predicate defined on the transition returns a boolean result to guard the transition. Thus, the state transition is controlled by the variable and predicate. In addition, the variable *v* is not static, but dynamically changed by the update function.

$$T : S \times I \times P(V) \rightarrow S \times O \times A(V)$$

Controller: Enabling or Disabling a Transition

Figure 6. The embedded supervisor in EFSM

In sum, EFSM can be understood from the perspective of supervisory control theory, and this contributes to the field of software cybernetics, which explores the interplay of software and the control [CCDM04, YCA16]. A major advance in EFSM is that control mechanism is jointly realized with the semantics of the variable-related symbols, e.g., v>7 or v<3, and when variable is updated the control is naturally implemented by the semantics.

As above we generalize how an EFSM model is to be transformed into the model in SCT, namely in the form of plant-FSM and supervisor. The next question is how to transform an exisitng supervisor into the form of variables and predicates, and incorporate them into the plant-FSM? This approach is illustrated in the above figure. In the RW Framework, the supervisor is originally modeled by $\Phi = (S, \psi)$, where S is a deterministic automaton and $\psi : \{\text{States of } S\} \rightarrow \{\gamma \in 2^I \mid I_{uc} \subseteq \gamma \subseteq I\}$ is a state feedback map. Here the automaton S evolves with the pace of the controlled machine, and its state set corresponds to the domain of the expected variable. While the state changes, this changing corresponds to updating of the variable. Then we use the state feedback map to regulate where the system goes, and this map actually plays the role of predicates.

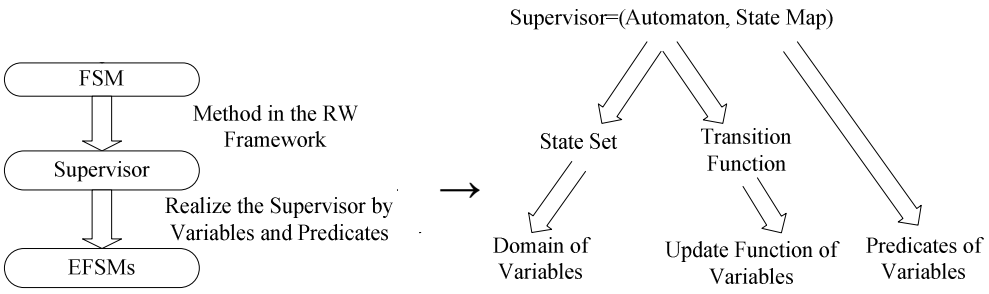

Figure 7 Transform Supervisor into Variables and Predicates

In brief an EFSM can be transformed to the control model in SCT and a control model in SCT can also be transformed to EFSM inversely. In the field of computer science and engineering, EFSM is a well-accepted model, and it is widely used in model checking. Here we highlight the control perspective of program flows,

where the control mechanism is studied in program flow graphs. This work contributes to the field of software cybernetics, which explores interplay of the software and the control, and we hope to merge a gap between certain concepts in computer science and supervisory control theory.

4. Controllablity of Variables in EFSM

A well-accepted assumption in existing framework of supervisory control theory (SCT) is that we cannot control everything of a system. Only certain factors are controllable. As a result, events imported to the automaton are partitioned into two disjoint subsets, i.e., a controllable subset and uncontrollable subset. The system is controlled by disabling certain controllable events that cause undesired state transitions. In the field of reactive systems or real-time systems, events from environments are generally deemed as uncontrollable factors to the system. Based on the SCT theory in the RW Framework [RW87, W04], the input event set I can be partitioned into a controllable subset and an uncontrollable subset, namely $I = I_c \cup I_{uc}$ and $I_c \cap I_{uc} = \varnothing$. In a similar manner we can also partition the transition set by a controllable subset and an uncontrollable subset, namely $T = T_c \cup T_{uc}$.

In EFSM model the concept of controllability is naturally extended to the variables in EFSM also, namely, some variables are controllable while others are not. In a similar way it is reasonable to assume that variables from environments are uncontrollable factors to the system. To define whether a variable is controllable or not, we study multiple EFSMs and they interact and communicate in certain protocols. When discussing one particular EFSM, other EFSMs become the environment. Thus, controllability of variables is with respect to a given EFSM. In the system composed of multiple EFSMs, a variable is possible to be controllable to one EFSM while uncontrollable to another.

In general controllable variables means that an EFSM can read and write the variables while other EFSMs cannot. The uncontrollable variables means that the EFSM cannot read or write the variable. Here an important issue is about readability and writability of a variable, and we provide a table as below to compare the controllability of variables with the existing concepts in computer programming.

Controllability of Variables	**Concepts in Computer Programming**
Controllable Variables	Have permission to read or write the variable while other EFSM cannot change the value. Such variable are usually called private variables.
Partially Controllable Variables	Have permission to read or write the variable while other EFSM can also change the value. Such variable are usually called Shared Variables / Global Variables.
Uncontrollable Variables	The EFSM do not have permission to read or write the variables. These variables are Private Variables in Other EFSM's or Read-Only External Variables.

For example, the fault or malfunction in a system are commonly considered as uncontrollable variables because any faults or malfunction happen unexpectedly and are generated from the environment. From the perspective the reactive systems, such flag variables for malfunction are only readable.

An common example is reading certain sensor output into the micro-computer, and the variables read in thus become a part of the logic system. For instance the micro-computer reads in the temperature of a room and controls the temperature within a given interval. The data of the temperature follows the physical laws and the computer is only able to control the data partly, usually it can only read in the data, not have the right to write the data in normal cases.

In addition there are discrete variables and continuous variables in mathematics. The discrete variables are commonly in type of integers or boolean. The continuous variables are real numbers in the type of either float or double. In computer science the float or double numbers are discretized and stored in a approximate manner. In the sense of mathematics, the real numbers are simply continuous variables, and when continuous variables are included in the system, EFSM can be considered as a hybrid system in a broad sense.

5. EFSM and FSM

Take the following graph as an example, and it models a typical behavior of a stack in computer science and engineering.

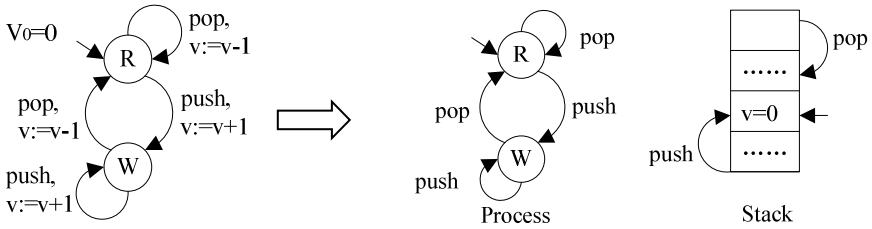

Figure 8. EFSM and FSM

Actually EFSM can be transformed into the normal FSM. Here we incorporate the variable because it can absorb a large number of states into the form of variables, and thus the state number will be greatly reduced. Consider the aforementioned example; it can be transformed into the product of a process and a stack as shown above, and if we formulate the same model in FSM, it has much more states than the model with variables. So compare the model with and without using variable, it is clear that the major difference exists in the number of states.

Besides, it is also reasonable to use languages to identify the behavior of EFSM. By the similar manner as introduced in Mealy Automata, we use the language to represent the behavior of the model. Here we give the definition of the input language as follows.

$$L(G) = \{ s \in I^* \mid T(y_0, s) \text{ is defined} \}$$

The state machine in Figure 8 equals the language of $(pop^* \cup push\, push^*\, pop)^* (\varepsilon \cup push\, push^*)$. Once again, one point to be emphasized is that the language of EFSM cannot give the entire information of the system dynamics because it cannot show the update of the variables, and this is a major difference from the

common FSM model.

How can the supervisor be incorporated into the controlled system by some variables and predicates? In the RW Framework, the supervisor is originally modeled by $\Phi = (S, \psi)$, where S is a deterministic automaton and $\psi : \{\text{States of } S\} \to \{\gamma \in 2^I \mid I_{uc} \subseteq \gamma \subseteq I\}$ is a state feedback map. Here the automaton S evolves with the pace of the controlled machine, and its state set corresponds to the domain of the expected variable. While the state changes, this changing corresponds to updating of the variable. Then we use the state feedback map to regulate where the system goes, and this map actually plays the role of predicates. The basic idea is also shown in Figure 6 and Figure 7.

Consider the aforementioned example; we propose the specification as the following language

$$L(SPEC) = (push^2 pop^2)^* (\varepsilon \cup push \cup push^2 \cup push^2 pop)$$

This specification requires the machine to circulate by pushing twice and poping twice. Based on the theory in the RW Framework, we have *SPEC* illustrated as below.

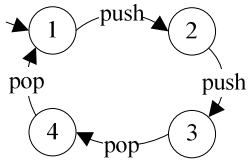

Figure 9. Specification

Define variable u to implement the supervisory control. Its domain is dom(u)={1,2,3,4} and its initial value is u_0=1. Then we embed variable u into the controlled state machine and it will fulfill the given specification as below.

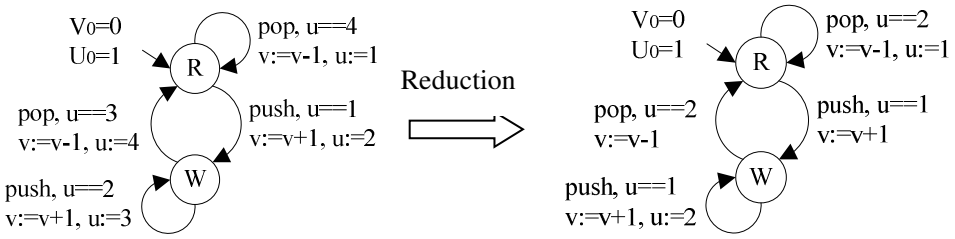

Figure 10. About Variables

This approach is clear in methodology, and it realizes the supervisor by using variables. However, this approach is not the optimized one because domain of the variable often exceeds the domain actually needed. Take the system in Figure 10 for example. Although variable u achieves the specification, its domain {1,2,3,4} exceeds the one needed. Actually we can only let dom(u)={1,2}, and the two values are enough to achieve the specification, as shown in Figure 10. Why? The reason lies in that the state of the controlled system can also be considered as a certain variable, and it can help u to identify where the controlled system stays. In the given example the minimal domain of u is 4/2=2, where 4 is the state number of the supervisor synthesized in the RW Framework, and 2 is the state number of the controlled system. Here we note that the minimal domain of the variable for supervisory control cannot be less than

#(state set of the supervisor)/#(state set of the controlled system)

In a practical situation we expect an effective approach to reduce the domain of the variable for supervisory control [SW04]. And this offers a research topic to be discussed in the future.

6. Develop the connector based on the shared variables

In the previous sections, we discuss EFSM and the relevant control concept. Based on the previous work this section will give a typical example to show a way of connecting the multi-FSMs with shared variables. This method can be a possible way to avoid or mitigate the state explosion problem in the RW Framework when a large system is composed by many subsystems.

The example has the background of communicating systems or manufacturing systems. It is composed by three parts, a sender machine, a receiver machine and a buffer. By the traditional way of modeling such a system by SUT, we construct the model as follows.

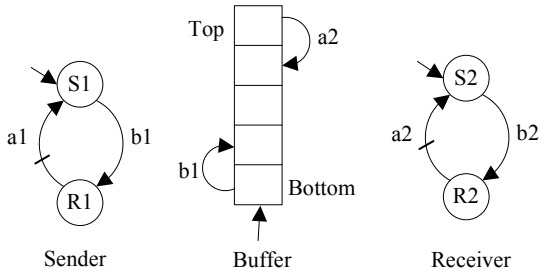

Figure 11

Then we can make product of the three sub-systems and show all possible behaviors of the whole system by a single FSM. The state number will be $2\times 2\times 5 = 20$. Then a specification refers to the buffer, that is, to avoid overflow or underflow, the buffer should be loaded no more than 3 pieces and be unloaded no less than 0 piece. Then applying the theory in the RW Framework we can formularize the specification and synthesize the required supervisor, and the work out this problem finally. However, the problem of the approach is that the state number increases too rapidly, and the cost for computing increases correspondingly.

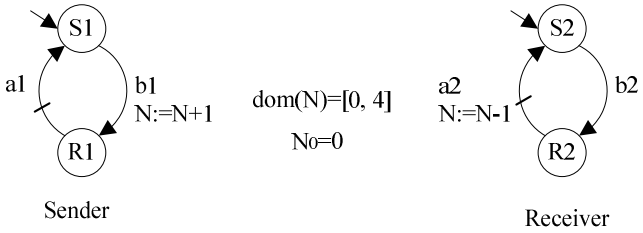

Figure 12

To improve the traditional solution to this problem, we apply the EFSM model to this problem. That is, we represent the buffer by a variable shared by both Sender and Receiver, and then we model the system as Figure 12 shows. Then the specification is proposed by $N \geq 0$ and $N \leq 3$. The next step is to synthesize the predicates for the controllable transitions. Apply the algorithm of the previous section, and we work out the predicates obtain the supervised system as follows.

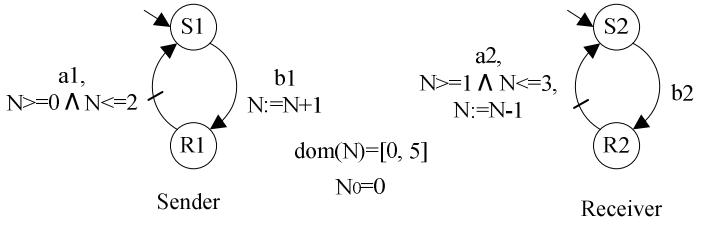

Figure 13

Through the simple example above, we note that the new model does not make product of the subsystems because the shared variable plays the role of connecter of the subsystems. Therefore when many subsystems are considered, the state number increases by "sum", but not by "multiply". Then the state explosion problem can be effectively avoided and the computing efficiency can be improved accordingly.

Besides, the result of the new approach is given in the form of EFSM. It is more related to the field of computer science and engineering, so it can be easily simulated by some software design tools such as SDL or be applied to the computer program.

Reference

[BDL04] G. Behrmann, A. David, and K. G. Larsen, A Tutorial on Uppaal, Lecture Notes in Computer Science, November 2004.

[BADR01] C. Bourhfir, E. Aboulhamid, R. Dssouli, N. Ricob, "A Test Case Generation Approach for Conformance Testing of SDL Systems", Computer Communications, vol. 24, pp. 319-333, 2001.

[CCDM04] K. Y. Cai, J. W. Cangussu, R. A. DeCarlo, A. P. Mathur, "An Overview of Software Cybernetics", Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice, IEEE Computer Society Press, 2004.

[F06] M. Fabian, Discrete Event Systems, Lecture Notes (ESS 200), Control and Automation Laboratory,, Chalmers University of Technology, 2006.

[RW87] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", SIAM Journal of Control and Optimization, vol. 25, no. 1, pp. 206-230, 1987.

[SW04] R. Su, W. M. Wonham, "Supervisor Reduction for Discrete-Event Systems," Discrete

Event Dynamic Systems, Jan 2004.

[WC06] P. Wang, K. Y. Cai, “Representing Extended Finite State Machines for SDL by A Novel Control Model of Discrete Event Systems”, QSIC 2006.

[W04] W. M. Wonham, “Supervisory Control of Discrete-Event Systems”, ECE 1636F/1637S 2004-05, Revised 2004.07.01, Dept. of Electrical & Computer Engineering, University of Toronto.

[YCA16] H. Yang, F. Chen, S. Aliyu, "Modern Software Cybernetics: Trends with New Cybernetics," Journal of Systems and Software. 2016.

[YG05] Y. Yang, R. Gohari, “Embedded Supervisory Control of Discrete-Event Systems”, IEEE International Conference on Automation Science and Engineering , pp. 410-415, 2005.