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. 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 be the computing model in Specification and Description Language (SDL) [1, 3, 4, 8]. As well known in computer science and engineering, SDL is a language to specify the communication of several processes, where each process is modeled by an EFSM, and a whole communicating system is modeled and computed by CEFSM, namely Communicating Extended Finite State Machines [2, 4]. This provides one of the backgrounds of our research work.

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.

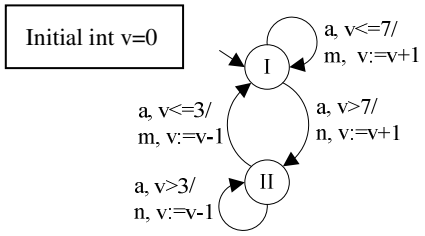

Figure 1 An Example of EFSM.

Here the states of EFSM are either *I* or *II*, and the initial state is *I*. (Intuitive Explanation Needed)

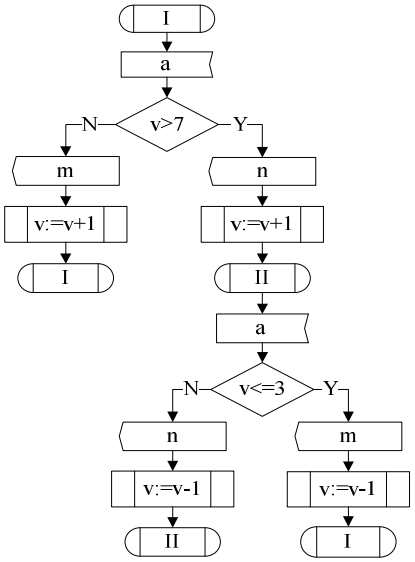

Figure 2. Flow graph of EFSM

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;
      }
}
```

1. 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 [7, 8]. The above structure can be linked with the framework established by Ramdage and Wonham, 1987 [5, 6].

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 [5, 6]. 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) 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 (UPPAAL, 2018), and thus it is possible to specify time-related logic to construct different system behaviour.

(c) Accordion to theory of automata and formal language, it is reasonable to use languages to identify the behavior of the EFSM. And by the similar manners as introduced in Mealy Automata, we can 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 can be well explained in the following section.

(d) 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.

2. About Deterministic of EFSM and Empty Events

In a mathematical sense 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. 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.

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 the following chart as an 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 imply non-deterministic transition because the transition is still controlled by the variables.

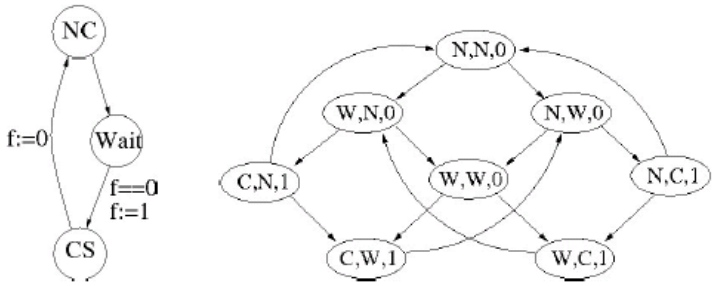

Figure 3. A Protocol Example

In supervisory control theory if the output event set includes empty event, the system is called 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. 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. Interacting EFSM and Controllablity of Variables

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, events from environments are generally considered as uncontrollable factors to the system. Based on the theory in the RW Framework [5, 6], 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, namely, some variables are controllable while others are uncontrollable. 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 assume that several EFSM's interact and consist of the entire system. When discussing one particular EFSM, other EFSM's become the environment. Thus, controllability of a variable is with respect to a certain EFSM. In the system composed of several EFSM's, a variable may be controllable to one EFSM while uncontrollable to another.

In general controllable variables means that the EFSM can read and write the variables while other EFSM considered as environments cannot. The uncontrollable variables means that the EFSM cannot read or write the variable. Here another important issue is about readability and writability of a variable.

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 not controlled by any program or machines.

However, such flag variables are readable.

In the sense of mathematics there are discrete variables and continuous variables. The discrete variables are commonly in type of integers or boolean. The continuous variables are real numbers which are 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 the 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.

4. EFSM and FSM

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

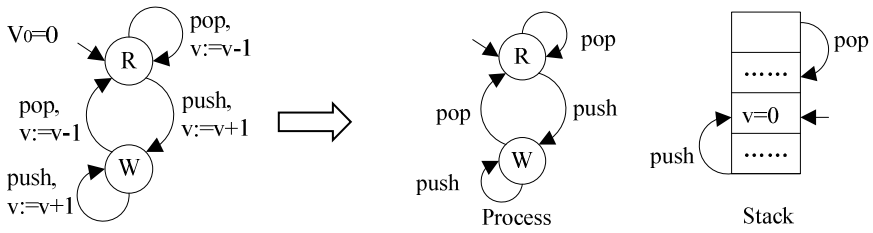

Figure 4. EFSM and FSM

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 4 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 3.

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.

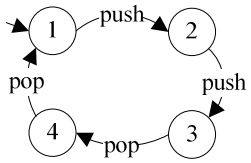

Figure 5. Specification

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.

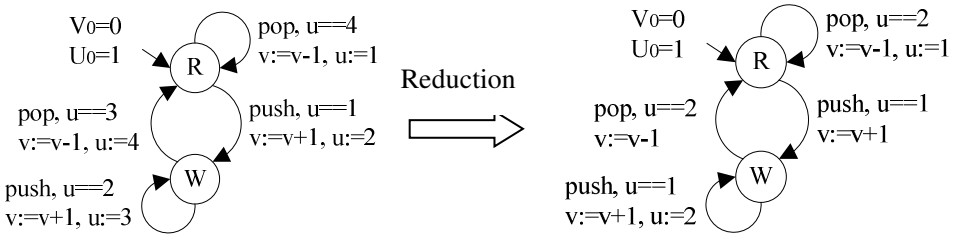

Figure 6. About Variables

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 6 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 6. 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

$$\#(\text{state set of the supervisor}) / \#(\text{state set of the controlled system})$$

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

Reference

[1] C. Bourhfir, E. Aboulhamid, F. Khendek, R. Dssouli, “Test Cases Selection from SDL Specifications”, Computer Networks, vol. 35, pp.693-708, 2001.

[2] R.E. Miller, Yong Xue, “Bridging the Gap between Formal Specification and Analysis of Communication Protocols”, Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications, pp. 225-231, March 1996.

[3] Introduction to SDL 88. http://www.sdl-forum.org/sdl88tutorial/index.html, 2012.

[4] 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.

[5] 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.

[6] 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.

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

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