Control of Finite State Machines with Variables

Peng Wang Xiang-Yun Wang Kai-Yuan Cai Walter Wonham

Abstract

This article discusses control and verification problem of finite state machines with variables 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 supervisor 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 design of computer programs.

1. About Formal Definition of FSMwV

In the past few decades formal methods were widely studied and used in verification of logic systems, especially for design of computer programs in safety-critical areas. This paper focuses on the control and verification of discrete event systems. A discrete event system is modeled by finite state machines (FSM) in this paper. Variables are attached to FSM, and the resulting model is called finite state machines with variables (FSMwV). The model is discrete if the variables are all discrete. In case of continuous variables, the model is hybrid in nature. In this manuscript we focus on FSM with discrete variables.

1.1 An Illustrative Example

Before we introduce the mathematical notations of FSMwV, an illustrative example of FSMwV is first presented as below, and it is also represented by a flowchart and computer program written in C Language.

Here the states of FSMwV are *I* or *II*, and the initial state is *I*. The input event is *a*, and you can consider it as a user input signal (e.g., keyboard input) or input message from other FSMwV, or a sequential procedure that users define well. 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 decrease to 2, and then *v* oscillates from 2 and 9.

In brief the above FSMwV is a signal generator that produces signal *v*.

More illustratively, the FSMwV in Figure 1 is represented by a flow graph, and the programming code is also given as below by C/C++. The code is structured exactly based on the mathematical model as presented in the next subsection.

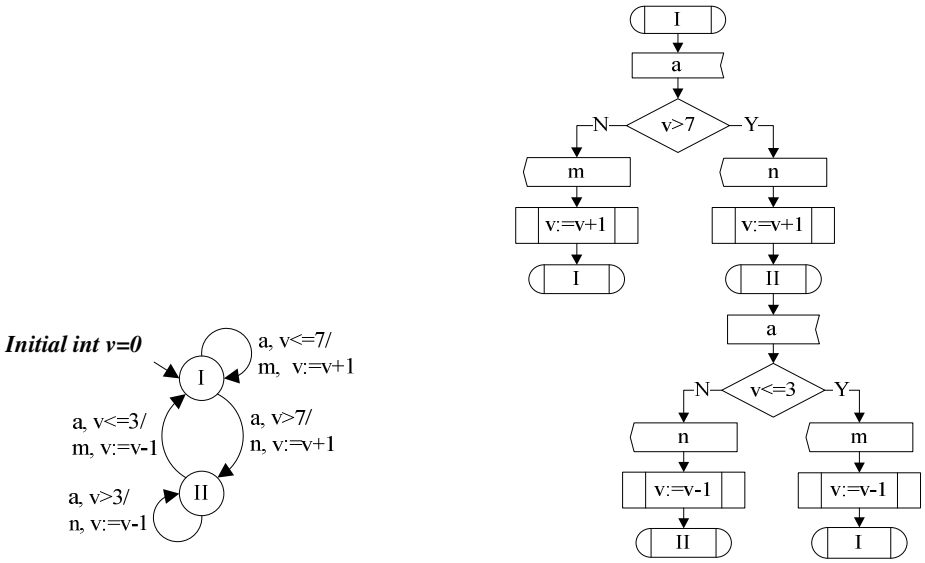


Figure 1. An Illustrative Example of FSMwV

```
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;
      }
}
```

1.2 Mathematical Definition of FSMwV

A finite state machine with variables is structured by a seven-tuple

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

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, ... 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 labeling function is to check whether $s \in S$ is a final state or not, and L will label the state $s \in S$ 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 variable V. $A_t(V)$ is the update function, which updates the values of the variables. The above definition specifies a static structure of FSMwV. The system dynamic further needs the initial condition including initial state $s_0 \in S$ and initial values of the variables $V_0 = \{v_{10}, v_{20} \cdots v_{n0}\}$.

With the above definition a FSMwV model forms a system dynamics. In specific, for a transition given by $t = (s_t{}^{src}, s_t{}^{dest}, i_t, o_t, P_t, A_t)$, the system is updated 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, 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 variable V remain unchanged. In other words, the predicates and update function like a controller that regulates the system behavior [YG05, WC06]. The above formal definition of FSMwV is compared with supervisory control framework established by Ramdage and Wonham in 1987 [RW87, W04], and we will discuss it in detail in the following sections.

Based on the above mathematical definition, the example of FSMwV in Figure 1 is formalized as below. 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 1, 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 FSMwV is I, and the initial value of v is v_0=0. The FSMwV evolves in a deterministic manner.

Behavior of discrete event systems is commonly described by event sequences, and FSMwV has both input event sequences and output event sequences. In FSMwV the complete system dynamics is represented by the several sequences combined together, including input event sequence, output event sequence, state sequence and variable sequence. The input event sequence generates other three types of sequences and it is also confined by the variables sequence in a feedback loop. The idea is illustrated as below.

Table 1.

	Math Notations
Input Event Sequence	i_1 i_2 ... i_n
Output Event Sequence	o_1 o_2 ... o_n
State Sequence	s_1 s_2 ... s_n
Variable Sequence	V_1 V_2 ... V_n

Predicate (Feedback) **Update Function**

By the above sequence the system dynamics of FSMwV is completely described. Although the four types of sequences provide complete information of system dynamics, some of them are redundant. By the theory of finite automaton and formal language, the input event sequence and state sequence are duplicate

information while the output sequence only provides partial information of the system dynamics, or as called partial observability in SCT. Very importantly, a major difference between the traditional FSM and FSMwV is that the input event sequence does not describe update of variables, and it does not provide the complete information of system behavior because variables are part of the system state. We will discuss this issue in the later section when FSMwV is compared with traditional FSM. Combine four types of sequence together, the transition sequence is also denoted as below in this manuscript.

$$s_1, v_1 \xrightarrow{i_1 / o_1} s_2, v_2 \xrightarrow{i_2 / o_2} s_3, v_3 \cdots$$

FSMwV is mathematically defined as above, and it is emphasized in different aspects when the model is applied in different areas. For example, when the model is applied in a compilation system that translates one formal language into another, the input event sequence and output event sequence are the major study topics, and the analysis is language-based. If we consider a signal generator as illustrated in Figure 1, which produce a cyclic variable v, the variable becomes a major topic and the analysis is variable-based. In sum, when the mathematical model of FSMwV is applied in various fields, the model is highlighted in different aspects.

2.1 Supervisory Control Theory and FSMwV

In this section the FSMwV model is compared with the classical model in the supervisory control theory (SCT) [RW87], and we will try to establish a link between FSMwV and SCT, especially review FSMwV from the control 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.

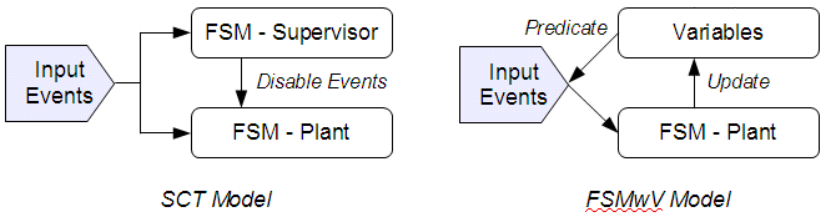


Figure 2. Supervisory Control Theory and Finite State Machines with Variables

In FSMwV the state transition is controlled by variables and predicates. The variables, predicates and update function together correspond to the supervisor in RW framework, and they jointly control transition of state in plant FSM. The detailed work is presented in [YG05, WC06].

For example, the state transition in Figure 1 is controlled by variable *v*. If event <*a*> occurs, whether the state changes depends on the current value of variable *v*. The predicate defined on the transition returns a boolean result and the transition is enabled if the result is true. Thus, the state transition is controlled by the variable and predicate, and the variable is also dynamically updated if the state transition happens.

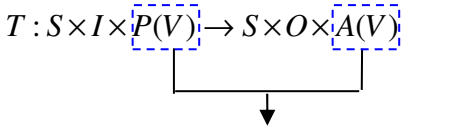


Figure 3. The embedded supervisor in FSMwV

In sum FSMwV can be understood from the perspective of supervisory control theory, and contributes to the field of software cybernetics, which explores the interplay of software and the control [CCDM04, YCA16]. An advance in FSMwV is that control mechanism is realized by 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 a FSMwV model is transformed into the model in SCT, namely in the form of plant-FSM and supervisor. Inversely it is also feasible to transform an existing supervisor into the form of variables and predicates, and incorporate them into the plant-FSM. The approach is illustrated in Figure 4. In RW Framework, the supervisor is initially defined 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 is updated with the pace of the plant FSM, 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 the system , and this map actually plays the role of predicates.

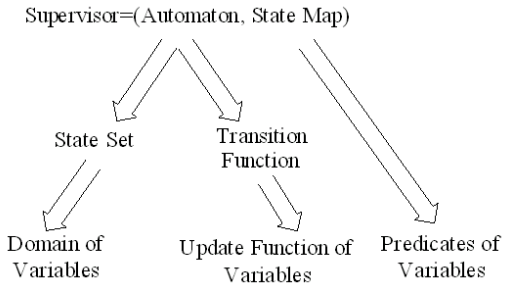


Figure 4 Transform Supervisor into Variables and Predicates

In brief a FSMwV can be transformed to the model in SCT and the model in SCT can also be transformed

to FSMwV inversely. In the field of computer science and engineering, FSMwV, or as commonly called extended finite state machines (EFSM), is a well-accepted model widely used in model checking. In this paper we highlight the control perspective of FSMwV, exploring control mechanism in program flow graphs. We hope to bridge a gap between certain concepts in computer science and supervisory control theory, and our study contributes to software cybernetics [YCA16, CCDM04].

Compare the model in SCT and FSMwV, there is another issue about labeling function. In SCT the labeling fuction is defined by $S \to \{0,1\}$, which classifies the states into two disjoint subsets, namely ordinary states and marker states [W04]. The marker state represents the final state where a system terminates (e.g., complete a computing task). The labeling function identifies whether the marker state is reached or not. If the marker state is reached, the labeling function outputs *1*, otherwise the output is *0*.

3. Variable-Based Supervisory Control

In FSMwV model the variable is used to abstract a large number of states in form of variables, and thus the state number is greatly reduced. In this section we further present the idea of variable-based supervisory control. The idea is presented by an example, describing a typical behavior of a stack in computer science.

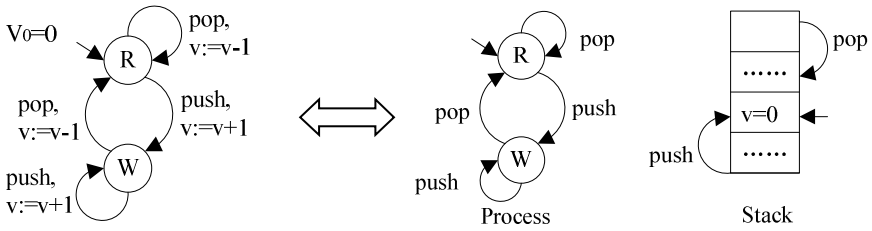


Figure 5. Plant FSM to be Controlled: The state is either R(Read) or W(Write), which identifies the action of two events (i.e., pop or push) before reaching the state. It is also possible to integrate the two states into a single state to simplify the model.

The FSMwV example as above can be transformed into the product of a Pop/Push process and a stack memory. The left part of Figure 5 illustrates a FSMwV model, and it is decomposed to FSM in the right part. Consider the paradigm illustrated in Figure 2, the behavior of control plant FSM is specified by its input event sequence, namely L(FSM)={s | s is an input event sequence}. By the theory of finite state automata and formal language, the input event sequence of the above model is given by a regular expression. The plant FSM in Figure 5 equals the language of $(pop^* \cup push\ push^*\ pop)^*(\varepsilon \cup push\ push^*)$. In specific the plant FSM refers to the system without implementing any control strategy, and it is the FSMwV without predicates included.

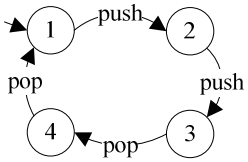
How can the supervisor be incorporated into the controlled system by some variables and predicates? In the RW Framework, the supervisor is modeled by $\Phi = (S, \psi)$, where S is a deterministic automaton and $\psi : \{\text{States of } S\} \to \{\gamma \in 2^I \mid I_{uc} \subseteq \gamma \subseteq I\}$ is a state feedback map. The state of automaton S is updated with the

pace of the plant FSM, and the state is mapped to a subset of input events to regulate the plant FSM, and this map actually plays the role of predicates.

Considering the stack example, we propose the specification by the following regular language.

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

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



State Feedback Map

State:	1	2	3	4
Disable Events:	pop	pop	push	push

Figure 6. Specification and Supervisor: Pushing twice and poping twice

Next, we will use variables to implement the supervisory control. Define variable u to index the state of the supervisor, namely dom(u)={1,2,3,4} and its initial value is u_0=1. The domain of variable u corresponds to the state set of SPEC FSM. The variable u is updated to circulate from *1* to *4* as shown in Figure 6. Then we embed variable u into the plant FSM to fulfill the given specification. The update function of variable u realizes the state transition of the supervisor and the predicate corresponds to the state feedback map. Here the coding of the supervisor state is not unique. We can use {1,2,3,4} to code it, and we can also use other strategy such as {10, 20, 30, 40}. The resulting FSMwV is illustrated in Figure 7, and this approach is named by variable-based supervisory control.

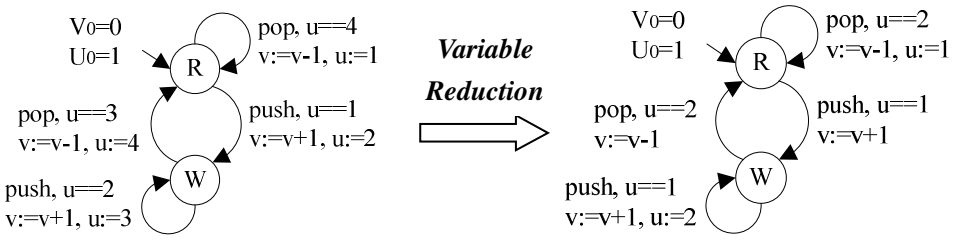


Figure 7. Variable-Based Supervisory Control

(a) Variable Reduction

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 7 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 7. The reason is that the state of the plant FSM can also be considered as a

certain variable, and it helps u to identify where the 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 plant FSM. 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 plant FSM)

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.

(b) Incremental Design of FSMwV

Suppose we have several requirements or specifications for plant FSM. Can we implement them one by one? An important issue is to check whether these specifications are independent or conflicting. For several independent specifications it is feasible to synthesize supervisory control by independent variables and realize a paradigm as shown below.

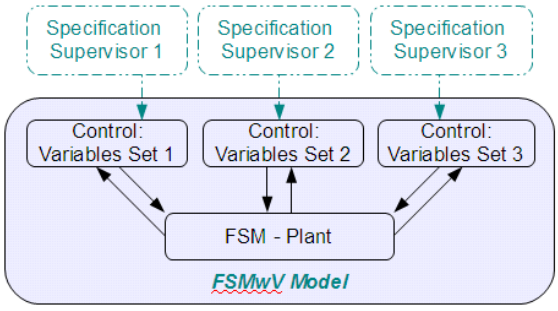


Figure 8. Incremental Design of FSMwV

However, sometimes the things are not straightforward, and we cannot judge it simply by intuition. In particular, if a specification is related with others, we have to examine if there is any conflict between specifications. Following the above stack example, we further present some cases as below.

Table 2. Incremental Design of Stack FSMwV

Specifications	Case 1	Case 2	Case 3
SPEC1	Push twice and pop twice	Push twice and pop twice	The stack memory is limited to 2 units. (v=0,1,2)
SPEC2	Uninterrupted push no more than 3 times	Push three times and pop three times	Uninterrupted push no more than 3 times
SPEC1∩SPEC2	Push twice and pop twice	Push twice and stop	SPEC1 and SPEC2 are realized by incremental control

As listed in Table2 there are three study cases. In Study Case 1 it is clear that SPEC2 includes SPEC1. In brief, pushing twice and popping twice defines a behaviour range smaller than uninterrupted pushing no more than 3 times. Because SPEC1 has no conflict with SPEC2, SPEC1∩SPEC2=SPEC1 in Study Case1. In Study Case 2 the two specifications are conflicting. After pushing twice, SPEC2 requires keeping pushing one more time while SPEC1 demand swift to popping. So there is no feasible control to realize both of them, and control will permit pushing twice and then the system stops there.

Study Case 3 is more complicated, but it is more close to practical problems. For this case it is not easy to directly judge if the two specifications are related or not. Recall Figure 5 and it shows how to decompose the control plant FSM in a push/pop process and a stack memory. Correspondingly, SPEC1 confines the upper limit and lower limit of stack memory while SPEC2 specifies the push/pop behavior of the process. SPEC1 and SPEC2 seem related issues, but it is feasible to synthesize supervisory control of FSMwV in a decoupling manner. The final system is illustrated as below by incremental design of FSMwV, where the variable *v* and *k* are independent variables, and SPEC1 and SPEC2 are realized by *v* and *k*, respectively.

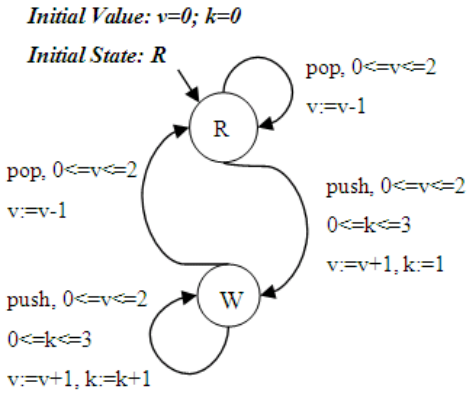


Figure 9 Variable-Based Supervisory Control: Case Study 3 for Incremental Design

Reference

[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.