Modeling of Discrete Event Systems Using Finite State Machines with Parameters

Yi-Liang Chen†
Rockwell Science Center
Thousand Oaks, CA 91360
ylchen@rsc.rockwell.com

Feng Lin
Dept. of ECE, Wayne State University
Detroit, MI 48202
felin@ece.eng.wayne.edu

Abstract

Control theories for discrete event systems modeled as finite state machines have been well developed over the years in addressing various fundamental control issues. However, modeling in finite state machines has long suffered the potential problem of state explosion that renders it unsuitable for many practical applications. In this paper, we propose an approach that appends finite sets of parameters to finite state machines in modeling discrete event systems. We show with an example that many discrete event applications can then be represented efficiently in this approach and the state explosion problem can then be mitigated.

Keywords: Discrete event systems, Finite state machines, Supervisory Control, Modeling, Model Synthesis

1 Introduction

Modeling and control of discrete event systems (DES) have been studied by control engineers for almost 20 years. During this period, many modeling approaches have been proposed, including most notably automata or finite state machines [RW87, CL99], Petri nets [Pet81, HKG97] and their variations such as vector DES [LW93, LW94] and event graphs [CGQ98], queuing systems [CL99] and generalized semi-Markov processes [Gly89].

Among these models, finite state machines are most straightforward for control. In fact, the supervisory control theory [RW87, RW89, LW88, CL99] developed based on finite state machines has been well developed as it addresses the fundamental issues in control of DES. As a result, we now have a good understanding of problems such as controllability, observability, coobservability, normality, decentralization, nondeterminism, etc.

The authors believe that an important reason we have got this far in a relatively short period of time is that we adopted a simple model of finite state machines. Because of this, we can focus our attention and see the essence of the control problem. Now with many of these fundamental problems resolved, we would like to pay more attention to models that are more useful in applications.

Modeling in finite state machines has long suffered the potential problem of state explosion that renders it unsuitable for many practical applications. For example, to model a buffer of $n$ capacity using a finite state machine would require at least $n$ states. On the other hand, by using an integer parameter to describe the content of the buffer, the number of states required can be drastically reduced. Furthermore, in the event that the capacity of the buffer changes, we can simply modify the range of the parameter without remodeling the system.

In this paper, we propose to employ both finite state machines and sets of parameters in modeling discrete event systems. We focus this paper in presenting this modeling mechanism and the related characterizations. Without doubts, many issues need to be resolved before a valid approach can be developed for analysis and synthesis of DES control using our proposed Finite State Machines with Parameters (FSMwP) mechanism. For some initial results on the DES control applying to this modeling mechanism, please refer to [CL00].

Note that the definition of our FSMwP is similar to the Extended Finite State Machines (EFSM) described in [CK96]. Nevertheless, the EFSM mechanism was developed for design verification of circuits and not for modeling of general discrete event systems. Hence, parameters are mainly for describing the contents of the circuit inputs/outputs rather than for describing system resources and possible time/resource constraints.
that FSMwP is designed for. Furthermore, neither concepts of system composition nor control synthesis were developed under the EFSM scheme.

The rest of the paper is organized as follows. We define the FSMwP and present its system composition mechanism in Section 2. We then describe a simple logistic system and show how we can model the system using the FSMwP framework in Section 3. In Section 4 we show some of the characteristic of FSMwP. We conclude the paper with comments on the further modeling and control extension of this approach in Section 5.

2 Finite State Machines with Parameters

Let us now formally introduce our model of finite state machines with parameters. First, let us recall that a finite state machine (FSM) is described by a 5-tuple [Cl99]

$$FSM = (\Sigma, Q, \delta, q_0, Q_m),$$

where $\Sigma$ is the event set, $Q$ the state set, $\delta : \Sigma \times Q \rightarrow Q$ the transition function, $q_0$ the initial state, and $Q_m$ the marked (or final) states.

To introduce parameters into an FSM, let $p \in P$ be a vector of parameters, where $P$ is some vector space. More often, $P$ is over the set of natural numbers. We also introduce guards $g \in G$ that are predicates on the parameters $p$. The transition function $\delta$ can be defined as a function from $\Sigma \times Q \times G \times P$ to $Q \times P$ as illustrated in Figure 1. The transition shown is to be interpreted as follows: If at state $q$ the guard $g$ is true and the event $\sigma$ occurs, then the next state is $q'$ and the parameters will be updated to $f(p)$.

$$g \land \sigma /p:=f(p)$$

$q$ $q'$

Figure 1: A transition in FSMwP.

If $g$ is absent in the transition, then the transition takes place when $\sigma$ occurs. Such a transition is called event transition. If $\sigma$ is absent, then the transition takes place when $g$ becomes true. Such a transition is called dynamic transition. If $p := f(p)$ is absent, then no parameter is updated during the transition.

Any transition with both event and guard can be decomposed into event transitions and dynamic transitions by introducing some extra states. For example, the transition in Figure 1 can be decomposed as shown in Figure 2. In the figure, $\neg g$ stands for the negation of $g$. For simplicity of discussion, we assume that such decomposition has been performed before any other operations is to be performed. After the decomposition, all transitions can be denoted as $q \rightarrow q'$, where $l$ can be either an event or a guard. We often view $\delta$ as a set of transitions and use the notation $q \xrightarrow{l} q' \in \delta$ to denote that $q \xrightarrow{l} q'$ is defined by $\delta$.

$$g \land \sigma /p:=f(p)$$

$q$ $q'$ $\neg g$

Figure 2: Transition decomposition.

In summary, a finite state machine with parameters can be viewed as a 7-tuple

$$FSMwP = (\Sigma, Q, \delta, P, G, (q_0, p_0), Q_m),$$

where $p_0$ is the initial value of parameters at the initial state $q_0$.

Without much difficulty, we can regard finite state machines with parameters as a special type of Hybrid Machines (HMs) introduced in [HLM97]. In particular, an FSMwP has no continuous dynamics (i.e., $\dot{p} = 0$ at any state). The only way to change parameters is by updating (or re-initialization).

2.1 Parallel Composition of FSMwPs

Similar to FSMs and HMs, we can define parallel composition of several FSMwPs running in parallel to form a composite finite state machine with parameters (CFSmwP)

$$CFSMwP = FSMwP_1 || FSMwP_2 || ... || FSMwP_n.$$
event transitions and dynamic transitions. Hence,

$$CFSM_{wP} = FSM_{wP_1} || ... || FSM_{wP_n}$$

$$= (\Sigma_1, Q_1, \delta_1, P_1, G_1, (q_{01}, p_{01}), Q_{m1}) || ... $$

$$||(\Sigma_n, Q_n, \delta_n, P_n, G_n, (q_{0n}, p_{0n}), Q_{mn})$$

$$= (\Sigma_1 \cup ... \cup \Sigma_n, Q_1 \times ... \times Q_n, \delta_1 \times ... \times \delta_n, $$

$$P_1 \cup ... \cup P_n, G_1 \cup ... \cup G_n, $$

$$(q_{01}, ..., q_{0n}, p_{01}, ..., p_{0n}), Q_{m1} \times ... \times Q_{mn})$$

$$= (\Sigma, Q, \delta, P, G, (q_0, p_0), Q_m),$$

where the transition function $\delta = \delta_1 \times ... \times \delta_n$ is defined as illustrated in Figures 3 and 4 for $n = 2$. In the figures, $l_i$ can be either an event ($l_i = \sigma_i$) or a guard ($l_i = g_i$). If $l_1 \neq l_2$, then the situation is illustrated in Figure 3. For example, if the transition $l_1$ occurs at state $(q_1, q_2)$, then the next state is $(q'_1, q'_2)$. Parameter $p_1$ is updated to $f_1(p_1)$ while $p_2$ is not updated. On the other hand, if $l_1 = l_2 = l$, then the situation is illustrated in Figure 4. That is, if the transition $l$ occurs at state $(q_1, q_2)$, then the next state is $(q'_1, q'_2)$. Parameters $p_1$ and $p_2$ are updated to $f_1(p_1)$ and $f_2(p_2)$ respectively.

$$l_1/p_1 = f_1(p_1) \quad l_2/p_2 = f_2(p_2)$$

$$l_1/p_1 = f_1(p_1) \quad (q_1, q_2) \quad l_2/p_2 = f_2(p_2) \quad (q'_1, q'_2)$$

$$l_1/p_1 = f_1(p_1) \quad (q_1, q_2) \quad l_1/p_1 = f_1(p_1) \quad (q_1, q'_2)$$

Figure 3: Parallel composition: $l_1 \neq l_2$.  

$$l_1/p_1 = f_1(p_1) \quad l_2/p_2 = f_2(p_2)$$

$$(q_1, q_2) \quad (q_1, q'_2)$$

Figure 4: Parallel composition: $l_1 = l_2 = l$.

We note that this definition is an extension to that of FSM [CL99]. Using this parallel composition, we can build large systems from simple components. This procedure can be automated.

In practice, it is possible that one event may have different labels in two interacting processes. For example, the output from Workstation A (labeled $\alpha$) could be the input to a cart (labeled $\gamma$). To model such a system, it is not always convenient to insist that $\alpha$ and $\gamma$ shall be given the same label. This is because that the cart may move from one station to another. So $\gamma$ may correspond to the output from Workstation B (labeled $\beta$) the next time. A convenient way to deal with this situation is to introduce dynamically masked parallel composition as follows.

A mask $M_i$ on events $\Sigma_i$ is a mapping $M_i : \Sigma_i \rightarrow \Sigma_i$, where $\Sigma_i$ is some interface events. What $M_i$ will do is to rename (or re-label) events in $\Sigma_i$ to events in $\Sigma_i$. There is no restriction on what $\Sigma_i$ may be. For example, some events in $\Sigma_i$ may be the same as the events in $\Sigma_i$, which means these events are not re-labeled. If no events are re-labeled, then we have the normal parallel composition and $\Sigma_i = \Sigma_i \cup ... \cup \Sigma_n$. In principle, $\Sigma_i$ may contain the empty string $\epsilon$. However, this will cause nondeterminism that is beyond the scope of this paper.

A masked $FSM_{wP}$, denoted by $M_i(FSM_{wP_i})$, is obtained from $FSM_{wP_i}$ by re-labeling events in $\Sigma_i$ by events in $\Sigma_i$ according to $M_i$.

Masked parallel composition is then defined in the same way as parallel composition.

$$CFSM_{wP} = M_1(FSM_{wP_1}) || M_2(FSM_{wP_2}) || ...$$

$$|| M_n(FSM_{wP_n}).$$

A similar concept for finite state machines, termed masked prioritized synchronous composition, was introduced in [KH00]. Different from the constant masks in their approach, our masks $M_i$ can be dynamic. This allows us, as in the previous example, to change the mask of $\gamma$ (input to the cart) from $\alpha$ (output from Workstation A) to $\beta$ (output from Workstation B), when the cart moves from Workstation A to Workstation B.

To this end, we define a dynamic mask as $M_i^d : \Sigma_i \rightarrow \Sigma_i$. Depending on $d \in D = \{d_1, d_2, ...,\}, M_i^d$ will give different mapping. Dynamically masked parallel composition is then defined as

$$CFSM_{wP} = M_1^d(FSM_{wP_1}) || M_2^d(FSM_{wP_2}) || ...$$

$$|| M_n^d(FSM_{wP_n}).$$

where $d$ may change from state to state, or from time to time.
2.2 Parallel Composition without Transition Decomposition

In the above construction of parallel composition, we assume that the guarded event transitions in EHMs have been decomposed into event transitions and dynamic transitions. This decomposition makes the theoretical discussions simple and straightforward. However, it may not be very practical to always decompose guarded event transitions in applications as this may increase the number of states significantly. Therefore, we would like to investigate the possibility of doing parallel composition without decomposition of guarded event transitions.

It is not difficult to prove that, by comparing the results of parallel composition after decomposition with the results of decomposition after parallel composition, if all FSMwP have disjoint event sets (i.e., $\Sigma_i \cup \Sigma_j = \emptyset$, for all $i \neq j$), then the parallel composition can be done without decomposition. Similar to Figure 3, the parallel composition without decomposition is illustrated in Figure 5. Notice that, since event sets are disjoint, the case similar to that in Figure 4 is no longer possible. If the event sets are not disjoint, then the situation will be more complicated and the parallel composition without decomposition may not produce the correct results.

![Figure 5: Parallel composition without transition decomposition.](image)

3 A Logistic System Example

To illustrate our FSMwP modeling framework, let us consider a simple logistic system as shown in Figure 6. The system consists of two depots, two stations and two trams. The trams can move between depots and stations along some rails. The task in the system is to transport parts between depots and stations. There may be more than one type of parts to be transported in the system. The parts are loaded at the depots by loading robots.

To model the system, let us divide the floor space into grids of $3 \times 4$ as shown in Figure 7. The depots ($D_1$ and $D_2$), stations ($S_1$ and $S_2$), and the rail are shown in Figure 7 as well.

To find the FSMwP model for Tram 1, we assume that Tram 1 starts at $D_2$ initially and it can move in four directions, denoted by E (east), N (north), W (west), and S (south) respectively. The current coordinates of the tram are treated as parameters, denoted by $X$ (horizontal) and $Y$ (vertical) respectively.

![Figure 7: Floor space grids for the logistic system.](image)

Since a move to the east will increase $X$ by 1, we have $E/X := X + 1$. Similarly,

$\frac{N}{Y} := Y + 1$;
$\frac{W}{X} := X - 1$; and
$\frac{S}{Y} := Y - 1$.

Initially, $X = 1, Y = 1$. Because of the rail, the tram can only move east if $Y = 2$ and $X < 4$, that is,


Similarly,

$((X = 1) \lor [X = 4]) \land [Y < 3] \land N/Y := Y + 1$;

$([Y = 2] \land [X > 1] \land W/X := X - 1$; and

$((X = 1) \lor [X = 4]) \land [Y > 1] \land S/Y := Y - 1$.
Since movements take time, we introduce a time transition \( \omega \). Assume that each movement takes 2 time units (e.g., seconds). Then the time transition is defined as

\[
\omega: q \overset{[t-t_o \geq 2]}{\longrightarrow} q'
\]

where \( t \) is the global clock value and \( t_o \) is the time when the system enters state \( q \) (\( t_o := t \) at the time of entry).

The tram will be loaded (L) and unloaded (U) at the depots and stations. Each occurrence of L will increase the number of parts on the tram by 1. Let \( p \) be the number of parts on the tram, then

\[
L/p := p + 1.
\]

Similarly,

\[
U/p := p - 1.
\]

Based on these elements described above, the FSMwP model of the tram is shown in Figure 8.

![Figure 8: An FSMwP model of a tram.](image)

Loading and unloading also takes time, but they depend on the speeds of robots. Therefore, it will be considered in the robot model.

To find the FSMwP model for a robot, note that the robot is fixed on the floor but its arm can move back (B) and move forward (F) into three positions: Bin, Home, and Dock. At Bin, the robot can pick up (PU) a part if its payload is 0 or put down (PD) a part if its payload is 1. Similar, it can execute PU and PD at Dock. All these events take time (assuming one second in each case). Therefore, the robot is modeled as in Figure 9.

![Figure 9: An FSMwP model of a robot.](image)

**4 Characteristics of FSMwP**

To describe the behavior of an FSMwP,

\[
FSMwP = (\Sigma, Q, \delta, P, G, (q_0, p_0), Q_m),
\]

we define a run of an FSMwP as a sequence

\[
r = (q_0, p_0) \overset{l_1}{\longrightarrow} (q_1, p_1) \overset{l_2}{\longrightarrow} (q_2, p_2) \overset{l_n}{\longrightarrow} (q_n, p_n) \ldots,
\]

where \( l_i \) is (the label of) the \( i \)th transition and \((p_i, q_i)\) is the state and parameter values after the \( i \)th transition. We denote the set of all possible runs of FSMwP as

\[
R(FSMwP) = \{ r : r \text{ is a run of } FSMwP \}.
\]

A trace of a run is the sequence of event or dynamic transitions in the run

\[
s = l_1l_2l_3 \ldots.
\]

That is, \( s \) is obtained from \( r \) by deleting the state information and parameter values.

If an FSMwP is deterministic (which we assume throughout this paper), then a run is uniquely determined by its trace. That is, we can reconstruct a run by looking at its trace and the FSMwP. The set of all traces of an FSMwP is a language denoted by

\[
L(FSMwP) = \{ s : s \text{ is a trace of } FSMwP \}.
\]

This language is called the language generated by FSMwP. The language marked by FSMwP is defined as

\[
L_m(FSMwP) = \{ s \in L(FSMwP) : \text{ } s \text{ ends in a marked state } q \in Q_m \}.
\]
Clearly, the set of all languages marked by FSMwPs is strictly larger than the set of regular languages. This can be seen by the following example.

Let

\[ L = \{ a^n b^n : n \text{ is a natural number} \} . \]

Obviously, \( L \) is not a regular language. However, the FSMwP in Figure 10 marks this language.

- denotes the marked state

**Figure 10: FSMwP model for \( L \).**

Since CFMSwP and FSMAwP have the same structure, runs, traces, and languages for CFMSwP are defined similarly.

5 Conclusion

In this paper, we have introduced a discrete event system modeling framework that appends additional parameters to finite state machines. We have shown, through an example, that this framework can represent efficiently systems that cannot normally be represented by purely finite state machines without huge state space. Characteristics of this modeling framework have been presented along with model synthesis techniques that can be used to build models for systems consisting of many components.

In addition to using parameters, modeling of very complex discrete event systems can be further managed by adopting a hierarchical structure. We are developing a multi-level hierarchical representation based on our FSMwP framework that enables automatic synthesis of higher-level models [CL00].

The ultimate goal of this approach is not only to model a complex discrete event system efficiently, but also to control the system efficiently. To this end, we are studying how to design controllers for systems modeled as FSMwP. Two types of controllers are currently under investigation: One is a safety controller to ensure the system never entering illegal states. This is similar to the supervisory control theory. However, with the introduction of parameters, the synthesis procedure is more complicated. State splitting according to parameter values may sometimes be involved. The other type is an optimal controller that guarantees effectiveness of a system using minimal cost. Initial results of our efforts are also described in [CL00].

References


