ORIGINAL ARTICLE

Effective Supervisor Synthesis for Automated Manufacturing Systems Based on Finite State Automata

Ho-Shan Chiang[1] and Yen-Liang Pan[2, *]

[1] *Communications, Electronics and Information Division, Army Command Headquarters, Ministry of National Defense, Taiwan, R.O.C.*

[2] ** Mathematics and Physics Division, General Education Center, Air Force Academy, Kaohsiung 820, Taiwan, R.O.C.*

(**Received** 8 September2017; **Accepted** 28 September 2017; **Published on line** 1 June 2018)

*Corresponding author: peterpan960326@gmail.com

DOI: 10.5875/ausmt.v8i2.1698

Abstract: This paper proposes a novel supervisor for automated manufacturing systems (AMSs) using finite state automata (FSA). The synthesis method is useful for obtaining a legal supervisor, but the transitional method cannot prevent state explosion. Therefore, this paper uses a new synthesis policy to significantly reduce the number of illegal states. The proposed method effectively controls the inherent problem of exponential numbers of states by using the conventional method. In addition, a real world AMS is introduced to significantly improve performance efficiency.

Keywords: Automated manufacturing systems; finite state automata; synthesis

I. Introduction

An AMS is a complex mechanical system capable of fabrication, machining and assembly operations, and entails a high degree of resource sharing [1]. It comprises many interrelated combinations of subsystems, including treatment stations, transport systems, communications systems and monitoring systems [2]. By integrating computer systems and hardware, AMS provides manufacturers with enhanced flexibility and efficiency.

The interacting parts and event interaction in an AMS require communication and information dissemination characterized as a system synthesis process that can raise a state explosion problem where the number of states in the global system model grows exponentially with the number of subsystems [3]. The state explosion problem is a complex and important issue in discrete event systems (DESs) [4][5].

This paper proposes a novel approach based on [6] to reduce the computational complexity of the synthesis process and the supervisor policy size (i.e. the number of feasible states) for AMSs. Moreover, to address the exponential state problem, we also develop another novel synthesis method to enhance modeling efficiency and to reduce the state explosion problem using leader event (LE) and follower event (FE).

The remainder of this paper is organized as follows. Section II introduces some relevant concepts, notations and properties of FSA. Section III shows our proposed new synthesis method of AMSs. Discussion and conclusions are presented in Section IV.

II. Preliminaries

2.1. Basic Concepts of Automata

A deterministic finite automaton (DFA) is represented by the 5-tuple $A = (\Sigma, Q, q_0, \delta, F)$ where Σ is a nonempty finite set of events, Q is a nonempty finite set of element call states, $q_0 \in Q$ is an initial (or start) state, that is, the automaton state when no input has yet been processed, δ is a state transition partial function given by $\delta: Q \times \Sigma \rightarrow Q$, where the *Cartesian product* $\times$ means that an ordered pair of elements from Q and Σ is mapped into an element of set Q, and the term "partial function" means that the function δ may not be defined for all ordered pairs that can be created of the sets Q and Σ, and F is a set of final states given as a subset of Q: $F \subseteq Q$ called accepts states, where F can be the empty set. A sample transition diagram of a finite acceptor is shown in Fig. 1.

There are two states q_0 and q_1. The initial state, q_0, is the designated start state and the second state, q_1, is a designated final state, respectively indicated by an inward-pointing arrow and a double circle. The alphabet Σ is defined as $\{\alpha, \beta\}$. This automaton describes the language where all strings contain an odd number of the symbol α, for it is only with an input string that satisfies that restriction that the automaton will end up in state q_1. In Fig.1 we mark the initial state by an inward-pointing arrow, and the

terminal states by double circles. In this example, it is rather easy to see that $\Sigma = \{\alpha, \beta\}$, $Q = \{q_0, q_1\}$, $F = \{q_1\}$. The transition function δ: $Q \times \Sigma \to Q$ is given by

$$\delta(q_0, \beta) = q_0, \quad \delta(q_0, \alpha) = q_1$$
$$\delta(q_1, \beta) = q_1, \quad \delta(q_1, \alpha) = q_0$$

Thus, the function δ can be represented either graphically as arcs, as in Fig. 1, or textually as a table, as in the following Table 1.

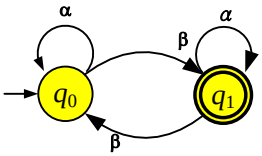


Fig. 1. A deterministic finite automaton.

Table 1. Transition table of DFA.

State/Event	β	α
q_0	q_0	q_1
q_1	q_1	q_0

2.2. Basic Definitions for Synthesis

Definition 1. [7] The FSC of n automata M_1, M_2...M_n, is denoted $M_1 \| M_2 \| ... \| M_n$ where

$$Q = Q^1 \times Q^2 \times .....Q^n$$
$$\Sigma = \Sigma^1 \cup \Sigma^2 \cup .....\Sigma^n \quad (1)$$
$$q_0 = \langle q_0^1, q_0^2, ..., q_0^n \rangle$$

and the transition function is $\delta(\langle q^1, q^2, .., q^n \rangle, \sigma) =$

$$\begin{Bmatrix} \text{undefined if } \exists A^k : \sigma \in \Sigma^k \wedge \sigma \notin \Gamma^k\left(q^k\right) \\ \langle q^{1+}, q^{2+}, .. q^{n+} \rangle \text{ otherwise} \end{Bmatrix} \quad (2)$$

where

$$q^{k+} = \begin{Bmatrix} \delta^k\left(q^k, \sigma\right) & \text{if} & \sigma \in \Gamma^k\left(q^k\right) \\ q^k & \text{if} & \sigma \notin \Sigma^k \end{Bmatrix} \quad (3)$$

Based on the definition above, the FSC of two automata M_1, M_2 is defined as $M = M_1 \| M_2$.

Definition 2: A string $\omega \in \Sigma^*$ is accepted by the DFA $A = (\Sigma, Q, q_0, \delta, F)$ if $\delta^*(q_0, \omega) \in F$

Definition 3: The language $L(A)$ accepted by the DFA $A = (\Sigma, Q, q_0, \delta, F)$ is $L(A) = \{\omega \in \Sigma^* \mid \delta^*(q_0, \omega) \in F\}$

Definition 4: A marker state (sometimes referred to as an accepting state) is a state at which the machine has successfully performed its procedure.

Definition 5: An event of a DFA that can trigger an event called a *leader event* (L_E).

Definition 6: An event of a DFA that can be triggered by a *leader event* called a *follower event* (F_E).

Definition 7: A *follower event* (F_E) exists if its *leader event* (L_E) exists.

In this paper, we use a long dashed line to connect a L_E with a F_E. The long dashed line begins from the L_E and ends at the F_E. In Fig. 2, the event $\beta(\tau)$ is called the *leader event* of the event $\rho(\gamma)$. On the other hand, the event $\rho(\gamma)$ is called the *follower event* of the event $\beta(\tau)$. Notably, leader events and follower events are located in different automata.

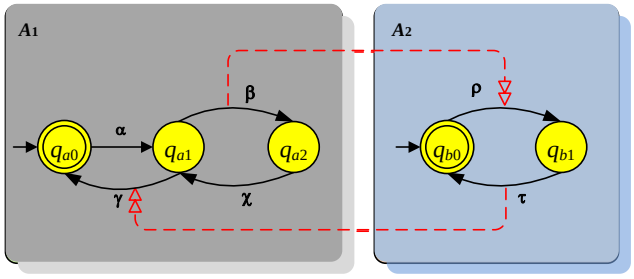


Fig. 2. Relation beween leader and follower events.

Definition 6 suggests that L_E and F_E will be presented by a pair form and they are placed in different DFAs. Here, we use the notation E_{LF} to represent the pair events of (L_E, F_E) (i.e. $E_{LF} = (L_E, F_E)$).

A F_E can be triggered if it satisfies the following two conditions: 1) $E_{LF} = (L_E, F_E)$ exist; 2) L_E happens and F_E is in an actual node.

The machine switches from state to state based on what symbol is read at each point. If the machine ends up in one of a set of particular states, then the string of symbols is called a legal state

Ho-Shan Chiang received the B.S. degree in electrical engineering from Republic of China Military Academy, Kaohsiung, Taiwan, in 1994, and the M.S. degree in Computer Science and the Ph.D. degree in control theory and control engineering from Chung Cheng Institute of Technology National Defense University, Taoyuan, Taiwan, in 2006 and 2012, respectively. His current research interests include supervisory control of discrete event system, automata applications, and automatic manufacturing systems.
hoshanjiang@gmail.com

Yen-Liang Pan received the B.S. and M.S. degrees in aeronautical engineering from Chung Cheng Institute of Technology, Taoyuan, Taiwan, in 1996 and 2001, respectively, and the Ph.D. degree in electrical and electronic engineering from National Defense University, Taoyuan, Taiwan, R.O.C. in 2011. He is currently Associate Professor and Chair in Mathematics and Physics Division, General Education Center, Air Force Academy, Kaohsiung, Taiwan. He was Visiting Scholar at the Discrete Event System Laboratory, New Jersey Institute of Technology (NJIT) in the summer of 2009. His research interests include Petri net theory and applications, computer-integrated manufacturing, automation, and supervisory control of discrete event systems. In 2013, He was invited be the chairman in the section of Artificial Intelligence and Soft Computing at AsicMIC 2013. Also, he was invited to organize one special section "Petri Nets' Applications in Intelligent Technologies and Engineering Systems" and be the chairman at ICITES 2013. In 2012, 2013, 2014 and 2015, he is awarded the excellent research at Air Force Academy.
peterpan960326@gmail.com

(F_i) (or is said to be accepted). If it ends up in any other state, then the string is an illegal state ($F_{il.}$) (i.e, not accepted).

A feasible event that is one which can be deterministic, and the action of the machine is completely determined by its current state and current input and no choice is allowed. In addition to being deterministic, an arrow must leave a given state for each of the input letters; there can be no missing arrows or the event is infeasible event.

III. Supervisor Synthesis

This section proposes a novel synthesis method to improve the state explosion problem. A real case of AMS is used to test the proposed synthesis method. We first introduce how to model one real-world AMS.

3.1 Modeling One AMS

The components of one real-world AMS are shown in Fig. 3. This AMS consists of robots, a CNC (i.e., CNC lathe and CNC MC), a flexible assembly system (FAS), an automated storage and retrieval system (AS/RS) unit, a conveyor and coordinates measuring machines (CMM).

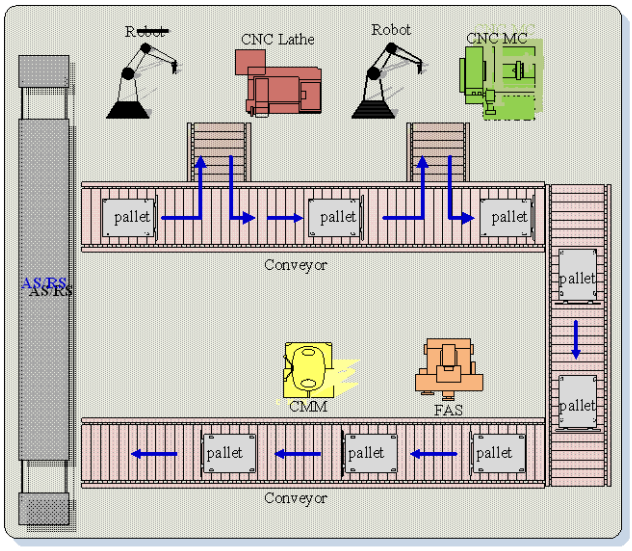


Fig. 3. Real World AMS layout.

For convenience, the behavior specifications of the AMS are described in terms of one component (shown in Fig. 4.). The plant model is based on the manufacturing system resources and their states. Resources may be machines, robots, transporters, and other equipment used to process parts.

In our modeling framework we constrain each resource to have unit capacity. The plant model describes the possible states of the plant and the transitions that link those states. An example of a plant is shown in Fig. 4, where the plant is composed of three resources, two machines (M_1, M_2) and a robot (R). The plant FSA model is shown in Fig. 5.

Each major component of the behavioral specifications is listed below

M_1 is used to transport pallets to give R for loading.

Robot (R) is used to load and unload the pallet between M_2 and M_1. In other words, R has two processes: *i*) it is used to load working in piece from the pallet to M_2, *ii*) it unloads working in piece put on the pallet from M_2 when M_2 finished it's processed.

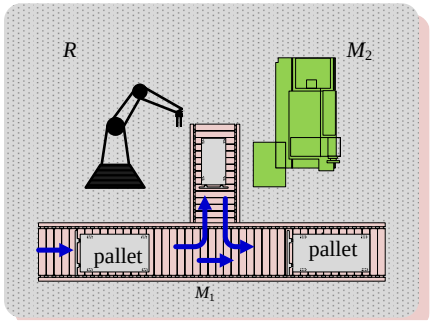
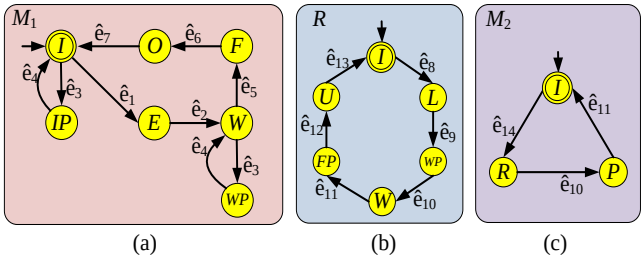


Fig. 4. A plant of the AMS.



(a) (b) (c)

Fig. 5. Three simplified FSA models of M_1 (a), R (b) and M_2 (c).

Here, we explain the relation specifications as follows;

M_1: the specification of M_1 is given below (i.e. Fig. 5a):

- initially M_1 is in idle state (I);
- supervisory instructions are issued after a pallet enters (E); it remains in idle state and passes state (IP) when there is not a pallet enter;
- if a pallet has entered the workstation state (W) then it starts work; if no pallet has entered then it passes (WP);
- M_1 moves to work finished state (F) when the M_2 process end; and
- the pallet enters the moving out state (O), followed by state I.

R: the specification of R is given below (i.e., Fig. 5b):

- initially R is in idle state (I);
- if a pallet has entered the workstation then loading (L) starts;
- if R loading ends then the cell control supervisor feeds the command for M_2 to initiate the enter process state (WP);
- R is in waiting state (W) when the M_2 process starts;
- R enters the finish process state (FP) when it receives the M_2 finished process message; and
- In the meantime, unloading (U) starts followed by the idle state.

M_2: the specifications of M_2 are depicted in Fig. 5c:

- initially M_2 is in idle state (I);

- when M_2 is in the ready state (R), M_2 will receive a work command from the supervisor; and
- M_2 process (P) begins.

The transition arcs that connect the states are labeled with the events that cause a change in state. These events are the alphabet from which strings (sequences) of events may be formed. The events are described by the notation $\hat{e}_1$, $\hat{e}_2$, etc. Their events are explained in Table 2.

Table 2. Notation of Events of Fig. 5.

Event	*Interpretation*	*Action sequence*
$\hat{e}_1$	moving in start	(1)
$\hat{e}_2$	moving in complete	(2)
$\hat{e}_3$	passed start	
$\hat{e}_4$	passed complete	
$\hat{e}_5$	processing end	(9)
$\hat{e}_6$	moving out start	(10)
$\hat{e}_7$	moving out complete	(11)
$\hat{e}_8$	loading start	(3)
$\hat{e}_9$	loading end	(5)
$\hat{e}_{10}$	R waiting	(6)
$\hat{e}_{11}$	R finishes process	(7)
$\hat{e}_{12}$	unloading start	(8)
$\hat{e}_{13}$	processing start	(9)
$\hat{e}_{14}$	M_2 available	(4)
$\hat{e}_{15}$	M_2 start	(6)
$\hat{e}_{16}$	M_2 end	(7)

For example when "$\hat{e}_1$" occurs, equipment M_1 state is being moved from state I to state E. In this application, all events are controllable events.

For a manufacturing plant, the initial state is the idle state and, after all production is completed, the plant is idle again.

3.2. An Efficient Synthesis Method

In any interactive application, the use of a formalism describing the system's dynamics is of capital importance [9]. Indeed, modeling is all the more necessary since it provides formal analysis techniques which make it possible to prove the design properties before implementation [10,11].

Our new synthesis method is described as follows. Our method adds L_E and F_E to the original net. From the above discussion, a new interactive model of the plant is constructed in Fig. 6. It uses the dashed line to connect two events of the same name, where the dashed line is denoted by L_E and the solid line is denoted by F_E.

The FSA of Fig. 6 has L_E with F_E, e.g. $\hat{e}_{10}$ has a F_E $\hat{e}_{15}$. This means that, when event $\hat{e}_{10}$ occurs, equipment R and M_2 are respectively changed to state W and P form WP and R. Here, equipment M_1 is still held at state W. That is, $M_1 \rightarrow \delta(W, \varepsilon) = W$, $R \rightarrow \delta(WP, \hat{e}_{10}) = W$ and $M_2 \rightarrow \delta(R, \hat{e}_{15}) = P$.

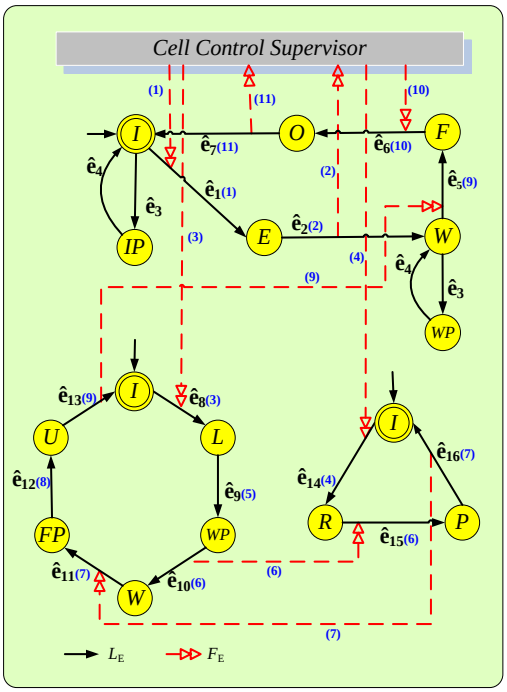


Fig. 6. Interactive events diagram.

This interactive event diagram is constructed by combining the component of the plant with the corresponding states of Fig. 5. The interactive event diagram process is determined as described above in Section 3.1. Our goal is to further reduce the size of the state space. This approach to state space reduction saves computational time without restricting the marked language of the coupled system. Finally, the following steps describe how to synthesize a net by using this method:

Algorithm.1: *FSC method*

Given a DFA:

Step 1: Find all strict states of a given DFA.

Step 2: According the DFA specification, remove the illegal states F_{il}.

Step 3: Mark the distinguishable state pairs.

To fulfill this task, this method first marks all pairs F_{il}; F_l, where $F_{il} \notin F$ and $F_l \in F$ are distinguishable. Next, we proceed as follows:

repeat

for all non-marked pairs F_{il}; F_l do

for each letter ω do

if the pair (F_{il}; ω),(F_l; ω) is marked

then mark F_{il}; F_l

until no new pairs are marked

Step 4: Construct the reduced DFA.

Please note that, in traditional theory, computation is a critical task for the development of synthesis policies. In a general case, the solution to synthesis depends on the reference to the original DFA specification and removes F_{il} which imposes time-consuming computation for a large-size net. Hence, when the number of such states is large, it leads to a much more structurally complex FSA supervisor than the plant model.

IV. Experimental Result and Comparison

One classical example is used to evaluate our proposed method, as shown in Fig. 7.

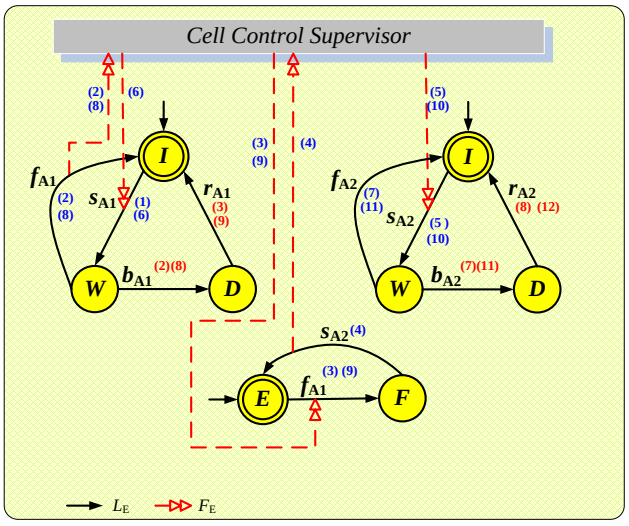


Fig. 7. An example of [6] interactive behavior diagram.

In Fig. 7, the machines initially should be in *idle* (*I*, *I*) state, and *B* is *empty* (*E*), respectively states *I*, *E* and *I*. For example, when event s_{A1} occurs, equipment A_1 is changed to state *W* from *I*. At this time, equipment *B* and A_2 are still respectively held at state *E* and *I*. That is, $A_1 \rightarrow \delta(I, s_{A1}) = W$, $B \rightarrow \delta(I, \varepsilon) = E$ and $A_2 \rightarrow \delta(I, \varepsilon) = I$, etc.

Depending on what event happens next it may either:

ACTION(1):

A_1. Initially it is idle in state *I*. Once it starts operating, i.e., the event s_{A1} occurs, its state changes to *W* and equipment *B* and A_2 are still respectively kept at states *E* and *I* due to empty event ε. That is, $A_1 \rightarrow \delta(I, s_{A1}) = W$, $B \rightarrow \delta(E, \varepsilon) = E$, $A_2 \rightarrow \delta(I, \varepsilon) = I$.

ACTION(2):

- When event f_{A1} occurs, A_1 changes its state to working (*W*), if it successfully finishes its work cycle (*F*). That is, $A_1 \rightarrow \delta(W, f_{A1}) = I$, $B \rightarrow \delta(E, f_{A1}) = F$, $A_2 \rightarrow \delta(I, \varepsilon) = I$; or
- When event b_{A1} occurs, A_1 changes its state to down (*D*), if it breaks down (b_{A1}). That is, $A_1 \rightarrow \delta(W, b_{A1}) = D$, $B \rightarrow \delta(E, \varepsilon) = E$, $A_2 \rightarrow \delta(I, \varepsilon) = I$. Then A_1 is repaired and returned to service (r_{A1}), bringing the machine back to its idle state (*I*).

ACTION(3):

When event s_{A2} occurs, *B* and A_2 change their states to *E* (*W*) and equipment A_1 is still kept at state *I* due to empty event ε. That is, $A_1 \rightarrow \delta(I, \varepsilon) = I$, $B \rightarrow \delta(F, s_{A2}) = E$, $A_2 \rightarrow \delta(I, s_{A2}) = W$.

ACTION(4):

- When event s_{A1} occurs, A_1 changes its state to working (*W*), if it successfully finishes its work cycle. Equipment *B* and A_2 are respectively kept at states *E* and *W* due to empty event ε. That is, $A_1 \rightarrow \delta(I, s_{A1}) = W$, $B \rightarrow \delta(E, \varepsilon) = E$, $A_2 \rightarrow \delta(W, \varepsilon) = W$; or
- When event f_{A2} occurs, A_2 changes its state to idle (*I*), if it successfully finishes its work cycle. That is, $A_1 \rightarrow \delta(I, \varepsilon) = W$, $B \rightarrow \delta(E, \varepsilon) = E$ and $A_2 \rightarrow \delta(W, f_{A2}) = I$.
- When event b_{A2} occurs, A_2 changes its state to down (*D*), if it breaks down (b_{A2}). That is, $A_1 \rightarrow \delta(I, \varepsilon) = I$, $B \rightarrow \delta(F, s_{A2}) = E$, $A_2 \rightarrow \delta(W, b_{A2}) = D$.

And A_2 is repaired and returned to service (r_{A2}), bringing the machine back to its idle state (*I*). That is, $A_1 \rightarrow \delta(I, r_{A2}) = I$, $B \rightarrow \delta(E, \varepsilon) = E$, $A_2 \rightarrow \delta(D, r_{A2}) = I$,; or event s_{A1} occurs, A_1 changes its state to working (*W*). That is, $A_1 \rightarrow \delta(I, s_{A1}) = W$, $B \rightarrow \delta(E, \varepsilon) = E$, $A_2 \rightarrow \delta(W, \varepsilon) = D$. Then A_2 is repaired and returned to service (r_{A2}), bringing the machine back to its idle state (*I*), and so on.

Using the proposed L_E/F_E approach, we can obtain results identical to those in [12]. Table 3 compares the results of the proposed approach against all existing approaches to deal with synthesis in aMS using PNs and FSAs [6,12-16].

The first column of Table III lists the references, while the second one denotes the model used to describe a system and the interactions between events and states. The third column shows some key words used to describe the used approach used to characterize synthesis. The system model is built by merging these modules through their common transitions and common transition subsystems. Finally, last three columns respectively show the AMS type respectively considered in the model, the complexity of the proposed solution algorithm, and the synthesis methods.

For instance, there are 146 states (i.e. 133 non-accepted states and 13 accepted states) using the FSC policy. In contrast, we can obtain the 13 legal states by directly using our synthesis method.

The method in [12] formulates a special class of shared-resource systems with restrictions using structural and behavioral conditions on PN models. In [13] a PN model with hierarchical modeling methodology facilitates the modeling task incorporating a top-down stepwise refinement technique. This approach necessitates the development of new PN model to represent primitive rules and another to represent metarules that can be further refined into subnets.

Zhou et al. [14] focused on the refinement-based PN theory, using stepwise refinement to reduce synthesis complexity. Based on the concept of S-decreases, Chen [15] also presented a method to calculate the criteria of the admissible markings in the form of linear inequalities. Furthermore, Zhou et al. [16] presented a top-down modular approach for the systematic PN synthesis of AMSs.

Replacing places and/or transitions of the original nets with basic PN design modules, the resulting refined PN can be guaranteed to preserve behavioral properties.

Tables 3. Comparisons of different approaches synthesis methods.

References	*System model*	*Approach to characterize synthesis*	*Merged model*	*Type of system*	*Complexity*	*Synthesis method*
[12]	PN	Aggregate and refines the model progressively	Multiplex	Independent subsystems	Polynomial	Parallel and sequential mutual exclusions
[13]	PN	Hierarchical control	Multiplex	Subsystems	Polynomial	Bottom-up and top-down techniques
[14]	PN	Decentralized control	Multiplex	Large AMSs	Polynomial	Stepwise Refinement-Based
[6]	FSA	Searching over the state space	Single	Subsystems	Exponential	FSC
[15]	PN	Generalized mutual exclusion constraint	Multiplex	Subsystems	Polynomial	The concept of minimal support S-decreases.
[16]	PN	Replacing places and/or transitions of original nets	Multiplex	Subsystems	Polynomial	Top down modular approach
Proposed Approach	FSA	Referring to the specific application	Single	Large AMSs	Original	The concept of L_E/F_E

Conclusions

This paper proposes an efficient synthesis method for AMSs based on automata approaches. The different models and synthesis policy techniques are presented in their historical evolution to describe the development of resolution methods over the last ten years. Indeed, starting from the analysis of deadlocked operating systems, solution methods have been specialized for AMS and for component structures, which are increasingly complex. Tractable synthesis techniques show the most promise for reducing complexity and preventing the state explosion problem while achieving desired system performance. Synthesis methods build the models systematically and progressively such that the desired properties are preserved all along the design process. Hence, research studies have particularly focused on one of the major issues which have received wide attention in the context of automated manufacturing.

The main advantages of the proposed approach include:

1. It controls the inherent problem of the exponential states which arises from the conventional method.

2. It reveals the underlying relationships among different FSA component models, and provides a refined model for system synthesis.

3. It preserves modularity, thus avoiding computational complexity which would necessitate the development of a new model framework.

4. A modular supervisor is easier to understand.

Future work will focus on the further development of less complex synthesis techniques for practical system synthesis problems.

References

[1] M. P. Fanti and M. C. Zhou, *IEEE Transactions on systems, man, and cybernetics- part A: systems and humans,* vol. 34, no. 1, 2004.

[2] M. R. Groover, *Automation, Production Systems, and Computer-Integrated Manufacturing*, Prentice Hall International, 2008.

[3] A. Valmari, "The state explosion problem," Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science, vol. 1491, pp. 429–528, Springer, Berlin, 1998. doi: 10.1007/3-540-65306-6_21

[4] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis," Modeling with Generalized Stochastic Petri Nets," *Wiley Series in Parallel Computing John Wiley and Sons*, New York: John Wiley & Sons, 1995.

[5] A. Valmari, "The state explosion problem," *Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science*, vol. 1491 pp. 429–528, Berlin: Springer, 1998.

[6] B. A. Brandin, "The real-time supervisory control of an experimental manufacturing cell," *IEEE Trans. Robotics and Automation*, vol. 12, no. 1, pp.1-14, 1996.

[7] C. A. R. Hoare, "Communicating Sequential Processes," *Prentice-Hall International Series in Computer Science*, 1985.

[8] F. Chu and X.-L. Xie, "Deadlock Analysis of Petri Nets Using Siphons and Mathematical Programming," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 793-804, 1997.

[9] H. Ezzedine, A. Trabelsi, and C. Kolski, "Modelling of an interactive system with an agent-based architecture using Petri nets, application of the method to the supervision of a transport system," *Mathematics and Computers in Simulation Computational Engineering in Systems Applications*, vol. 70, no. 5, pp. 358-376, 2006. doi: 10.1016/j.matcom.2005.11.006

[10] P. Palanque and R. Bastide, "Synergistic modelling of tasks, users and systems using formal specification techniques," *Interacting with Computers*, vol. 9, no. 2, pp. 129–153, 1997. doi: 10.1016/S0953-5438(97)00013-1

[11] M. Abed, H. Ezzedine, and C. Kolski, Modélisation des tâches dans la conception et l'évaluation des systèmes interactifs: la méthode SADT/Petri, Paris: Hermes, vol. 1, pp. 145–174, 2001.

[12] M. Zhou and F. DiCesare, "Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources," *IEEE Transactions on Robotics and Automation,* vol. 7, no. 4, pp. 515-527, 1991. doi: 10.1109/70.86081

[13] M. Der. Jeng and F DiCesare, "A review of synthesis techniques for Petri nets with applications to automated manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 23, no.1, pp. 301-312, 1993. doi: 10.1109/21.214792

[14] Z.-J. Ding, C.-J. Jiang, M.-C. Zhou, and Y.-Y. Zhang, "Preserving languages and properties in stepwise refinement-based synthesis of Petri nets," *IEEE Transactions on System, Man and Cybernetics – Part A*, vol. 38, no.4, pp. 791-801, 2008. doi: 10.1109/TSMCA.2008.923064

[15] H., Chen, "Control Synthesis of Petri Nets Based on S-Decreases", *Discrete Event Dynamic Systems: Theory and Applications*, vol. 10, no. 3, pp. 233-249, 2000. doi: 10.1023/A:1008397810443

[16] M. C. Zhou, K. McDermott, and P. A. Patel, "Petri net synthesis and analysis of a flexible manufacturing system cell," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 2, pp. 523–531, 1993. doi: 10.1109/21.229464