Supervisor Computation and Representation: A Case Study

S. Miremadi [*] **K. Åkesson B. Lennartson M. Fabian**

[*] *Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: miremads@chalmers.se).*

Abstract:
When supervisory control theory is applied to industrial problems the need for a more expressive modeling formalism than plain event based automata is crucial. The models are typically built in a bottom-up structure where multiple sub-plant and sub-specifications together compose the full plant and specification, respectively. Typically, the enabling of an event in a sub-model may depend on the state of other sub-models. The standard approach is to synchronize on shared events. However, to build models of large industrial problems with complex constraints between sub-models are beyond many engineers abilities. One attempt to deal with this problem is to extend the plain automata with variables and allow guard conditions and action functions to be associated with transitions. This paper discusses the strengths and weaknesses of one such formulation that fits well together with the standard supervisory control theory. A related problem is how to represent the result after the synthesis procedure, i.e., the supervisor. In this paper we present an approach where the supervisor may be represented as extended guard conditions on the original sub-models. This allows an efficient and comprehensible representation of complex supervisors. Hence, it is preferable both from a user, as well as an implementation perspective. Both the modeling formalism based on extended finite automata and the way to represent the supervisor as extended guard conditions have been implemented in a supervisory control tool.

Keywords: Discrete event systems, supervisory control theory, automata.

1. INTRODUCTION

In the supervisory control theory (SCT) Ramadge and Wonham (1989); Cassandras and Lafortune (2008) a model of the uncontrolled *plant* and the *specification* is used to synthesize a *supervisor* that restricts the behavior of the plant such that the given specification is fulfilled. SCT can be used in various applications including automated manufacturing and embedded systems, some applications are presented in Balemi et al. (1993); Feng et al. (2007); Andersson et al. (2010).

When applying SCT to industrial problems, two issues arise. (i) How to model the plant and the specification. (ii) How to represent the synthesized supervisor. Most of the formal languages such as *finite automata* and *Petri nets* that are used as modeling formalisms in SCT are more convenient for modeling the plant and the specification. It is also common to include a composition operator, typically the full synchronous composition operator, that can be used to derive the plant and the specification models from sub-models. Usually, the same framework used for modeling is also used to represent the supervisor.

A basic problem with using finite automata and event-based synchronization between sub-models, in our view, is that it is common that the condition for a state-change in one sub-model depends on the current state of some other sub-models. In simple cases these conditions can be described with event-based synchronization but when the logic conditions become more complex and involve conjunctions, disjunctions, and negations doing this manually with event-based synchronization is both time-consuming and error-prone. In SCT, the supervisor influences control by possibly disabling controllable events from being generated by the plant. While this is a useful abstraction, it does not capture that commonly signals are used to interact between different sub-plants (e.g. different machines) and also between the logic controllers and the machines. Typically, signals can also be used to model sensors and actuators. Since signals can conveniently be modeled using shared variables, there are good reasons for including variables in a modeling framework suited for modeling large-scale applications. As a consequence of introducing variables it is natural to allow *guard* conditions, boolean-valued functions, that restrict when state transitions should be allowed. *Action* functions that update the current values when a state-transition takes place are also useful.

The idea of using finite automata extended with variables, guard conditions, and action functions is not new. Statecharts, Harel (1987), an extension of ordinary automata with hierarchy, concurrency and communication using variables, guards and actions; a variant is used in the Unified Modeling Language (UML).

When discussing modeling frameworks it is important to highlight the difference between modeling of a control logic and the modeling of plants. In the SCT, plant models spontaneously generate events. Thus, a number of different events may be generated in each plant state but which event that is generated and when the event is generated is determined by the plant itself. Compare, this to the requirements for modeling of control logic. When modeling control logic it is desirable to have a deterministic behavior in the sense that the same input sequences should always result in the same output sequence because the control logic will be hard to analyze if it produces different results in different runs. This, is not desirable when modeling plants, consider for example the situation where a machine can do an operation and finish successfully but it is also a possibility that the operation finish unsuccessfully. Thus, two different scenarios should be expressed by the plant model.

The main objective of Statecharts is to model the software, i.e. control logic, and thus not suitable, in the standard formulation, for modeling of plant behavior that is necessary within the SCT framework. In addition, there is a causality between subsystems in Statecharts, which is not desired in the SCT framework.

There also exist a number of other frameworks that are based on automata extended with variables such as Chen and Lin (2000); Yang and Gohari (2005); Gaudin and Deussen (2006). In Chen and Lin (2000), it is assumed that a variable can be updated by at most one extended automaton. In Yang and Gohari (2005), finite state machines with variables are used to implement a supervisor. The authors encode the states of a given supervisor using Boolean variables. The variables are used in guards and actions attached to the events (not transitions) of the model. In Gaudin and Deussen (2006), to ensure a least restrictive supervisor, it is assumed that all variables are local i.e. not shared between automata.

Another type of discrete event models that are used in SCT are Petri nets Giua (1992); Giua and DiCesare (1994). Petri nets can conveniently model systems with huge and also infinite state-spaces due to its explicit modeling of concurrency. While Petri nets are an attractive approach, in some situations they have, in the standard formulation, weak support for adding state-dependent conditions, with complex logic conditions since arbitrary guard conditions and action functions are not supported. Furthermore, the synthesis problem is not, in general, decidable for systems with infinite state-spaces. Since arcs are used to model constraints, standard Petri nets are in one sense a monolithic approach. Thus, Petri nets are, in our view, missing a few properties that are necessary for being suitable in large-scale SCT applications.

In this paper, we present on approach to handle both the plant and specification modeling, synthesis and representation of large-scale systems. The modeling framework is based on automata extended with variables and guard conditions and action functions, called *Extended Finite Automata (EFAs)* Sköldstam et al. (2007). EFAs do not have many of the restrictions presented in the other frameworks based on state transition models extended with variables. The supervisory synthesis has been implemented symbolically using binary decision diagrams and consequentially much larger state-spaces can be handled than what is possible using explicit state-space enumeration.

Additionally, a method for representing the synthesized supervisor as extended guard conditions on the original plant and specification models has been developed as part of this research. A goal with our research has been on generating compact guard conditions that represent a possible complex supervisor in as comprehensible way as reasonably possible. Also, the generation of the compact guard conditions is implemented symbolically using binary decision diagrams, thus allowing large state-spaces to be handled. While specific aspects of this research has been presented in previous papers, the goal of this paper is to show using an example how all pieces fit together and, hopefully, also show how the proposed approach is beneficial to users who work with large-scale systems. The full framework has been implemented in the supervisory control tool Supremica (available online); Åkesson et al. (2003); Miremadi et al. (2008).

Furthermore, there are a number of advantages in having the output as EFAs. A supervisor that is represented in a modular manner is more comprehensible and tractable for the users. In addition, typically, a modular supervisor consumes less memory on a controller. The reason is that the synchronization will be performed online on the controller, see Hellgren et al. (1999, 2002); Åkesson (2002), which can alleviate the problem of exponential growth of the number of states in the synchronization. Furthermore, since EFAs include guards and actions, they can easily be converted to controller programming languages e.g. SFC or ladder diagrams and to well-known verification tools such as NuSMV Voronov and Akesson (2009).

This paper is organized as follows: In Section 2, two types of modeling formalisms are described. Section 3 presents a case study related to automated guided vehicles. The supervisory control theory is briefly explained in Section 4 and is applied to the example. Finally, Section 5 provides some conclusions.

2. MODELING FORMALISMS

This section provides a brief introduction to two types of state transition models. We are interested in deterministic systems, and thus, we describe deterministic models.

Definition 1 (Deterministic Finite Automaton):
A deterministic finite automaton (DFA) A is a 4-tuple

$$A = \langle Q^A, \Sigma^A, \delta^A, q_0^A \rangle,$$

where:

(*i*) Q^A is a finite set of states;
(*ii*) Σ^A is a nonempty finite set of events;
(*iii*) $\delta^A\colon Q^A \times \Sigma^A \to Q^A$ is a partial transition *function* that describes the state transitions;
(*vi*) $q_0^A \in Q^A$ is the initial state;

For a transition function $\delta(q, \sigma) = \acute{q}$, q is called the *source-state* and $\acute{q}$ is called the *target-state*.

A sequence of events is called a *string* of events. An empty string is denoted by ε and all possible strings consisting of events from Σ is denoted by Σ^*.

The domain of the transition function of an automaton can be recursively extended to strings of events:

$$\delta(q,\varepsilon) = q$$
$$\delta(q,s\sigma) = \delta(\delta(q,s),\sigma) \quad \text{for} \quad s \in \Sigma^* \quad \text{and} \quad \sigma \in \Sigma.$$

The composition of two automata is defined by the *full synchronous composition* operator $\|$, which is formally described in Hoare (1985).

A finite automaton can be extended with variables, guards and actions associated to the transitions of the automaton. Such an augmentation of an ordinary automaton is called an extended finite automaton (EFA) Sköldstam et al. (2007). The transitions in the EFA are enabled if and only if the guard formula is true and when a transition is taken, updating actions of a set of variables may follow.

Definition 2 (Deterministic Extended Automaton): An extended finite-state automaton E is a 6-tuple

$$E = \langle L \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, q_0 \rangle,$$

where:

(i) $L \times V$ is the extended finite set of states, denoted by Q, where L is a set of *locations* and V is the domain of definition of the variables;
(ii) Σ is a nonempty finite set of events (the alphabet);
(iii) $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V;
(iv) $\mathcal{A} = \{a \mid a \text{ is a function from } V \text{ to } V\}$ is a collection of action functions;
(v) $\rightarrow \subseteq L \times \Sigma \times \mathcal{G} \times \mathcal{A} \times L$ is the transition function;
(vi) $q_0 = (l_0, v_0) \in L \times V$ is the initial state.

The states of the ordinary automaton are extended to $L \times V$, where $V = V^1 \times ... \times V^n$. The finite set V is the domain of definition of an n-tuple of variables $v = (v^1, \ldots, v^n)$ with initial values $v_0 = (v_0^1, \ldots, v_0^n) \in V$. The *guards* are predicates over the variables that relate each element of V to either 1 (true) or 0 (false). The *actions* are considered as functions since we are interested in deterministic systems in our work. Guards and actions are written as

$$k \;=\; g(v), \text{ where } k \in 0,1;$$
$$\acute{v} \;:=\; a(v) = (a^1(v), \ldots, a^n(v)), \text{where } \acute{v} \in V.$$

The symbol Ξ is used to denote implicit actions that do not update the value of variables. For instance, if $a^i(v^j) = \Xi$, it means that action a^i does not update v^j.

The transition relation is written as $p \xrightarrow{\sigma}_{g/a} q$, where $p, q \in Q$, $\sigma \in \Sigma$, $g \in \mathcal{G}$ and $a \in \mathcal{A}$. If g is absent, denoted by $p \xrightarrow{\sigma}_{a} q$, it is assumed that g always evaluates to true. If a is absent, denoted by $p \xrightarrow{\sigma}_{g} q$, it is assumed that $a(v) = (\Xi, \Xi, \ldots, \Xi)$ and no variable is updated during the transition.

3. CASE STUDY: AUTOMATED GUIDED VEHICLES

In this section, we describe a flexible manufacturing cell which is a modified version of the cell introduced by Holloway and Krogh (1990). We will show how such a system can be modeled by EFAs to be used as input to the synthesis explained in Section 4. The cell, shown in Fig. 1, consists of three workstations, two input stations and one output station. There also exist five Automated Guided Vehicles (AGVs), each one responsible to route some parts through the cell by following certain paths. Each path has an index number associated to the AGV shown in Fig. 1.

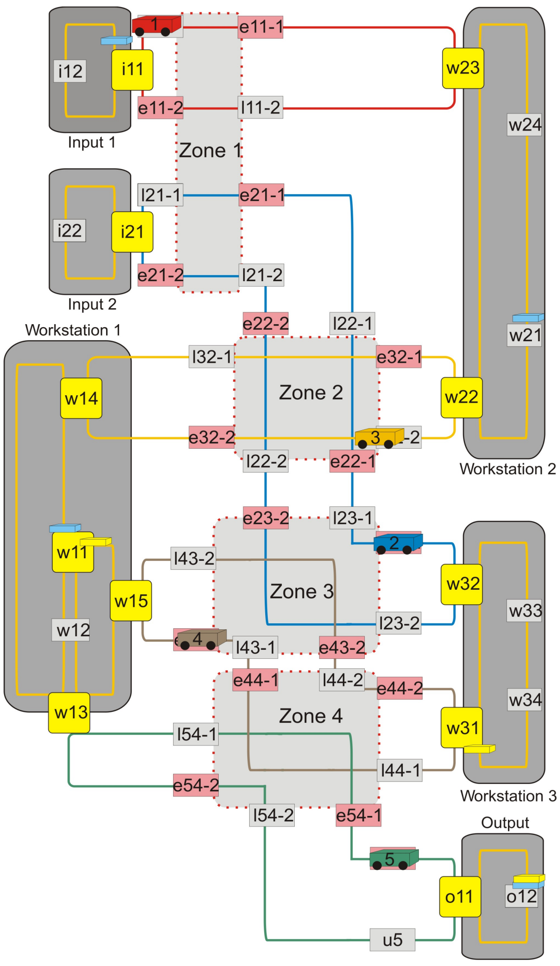

Fig. 1. A flexible manufacturing cell with AGVs transporting parts between different stations.

The control problem is that the routes intersect or are very close to each other and thus there are zones in which no two AGVs are allowed to be at the same time. These zones are shaded light-grey with dotted frames in Fig. 1. We assume that AGV 3 is inside Zone 2 when the system starts operating and the other AGVs are outside the zones. When AGV i *enters* Zone j, event eij-k is fired, where k is used to distinguish the event with other events in case the AGV enters the same zone from different places. For instance, AGV 1 enters Zone 1 from two different places. Entering events eij-k are controllable to the supervisor. Similarly, event lij-k is fired when AGV i leaves Zone j, which is uncontrollable to the supervisor. The input and output stations and workstations have their own events, all of them which are uncontrollable. Furthermore, the exists two other events in the middle of Path 5 labeled by $c5$ (covered by AGV 5 in Fig. 1) and $u5$ that are controllable and uncontrollable, respectively.

3.1 DFA Model

The DFA models of the AGVs, input and output stations and workstations can be considered as plants. The DFA model of AGV 2 is shown in Fig. 2. The DFA models of the other parts can be modeled in an analogous manner.

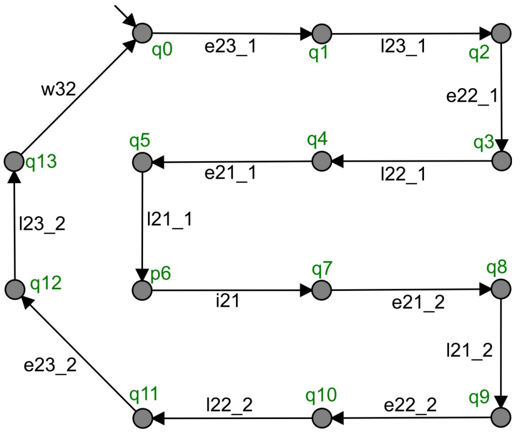

Fig. 2. DFA modeling AGV 2 presented in Fig. 1.

The restriction that at most one AGV can be inside a zone at a time can be considered as a specification, while the AGV paths can be viewed as the plant. Hence, there are four specifications in this system. The automaton modeling Zone 2 is shown in Fig. 3.

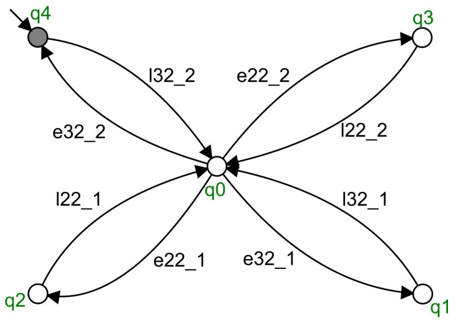

Fig. 3. DFA modeling Zone 2 presented in Fig. 1.

3.2 EFA Model

Since there are not restrictions on the input and output stations and workstations, the EFA models of these parts are equivalent to their corresponding DFA models. The zones are modeled by five integer variables, i.e z_j, with domains $\{0, 1\}$, rather than specification automata. When an AGV wants to enter Zone j, it is checked whether the zone is not occupied by any other AGV, i.e. $z_j > 0$. Since AGV 3 starts from inside Zone 2, the initial value is 0 for z_2 and 1 for the other variables. When the AGV enters and leaves Zone j, z_j will be decremented and incremented by one, respectively. The guards and actions are then attached to the DFA models of the AGVs. A part of the EFA modeling AGV 2 is shown in Fig. 4.

If $n \geq 2$ AGVs were allowed to enter Zone j, it is sufficient to extend the domain of variable z_j to $\{0, \ldots, n\}$. For the DFA case, the automaton for Zone j must be modified extensively, depending on the value of n. This shows one of the advantages of modeling using EFAs compared to DFAs.

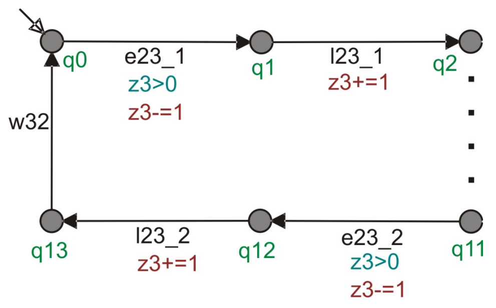

Fig. 4. EFA modeling AGV 2 presented in Fig. 1.

4. SUPERVISOR COMPUTATION AND REPRESENTATION

In this section, we briefly explain the supervisory control theory and the results of applying it to the AGV example. Furthermore, we modify the AGV example to face the uncontrollability problem and show how the new supervisor can be represented by a simple modification of the plant EFAs.

4.1 Supervisory Control Theory

Supervisory Control Theory (SCT) Ramadge and Wonham (1989); Cassandras and Lafortune (2008) is a general theory to automatically synthesize supervisors based on some given plant(s) and specification(s). Specifications describe the allowed and inhibited behaviors. A *supervisor* restricts the conduct of plants to guarantee that the system never violates the given specifications.

In SCT, some states of an automaton A can be considered as *marked states* Q_m^A, which are the states that are desired to be reached from the initial state. The set of marked states of a composed automaton $A_1 \parallel A_2$ is $Q_m^{A_1} \times Q_m^{A_2}$. In addition, some states are specified as *explicitly forbidden* Q_x^A, which are states that should not be reached from the initial state. The set of forbidden states of a composed automaton $A_1 \parallel A_2$ is $Q_x^{A_1} \times Q^{A_2} \cup Q^{A_1} \times Q_x^{A_2}$. Furthermore, the events are divided into two disjoint subsets: *controllable events*, denoted by Σ_c, that can be prevented from executing by the supervisor; and *uncontrollable events*, denoted by Σ_u, that can be executed independently from the supervisor Ramadge and Wonham (1989); Cassandras and Lafortune (2008).

In supervisory synthesis, a plant P can be described by the synchronization of a number of sub-plants $P = P_1 \parallel P_2 \parallel \ldots \parallel P_l$, and similarly for a specification $Sp = Sp_1 \parallel Sp_2 \parallel \ldots \parallel Sp_m$. A first candidate of the supervisor is the composed automaton $S_0 = P \parallel Sp$. After the synthesis procedure, some states, referred to as *forbidden states*, are identified as *blocking* and *uncontrollable*, which should be excluded from S_0 in order to obtain the *safe states*, i.e. the states belonging to the supervisor.

To be able to handle large systems, we use the symbolic representations of the models by Binary Decision Diagrams (BDDs) Akers (1978), powerful data structures for representing Boolean functions. In Miremadi et al. (2010b), it is explained how the EFAs can be represented by BDDs and how the full synchronous operator for EFAs can be represented symbolically. The synthesis computations is then carried out on the BDDs using fixed point computations described in Vahidi et al. (2006).

The AGV example mentioned in Section 3 is both non-blocking and controllable. The supervisor consists of 25731072 ($\sim 2.5 \times 10^7$) states which is, in this case, equal to $|Q_{ac}^{P\|Sp}|$. In other words, all the events are allowed to occur from the reachable states in $P \parallel Sp$.

4.2 Guard Generation

Recall that the supervisor influences the plant models by preventing them to execute some events in their current states in order to avoid violations of the given specifications.

Concerning the states that are retained or removed after the synthesis process, the states that enable an arbitrary event σ can be divided into three state sets called *basic state sets*: *forbidden state set*, *allowed state set* and *don't-care state set*.

The forbidden state set, denoted by $Q_{\mathbf{f}}^{\sigma}$, is the set of states in the supervisor where the execution of σ is defined for S_0, but not for the supervisor. The allowed state set, denoted by $Q_{\mathbf{a}}^{\sigma}$, is the set of states in the supervisor where the execution of σ is defined for the supervisor. In other words, for each event σ in S_0's alphabet, $Q_{\mathbf{a}}^{\sigma}$ represents the set of states where event σ *must* be *allowed* to be executed in order to end up in states belonging to the supervisor (an analogous argument can be given for $Q_{\mathbf{f}}^{\sigma}$).

In order to obtain compact and simplified guards, inspired from the Boolean minimization techniques, we determine a set of states where executing σ will not impact the result of the synthesis and utilize these states to minimize the guards.

Consequently, for a given event σ, the states that can impact the supervisor are only the states where σ *must* be allowed, $Q_{\mathbf{a}}^{\sigma}$, or forbidden, $Q_{\mathbf{f}}^{\sigma}$, to occur and the remaining states can be considered as don't-care. For formal definitions of the state sets and proofs, see Miremadi et al. (2010a).

Based on the basic state sets, some logic restrictions can be extracted, expressing under which conditions the events can be executed without violating the specifications.

For an event σ and a state $(q_u^{A_1}, q_v^{A_2}, \ldots, q_w^{A_M}) \in Q^{S_0}$, the following propositional function $G^{\sigma} : Q^{A_1} \times Q^{A_2} \times \ldots \times Q^{A_N} \rightarrow \mathbb{B}$, referred to as *guard*, is desired:

$$G^{\sigma}(q^{A_1}, q^{A_2}, \ldots, q^{A_N}) = \begin{cases} \text{true} & (q^{A_1}, q^{A_2}, \ldots, q^{A_N}) \in Q_{\mathbf{a}}^{\sigma} \\ \text{false} & (q^{A_1}, q^{A_2}, \ldots, q^{A_N}) \in Q_{\mathbf{f}}^{\sigma} \\ \text{don't care} & \text{otherwise} \end{cases}$$

where $\mathbb{B}$ is the set of Boolean values and q^{A_i} represents the current state of automaton A_i. In particular, σ is allowed to be executed from the state $(q^{A_1}, q^{A_2}, \ldots, q^{A_N})$ if the guard is true.

By applying minimization methods of Boolean functions (utilizing the don't-care state set) and some heuristic techniques, a more simplified guard can be obtained, denoted by $\mathcal{G}$. This procedure is defined in details in Miremadi et al. (2010a).

4.3 Uncontrollability in the AGV Example

To introduce some changes in the AGV system, let event $e23$-2 in Fig. 1 (upper part of Zone 3) be uncontrollable. For the DFA model, it is sufficient to only change the event's property. However, for the EFA model, since there do not exist any specifications (the zone specifications are replaced by guards using integer variables), we have to apply a small modification to the plant model. In general, when the uncontrollability property is going to be modeled on an EFA plant, the following rule can be applied:

$$\forall\ (p \xrightarrow{\sigma}_{g/a} q) \mid \sigma \in \Sigma_u :$$
$$\text{add } (p \xrightarrow{\sigma}_{\neg g} q_x) \text{ to the set of partial transitions.}$$

where Σ_u is the set of uncontrollable events, q_x is a new explicitly forbidden state added to the plant. The rule holds if the σ is a local event.

By looking at Fig. 4, we can observe that an uncontrollability problem will arise if $e23\_2$ is fired from location $q11$ when $z3 <= 0$. Hence, we can model this by adding a transition from location $q11$ to a forbidden location by a guard $z3 <= 0$. The new EFA is shown in Fig. 5.

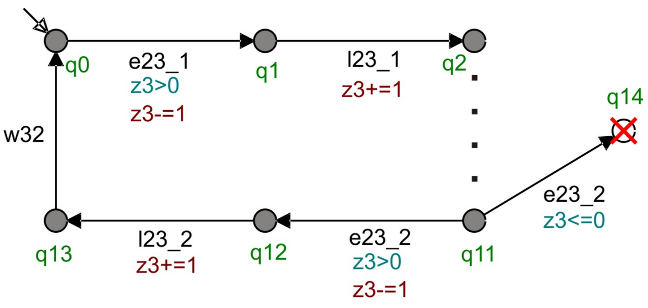

Fig. 5. EFA modeling AGV 2 including uncontrollability.

The supervisor for this model consists of 24993792 ($\sim 2.5 \times 10^7$) states. The EFA model has 27376128 reachable states where 2382336 ($\sim 2.4 \times 10^6$) of those are blocking. Representing the supervisor's states explicitly will not be possible and hard for the user to understand. Therefore, we apply the approach by Miremadi et al. (2010b), which is a way of representing the supervisor by symbolically generating guards and attaching them to the original EFAs.

By applying the mentioned approach to this example, two guards are generated for three different events:

$$\mathcal{G}^{e43\text{-}1} = \mathcal{G}^{e43\text{-}2} : \quad q^{AGV2} \neq q10$$
$$\mathcal{G}^{e22\text{-}2} : \quad z_3 == 1$$

The first guard says that AGV 2 should not be in state $q10$ when events $e43$-1 and $e43$-2 are going to be fired. By observing Fig. 2, this means that event $e22$-2 should not have been fired before. This can also be concluded from Fig. 1. If $e22$-2 is fired, then it is not possible to prevent AGV 2 of entering Zone 3 because $l22$-2 and $e23$-2 are uncontrollable events. Similarly for the second guard, if $e22$-2 is going to be fired, it should be checked whether Zone 3 is empty, i.e. $z_3 == 1$. In the next step, these guards are attached to the transitions in the plant EFAs where the mentioned events are included.

In this way, we have restricted the plant EFAs to end up in more than 2 million blocking states by merely two simple guards. The guards were generated in approximately three seconds.

Worth to mention that the program is implemented in JAVA programming language using Supremica libraries Åkesson et al. (2006, 2003), which uses JavaBDD (avilable online) as the BDD package. The example was conducted on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3GB RAM) running Windows XP.

5. CONCLUSIONS

In this paper, we have shown how to model a flexible manufacturing cell with AGVs by DFA and EFA models. Furthermore, for the uncontrollable version of the cell, the synthesis results were given. It was shown that two simple guards are sufficient to restrict the model to enter around 2 million forbidden states. Since the entire procedure is based on BDDs, the computations were carried out efficiently.

Consequently, the controller designers remain within the same scope, i.e. EFA models, during the modeling phase. Since designing a controller is typically an iterative process, this is considered as a significant feature of the approach. In essence, the users can make their modifications to the supervisors on the same type of models. [1]

[1] This work was carried out within the Wingquist Laboratory VINN Excellence Centre at Chalmers University of Technology.

REFERENCES

Akers, S. B., Jun. 1978. Binary decision diagrams. IEEE 27, 509–516.

Andersson, K., Richardsson, J., Lennartson, B., Fabian, M., 2010. Coordination of operations by relation extraction for manufacturing cell controllers. IEEE Trans. on Control Systems Technology 18 (2), 414–429.

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H., Franklin, G. F., 1993. Supervisory control of a rapid thermal multiprocessor. IEEE Transactions on Automatic Control 38 (7), 1040–1059.

Cassandras, C. G., Lafortune, S., 2008. Introduction to Discrete Event Systems, 2nd Edition. Springer.

Chen, Y.-L., Lin, F., Sep. 2000. Modeling of discrete event systems using finite state machines with parameters. In: Proceedings of the IEEE International Conference on Control Applications, CCA'00. pp. 941–946.

Feng, L., Wonham, W. M., Thiagarajan, P. S., 2007. Designing communicating transaction processes by supervisory control theory. Form. Methods Syst. Des. 30 (2), 117–141.

Gaudin, B., Deussen, P. H., 2006. Supervisory control on concurrent discrete event systems with variables (extended version). Tech. rep., Technical University, Berlin.

Giua, A., Jul. 1992. Petri nets as discrete event models for supervisory control. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York, USA.

Giua, A., DiCesare, F., Apr 1994. Blocking and controllability of petri nets in supervisory control. IEEE Transactions on Automatic Control 39 (4), 818–823.

Harel, D., 1987. Statecharts: A visual formalism for complex systems. Science of Computer Programming 8, 231–274.

Hellgren, A., Fabian, M., Lennartson, B., 1999. Synchronized execution of discrete event models using sequential function charts. In: 38th Decision and Control. Phoenix AZ, USA.

Hellgren, A., Lennartson, B., Fabian, M., 2002. Modelling and plc-based implementation of modular supervisory control. In: Proceedings of the Sixth International Workshop on Discrete Event Systems, WODES'02. pp. 371–376.

Hoare, C. A. R., 1985. Communicating sequential processes. Series in Computer Science. Prentice-Hall.

Holloway, L. E., Krogh, B. H., 1990. Synthesis of feedback control logic for a class of controlled Petri nets. IEEE Transactions on Automatic Control 35 (5), 514–523.

JavaBDD, avilable online.
URL `http://javabdd.sourceforge.net`

Åkesson, K., 2002. Methods and tools in supervisory control theory: operator aspects, computation efficiency and applications. Ph.D. thesis, Signals and Systems, Chalmers University of Technology, Göteborg, Sweden.

Åkesson, K., Fabian, M., Flordal, H., Malik, R., Jul. 2006. Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proceedings of the 8th international Workshop on Discrete Event Systems, WODES'08. Ann Arbor, MI, USA, pp. 384–385.

Åkesson, K., Fabian, M., Flordal, H., Vahidi, A., 2003. Supremica—a tool for verification and synthesis of discrete event supervisors. In: 11th Mediterranean Conference on Control and Automation. Rhodos, Greece.

Miremadi, S., Akesson, K., Fabian, M., Vahidi, A., Lennartson, B., May 2008. Solving two supervisory control benchmark problems using supremica. In: Proceedings of the 9th International Workshop on Discrete Event Systems, WODES'08. pp. 131–136.

Miremadi, S., Åkesson, K., Lennartson, B., 2010a. Symbolic computation of reduced guards in supervisory control, submitted to IEEE Transactions on Automation Science and Engineering.

Miremadi, S., Åkesson, K., Lennartson, B., 2010b. Symbolic supervisory synthesis on extended finite automata, submitted to IEEE Transactions on Control Systems Technology.

Ramadge, P., Wonham, W., Jan 1989. The control of discrete event systems. Proceedings of the IEEE 77 (1), 81–98.

Sköldstam, M., Åkesson, K., Fabian, M., Dec 2007. Modeling of discrete event systems using finite automata with variables. In: Proceedings of the 46th IEEE Conference on Decision and Control. pp. 3387–92.

Supremica, available online. `www.supremica.org`. The official website for the Supremica project.

Vahidi, A., Fabian, M., Lennartson, B., Oct. 2006. Efficient supervisory synthesis of large systems. Control Engineering Practice 14 (10), 1157–1167.

Voronov, A., Akesson, K., Aug. 2009. Verification of process operations using model checking. In: Proceedings of the 5th IEEE International Conference on Automation Science and Engineering, CASE'09. pp. 415–420.

Yang, Y., Gohari, R., Aug. 2005. Embedded supervisory control of discrete-event systems. In: Proceedings of the IEEE International Conference on Automation Science and Engineering, CASE'05. pp. 410–415.