ON THE SUPERVISORY CONTROL FOR STATE TRAJECTORY SPECIFICATIONS IN TIME-VARYING DISCRETE-EVENT SYSTEMS

I. Romanovski * **M. Guay** * **K. Rudie** **

* *Department of Chemical Engineering, Queen's University, Kingston, Ontario, Canada K7L 3N6*
** *Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada K7L 3N6*

Abstract:
In this paper, the Supervisory Control in Time-Varying Discrete-Event Systems is considered when the plant specifications are formulated as State Trajectory Specifications. The procedure for construction of on-line supervisors in this case is given and an example of re-using of the computed supervisor is given.

Keywords: Discrete-Event Systems, Time-Varying Automata, Supervisory Control, State Trajectory Specifications

1. INTRODUCTION

Systems in the areas of manufacturing, telecommunications, and transportation are often represented by networks of interacting objects modelled by Discrete-Event Systems (DES). Due to the inherent complexity of many physical networks, and, as a result, exponential growth of the state space, the analysis and control of such systems often gives rise to structural and computational problems of enormous complexity.

The essential requirements for the construction of DES supervisors are (i) the existence of a finite deterministic automaton (Ramadge and Wonham, 1987) that represents the model of a system and (ii) the existence of a finite deterministic automaton that represents the desired behavior (or, specification) of the system. However, it may be extremely difficult, if not impossible, in practice, to obtain an automaton that completely describes the future behavior of the modelled process (see (Chung *et al.*, 1992)). Moreover, the modelled process, as well as the desired behavior, may have dynamical components that would not allow us to get a full description as an automaton (Lin, 1993).

In the theory of supervisory control of DES (Ramadge and Wonham, 1987) several methods were developed to minimize the computational and modelling complexity of a DES supervisor construction. These methods include variable lookahead policies (Chung *et al.*, 1992), hierarchical control (Zhong and Wonham, 1990), vertical state aggregation (Hubbard and Caines, 1999), dynamical consistency based state aggregation, (Shen and Caines, 2002) horizontal state aggregation, (Romanovski and Caines, 2001), (Romanovski and Caines, 2002) and decentralized control (Cieslak *et al.*, 1988), (Rudie and Wonham, 1992).

The standard model for the interaction of parallel subsystems is that of the synchronous product (Wonham, 2000), (also called *parallel composition* in (Cassandras and Lafortune, 1999)) where the computation of a supervisor is performed off-line based on the complete information about the model and the desired behavior of an automaton (Fig. 1, top scheme). However, due to the above observations, such computation is not always possible. We would like to propose a new supervisory control scheme that permits the computation of supervisors in situations where models and specifications are time-varying (Fig. 1, bottom scheme). We call such structures *time-varying discrete-event systems*. Switching logic may occur due to various reasons: device breakdown, appearance of a new device, new information, and so on. However, the mechanism for logic switching is unmodeled. We intentionally leave this out of the scope of the current work because we wish to focus on supervisor design in the face of switches rather than on the cause of switches or on the transitional behavior during a switch.

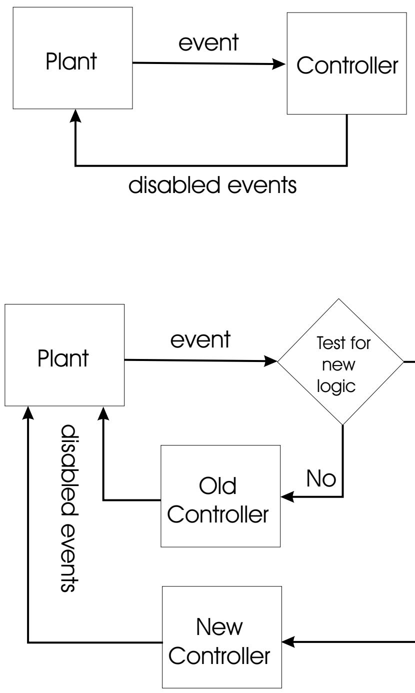

Fig. 1. The current control scheme (top) and the new control scheme (bottom)

2. ONLINE COMPUTATION OF SUPERVISORS FOR STATE TRAJECTORY SPECIFICATIONS

Natural specifications for manufacturing, transportation and telecommunications systems are often formulated in terms of the existence of safe transitions between members of a specified ordered sequence of states (with possible constraints on visiting other system states) regardless of the event sequence by which this is achieved. In this section we introduce an algorithmic procedure for computing the supervisor for such specifications.

Definition 1. ((Romanovski and Caines, 2001)) A *State Trajectory Specification* ($SPEC$) for a given automaton G is a 4-tuple of subsets of X, namely, $SPEC = \{X_I, X_T, X_{pc}, X_{bad}\}$, where $X_{pc} \cap X_{bad} = \emptyset$. The set X_I is termed the set of initial states (of the $SPEC$), X_T is termed the set of terminal states, X_{pc} is an *ordered* subset of X (possibly with repetitions) termed the set of ports of call and X_{bad} is termed the set of bad states.

The interpretation is that X_{pc} is the set of states which should be visited in a given order while X_{bad} is the set of states which must be avoided. Furthermore, unless otherwise stated, X_I and X_T are singletons ($\{x_I\}$ and $\{x_T\}$, respectively).

The term to *drive a state x (of an automaton) to a state y* signifies that there exists an input word of controllable and uncontrollable events u such that when the automaton is in the state x and accepts the word u the automaton terminates in state y. Equivalently, y is reachable from x via an input sequence $u \in \Sigma^*$.

Definition 2. We say that an automaton $G = (X, \Sigma, \delta, X_o, X_m)$ satisfies the $SPEC = \{x_I, x_T, X_{pc}, X_{bad}\}$ if and only if for any initial automaton state x_o there exists a system trajectory that satisfies all of the following:

(1) The initial automaton state x_o is driven to the state x_I without entering the set X_{pc};
(2) The state x_I is driven to state x_T along a trajectory which contains all the elements of X_{pc} in the specified order;
(3) The trajectory from x_o to the x_T does not pass through any state in the set of potentially bad states,

where a potentially bad state is a state in X_{bad} or a state which can be driven to a bad state by a sequence of uncontrollable events (or from which a bad state is reachable by a sequence of uncontrollable events). The set of potentially bad states is denoted by X_{pbad}.

For any given $SPEC$ and automaton G we denote by $L(SPEC)_G$ the set of all legal strings (or, trajectories) $v \in L(G)$ that satisfy the definition above (for the formal definition of $L(SPEC)_G$ see (Romanovski and Caines, 2002)).

2.1 State Trajectories with Bad States Only

First, we consider a special class of specifications for which $X_I = X_T = X_{pc} = \emptyset$; we denote this class by $\mathcal{K}$. In other words, if $SPEC \in \mathcal{K}$, for trajectory $v \in L(G)$ we have that $v \in L(SPEC)_G \iff v$ does not pass through any state from X_{pbad}, that is, we must only avoid potentially bad states. Below we present the algorithm for calculation of potentially bad states for a given automaton and specification.

Algorithm 3.1 Computes the set of potentially bad states

Inputs: $G = (X, \Sigma, \delta, x_o, X_m)$,
$SPEC = \{x_I, x_T, X_{pc}, X_{bad}\}$.
$X_{pbad} \leftarrow X_{bad}$
for all $(x', u, x'') \in (X - X_{pbad}) \times \Sigma_{uc} \times X_{pbad}$ **do**
$X_{pbad} \leftarrow X_{pbad} \cup \{x'\}$
end for
Output: X_{pbad}

Note that if $\Sigma_{uc} = \emptyset$ then $X_{pbad} = X_{bad}$.

Proposition 3. For any automata $G = (X, \Sigma, \delta, x_o, X_m)$ and a specification $SPEC \in \mathcal{K}$, G satisfies $SPEC$ if and only if $x_o \notin X_{pbad}$.

Proof. Follows from Definition 2 and Algorithm 3.1.

In the rest of the paper we assume that $x_o \notin X_{pbad}$, that is, $L(SPEC)_G \neq \emptyset$ for $SPEC \in \mathcal{K}$.

Let us consider a collection of automata $\mathbf{G} = \{G(0), \ldots G(N) \ldots\}$ and legal specification K represented as a state trajectory specification $SPEC \in \mathcal{K}$. We start with the synthesizing of $S(0)$ using Algorithm 3.1 as follows:

(1) Compute X^0_{pbad} for input $G(0) = (X_0, \Sigma_0 = \Sigma_{c_0} \cup \Sigma_{uc_0}, \delta_0, x_{o_0}, X_{m_0})$ and $SPEC$;
(2) For any $x \in X_0$ set

$$S(0)[x] =$$

$$\Sigma_{uc_0} \cup \{a \in \Sigma_{c_0},\ s.t.\ \delta_0(x, a) \notin X^0_{pbad}\} \quad (1)$$

Let $L^0(SPEC)$ be the language realized by $S(0)$ acting on $G(0)$.

Lemma 2.1. $v \in L^0(SPEC) \iff \delta^*_0(x_{o_0}, v) \notin X^0_{pbad}$.

Proof. We use induction on the length of v.

$\Longrightarrow$ Since the initial state, x_0, does not belong to X^0_{pbad}, we have that $\delta^*_0(x_{o_0}, \epsilon) \notin X^0_{pbad}$. Assume that $\delta^*_0(x_{o_0}, v) \notin X^0_{pbad}$ for any $v \in L^0(SPEC)$ for which $|v| \leq M$, and consider $va \in L^0(SPEC)$. If $a \in \Sigma_{uc_0}$ and $\delta^*_0(x_{o_0}, va) \in X^0_{pbad}$, then, by the definition of X^0_{pbad} we must have that $\delta^*_0(x_{o_0}, v) \in X^0_{pbad}$; this fact contradicts the inductive hypothesis. If $a \in \Sigma_{c_0}$, we have that $\delta^*_0(x_{o_0}, va) \notin X_{pbad}$ by the construction of $S(0)$.

$\Longleftarrow$ Similarly, if $\delta^*_0(x_{o_0}, \epsilon) \notin X_{pbad}$, we have that $\epsilon \in L^0(SPEC)$ by the construction of $S(0)$. Assume $\delta^*_0(x_{o_0}, v) \notin X_{pbad} \Longrightarrow v \in L^0(SPEC)$ for any $|v| \leq M$. Set $\delta^*_0(x_{o_0}, v) = x$. Then, by the construction of $S(0)$ for any $a \in \Sigma_0$ such that $\delta_0(x, a)$ is defined, we have that

$$\delta_0(x, a) \notin X^0_{pbad} \Longrightarrow va \in L^0(SPEC)$$

Proposition 4. $L^0(SPEC)$ is the supremal controllable sublanguage of $L(SPEC)_{G(0)}$ w.r.t. $G(0)$ (i.e., $L^0(SPEC) \subseteq L(SPEC)_{G(0)}$ and for any L' with $L^0(SPEC) \subset L'$ we have that if $L' \subseteq L(SPEC)_{G(0)}$, then L' is uncontrollable).

Proof. By definition (see (Wonham, 2000)), $L^0(SPEC)$ is controllable if and only if for any $v \in L^0(SPEC)$ and $a \in \Sigma_{uc_0}$ such that $va \in L(G(0))$, we have that $va \in L^0(SPEC)$. If $v \in L^0(SPEC)$ and $a \in \Sigma_{uc_0}$ such that $va \in L(G(0))$, we have that $\delta^*_0(x_{o_0}, va) \notin X^0_{pbad}$, otherwise $\delta^*_0(x_{o_0}, v) \in X^0_{pbad}$ and, as a result, $v \notin L^0(SPEC)$ by Lemma 2.1. Thus, $va \in L^0(SPEC)$, and $L^0(SPEC)$ is controllable w.r.t. $L(G(0))$. Let now $v \in L(G(0)$ and $v \in L(SPEC)$. Then $\delta^*_0(x_{o_0}, v) \notin X^0_{pbad}$ by Algorithm 3.1, and as a result, $v \in L^0(SPEC)$ by Lemma 2.1, so that $L^0(SPEC)$ is indeed a supremal controllable sublanguage of $L(SPEC)_{G(0)}$ w.r.t. $G(0)$.

Once we detect that logic of the plant has changed to $G(i)$ for some $i = 1, 2, 3, \ldots, N, \ldots$, we run Algorithm 3.1 with the new input $G(i)$, compute X^i_{pbad} and set

$$S(i)[x] = \Sigma_{uc_i} \cup \{a \in \Sigma_{c_i},\ s.t.\ \delta_i(x, a) \notin X^i_{pbad}\} \quad (2)$$

We assume that once we are at state x no uncontrollable events are added to x when the plant logic is changed to $G(i)$. This way we guarantee that $x \notin X^i_{pbad}$.

Corollary 5. $S(i)$ realizes the supremal controllable sublanguage $L^i(SPEC)$ of $L(SPEC)_{G(i)}$ w.r.t. $G(i)$.

From Proposition 4 and Corollary 5 we see that at each point in time the supervisors constructed according to (1) and (2) guarantee that for the current plant, the minimally restrictive behavior of $SPEC$ is generated.

2.2 State trajectories with x_T

Here we consider the $SPECs$ of the type $(\emptyset, \emptyset, x_T, X_{bad})$ for the collection of automata $\mathbf{G}$. The class of such specifications is denoted by $\mathcal{K}_1$.

We assume that for every index $j = 0, 1, \ldots N \ldots$ and $SPEC' = (\emptyset, \emptyset, \emptyset, X_{bad})$, $L^j(SPEC') \neq \emptyset$ there is at least one index i such that $L^i(SPEC) \neq \emptyset$, and our aim is to produce the scheme that synthesizes the supervisor for $L^i(SPEC)$ for any such i.

In order to do that, we must find the set of states $X_i' \subseteq X_i$ through which all possible trajectories from x_{o_0} to x_T go. First, we run Algorithm 3.1 for $G(i)$ to get X^i_{pbad}. Then we compute all states that are reachable from x_{o_i}, denoted by $X_i(x_{o_0}) \subseteq X_i - X^i_{pbad}$ and all states from which x_T can be reached, denoted by $X_i(x_T) \subseteq X_i - X^i_{pbad}$. Clearly, $X_i' = X_i(x_{o_0}) \cap X_i(x_T)$

Algorithm 3.2 Computes $X_i(x_{o_0})$.
Inputs: $G(i)$, X^i_{pbad}, $SPEC$
$X_i(x_{o_0}) \leftarrow \{x_{o_0}\}$
for all $(x', u, x'') \in X_i(x_{o_0}) \times \Sigma_i \times (X - X^i_{pbad})$
do
$X_i(x_{o_0}) \leftarrow X_i(x_{o_0}) \cup \{x''\}$
end for
Output: $X_i(x_{o_0})$

Algorithm 3.3 Computes $X_i(x_T)$.
Inputs: $G(i)$, $X^i_{pbad}\}$, $SPEC$
$X_i(x_T) \leftarrow \{x_T\}$
for all $(x', u, x'') \in (X - X^i_{pbad}) \times \Sigma_i \times X_i(x_T)$
do
$X_i(x_T) \leftarrow X_i(x_T) \cup \{x'\}$
end for
Output: $X_i(x_T)$

Now, some of these trajectories may not be safe, i.e., for some $x \in X_i'$, there may exist an uncontrollable $u \in \Sigma_{uc_i}$ such that $\delta_i(x, u) \notin X_i'$, so we must exclude these states.

Algorithm 3.4 Computes the X_i'': states of safe trajectories from x_{o_i} to x_T.
Inputs: $G(i)$, X_i'.
$X_{unsafe} \leftarrow \emptyset$
for all (x', u, x'') such that
$(x', u, x'') \in (X_i' - X_{unsafe}) \times \Sigma_U \times ((X - X_i')$
$\cup X_{unsafe})$ **do**
$X_{unsafe} \leftarrow X_{unsafe} \cup \{x'\}$
end for
Output: $X_i'' \leftarrow X_i' - X_{unsafe}$

It is clear that if $x_o \in X_{unsafe}$ or $x_T \in X_{unsafe}$, there are no safe trajectories from x_o to x_T. If it is not the case, we need to modify $G(i)$ to see if there are trajectories from x_{o_i} to x_T going through safe states only. For each $G(i) = (X_i, \Sigma, \delta_i, x_{o_i}, X_{m_i})$ we form $G'(i)$ as follows:

$$G'(i) = (X_i'', \Sigma, \delta|_{X_i''}, x_{o_i}, X_{m_i})$$

and do over Algorithms 3.2 and 3.3 with input $G'(i)$ to get new $X_i^{safe}(x_{o_i})$ and $X_i^{safe}(x_T)$.

Now we are ready to define $S(i)$:

$$S(i)[x] = \Sigma_{uc_i} \cup$$

$$\{a \in \Sigma_{c_i},\ s.t.\ \delta_i(x, a) \in X_i^{safe}(x_{o_0}) \cap X_i^{safe}(x_T)\}.$$

Note that when $x_T \notin X_i$, or when $X_i^{safe}(x_{o_0}) \cap X_i^{safe}(x_T) = \emptyset$, we just wait, i.e., disable everything we can, until the logic of the plant changes (in which case, there is a possibility that for some $m > i$ the plant has evolved so that $G(m)$ will lead to $X_m'' \neq \emptyset$). Since each $G(i)$ satisfies $(\emptyset, \emptyset, \emptyset, X_{bad})$, we will never visit a potentially bad state by uncontrollable string.

Next, we show how the general $SPEC = \{x_I, x_T, X_{pc}, X_{bad}\}$ can be represented as a sequence of specifications from class $\mathcal{K}_1$.

Since we are considering the case where the initial and terminal states are singletons ($\{x_I\}$ and $\{x_T\}$), we can represent any $SPEC$ as a pair $\{X_{pc}, X_{bad}\}$ with the singletons $\{x_I\}$ and $\{x_T\}$ included in the X_{pc} as the first and the last state, respectively.

In (Romanovski and Caines, 2002) it was proven that the automaton G satisfies the specification $SPEC = \{X_{pc}, X_{bad}\}$ if and only if

$$SPEC^0 = \{< x_I >, X_{bad} \cup X_{pc} - \{x_I\}\}$$

is satisfied by

$$G^0 = (X, \Sigma, \delta, x_o, x_I);$$

and

$$SPEC^j = \{< x_j >, X_{bad} \cup X_{pc} - \{x_j\}\}$$

is satisfied by

$$G^j = (X, \Sigma, \delta, x_{j-1}, x_j),\ j = 1, \ldots n+1,$$

where $x_0 = x_I$, $x_{n+1} = x_T$ and n is the number of ports of call. In fact, since by the construction of G^j the initial state of the current specification is x_{j-1}, we can return to a 4-tuple representation of $SPEC^j$ as $\{\emptyset, x_j, \emptyset, X_{bad} \cup X_{pc} - \{x_{j-1}, x_j\}\} \in \mathcal{K}_1$. That allows us to formalize the scheme that synthesizes $S(i)$ for $SPEC = \{X_{pc}, X_{bad}\}$ as follows:

(1) For each $SPEC^j \in \mathcal{K}_1$ we synthesize $S^j(i)$ that realizes $L^i(SPEC^j)$ for $G(i)$.
(2) Then,
$$S(0)[x_{o_0}] = S^0(0),$$
and for each $i = 1, 2, 3 \ldots,\quad 0 \leq j \leq n+1$
$$S(i)[x] = \begin{cases} S^j(i), & until\ \ x \neq x_j \\ S^{j+1}(i), & if\ \ x = x_j \\ \Sigma_{uc_i} & if\ \ x = x_T \end{cases}$$

3. EXAMPLE

This example is adapted from (Rudie *et al.*, 1994). Consider a production line which consists of a conveyer belt and a series of identical machines that work in parallel to process video cassettes. Each machine is designed to wind a recorded tape

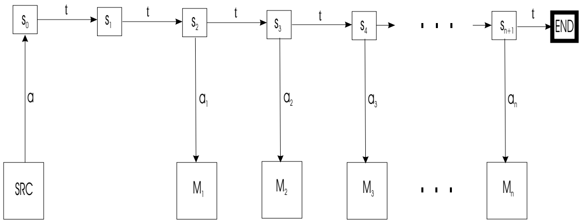

Fig. 2. Belt and N machines

onto empty cassettes. Cassettes are to be taken from the belt located above the machines, and there is a special machine that is responsible for placing empty cassettes (one at a time) on the conveyer belt. A detailed description of the line can be found in (Rudie *et al.*, 1994).

We assume that the maximum number of cassettes that can be in any machine at once is 3. Also, We define the following time constants. Let T_s be the time it takes for a video cassette to reach the first machine and T_b the time it takes for a cassette to travel from one machine to the next. As in (Rudie *et al.*, 1994), $T_s = 10$ seconds and $T_b = 5$ seconds. The minimum processing time, T_p, is equal to 25 seconds. We model a basic time unit as an uncontrollable event t with the assumptions that $T_s = 2t$ and $T_b = t$.

The model of the conveyer belt with N machines is given in Figure 2. In this figure, controllable event a denotes a cassette being dropped on the conveyer belt and each controllable event a_i denotes the machine M_i grabbing a cassette from the belt above the machine. The belt can be modelled by an automaton, and state of this automaton can be completely described by a binary string $[s_0, s_1, \ldots, s_{n+1}]$, where $s_i \in \{0, 1\}$, $s_i = 1$ when there is a cassette in the spot s_i on the belt and $s_i = 0$ otherwise. There is also a special state called "END". Transition function δ_{belt} is defined as follows:

(1) First,
$$\delta_{belt}([s_0, s_1, \ldots, s_{n+1}], t) =$$
$$\begin{cases} [s'_0, s'_1, \ldots, s'_{n+1}] & if\ s_{n+1} \neq 1, \\ END & otherwise \end{cases}$$
where $s'_0 = 0$, $s'_i = s_{i-1}$.

(2) Further, an event a_i, $i = 1, \ldots N$, is defined whenever $s_{i-1} = 1$ and $\delta_{belt}([s_0, s_1, \ldots, s_{i-2}, 1, s_i, \ldots, s_{n+1}], a_i) = [s_0, s_1, \ldots, s_{i-2}, 0, s_i, \ldots, s_{n+1}]$.

(3) Finally, event a is defined whenever $s_0 = 0$ and $\delta_{belt}([0, s_1, \ldots, s_{n+1}], a) = [1, s_1, \ldots, s_{n+1}]$.

The automaton modelling machine M_i is given in Figure 3. Each state in this automaton corresponds to the number of cassettes currently in the machine. The uncontrollable event b_i represents the end of the processing and must occur after T_p or more time units when the machine is in state 1, 2 or 3.

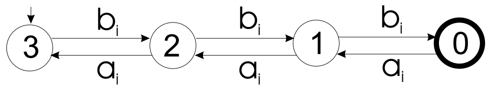

Fig. 3. Machine M_i

Following (Rudie *et al.*, 1994), we need to ensure that (i) no machine will ever be idle for lack of empty cassettes to process, (ii) no input stack will exceed its capacity of three cassettes, and (iii) no cassette will go beyond the last machine in the row without being grabbed from the overhead conveyor belt.

For $N = 2$ this problem was solved in (Rudie *et al.*, 1994). Here we denote by A_n the automaton that represents the production line with n machines processing the cassettes and consider the above specifications for a set of finite deterministic automata $\{A_2, \ldots A_n \ldots\}$. The change of a model from A_i to A_j can be caused by various reasons (e.g., by adding one or more new machines, from the breakdown of a working machine, and so on).

We model the first and the third specifications as state trajectory specifications for the belt and for each machine M_i respectively, namely, $SPEC_{belt} = \{\emptyset, \emptyset, \emptyset, X_{bad}^{belt}\}$, where $X_{bad}^{belt} = \{END\}$, and $SPEC_{M_i} = \{\emptyset, \emptyset, \emptyset, X_{bad}^{M_i}\}$, where $X_{bad}^{M_i} = \{0\}$. The second specification is modelled by construction of the automata for M_i, namely, there is no transition a_i from state 3.

Here, due to the actual physical implementation of controlled events a and a_i, we consider them to be not only controllable, but *forcible* (see (Rudie *et al.*, 1994)), that is, it is possible to force the occurrence of events a or a_i before the next occurrence of t or b_i. Without this assumption our specifications would be impossible to realize, since there is a string of uncontrollable events that leads to a bad state from the initial state for each machine as well as for the belt.

It is shown in (Rudie *et al.*, 1994) that when $N = 5$ the number of states in the automaton representing the whole system could be over 319 million. Clearly, it is impossible to compute a supervisor directly each time a new machine is added to the system, for example. However, due to the nature of our specification, the supervisor can appear as a set of supervisors for each machine, which command, for each state of the belt, for a_i to be disabled or forced. For example, for the last machine, M_N, the event a_N must be forced whenever $s_{N+1} = 1$. We will explain how the supervisor for the A_i can be used for A_{i+1}, that is, when a new machine is added.

Let S_i^{old} be a supervisor for machine M_i in an automaton A_N, S_0^{old} be a supervisor for the machine that controls the cassette being put on the belt. Suppose that a new machine M' is added between machines M_j and M_{j+1} in A_N. Then, for A_{N+1}, $i = 2, \ldots, N+1$,

$$S_i^{new} = \begin{cases} S_i^{old} & if \quad i > j, \\ S_{i-1}^{old} & if \quad i \leq j. \end{cases}$$

That is, if the machine appeared in a row before M_i, S_i^{new} disables or forces a_i for any string $[s_0, s_1, \ldots, s_{n+2}]$ if and only if a_i was disabled or, respectively, forced by S_i^{old} for the string $[s'_0, s'_1, \ldots, s'_{n+1}]$, where $s'_i = s_{i+1}$, $i = 0, \ldots, n$. And if the new machine appeared in a row after (or at the place of) M_i, S_i^{new} disables or forces a_i for any string $[s_0, s_1, \ldots, s_{n+2}]$ if and only if a_i was disabled or, respectively, forced by S_{i-1}^{old} for the string $[s'_0, s'_1, \ldots, s'_{n+1}]$, where $s'_i = s_{i+1}$, $i = 0, \ldots, n$.

To see this, it is enough to show that a decision of forcing or disabling each a_i at the string (belt's state) $[s'_0, s'_1, \ldots, s'_{n+1}]$, where the information for how many units of time the cassette currently being processed in each machine is given, depends only on the position of M_i in the row; namely, we disable a_i if and only if there is a machine in a row after M_i, say, M_{i+k} that would lead to a state "0" after kt seconds if we would force a_i. For example, when $N = 2$, at the string 1010, when in machine M_2 there is only one cassette processed for $4t$ seconds, we need to disable a_1, as it was computed in (Rudie *et al.*, 1994), since otherwise it is possible that M_2 finishes processing the cassette and there is no cassette on the belt to grab, so M_2 may enter state "0". Similarly, we force the event a_i only if there is a possibility of that machine M_i reaching the state "0", or there is a possibility that this cassette could not be grabbed by any machine further in the row since they all packed, so the belt reaches the state "END". But in both cases, if any of these possibilities exist at the string $[s'_0, s'_1, \ldots, s'_{n+1}]$ for machine M_i, then this possibility will remain the same for a new string $[s_0, s_1, \ldots, s_{n+2}]$, since it is obtained from the old string by adding 0 or 1 at the beginning! Thus, for example, if new machine M_j is added between M_1 and M_2 in A_2, we need to disable a_j at strings 01010 and 11010 by the same reasons we disabled a_1 in A_2. We conclude that we leave the supervisor for M_i unchanged if the new machine is added at the j^{th} place in the row and $j < i$, and we use the old supervisor for M_{i-1} for each M_i where $i = 1, \ldots, j$.

Thus, when a new machine is added, we need to recompute only S_0 and S_1, and use the old supervisors for the rest of the machines by the scheme described above.

4. CONCLUSIONS

A new scheme is presented that allows us to compute on-line supervisors for state trajectory specifications. This approach is especially useful when it is impossible to form a union of time-varying automata. In future work, the mechanisms for the plant logic evolution will be formalized to consider methods of constructing "hierarchical" supervisors and to invoke new methods for model-predictive control.

REFERENCES

Cassandras, C.G. and S. Lafortune (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers. New York.

Chung, S.L., S. Lafortune and F. Lin (1992). Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. on Automatic Control* **37**, 1921–1935.

Cieslak, R., C. Desclaux, A.Fawaz and P.Varaiya (1988). Supervisory control of discrete event processes with partial observation. *IEEE Trans. on Automatic Control* **33**, 249–260.

Hubbard, P. and P.E. Caines (1999). Initial investigations of hierarchical supervisory control for multi-agent systems. *Proceedings of the 38th IEEE CDC* **3**, 2218–2223.

Lin, F. (1993). Robust and adaptive supervisory control of discrete event systems. *IEEE Trans. on Automatic Control* **38**, 1848–1852.

Ramadge, P.J. and W.M. Wonham (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control Optimization* **25**, 206–230.

Romanovski, I. and P.E. Caines (2001). On vector trajectory specifications for multi-agent product systems. *Proceedings of the 40th IEEE CDC* **1**, 2333–2334.

Romanovski, I. and P.E. Caines (2002). Multi-agent products and trajectory specifications in supervisory control theory. *Proceedings of the 2002 ACC* **1**, 722–723.

Rudie, K. and W.M. Wonham (1992). Think globally, act locally: Decentralized supervisory control. *IEEE Trans. on Automatic Control* **37**, 1692–1708.

Rudie, K., N. Shimkin and S.D. O'Young (1994). Timed discrete-event systems: A manufacturing application. *Proceedings of the CISS* **1**, 374–381.

Shen, G. and P. E. Caines (2002). Hierarchically accelerated dynamic programming for finite state machines. *IEEE Trans. on Automatic Control* **47**, 271–283.

Wonham, W.M. (2000). Notes on control of discrete event systems, http://www.control.utoronto.ca/cgi-bin/dldes.cgi.

Zhong, H. and W.M. Wonham (1990). On the consistency of hierarchical supervision in discrete-event systems. *IEEE Trans. on Automatic Control* **35**, 1125–1134.