On Computation of State Avoidance Control for Infinite State Systems in Assignment Program Framework *†

Ratnesh Kumar
Dept. of Elec. & Comp. Eng.
Iowa State Univ., Ames, IA

Vijay K. Garg
Dept. of Elec. & Comp. Eng.
Univ. of TX at Austin, TX

Abstract

In this paper we study supervisory control of discrete event systems with potentially infinite state-space using state variables for representation and specification. An *assignment program* model consisting of state variables and a finite set of conditional assignment statements is used for representing a discrete event system, and a predicate over state variables is used for representing a state avoidance control specification. The contribution of this paper is to show how to perform supervisory control computations symbolically. In the case of a Petri net (vector addition system) with the set of forbidden states being a right-closed set, we present a finitely terminating algorithm for maximally permissive supervision.

Keywords: Discrete event systems, Assignment Programs, Supervisory control, State avoidance, Petri nets

1 Introduction

Algorithms for computation of supervisors for discrete event systems have been well developed for plants with *finitely* many states and specifications given as *regular* languages of event traces. There has been little work on synthesis of supervisors for general systems which have infinitely many states except for some classes of Petri nets. In this paper we present a state variable approach to supervisory control of such systems, which allows a concise description of an infinite state system.

Prior work on control of infinite state systems has mainly been in the framework of Petri nets. This includes the work on various classes of controlled Petri net [6], and the equivalent vector discrete event systems [11]. In this paper we study the synthesis problem for general infinite state systems modeled using assignment programs. This model was first used in

*A preliminary version of this paper appeared in [2, 8].

†This research was supported in part by the National Science Foundation under the grants ECS-9709796, ECS-0099851, ECS-0218207, ECS-0244732, ECS-9907213, and CCR-9988225, a ONR grant N000140110621, a Texas Education Board grant ARP-320, an Engineering Foundation Fellowship, and an IBM Grant.

[10] to obtain a necessary and sufficient condition for the existence of a supervisor, and to provide a formula for a maximally permissive supervisor. In this paper, we study the issue of algorithmic computation of the supervisors.

We consider the problem of safety control formulated as a state avoidance problem and specified as a predicate over state variables. Using predicate transformers, we obtain a supervisor using symbolic computations. This computation is in general undecidable (as expected), which however, only indicates the non-existence of a general purpose automated solution technique. Special case analysis can be carried out for obtaining the solution, and we illustrate the methodology using several examples.

An interesting special case arises when the system is a Petri net, and the set of forbidden states is a right-closed set, i.e., if a certain Petri net marking is forbidden, then any other marking *covering* it is also forbidden. The importance of this class of sets has been recognized by others in the Petri-net community. See for example [16] where the structure of a right-closed set is studied and is shown that it can be represented as a finite union and intersection of sets expressed using linear inequalities. We show that the supervisory control problem for state avoidance is *decidable* for such systems and provide an algorithmic technique to compute a maximally permissive supervisor. The approach we have taken is similar in spirit to that in [4] on recursive equivalence of the liveness and reachability problems for the Petri nets.

This decidability result generalizes the existing results on controlled Petri nets where supervisory synthesis for state avoidance of right-closed markings (typically specified as the simultaneous marking of certain combination of places) has been solved for various restricted class of Petri nets such as a subclass of marked graphs [6], a subclass of state machines [1], and recently a more general class of Petri nets satisfying the "transition conflict (TC)" condition [5]. Our result also generalizes the existing results on controlled vector discrete event systems, which are precisely the controlled Petri nets, where the problem has been solved under the restriction of "loop-freedom" in the uncontrolled subnet, and the set of forbidden states specified as a "linear-inequality predicate", which is a right-closed set (see [11, 17]). It also generalizes the results based on place invariance techniques presented in [18] where the set of forbidden states is also specified as a linear inequality predicate—a right-closed set, and sufficient conditions are given for the existence of a supervisor, and the supervisor computed is not necessarily maximally permissive.

An expanded version of this paper, which includes all omitted proofs, as well as more examples is available on the World Wide Web [9].

2 Notation and Preliminaries

A discrete event system, denoted G, is a 4-tuple $G := (X, \Sigma, \rightsquigarrow, X_0)$, where X denotes the state set, Σ is the finite event set, $\rightsquigarrow \subseteq X \times \Sigma \times X$ is the set of state transitions, and $X_0 \subseteq X$ is the set of initial states. We use state variables to represent the states and a finite set of conditional assignment statements to represent the state transitions. The desired behavior is specified as a predicate over the state variables representing a set of forbidden states.

The notation v is used to denote the vector of state variables of G. If v is n-dimensional, then $v = [v_1, \ldots, v_i, \ldots, v_n]$, where v_i is the ith state variable. The state space X of G equals the Cartesian product of domains of all state variables, i.e., $X := \prod_{i=1}^{n} \mathcal{D}(v_i)$, where $\mathcal{D}(v_i)$ is the domain of v_i. By definition $\mathcal{D}(v_i)$ is a countable set and can be identified with the set of natural numbers $\mathcal{N}$.

We use *predicates* for describing various subsets of the state space. Let $\mathcal{P}(v)$ denote the collection of predicates defined using the state variable vector v, i.e., if $P(v) \in \mathcal{P}(v)$, then it is a boolean valued map $P(v) : X \to \{0, 1\}$. Consider for example a two dimensional state space $X = \mathcal{Z}^2$. Then the predicate $P(v) = [v_1 \geq v_2]$ refers to all the states in which the value of variable v_1 is at least as large as the value of variable v_2. The symbols $true$ and $false$ are used for denoting predicates that hold on all and none of the states respectively. With every predicate $P(v) \in \mathcal{P}(v)$, we associate a set $X_P \subseteq X$ on which $P(v)$ takes the value one. Thus the collection of predicates $\mathcal{P}(v)$ has a one-to-one correspondence with the power set 2^X, and the names predicates and state-sets can be used interchangeably. We say that the predicate $P(v)$ holds on $\hat{X} \subseteq X$ if $\hat{X} \subseteq X_P$.

Given $P(v) \in \mathcal{P}(v)$, its *negation* is denoted by $\neg P(v)$. Given an indexing set Λ such $P_\lambda(v) \in \mathcal{P}(v)$ for each $\lambda \in \Lambda$, the *conjunction* and *disjunction* over Λ is denoted by $\bigwedge_{\lambda \in \Lambda} P_\lambda(v)$ and $\bigvee_{\lambda \in \Lambda} P_\lambda(v)$ respectively. The quadruple $(\mathcal{P}(v), \neg, \wedge, \vee)$ forms a boolean algebra that is isomorphic to the algebra of the subsets of X with the operations of complementation, intersection, and union. For predicates $P(v)$ and $Q(v)$, we say that $P(v)$ is *stronger* than $Q(v)$ (equivalently, $Q(v)$ is *weaker* than $P(v)$), denoted $P(v) \preceq Q(v)$, if $P(v) \wedge Q(v) = P(v)$, or equivalently, $P(v) \vee Q(v) = Q(v)$. It is easily verified that $(\mathcal{P}(v), \preceq)$ is a *complete Boolean lattice*.

State transitions map a state to another state. Such mappings are extended to set of states or predicates in a natural way, and are known as *predicate transformers*. We use $\mathcal{F}$ to denote the collection of all predicate transformers, i.e., if $f \in \mathcal{F}$, then $f : \mathcal{P}(v) \to \mathcal{P}(v)$. The *conjunctive closure* of f, denoted f_*, and *disjunctive closure* of f denoted f^* is defined to be $\bigwedge_{i \geq 0} f^i$ and $\bigvee_{i \geq 0} f^i$ respectively, where f^0 is the identity predicate transformer and $f^{i+1} := f(f^i)$. Given $f : X \to X$, the *substitution predicate transformer* "$v := f(v)$" maps a predicate $P(v)$ to $P(f(v))$.

Next we review the assignment program model, first introduced in [10], for a discrete event system G described above. The initial state set of G is specified as an *initial predicate*, denoted $I(v)$, which implies $X_0 = X_I$. The state transitions "$\rightsquigarrow$" of G is specified using a finite set of conditional assignment statements of the form:

$$\sigma : [C_\sigma(v)] \Rightarrow [v \rightsquigarrow f_\sigma(v)],$$

where $\sigma \in \Sigma$ is an event, $C_\sigma(v)$ is a predicate, called the *guard*, and $f_\sigma : X \to X$ is a map defined on the state space. If no guard is present, then $true$ is treated as the guard. A conditional assignment statement of the above type is *enabled* if the condition $C_\sigma(v)$ holds. An enabled assignment statement may *execute*. Upon execution, new values are assigned to the state variables according to the map f_σ and a state transition on the event σ occurs. For simplicity, we assume that if multiple assignment statements are simultaneously enabled

only one of them nondeterministically executes at any time. This assumption may be relaxed to allow *concurrency* of execution.

The substitution predicate transformer can be used to define the *forward one-step reachable*, fr, and *backward one-step reachable*, br, predicate transformers for G. fr determines the "postcondition" after the occurrence of a state transition for a given "precondition", whereas br determines the "precondition" prior to the occurrence of a state transition for a given postcondition. [1]

For the assignment statement $\sigma : [C_\sigma(v)] \Rightarrow [v \rightsquigarrow f_\sigma(v)]$ and a condition $P(v)$, these are formally defined as follows:

$$fr(P(v), \sigma) := C_\sigma(f_\sigma^{-1}(v)) \wedge P(f_\sigma^{-1}(v)); \quad br(P(v), \sigma) := C_\sigma(v) \wedge P(f_\sigma(v)).$$

Note that the computation of br is easier as compared to that of fr, since its computation does not require the extra computation of f^{-1}.

For $\hat{\Sigma} \subseteq \Sigma$, we define $fr(P(v), \hat{\Sigma}) := \bigvee_{\sigma \in \hat{\Sigma}} fr(P(v), \sigma)$, and similarly, $br(P(v), \hat{\Sigma}) := \bigvee_{\sigma \in \hat{\Sigma}} br(P(v), \sigma)$. Finally, note that $fr^*(P(v), \hat{\Sigma})$ denotes the set of states which are reachable from a state in $P(v)$ by execution of zero or more transitions of events in $\hat{\Sigma}$. Similarly, $br^*(P(v), \hat{\Sigma})$ denotes the set of states from where a state in $P(v)$ can be reached by execution of zero or more transitions of events in $\hat{\Sigma}$. Clearly, fr^* is useful in characterizing the *forward reachability*, whereas br^* is useful in characterizing the *backward reachability*.

Let $B(v) \in \mathcal{P}(v)$ be a *forbidden predicate*, i.e., it specifies the set of forbidden states. A supervisor S is designed which restricts the behavior of G by disabling some of its controllable events at its various states so that the controlled system G_S does not visit any of the forbidden states. We use $\Sigma_u \subseteq \Sigma$ to denote the set of *uncontrollable* events; the events in $\Sigma - \Sigma_u$ are *controllable*. Then S is a map that assigns to every event $\sigma \in \Sigma$ a predicate $S_\sigma(v)$ where σ is disabled by S. Since uncontrollable events cannot be disabled, we require $S_\sigma(v) = false$ for each $\sigma \in \Sigma_u$. Each assignment statement of the controlled system then takes the form:

$$\sigma : [C_\sigma(v) \wedge \neg S_\sigma(v)] \Rightarrow [v \rightsquigarrow f_\sigma(v)].$$

Remark 1 Note that in our setting concurrency of state transitions is not allowed. This is restrictive compared to the setting in controlled Petri nets [6]. An advantage of this restriction is that a unique maximally permissive supervisor exists. (In the setting of controlled Petri nets a unique maximally permissive supervisor need not exist as demonstrated in [3].)

Since the effect of supervision is to strengthen the guard for each conditional assignment statement in G, it is clear that S restricts the behavior of G. We use fr_S and br_S to denote the associated predicate transformers of G_S. The control task is to design S such that

$$fr_S^*(I(v), \Sigma) \preceq \neg B(v). \tag{1}$$

[1] Earlier references such as [14, 10] used wp (resp., sp) instead of br (resp., fr) for backward (resp., forward) one-step reachability. This, however, was a poor notation since the predicate transformers wp and sp are used to mean something different in computer science literature: For example, $wp(P(v), \hat{\Sigma})$ denotes the set of initial states from where the program consisting of assignment statements in $\hat{\Sigma} \subseteq \Sigma$ terminates and reaches states where $P(v)$ holds.

Remark 2 A special case of this problem with $I(v) = \neg B(v)$ was first studied in [14] where it was shown that a supervisor exists if and only if $\neg B(v)$ is Σ_u-*invariant*, i.e., $\neg B(v) \preceq \neg br(B(v), \Sigma_u)$. A related problem was studied in [10, 11] where conditions were derived for the existence of a supervisor S such that $fr^*_S(I(v), \Sigma) = \neg B(v)$, i.e., the inequality of (1) replaced by an equality. Below we present conditions for the problem of (1). It turns out that the existence conditions for the two cases (inequality vs. equality) differ, nonetheless the formula for the maximally permissive supervisor is the same.

Proposition 1 [9] Given a plant G and a forbidden predicate $B(v)$, $[\exists S : fr^*_S(I(v), \Sigma) \preceq \neg B(v)] \Leftrightarrow [I(v) \preceq \neg br^*(B(v), \Sigma_u)]$.

From Proposition 1, the test for the existence of a supervisor requires the computation of $br^*(B(v), \Sigma_u)$. We show below that the same computation is also needed for obtaining a maximally permissive supervisor. (Given two supervisors S^1 and S^2 we say that S^1 is *more permissive* or *less restrictive* than S^2, denoted $S^1 \leq S^2$, if for each $\sigma \in \Sigma$, $S^1_\sigma(v) \preceq S^2_\sigma(v)$. S is said to be the least permissive or the most restrictive supervisor if it disables all the controllable events in all the states.)

Let $\mathcal{S}$ be the family of supervisors such that for each $S \in \mathcal{S}$, $fr^*_S(I(v), \Sigma) \preceq \neg B(v)$. Then since we impose the requirement of non-concurrency of transition execution, $\mathcal{S}$ contains a maximally permissive supervisor $S^\uparrow$ whenever it is nonempty, given by $S^\uparrow_\sigma(v) := \wedge_{S \in \mathcal{S}} S_\sigma(v)$ for each $\sigma \in \Sigma$. The following theorem gives an explicit characterization of $S^\uparrow$ and can be obtained similar to [14, Corollary 7.1 and Proposition 8.1].

Proposition 2 [9] Given a plant G and a forbidden predicate $B(v)$, if $\mathcal{S} \neq \emptyset$, then a maximally permissive supervisor is given by: $S^\uparrow_\sigma(v) = br(br^*(B(v), \Sigma_u), \sigma)$ for any $\sigma \in (\Sigma - \Sigma_u)$.

3 Computation of br^*: General Case

The test for the existence of a supervisor as well as the computation of a maximally permissive supervisor (when one exists) require the computation of $br^*(B(v), \Sigma_u)$. The following theorem shows that the computation of $br^*(B(v), \Sigma_u)$ is in general undecidable, and we need systematic methods to decide the termination of the recursive computation in special cases.

Theorem 1 For a plant G, and a forbidden predicate $B(v)$, it is in general undecidable to determine whether a given state is in $br^*(B(v), \Sigma_u)$.

Proof: Our proof is based on the undecidability of the emptiness of a recursively enumerable language, i.e., a language generated by a Turing machine [7, Theorem 8.6]. We show that the emptiness problem of a recursively enumerable language can be *reduced* to the problem of determining membership in $br^*(B(v), \Sigma_u)$ for some G and $B(v)$. Consider a Turing machine T with its event set being Σ_u. Let G be an assignment program that emulates T (this is

possible since an assignment program is Turing equivalent [9, Theorem 1]), and let $B(v)$ be the predicate representing the accepting states of T. Then the language accepted by T is nonempty if and only if it is possible to reach one of its accepting states (by a sequence of uncontrollable event transitions) from the initial state of T, i.e., if and only if the initial state of T is in $br^*(B(v), \Sigma_u)$. Since the problem of determining the emptiness of the language accepted by T is in general undecidable, it follows that it is undecidable to determine whether the initial state of T is in $br^*(B(v), \Sigma_u)$. ■

Remark 3 Note that Theorem 1 provides the expected result that generally the computation of maximally permissive supervisor cannot be achieved in an automated fashion. A similar result has been derived for the case of Petri nets when both the plant and the specification are given as general Petri nets [15]. Theorem 1 illustrates that for systems for which the *language emptiness* problem is undecidable, the existence (and computation) of a supervisor for state avoidance control is also undecidable. This complements the result of Sreenivas [15] which shows that for systems for which *language containment* problem is undecidable, the existence (and computation) of a supervisor for language avoidance control is undecidable.

Undecidability of the computation, however, only indicates the non-existence of a general purpose automated computation technique. Analysis for special cases may be carried out for the computation of $br^*(B(v), \Sigma_u)$. We first discuss the case when a single uncontrollable event is present and later generalize this to the case of multiple uncontrollable events.

3.1 Single Uncontrollable Event

Note when there is a single uncontrollable event σ, then $br^*(B(v), \Sigma_u) = br^*(B(v), \sigma) = [\exists n \in \mathcal{N} : br^n(B(v), \sigma)]$. In the following theorem we give a formula for $br^n(B(v), \sigma)$ which is used to compute $br^*(B(v), \sigma)$.

Theorem 2 For all $n \geq 1$, $br^n(B(v), \sigma) = B(f_\sigma^n(v)) \bigwedge_{i \leq n-1} C_\sigma(f_\sigma^i(v))$.

Proof: The proof is based on induction on n. The assertion trivially holds for $n = 1$, establishing the base step. For induction step assume it is true for $n \leq k$. Then

$$
\begin{aligned}
& br^{k+1}(B(v), \sigma) \\
&= br\left(br^k(B(v), \sigma), \sigma\right) \\
&= br\left(B(f_\sigma^k(v)) \bigwedge_{i \leq k-1} C_\sigma(f_\sigma^i(v)), \sigma\right) \\
&= [B(f_\sigma^k(f_\sigma(v))) \bigwedge_{i \leq k-1} C_\sigma(f_\sigma^i(f_\sigma(v)))] \wedge C_\sigma(v) \\
&= [B(f_\sigma^{k+1}(v)) \bigwedge_{i \leq k-1} C_\sigma(f_\sigma^{i+1}(v))] \wedge C_\sigma(v) \\
&= [B(f_\sigma^{k+1}(v)) \bigwedge_{1 \leq i \leq k} C_\sigma(f_\sigma^i(v))] \wedge C_\sigma(f^0(v)) \\
&= B(f_\sigma^{k+1}(v)) \bigwedge_{i \leq k} C_\sigma(f_\sigma^i(v)),
\end{aligned}
$$

where the second equality follows from the induction hypothesis. This establishes the induction step and completes the proof. ■

The following example illustrates the application of Theorem 2.

Example 1 Suppose $B(v) = [v = m]$, $C_\sigma(v) = [v < m]$, and $f_\sigma(v) = v + 1$, which implies $f_\sigma^n(v) = v + n$. Then using the formula of Theorem 2 we obtain: $br^n(B(v), \sigma) = [v = m - n]$. Hence $br^*(B(v), \sigma) = [v \leq m]$.

The formula in Theorem 2 requires multiple conjunctions. However, as shown in the following corollary the formula is much simplified when f_σ satisfies certain conditions with respect to $C_\sigma(v)$.

Definition 1 A function $f : X \to X$ is said to be *increasing* with respect to a predicate $P(v) \in \mathcal{P}(v)$ if $P(v) \preceq P(f(v))$; it is said to be *decreasing* with respect to $P(v)$ if $P(f(v)) \preceq P(v)$.

Corollary 1 [9] If f_σ is increasing with respect to $C_\sigma(v)$, then $br^*(B(v), \sigma) = B(v) \vee [C_\sigma(v) \wedge \{\vee_{n \geq 1} B(f_\sigma^n(v))\}]$. If f_σ is decreasing with respect to $C_\sigma(v)$, then $br^*(B(v), \sigma) = B(v) \bigvee_{n \geq 1} [C_\sigma(f_\sigma^{n-1}(v)) \wedge B(f_\sigma^n(v))]$. If f_σ is increasing with respect to $C_\sigma(v)$ and decreasing with respect to $B(v)$, or it is decreasing with respect to $C_\sigma(v)$ and $B(v)$, then $br^*(B(v), \sigma) = B(v)$.

We illustrate the above results by applying it to a "linear" system.

Example 2 Assume that $f_\sigma(v) = av + b$, where a, b are constants of appropriate sizes (if $v \in \mathcal{Z}^n$, then $a \in \mathcal{Z}^{n \times n}$ and $b \in \mathcal{Z}^n$). Then using induction on n it is easy to show that

$$f_\sigma^n(v) = \begin{cases} a^n v + (a-1)^{-1}(a^n - 1)b & \text{if } a \neq 1 \\ v + nb & \text{otherwise.} \end{cases}$$

Suppose for example that the assignment f_σ is of the form: $v_1, v_2 := v_1 + v_2, v_1 - v_2$. Then in this case

$$a = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Therefore from the formula above

$$f_\sigma^n(v_1, v_2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^n \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},$$

where it can be verified that

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^n = \begin{cases} \begin{bmatrix} 2^{\frac{n}{2}} & 0 \\ 0 & 2^{\frac{n}{2}} \end{bmatrix} & \text{if } n \text{ even} \\ \begin{bmatrix} 2^{\frac{n-1}{2}} & 2^{\frac{n-1}{2}} \\ 2^{\frac{n-1}{2}} & -2^{\frac{n-1}{2}} \end{bmatrix} & \text{if } n \text{ odd.} \end{cases}$$

In other words,

$$f_\sigma^n(v_1, v_2) = \begin{cases} 2^{\frac{n}{2}} v_1, 2^{\frac{n}{2}} v_2 & \text{if } n \text{ even} \\ 2^{\frac{n-1}{2}} v_1 + 2^{\frac{n-1}{2}} v_2, 2^{\frac{n-1}{2}} v_1 - 2^{\frac{n-1}{2}} v_2 & \text{if } n \text{ odd.} \end{cases}$$

Suppose $B(v_1, v_2) = [v_1 \geq v_2]$ and $C_\sigma(v_1, v_2) = true$. Then from Corollary 1

$br^*((v_1 \geq v_2), \sigma)$
$= \exists n \in \mathcal{N} : B(f_\sigma^n(v_1, v_2))$
$= [\exists n \text{ even } : (2^{\frac{n}{2}} v_1 \geq 2^{\frac{n}{2}} v_2)]$
$\bigvee [\exists n \text{ odd } : (2^{\frac{n-1}{2}} v_1 + 2^{\frac{n-1}{2}} v_2 \geq 2^{\frac{n-1}{2}} v_1 - 2^{\frac{n-1}{2}} v_2)]$
$= [v_1 \geq v_2] \bigvee [(v_1 + v_2) \geq (v_1 - v_2)]$
$= [v_1 \geq v_2] \bigvee [v_2 \geq 0]$.

Thus if the system starts from a state where the last condition does not hold, i.e., where $[v_1 < v_2 < 0]$, then it never reaches a state in $[v_1 \geq v_2]$ on an event sequence of σ.

3.2 Multiple Uncontrollable Events

Note when $\Sigma_u = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, $\Sigma_u^* = (\sigma_1^* \sigma_2^* \ldots \sigma_m^*)^*$. So the reachability computation with respect to traces in Σ_u^* can equivalently be obtained by repeated reachability computation with respect to traces in $\sigma_1^* \sigma_2^* \ldots \sigma_m^*$, which in turn is a m-fold repeated reachability computation, each with respect to sequences of a *single* uncontrollable event. Hence the following iterative computation gives $br^*(B(v), \Sigma_u)$:

- $B_0(v) := B(v)$
- $B_{i+1}(v) := br^*(B_i(v), \sigma_{i+1})$

Then $br^*(B(v), \Sigma_u) = \exists n \in \mathcal{N} : (B_m)^n(v)$.

In the following example we illustrate our technique for an infinite state system that cannot be modeled as a Petri net. This example appears in [13, pp. 194-196] to illustrate the limitation of the modeling power of Petri nets and to show how Petri nets with inhibitor arcs can be used to model the given system.

Example 3 Consider the following producer-consumer problem consisting of two producers and two consumers. Producer $P_i (i = 1, 2)$ produces items for consumer C_i, and upon production it deposits the item in buffer B_i. The transmission of the items from buffer B_i to consumer C_i is via a shared channel which can transmit only one item at a time and gives priority of transmission to items of buffer B_1 (items of buffer B_2 can only be transmitted to consumer C_2 over the channel when B_1 is empty). A Petri net with inhibitor arc model for this system is given in [13, p. 196], which can be translated to an assignment program model as follows. $p_i/c_i/b_i$ denotes number of items at $P_i/C_i/B_i$. Initially, $p_i = c_i = 1$ and $b_i = 0$.

$$
\begin{array}{lll}
prod_1 : & [p_1 > 0] & \Rightarrow \quad [b_1 \rightsquigarrow b_1 + 1] \\
prod_2 : & [p_2 > 0] & \Rightarrow \quad [b_2 \rightsquigarrow b_2 + 1] \\
con_1 : & [b_1 > 0 \wedge c_1 > 0] & \Rightarrow \quad [b_1 \rightsquigarrow b_1 - 1] \\
con_2 : & [b_2 > 0 \wedge c_2 > 0 \wedge b_1 = 0] & \Rightarrow \quad [b_2 \rightsquigarrow b_2 - 1]
\end{array}
$$

first two events are the production events that are controllable and the last two events are the consumption events that are uncontrollable. The control task is to ensure the stability

of the buffers by requiring that the cumulative contents of the two buffers never exceed a fixed number m, i.e., the forbidden states are given by $B(v) = [b_1 + b_2 > m]$.

We proceed with the computation of $br^*(B(v), \Sigma_u)$, where $\Sigma_u = \{con_1, con_2\}$ as follows. We first compute $br^*(B(v), con_1)$. Note that $C_{con_1}(f_{con_1}(v)) = [b_1 - 1 > 0 \wedge c_1 > 0] \preceq [b_1 > 0 \wedge c_1 > 0] = C_{con_1}(v)$, and $B(f_{con_1}(v)) = [b_1 - 1 + b_2 > m] \preceq [b_1 + b_2 > m] = B(v)$, i.e., f_{con_1} is decreasing with respect to both $C_{con_1}(v)$ and $B(v)$. Hence from the third part of Corollary 1, $br^*(B(v), con_1) = B(v) = [b_1 + b_2 > m]$. Similarly it can be shown that f_{con_2} is also decreasing with respect to both $C_{con_2}(v)$ and $B(v)$. So $br^*(br^*(B(v), con_1), con_2) = br^*(B(v), con_2) = B(v) = [b_1 + b_2 > m]$. Hence $br^*(B(v), \Sigma_u) = B(v) = [b_1 + b_2 > m]$. Since $I(v) \preceq [b_1 + b_2 = 0] \preceq \neg br^*(B(v), \Sigma_u) = [b_1 + b_2 \leq m]$, it follows from Proposition 1 that a supervisor that achieves the desired buffer stability exists.

The maximally permissive supervisor can be computed using the result of Theorem 2 as follows:

1. $S_{prod_1}(v) = br(br^*(B(v), \Sigma_u), prod_1) = [b_1 + b_2 \geq m]$
2. $S_{prod_2}(v) = br(br^*(B(v), \Sigma_u), prod_2) = [b_1 + b_2 \geq m]$

Thus the supervisor disables both the production events when the cumulative buffer content is at least the desired maximum.

4 Computation of br^*: Petri net Case

When G is a Petri net, its state space is $X = \mathcal{N}^n$, i.e., it is the set of n-dimensional positive integer valued vectors. We show that in the case of Petri nets the recursive computation of $br^*(B(v), \Sigma_u)$ terminates whenever the set of forbidden states is right-closed.

Definition 2 Given a set of positive integer valued vectors $\hat{X} \subseteq \mathcal{N}^n$, it is said to be *right-closed* if $x \in \hat{X}$ and $x' \geq x$ implies $x' \in \hat{X}$. A predicate $P(v) \in \mathcal{P}(v)$ is said to be right-closed if X_P is a right-closed set.

Thus $\hat{X}$ is right-closed if whenever it contains a state, it contains all states "covering" it. The following well known lemma describes a property of right-closed subsets of positive integer valued vector sets.

Lemma 1 Given a right-closed vector set $\hat{X} \subseteq \mathcal{N}^n$, it contains finitely many minimal elements (with respect to the standard partial ordering of vectors).

In the following lemma we show that for a Petri net plant $br^*(B(v), \Sigma_u)$ is right-closed whenever $B(v)$ is right-closed.

Lemma 2 If the plant G has a Petri net model, and the forbidden predicate $B(v)$ is right-closed, then $br^*(B(v), \Sigma_u)$ is also a right-closed predicate.

Proof: Consider a state x in $br^*(B(v), \Sigma_u)$. Then there exists a sequence u of transitions of uncontrollable events from x to a state $\bar{x}$ in $B(v)$. If x' is a state covering x, i.e., if $x' \geq x$, then from a property of Petri nets the transition sequence u can also occur at x' resulting in a state $\bar{x}'$ that covers $\bar{x}$. Since $B(v)$ is a right-closed predicate, $\bar{x}'$ is in $B(v)$. So we conclude that x' is a state in $br^*(B(v), \Sigma_u)$, proving that it is right-closed. ■

The above two lemmas can be used to obtain the following main result of this section:

Theorem 3 For a Petri net plant G and a right-closed forbidden predicate $B(v)$, the recursive computation of $br^*(B(v), \Sigma_u)$ terminates in a finite number of iterations.

Proof: From Lemma 2, $br^*(B(v), \Sigma_u)$ is right-closed. So from Lemma 1, it contains a finite number of minimal positive integer valued vectors. Let $\hat{X}$ be the set of such minimal vectors. For each $x \in \hat{X}$, define $N_x \in \mathcal{N}$ to be the minimum number of uncontrollable transitions it takes to reach a state in $B(v)$ from state x, and let $N := \max_{x \in \hat{X}} N_x$. Then since N_x is finite for each $x \in \hat{X}$ and since $\hat{X}$ is also finite it follows that N itself is finite. We claim that the recursive computation of $br^*(B(v), \Sigma_u)$ terminates in at most N iterations, i.e., $br^*(B(v), \Sigma_u) = \bigvee_{n \leq N} br^n(B(v), \Sigma_u)$. It suffice to show that given any state x in $br^*(B(v), \Sigma_u)$ it takes fewer than N uncontrollable transition to reach a state in $B(v)$ from x. Since x is a state of $br^*(B(v), \Sigma_u)$, it covers some minimal vector $x' \in \hat{X}$. By definition of N, there exists a transition sequence u of uncontrollable events of length at most N from x' to a state $\bar{x}'$ in $B(v)$. Since x covers x', u can also occur at x resulting in a state $\bar{x}$ that covers $\bar{x}'$. Finally, since $B(v)$ is right-closed, it follows that $\bar{x}$ is in $B(v)$. This completes the proof. ■

Remark 4 Since the computation of $br^*(B(v), \Sigma_u)$ does not depend on the conditional assignment statements corresponding to the controllable events, the result of Theorem 3 can be strengthened to a more general setting where only the *uncontrollable sub-system*, i.e., the system described by the conditional assignment statements corresponding to the uncontrollable events, can be represented as a Petri net.

The proof of Theorem 3 is existential in nature. It does not provide an upper bound on the number of iterations needed for the recursive computation of $br^*(B(v), \Sigma_u)$ to terminate, but it can be seen that it is bounded by the depth of the deepest *coverability tree* [13] associated with the uncontrollable sub-system.

5 Conclusion

We have examined the suitability of using assignment programs to model discrete event systems. The problem of supervisory synthesis for state avoidance is in general infeasible. We believe that techniques need to be developed so that the synthesis can be done with some human assistance. To this end, we have presented certain symbolic techniques. The success of theorem prover softwares such as PVS [12] that can do many types of symbolic computations automatically, confirms that the symbolic computation presented in the paper

can be automated. However, the iterative computation is not guaranteed to terminate in general. We show that when the system can be represented as a Petri net and the forbidden state set is right-closed, such as when it is specified as simultaneous marking of certain combination of places, then the iterative computation does terminate.

References

[1] R. K. Boel, L. Ben-Naoum, and V. Van Breusegem. On forbidden state problems for a class of controlled Petri nets. *IEEE Transactions on Automatic Control*, 40(10):1717–1731, 1995.

[2] V. K. Garg and R. Kumar. State-variable approach for controlling discrete event systems with infinite states. In *Proceedings of 1992 American Control Conference*, pages 2809–2813, Chicago, IL, July 1992.

[3] C. H. Golaszewski and P. J. Ramadge. Supervisory control of discrete event processes with arbitrary controls. In P. Varaiya and A. B. Kurzhanski, editors, *Discrete Event Systems: Models and Applications*, pages 459–469. Springer-Verlag, 1987.

[4] M. H. T. Hack. The recursive equivalence of the reachability problem and the liveness problem for Petri nets and Vector addition systems. In *15th Annual Symposium on Switching and Automata Theory*, pages 156–164, New Orleans, LA, October 1974.

[5] L. E. Holloway, X. Guan, and L. Zhang. A generalization of state avoaidance policies for controlled Petri nets. *IEEE Transactions on Automatic Control*, 41(6):804–816, June 1996.

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

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

[8] R. Kumar and V. K. Garg. Assignment program model for control of infinite state systems. In *Proceedings of 1995 Annual Allerton Conference*, pages 166–175, Urbana, IL, October 1995.

[9] R. Kumar and V. K. Garg. Computation of state avoaidance control for infinite state systems in assignment program framework—the expanded version. Technical report, http://www.eng.iastate.edu/~rkumar/PUBS/cstmttech.ps, 1997.

[10] R. Kumar, V. K. Garg, and S. I. Marcus. Predicates and predicate transformers for supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 38(2):232–247, February 1993.

[11] Y. Li and W. M. Wonham. Control of vector discrete event systems I - the base model. *IEEE Transactions on Automatic Control*, 38(8):1214–1227, August 1993.

[12] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS System guide, Version 2.4. Technical report, SRI International, Melno Park, CA, 2001.

[13] J. L. Peterson. *Petri Net Theory and Modeling of Systems.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.

[14] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[15] R. S. Sreenivas. On a weaker notion of controllability of a language K with respect to to a language L. *IEEE Transactions on Automatic Control*, 38(9):1446–1447, 1993.

[16] G. Stremersch. *Supervision of Petri nets.* Kluwer Academic Publishers, Boston, MA, 2001.

[17] G. Stremersch and R. K. Boel. Reduction of the supervisory control problem for petri nets. *IEEE Transactions on Automatic Control*, 45(12):2358–2363, 2000.

[18] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis. Feedback control of Petri nets based on place invariants. *Automatica*, 32(1):15–28, 1996.