Discrete Event Dynamic Systems: Theory and Applications, 9, 147–169 (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Synthesis of Controllers of Processes Modeled as Colored Petri Nets

MBI MAKUNGU
Département de mathématiques et d'informatique, Université de Sherbrooke, Sherbrooke (Québec) CANADA J1K 2R1

MICHEL BARBEAU michel.barbeau@dmi.usherb.ca
Département de mathématiques et d'informatique, Université de Sherbrooke, Sherbrooke (Québec) CANADA J1K 2R1

RICHARD ST-DENIS richard.st-denis@dmi.usherb.ca
Département de mathématiques et d'informatique, Université de Sherbrooke, Sherbrooke (Québec) CANADA J1K 2R1

Abstract. This paper presents an adaptation of a supervisory control theory and a supervisor synthesis problem to a class of colored Petri nets. More specifically, the forbidden state control problem with full observation, in which a discrete-event system is modeled as a colored Petri net with a symmetry specification, is investigated. This problem is decidable if the colored Petri net has finite color sets and bounded places. A new algorithm for deriving a controller is presented in detail with a proof of correctness. Unlike conventional algorithms that explore the entire reachable set of states, our algorithm avoids an exhaustive search of the state space by exploiting a symmetry specification. It performs particularly well when applied to large but structured processes with similar components. Furthermore, this approach leads to a representation of controllers which are smaller than those obtained with automaton-based approaches.

Keywords: discrete-event systems, supervisory control, colored Petri net, symmetry specification, synthesis

1. Introduction

A control theory has been devised by Ramadge and Wonham for modeling supervised discrete-event systems (DESs) and synthesis of controllers. This theory has been primarily studied in the context of automaton-based models (Ramadge and Wonham, 1987). Automaton-based modeling is, however, cumbersome, particularly in representing large systems consisting of numerous similar interacting components. Even though the computational complexity is polynomial in the number of system states, it grows exponentially with the number of components (Ramadge and Wonham, 1989). This phenomenon, called the *state explosion problem* (Clarke et al., 1994), can be overcame by partitioning the components into a small number of equivalence classes so that all components in a given class are essentially similar. This paper makes a contribution in this direction and presents an algorithm that reduces the complexity of a supervisory control problem under full observation for DESs consisting of large but structured processes with similar behaviors. DESs are modeled as colored Petri nets. The purpose of this algorithm is to rigorously scale up the approach of Ramadge and Wonham.

Recently, Petri nets have begun to receive attention as models for investigating the control of DESs. The Petri net-based models are more powerful than the automaton-based models in that the set of Petri net languages is a superset of regular languages. Furthermore, they allow smaller representations of multiple interacting components than equivalent automaton models. Indeed, a single Petri net structure can describe the behavior of several concurrent components modeled as tokens marking the structure.

One may identify mainly two different approaches. In the first approach, introduced in (Krogh, 1987) and (Ichikawa and Hiraishi, 1988), a DES is described by using a *Controlled Petri net* (CtlP-net). A CtlP-net is an extension of a standard Petri net in which the enabling of transitions can be influenced by external binary control inputs. A tutorial survey of work related to this approach can be found in (Holloway, Krogh, and Giua, 1997). Holloway and Krogh have developed methods for avoiding forbidden markings for a class of Petri nets called cyclic marked graphs (Holloway and Krogh, 1990). Later, Holloway et al. have extended this method to guarantee liveness of the control (Holloway et al., 1996). Ushio has discussed modular control synthesis and presented a necessary and sufficient condition for the unique existence of a maximally permissive feedback (Ushio, 1990). Boel et al. have treated the forbidden state problem for a class of Petri nets called state machines (Boel et al., 1995). Sreenivas has studied the way to enforce liveness via supervisory control in DES modeled as controlled Petri nets (Sreenivas, 1997b).

In contrast, the second approach does not include the concept of control input. The control is obtained by synchronizing the Petri net of the plant with a controller which may be an automaton or an ordinary Petri net. Denham has described a supervisory control theory in which the Petri net of the system is viewed as a two-sorted algebra and the required behavior is specified as Petri net invariants (Denham, 1988). Li and Wonham have extended the notions of controllability and observability to Petri nets which they call vector DES (Li and Wonham, 1993). Giua and DiCesare have examined the synthesis of controllers when the systems are modeled as conservative Petri nets (Giua and DiCesare, 1991a). They have also presented necessary and sufficient conditions for the existence of a Petri net supervisor when both the system and control specifications are given as Petri net languages (Giua and DiCesare, 1991b). Moody and Antsaklis, and Yamalidou et al. have described a method for synthesizing a Petri net controller in the presence of uncontrollable and unobservable transitions (Moody and Antsaklis, 1995; Yamalidou et al., 1996). Their method uses place invariants and is able to enforce logical and algebraic constraints containing elements of the marking and firing vectors. Sreenivas and Krogh have introduced the λ-free nets, a class of Petri nets for which controllability of event languages is decidable (Sreenivas and Krogh, 1992; Sreenivas, 1993). More general results on the decidability of controllability can be found in (Giua and DiCesare, 1995). Finally, Kumar and Holloway have shown that the problem of determining controllability for Petri net languages is reducible to the reachability problem of Petri nets (Kumar and Holloway, 1992).

Several types of control specifications have been applied to synthesis methods for Petri nets, including: avoiding a set of forbidden states (Krogh, 1987; Holloway and Krogh, 1990; Boel et al., 1995); enforcing event language specifications (Giua and DiCesare, 1991b; Kumar and Holloway, 1992); enforcing fairness (Sreenivas, 1997a); enforcing liveness (Holloway et al., 1996; Sreenivas, 1997b); and enforcing real-time control constraints

(Sathaye and Krogh, 1992; Park and Chong, 1995). These synthesis methods can be divided into two classes: methods that explore the state space and methods that examine only the Petri net structure. The main problem with the first class is that the state space can grow exponentially with respect to the size of the Petri net. The limitation of the second class is due to conditions on the net structure, that is, restrictions on the interactions among sub-systems in the modeling process.

Among the Petri net-based models, colored Petri nets (CP-nets) are powerful enough to describe complex systems in a manageable way, particularly when they contain many interacting components that are similar but not identical (Jensen, 1992). It is well-known that the class of CP-nets with finite color sets is equivalent to the class of place/transition nets (PT-nets). The CP-nets describe the same systems as the PT-nets but often in a more compact form and allow the use of equivalence classes to represent similar components. Specifications can be more readable and eventually more tractable. If no restrictions are imposed on the definition of CP-nets, then they are equivalent to Turing machines.

The work presented in this paper addresses the forbidden state control problem for a class of CP-nets in which color sets are finite and behaviors satisfy boundedness properties. Given a control specification expressed as a set of forbidden markings, a procedure computes the unique maximal set of admissible markings and a controller by means of an occurrence graph with symmetries. The symmetries guarantee that equivalent markings have similar behavior (Jensen, 1995). Our aim is not to include, as control logic, additional places and transitions to the CP-net being controlled. Such an approach is generally used when a simulation or a performance analysis is performed for a given control policy (Krogh, 1987). In this work, the process being controlled is separated from the control logic. This approach is suitable for controller synthesis in which the process to be controlled already exists and the aim is to compute, from a control specification, a separated controller which, when embodied with the process in a closed-loop system, satisfies the control specification. Thus, this approach permits formulations and solutions of different control synthesis problems for the same process.

The layout of the paper is as follows. The next section introduces the notation and definitions used in CP-nets. Section 3 extends supervisory control theory to processes modeled by CP-nets with a consistent symmetry specification. Section 4 formulates the control synthesis problem for avoiding a set of forbidden markings. Section 5 describes in detail a new synthesis algorithm and gives a proof of its correctness. Section 6 illustrates its application with an example. Finally, concluding remarks are provided in Section 7.

2. Notation and Preliminaries

The formal definition of CP-nets requires the notion of multi-set. A multi-set is defined as a set in which given elements may appear several times. A multi-set can also be defined as a function mapping each element of a set to a non-negative integer (the number of occurrences of that element). Given a set S, the set of all multi-sets over S is denoted S_{MS}. A multi-set over a set S can be represented by a sum. Each term of the sum consists of a coefficient and an element of S. The former indicates the number of occurrences of the latter. For the sake of simplicity, zero occurrence elements are omitted and the coefficient of a single

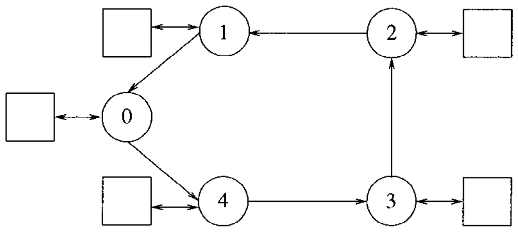


Figure 1. Example of a network.

occurrence element is omitted. For instance, if $S = \{x, y, z\}$, a multi-set in which there are one occurrence of x, three of y, and none of z can be represented as $x + 3`y$.

We use two operations over multi-sets. The union of two multi-sets m_1 and m_2 is represented as $m_1 + m_2$. Element-wise comparison of m_1 and m_2 is represented as $m_1 \leq m_2$ and holds true if m_1 is a sub multi-set of m_2. The expression $|m|$ denotes the number of elements of m.

A CP-net is a many-tuple $(\Sigma, P, T, A, N, C, E, M_0)$, where Σ is a finite set of non-empty types, called *color sets*; P is a finite set of *places*; T is a finite set of *transitions*; A is a finite set of *arcs* connecting places and transitions; N is a *node function* (defined from A to $P \times T \cup T \times P$) that maps each arc to a pair of nodes of different kinds (i.e., one is a place, while the other is a transition); C is a *color function* (defined from P to Σ) that associates a color set to each place; E is an *arc expression function* that maps each arc a, with attached place $p(a)$, to a multi-set of $C(p(a))_{MS}$; and M_0 is the initial marking. It maps each place p to a multi-set over the color $C(p)$ of p. The sets P, T, and A are pairwise disjoint. We consider only CP-nets with decidable properties as valid models for *DESs*, that is, the color sets are finite and contents of places are bounded. Furthermore, for the sake of simplicity, we assume that guards (occasionally used in the definition of CP-nets) always evaluate to true. Hence, guards are not modeled explicitly in this paper.

A token element is a pair (p, c), where $p \in P$ and $c \in C(p)$. Let *TE* be the set of all token elements. The state of a CP-net is given by its current *marking M* which is a multi-set over *TE*. A marking gives a distribution of tokens over the places. The sets of all markings is denoted by $\mathcal{M}$.

Let us consider an example. We assume a network of switches interconnected by point-to-point links. The network contains data sources and sinks, which are senders and receivers of cells, and are called *hosts*. A host can be both a source and a sink. The network supports permanent virtual circuits (PVCs). A PVC exists between two hosts, one with the source role and the other with the sink role, and passes through at least one switch. Every cell belongs to some PVC. All cells in the same PVC follow the same path through the network. For simplicity, we consider simplex PVCs so that a communication actually consists of two one-way PVCs connecting the same two hosts in opposite directions. Figure 1 shows a simple network. Switches are represented by circles numbered from 0 to 4, hosts by squares, and communication links by oriented edges.

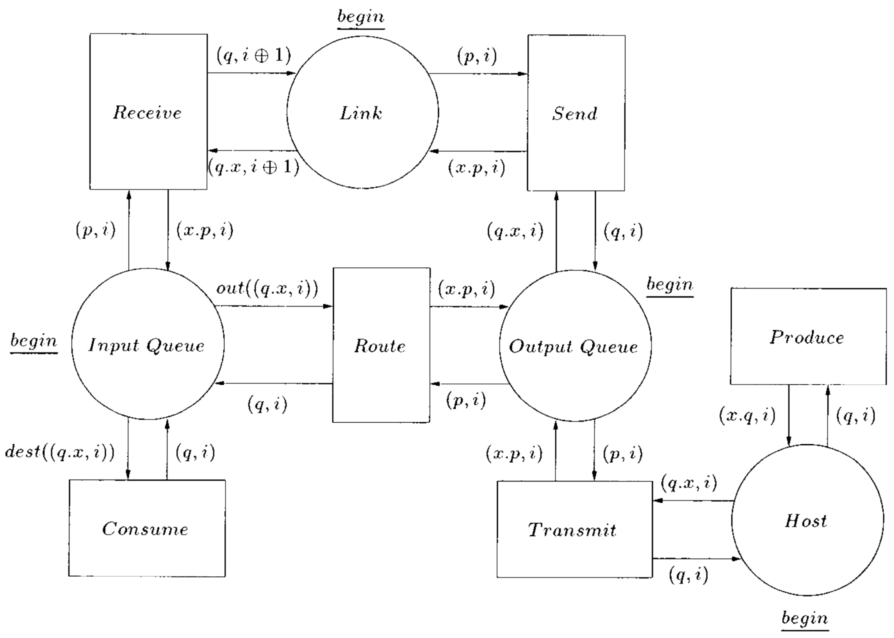


Figure 2. CP-net model of the network in Figure 1.

Each switch has two queues, an input queue and an output queue. The operation of a switch consists of three tasks: receive and add a cell in the input queue, remove the front cell from the input queue and either consume the cell (if destinated to the local host) or add it in the output queue (otherwise), and transmit a cell on a link. Each point-to-point link is also modeled as a queue of cells. Cells queued up for output are transmitted as rapidly as possible. If cells arrive too fast for the switch to process them, the network runs out of memory for arriving cells. When this saturation point is reached, the network is congested.

We model the network operation as a CP-net. Figure 2 shows a CP-net that models the network in Figure 1. Figure 3 shows declarations of token values, operators, functions, and variables used in the model. The constant *n* represents the total number of switches in the network. The constant *begin* represents the initial contents of a place: a multi-set of pairs, a queue contents and the number of the switch that owns the queue. Each switch is represented by a numerical value of type *Switch*. A cell is represented by a numerical value of type *Cell* that gives its destination switch. Each queue of cells is represented by a value of type *Queue* which is the Cartesian product of the *QList* and *Switch* color sets, where the *QList* color set represents a list of *Cell* tokens manipulated as a queue. The concatenation operator "." is used as a constructor of FIFO queues, that is, $x.q$ is a queue with rear cell x and rest q, and $q.x$ is a queue whose front cell is x and rest is q. The operator "$\oplus$" denotes modulo *n* addition.

val $n = 5$;	
val $begin = (\epsilon, 0) + (\epsilon, 1) + (\epsilon, 2) + (\epsilon, 3) + (\epsilon, 4)$;	
color	$Switch = 0 \ldots n - 1$;
color	$Cell = 0 \ldots n - 1$;
color	$QList =$ **list** $Cell$;
color	$Queue =$ **product** $QList \times Switch$;
$\epsilon \in QList$:	empty list;
.:	binary concatenation operator on $Cell$;
$\oplus$:	addition modulo n operator;
var	x: $Cell$;
	p, q: $QList$;
	i: $Switch$;
fun $dest((q.x, i)) =$ **if** $i = x$ **then** $(q.x, i)$;	
fun $out((q.x, i)) =$ **if** $i \neq x$ **then** $(q.x, i)$;	

Figure 3. Declaration of elements used in the CP-net model.

In Figure 2, switches, hosts, and links are represented by places (circles): *Input Queue* for the switch input queues, *Output Queue* for the switch output queues, *Host* for the host output queues, and *Link* for the communication links. Every place contains token values from the color set *Queue*. Each token value models a queue of a switch, host, or link. The host (link) number corresponds to the switch to which the host is attached (link source switch). For example, the token value $(q, 1)$ contained in place *Output Queue* models the state of the output queue of switch 1 with contents q. We assume that each queue has a fixed capacity of five cells.

Actions are represented by transitions (rectangles). The transition *Produce* represents production of cells by hosts. Consumption of cells is modeled as the transition *Consume*. The operations of a switch are represented by four transitions: the transition *Receive* for receiving incoming cells from an adjacent switch through a communication link, transition *Transmit* for receiving incoming cells from an attached host, transition *Route* for making routing decisions, and transition *Send* for sending outgoing cells. We have the following arc expressions: $dest((q.x, i))$ means that transition *Consume* is enabled if host i is the destination of cell x (i.e., $i = x$) and $out((q.x, i))$ means that transition *Route* is enabled if host i is not the destination of cell x (i.e., $i \neq x$). In the latter case, cell x is in transit and moved to the output queue. Initially, each place contains five empty queues represented by the constant *begin*; there are no cells in the network. This CP-net is represented as the many-tuple of Figure 4.

The definition of the behavior of CP-nets is based on the concepts of *enabling* and *occurrence* (Jensen, 1992). For all $t \in T$, let $Var(t)$ be the set of *variables* appearing on arcs that have t as source or destination. For instance, $Var(Consume) = \{i, q, x\}$. The type of a variable v is denoted as *Type(v)*. Let the function $A(p, t)$ be the set of arcs from place p to transition t, that is, $A(p, t) = \{a \in A | N(a) = (p, t)\}$. For all $(p, t) \in P \times T$, let $E(p, t) = \sum_{a \in A(p,t)} E(a)$ be the sum of the expressions on arcs from

$$\begin{array}{rcl}
\Sigma & = & \{Switch, Cell, QList, Queue\} \\
P & = & \{Input\ Queue, Output\ Queue, Host, Link\} \\
T & = & \{Produce, Consume, Receive, Transmit, Route, Send\} \\
A & = & \{(Input\ Queue, Consume), (Consume, Input\ Queue), \\
 & & (Input\ Queue, Route), \ldots\} \\
N(a) & = & a, \text{for all } a \text{ in } A \\
C(p) & = & Queue, \text{for all } p \text{ in } P \\
E(a) & = & \begin{cases} dest((q.x, i)) & \text{if } a = (Input\ Queue, Consume) \\ (q, i) & \text{if } a = (Consume, Input\ Queue) \\ \ldots \end{cases} \\
M_0(p) & = & begin, \text{for all } p \text{ in } P
\end{array}$$

Figure 4. The CP-net of Figure 2 represented as a many-tuple.

place p to transition t. In our example, $A(p, t) = \{(p, t)\}$ if $(p, t) \in A$, because between any given pair of nodes there is at most one arc in every direction. Hence, for instance, $E(\mathit{Input\,Queue}, \mathit{Consume}) = \mathit{dest}((q, x, i))$. $E(t, p)$ is defined in a similar fashion.

A *binding* b of a transition t is a substitution, $\langle v_1 = c_1, \ldots, v_n = c_n\rangle$, that assigns a color c_i (also called token) to every variable $v_i \in \mathit{Var}(t)$ $(1 \leq i \leq n)$. It is required that c_i is of the color set of v_i. The set of all bindings for a transition t is denoted as $B(t)$. A *binding element* is a pair (t, b), where $t \in T$ and $b \in B(t)$. The set of all binding elements is denoted as *BE*.

A *step* Y is a function that maps each $t \in T$ to a multi-set over $B(t)$ such that $Y(t)$ is finite for all $t \in T$ and non-empty for at least one $t \in T$. The expression $E(p, t)\langle b\rangle$ yields the multi-set of tokens removed from place p when transition t occurs with the binding b. Similarly, the expression $E(t, p)\langle b\rangle$ yields the multi-set of tokens inserted in place p when transition t occurs with the binding b. For instance, when the transition *Receive* occurs with the binding $b_1 = \langle i = 2, p = \varepsilon, q = \varepsilon, x = 2\rangle$, $E(\mathit{Link}, \mathit{Receive})\langle b_1\rangle$ extracts token $(\varepsilon.2, 3)$ from place *Link* while $E(\mathit{Input\,Queue}, \mathit{Receive})\langle b_1\rangle$ extracts token $(\varepsilon, 2)$ from place *Input Queue* and $E(\mathit{Receive}, \mathit{Link})\langle b_1\rangle$ deposits token $(\varepsilon, 3)$ in place *Link* while $E(\mathit{Receive}, \mathit{Input\,Queue})\langle b_1\rangle$ deposits token $(2.\varepsilon, 2)$ in place *Input Queue*.

A step Y is *enabled* in marking M if and only if

$$\sum_{t\in T}\sum_{b\in Y(t)} E(p, t)\langle b\rangle \leq M(p) \text{ for all } p \in P \tag{1}$$

The left term of the inequality yields the multi-set removed from place p when the transitions in step Y occur. The multi-set removed is required to be less than or equal to the contents of p. When a step Y is enabled in a marking M, it may *occur*. Occurrence changes M to another marking M', defined by:

$$M'(p) = (M(p) - \sum_{t\in T}\sum_{b\in Y(t)} E(p, t)\langle b\rangle) + \sum_{t\in T}\sum_{b\in Y(t)} E(t, p)\langle b\rangle \ \forall p \in P \tag{2}$$

The first sum defines the tokens removed from place p while the second specifies the inserted tokens. We say that M' is *directly reachable* from M by the *occurrence* of the step Y, which is also denoted by $M[Y\rangle M'$. For instance, the step Y_0 that maps transition *Produce* to binding $\langle i = 0, q = \varepsilon, x = 0\rangle$ and all other transitions to the empty multi-set is enabled in a marking M_0 where every place is marked with the expression *begin* (see Figure 3). If Y_0 occurs in M_0, we get a marking M_1 in which place *Host* is marked with the tokens $(0, 0) + (\varepsilon, 1) + (\varepsilon, 2) + (\varepsilon, 3) + (\varepsilon, 4)$ and every other place contains the expression *begin*. Cell 0 has been inserted in the queue of Host 0. The concept of *occurring step* is extended to a *finite occurrence sequence* of markings and steps as follows:

$$M_1[Y_1\rangle M_2[Y_2\rangle \ldots M_n[Y_n\rangle M_{n+1}$$

where $n \in \mathbb{N}$ is the *number of steps* in the sequence and $M_i[Y_i\rangle M_{i+1}$ for all $i \in \{1, \ldots, n\}$. The abbreviated form $M_1[Y_1Y_2 \ldots Y_n\rangle M_{n+1}$ is also used. A marking M' is *reachable* from a marking M if and only if there exists a finite occurrence sequence having M as start marking and M' as end marking. A marking M is reachable if and only if it is reachable from M_0. The set of markings that are reachable from M is denoted as $[M\rangle$. By definition, $M \in [M\rangle$. The marking directly reachable from M when the step Y is enabled is represented by $M[Y\rangle$. In this paper, we assume that every step maps one transition to one binding element, while all other transitions are mapped to the empty multi-set, although a number of binding elements can be concurrently enabled. Therefore, $M[Y\rangle M'$ is written as $M[(t, b)\rangle M'$ and $M_1[Y_1Y_2 \ldots Y_n\rangle M_{n+1}$ as $M_1[(t_1, b_1)(t_2, b_2) \ldots (t_n, b_n)\rangle M_{n+1}$.

Symmetry Specification

It is often the case that large systems have numerous interacting components with similar behaviors. These components are so alike that we may abstract the differences between them. As aforementioned, in the initial marking of Figure 2, Host 0 can produce Cell 0. From M_0, every other cell can be produced and every other host can produce a cell. In this problem, we supervise the length of queues, not their actual contents. Hence, all the immediate successors of M_0 are considered equivalent or symmetrical to each other. They are symmetrical in the sense that any one of them can be obtained from any of the others by a consistent exchange of colors (e.g., Host 1 for Host 0, Cell 1 for Cell 0).

The set of atomic color sets is denoted as Σ_{AT} and the set of structured color sets is denoted as Σ_{ST}. Let *SG* be a function that maps each atomic color set $S \in \Sigma_{AT}$ to a subgroup *SG(S)* of the set of all permutations of S. A permutation symmetry for *SG* is a function ϕ that maps each atomic color set $S \in \Sigma_{AT}$ to a permutation $\phi_S \in SG(S)$. The set of all permutation symmetries for *SG* is denoted Φ_{SG} and is called a *permutation symmetry specification* (Jensen, 1995).

The domain of each permutation symmetry $\phi \in \Phi_{SG}$ is extended from Σ_{AT} to Σ. For each permutation symmetry $\phi \in \Phi_{SG}$ and each structured color set $S \in \Sigma_{ST}$, ϕ_S is defined as follows:

1. if S = **list** A, $\phi_S(a_1a_2 \ldots a_n) = \phi_A(a_1)\phi_A(a_2) \ldots \phi_A(a_n)$.

2. if S = **product** $A_1 \times A_2$, $\phi_S((a_1, a_2)) = (\phi_{A_1}(a_1), \phi_{A_2}(a_2))$.

Let a permutation symmetry $\phi \in \Phi_{SG}$, a marking M, a binding b, and a binding element (t, b); $\phi(M)$, $\phi(b)$, and $\phi((t, b))$ are defined as follows:

1. for all $p \in P$, $\phi(M)(p) = \phi_{C(p)}(M(p))$.
2. for all $v \in Var(t)$, $\phi(b)(v) = \phi_{Type(v)}(b(v))$.
3. $\phi(t, b) = (t, \phi(b))$.

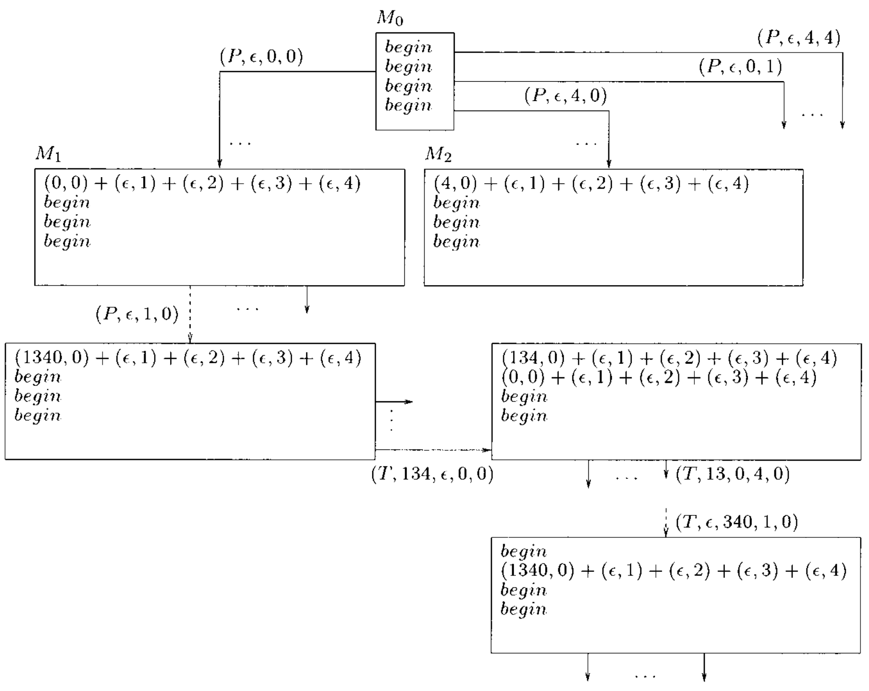
Note that $(\Phi_{SG}, \circ)$ is an algebraic group, where $\circ$ is the function composition operator and the neutral element is the identity function. In order to capture the fact that two symmetrical markings have similar properties, a permutation symmetry specification Φ_{SG} must be consistent with the behavior of the CP-net. A symmetry specification Φ_{SG} is *consistent* if and only if each permutation symmetry maps the initial marking to itself, $\phi(M_0) = M_0$, binding elements to binding elements, for all $b \in B(t)$ $\phi(b) \in B(t)$, and it does not matter whether the permutation symmetry is applied before or after the evaluation of an arc expression, if a is an arc and t the transition of a, for all $b \in B(t)$ $E(a)\langle\phi(b)\rangle = \phi(E(a)\langle b\rangle)$ (Jensen, 1995). It is the responsibility of a person developing a CP-net model to provide a permutation symmetry specification and to prove that it is consistent. An alternative approach is to leave to a software tool the task of finding a permutation symmetry specification. It is a topic of research (Jensen, 1995). In our example, it is easy to see that arbitrary permutation of cell and host numbers yields a consistent symmetry specification.

A consistent permutation symmetry specification Φ_{SG} induces the equivalence relations $\approx_M$ and $\approx_{BE}$ on markings and binding elements, respectively. We use $M_\approx$ and $BE_\approx$ to denote the set of all equivalence classes for $\approx_M$ and $\approx_{BE}$, respectively. The notation $[X]$, where $X \subseteq \mathcal{M}$, represents all the markings equivalent to a marking from X: $[X] = \{M \in \mathcal{M} \mid (\exists x \in X)\ M \approx_M x\}$. The notation $[\{M\}]$, where $M \in \mathcal{M}$, is simplified to $[M]$. Hereafter, Φ_{SG} is imply written as Φ and called a symmetry specification.

Occurrence Graph

An occurrence graph (OS-graph) models the behavior of a CP-net. A part of an occurrence graph for the CP-net of Figure 2 is shown in Figure 5. It has a set of nodes and a set of arcs. Each node represents a reachable marking. The marking is inscribed within the node. The node labeled M_0 represents the initial marking. The rows indicate the markings of places *Host*, *Output Queue*, *Link*, and *Input Queue*. Each arc represents a step. The arc is labeled with a binding element. For the sake of conciseness, we write $(P, \varepsilon, 0, 0)$ instead of (*Produce*, $\langle q = \varepsilon, x = 0, i = 0\rangle$) and $(T, 134, \varepsilon, 0, 0)$ instead of (*Transmit*, $\langle q = 134, p = \varepsilon, x = 0, i = 0\rangle$), analogously for the other steps.

Figure 5 shows just some of the reachable markings. Let us compute the total number of reachable markings. For each queue, there are $\sum_{i=0}^{5} 5^i = 3906$ token values. There are five queues stored in each place. Hence, for each place there 3906^5 markings. There are four places, it means that there are $(3906^5)^4 = 3906^{20} \approx 10^{72}$ different markings in the OS-graph.



M_0
begin begin begin begin
$(P,\epsilon,0,0)$
$(P,\epsilon,4,4)$
$(P,\epsilon,0,1)$
$(P,\epsilon,4,0)$
M_1
$(0,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ begin begin begin
M_2
$(4,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ begin begin begin
$(P,\epsilon,1,0)$
$(1340,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ begin begin begin
$(134,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ $(0,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ begin begin
$(T,134,\epsilon,0,0)$
$(T,13,0,4,0)$
$(T,\epsilon,340,1,0)$
begin $(1340,0)+(\epsilon,1)+(\epsilon,2)+(\epsilon,3)+(\epsilon,4)$ begin begin

Figure 5. Occurrence graph for the CP-net of Figure 2.

From a behavioral point of view, there is little difference between symmetrical markings M_1 and M_2 in Figure 5 under the permutation symmetry ϕ in which Cell 0 is mapped to Cell 4 and all other token values remain the same. Each occurrence sequence starting from M_1 as a symmetrical occurrence sequence starting from M_2, and vice versa. Occurrence sequences can be obtained from each other by application of ϕ. Hence, during behavior analysis it makes sense to explore a single marking for each class of symmetrical markings.

More formally, an occurrence graph is a 4-tuple (V, A, N, v_0), where

- V is the finite set of nodes $\{[M] \in M_{\approx} \mid [M] \cap [M_0\rangle \neq \emptyset\}$;
- A is the finite set of arcs $\{([M_1], [(t, b)], [M_2]) \in V \times BE_{\approx} \times V \mid \exists(M'_1, (t, b'), M'_2) \in [M_1] \times [(t, b)] \times [M_2]$ such that $M'_1[(t, b')\rangle M'_2\}$;
- N is the node function from A into $V \times V$. If $a = ([M_1], [(t, b)], [M_2])$, then $N(a) = ([M_1], [M_2])$;
- $v_0 \in V$ is the initial node ($v_0 = [M_0]$).

For the example of Figure 5, let us suppose that we explore just one marking in each class of symmetrical markings. We make abstraction of the actual contents of queues. We just retain their length. Every queue may have from zero to five elements. There are therefore six individual queue states. In every place there are five queues but we make also abstraction of the actual owner of every queue. We distinguish 252 different states of five queues (this figure corresponds to the number of multi-sets of cardinal five that can be constructed from six different elements). Since there are four places, we have a total of $252^4 \approx 10^{9.61}$ markings to explore. We have reduced the complexity of the problem for at least 62 orders of magnitude. It is still, however, a lot of markings. Note, although, that synthesis of a controller does not have to generate all these markings. Indeed, in the implementation of our algorithm we construct the OS-graph of the process while we synthetize the controller, on the fly. During synthesis many paths are cut off because they are outgoing from invalid states. Hence, the controller itself is much smaller than the OS-graph. Construction on the fly of reachability graphs is explained in (Barbeau et al., 1997).

3. CP-net Supervisory Design under Full Observation

The basic problem in supervisory control is to construct a controller that can turn off various events of an uncontrolled *DES*, called a *process* (Pr), according to some requirements. The process Pr is defined as a triple (CPN, Φ, K), where CPN is a CP-net with a consistent symmetry specification Φ and $K\colon P \to \mathbb{N}$ is a *capacity function* bounding the contents of every place. Therefore, we require that $|M_0(p)| \le K(p)$ and $|M'(p)| \le K(p)$ for all $p \in P$, in the enabling rule (1) (with $M'(p)$ defined by rule (2)).

Let Γ be the set of all functions γ, called *control patterns*, that assign a subset of $B(t)$ to every transition t to T, that is, $b \in \gamma(t) \Rightarrow b \in B(t)$. If $b \in \gamma(t)$, then the controller prevents the transition t from occurring with the binding b. Let T_c and T_u be fixed disjoint subsets of T denoting the sets of *controllable* and *uncontrollable* transitions, respectively. A controllable transition is one which occurrence can be disabled by the controller whereas an uncontrollable transition cannot. The basic notion of activity in a CP-net is that of binding element. For the applications we handled so far, we found more convenient to specify controllability of transitions which by extension applies to bindings because the former are not as numerous as the latter.

A *controlled DES* (*CDES*) is an ordered tuple $Pr_c = (CPN_c, \Phi, K, \Gamma)$. In a CPN_c, $M[_c(t, b)\rangle M'$ denotes that a marking M' is directly reachable from M by the occurrence of the binding element (t, b) under the control of $\gamma \in \Gamma$. This is defined as:

$$\begin{array}{rl} M[(t,b)\rangle M' & \text{if } t \in T_u \\ M[(t,b)\rangle M', b \notin \gamma(t) & \text{if } t \in T_c \text{ and } \gamma \in \Gamma \end{array}$$

A *controller* is a pair $S = (G, \varphi)$, where G is a subgraph of an OS-graph with symmetries and φ is the feedback function.

The *feedback function* $\varphi\colon V \to \Gamma$ satisfies the following conditions ($\wp(B(t))$ denotes the power set of $B(t)$):

$$\begin{array}{rl} \varphi(v)(t) = \emptyset & \text{if } t \in T_u \\ \varphi(v)(t) \in \wp(B(t)) & \text{if } t \in T_c \end{array}$$

The graph G can be interpreted as the transition graph of an automaton modeling the behavior of a controller as in the original framework of Ramadge and Wonham. It is

$$\begin{tikzcd}
\text{Process } Pr_c & \text{Controller } S \\
M_i \arrow[r, "\in_\phi"] \arrow[dd] & v_i \arrow[dd] \\
(t_i, b_i) \arrow[r, "\in_\phi"] & l \\
M_{i+1} \arrow[r, "\in_\phi"] & v_{i+1} \\
\gamma_{i+1} & \varphi(v_{i+1}) = \gamma'_{i+1} \arrow[l, "\phi^{-1}"']
\end{tikzcd}$$

$$\phi^{-1}(\gamma) = \{(t, [\phi^{-1}(b_1), \ldots, \phi^{-1}(b_l)]) \mid (t, [b_1, \ldots, b_l]) \in \gamma\}$$

Figure 6. An execution step of the closed-loop system.

driven by a sequence of steps occurring in *CPN*. That is, after the firing of a transition, the controller moves to a node v which represents the marking reached by *CPN*. The role of the feedback function φ is to provide, after each execution step of the process and the controller, the control pattern γ that represents the binding elements inhibited for the next step.

The CDES and controller are then embodied in a closed-loop system to constitute a *supervised DES* (SDES) $S/Pr_c = (S, Pr_c)$. A state of a $SDES$ is a pair (v, M) where $v \in V$ and M is a marking of CPN_c such that $M \in v$. The behavior of S/Pr_c is illustrated in Figure 6. Let M_i be the current marking of the process and $v_i = [M_i']$ be a node of G such that there exists $\phi \in \Phi$ with $M_i' = \phi(M_i)$, that is, $M_i \in v_i$. Let (t_i, b_i) be the next step such that $M_i[(t_i, b_i)\rangle M_{i+1}$ and $b_i \notin \gamma_i(t_i)$ with the current control pattern γ_i. First, the controller S moves to the next node $v_{i+1} = [M_{i+1}']$ by executing the transition from node v_i on the arc labeled $l = [(t_i, b_i')]$. The binding element (t_i, b_i') is the representative member of binding elements equivalent to the process step (t_i, b_i), that is, $(t_i, b_i') = (t_i, \phi(b_i))$. The controller S includes, by construction, control patterns for a representative marking of each equivalence class of reachable markings. Let $\gamma_{i+1}' = \varphi(v_{i+1})$ be the control pattern of node v_{i+1}. To compute the control pattern corresponding to the next marking M_{i+1} of the process, the controller uses the inverse of the permutation symmetry ϕ to map the control pattern γ_{i+1}' to the control pattern $\gamma_{i+1} = \phi^{-1}(\gamma_{i+1}')$. Since $(\Phi, \circ)$ is a group, ϕ^{-1} always exists. Formally, if (v_i, M_i) is the current state of S/Pr_c, then a next state is (v_{i+1}, M_{i+1}) if and only if there exists a binding element (t_i, b_i) such that $M_i[(t_i, b_i)\rangle M_{i+1}$, $b_i \notin \gamma_i(t_i)$ with $\gamma_i = \phi^{-1}(\varphi(v_i))$, and $(v_i, l, v_{i+1}) \in A$ with $M_i \in v_i$, $M_{i+1} \in v_{i+1}$ and $(t_i, b_i) \in l$. The control pattern of M_{i+1} is $\phi^{-1}(\varphi(v_{i+1}))$. The controller S must be *complete* in the sense that $(v_i, l, v_{i+1}) \in A$ whenever S/Pr_c is in state (v_i, M_i) with $M_i[_c(t_i, b_i)\rangle M_{i+1}$ and $b_i \notin \gamma_i(t_i)$.

4. The Forbidden State Control Problem

In this paper, we consider the forbidden state control problem in which the control specification is expressed as a set of forbidden markings $\mathcal{M}_b \subseteq \mathcal{M}$. If a marking is forbidden, all its equivalent markings are also forbidden. Therefore, only one representative per each equivalence class of forbidden markings is included in $\mathcal{M}_b$.

Inadmissible Marking

Given a set of forbidden markings $\mathcal{M}_b$, there is, in general, a larger set of markings which must be avoided, due to uncontrollable transition sequences. The markings from which the process can uncontrollably reach forbidden markings are characterized by the following predicate:

$$\begin{aligned} \mathit{Inadmissible}(M) \Leftrightarrow\ & (\exists n \geq 0)(\exists t_1, \ldots, t_n \in T_u) \\ & (\exists b_1 \in B(t_1)) \ldots (\exists b_n \in B(t_n)) \\ & M[(t_1, b_1) \ldots (t_n, b_n)\rangle \ \in [\mathcal{M}_b] \end{aligned} \tag{3}$$

A node with associated inadmissible markings is inadmissible. When the initial marking is admissible, and steps contain only one binding element, the maximally permissive solution to the forbidden state control problem exists and prevents the process from reaching any inadmissible marking (Holloway and Krogh, 1990).

Problem

The problem is expressed as follows. Given a set of forbidden markings $\mathcal{M}_b$, an uncontrolled DES $Pr = (CPN, \Phi, K)$, and an admissible initial marking $M_0 \notin [\mathcal{M}_b]$, derive a *maximally permissive* controller S, that is: (1) the closed-loop system S/Pr_c is safe (Pr_c cannot reach a forbidden marking under the control of S); and (2) a reachable marking of Pr, which is an unreachable marking of Pr_c under the control of S, is either forbidden, can uncontrollably lead to a forbidden marking, or can only be reached from the initial marking by sequences that pass through a forbidden or inadmissible marking. Before describing the synthesis algorithm, let us introduce an admissibility assessment predicate and the notion of latest controllable binding elements.

Latest Controllable Binding Elements

Let y denote a node in the *OS-graph*. The *latest controllable binding elements* of y is a set (denoted as *LCBE*) of all triples of the form (x, t, b) such that (the expressions M_x and $lcbe_x$ denote a representative marking M and *LCBE* of the node x, respectively):

1. x is a node in the *OS-graph*;
2. (t, b) is a binding element, where t is a controllable transition;

```
procedure Synthesize_Controller($Pr$,$\approx_M$,$\approx_{BE}$,$\mathcal{M}_b$,$T_c$)
$V \leftarrow \{\}$; $A \leftarrow \{\}$; $processed\_nodes \leftarrow \{\}$; $unprocessed\_nodes \leftarrow \{$New_Node($M_0$,$\{\}$)$\}$
repeat
  select $x$ in $unprocessed\_nodes$;
  if not ($M_x \approx_M M_y$) for some $y$ in $processed\_nodes$ then
    for all $(t, b)$ enabled in $M_x$ do
      $M \leftarrow$ New_Marking($M_x$,($t$,$b$));
      if $M \approx_M M'$ for some $M'$ in $\mathcal{M}_b$ then
        if $t \in T_c$ then
          $\varphi(x)(t) \leftarrow \varphi(x)(t) \cup \{b\}$
        else
          Inadmissible($x$); break
      else
        if not ($M \approx_M M_y$) for some $y$ being a son of $x$ then
          if $t \in T_c$ then $lcbe_z \leftarrow \{(x,t,b)\}$ else $lcbe_z \leftarrow lcbe_x$;
          $z \leftarrow$ New_Node($M$,$lcbe_z$);
          $unprocessed\_nodes \leftarrow unprocessed\_nodes \cup \{z\}$;
          $a \leftarrow$ New_Arc($x$,$(t, b)$,$z$); $A \leftarrow A \cup a$
    if $x.status = admissible$ then V $\leftarrow$ V $\cup \{x\}$
  else
    if $y.status = inadmissible$ then Inadmissible($x$) else $lcbe_y \leftarrow lcbe_y \cup lcbe_x$
  $processed \leftarrow processed \cup \{x\}$;
  $unprocessed\_nodes \leftarrow unprocessed\_nodes \setminus \{x\}$
until $unprocessed\_nodes = \{\}$;
```

Figure 7. Algorithm for synthesizing a compact controller.

3. $(\exists n \geq 0)$ $M_x[(t, b)(t_1, b_1) \ldots (t_n, b_n)\rangle M_y$ with $t_i \in T_u$, for $i = 1, \ldots, n$.

A triple (x, t, b) contained in the *LCBE* of y is interpreted as follows. The occurrence of the step (t, b) from M_x is controllable whereas the sequence of steps $(t_1, b_1) \ldots (t_n, b_n)$ from $M_x[(t, b)\rangle$ to M_y are uncontrollable. Therefore, to make M_y and its equivalent markings unreachable in the process, it is necessary to disable the binding b for t when the process is in marking M_x.

5. The Synthesis Algorithm

The basic idea behind our algorithm is to reduce the number of markings that must be examined by gathering the components that "behave in the same way" into the same equivalence class. Therefore, the set of all markings and the set of all steps are partitioned into disjoint nonempty equivalence classes. The algorithm, given in Figure 7, is based on the notion of *latest controllable binding elements*, *equivalent markings*, and *inadmissible markings*. It accepts as input a process Pr, a consistent symmetry specification represented by the equivalence relations $\approx_M$ and $\approx_{BE}$, a set of forbidden markings $\mathcal{M}_b$, and a set of controllable transitions T_c. A maximally permissive controller $S = (G, \varphi)$ is computed from representative members, one per class of equivalent markings and class of equivalent steps.

The algorithm uses many functions briefly described hereafter. Function `New_Node` creates a new node, of the controller graph G, from a marking and an *LCBE*. The marking

```
1 procedure Inadmissible($x$) :
2     $x.status \leftarrow inadmissible$; Remove_Sons($x$); $V \leftarrow V \setminus \{x\}$
3     for each $x'$ in From_Nodes($x$) such that $x'.status = admissible$ do
4         Inadmissible($x'$)
5     if $lcbe_x \neq \{\}$ then
6         for each $(x', t, b) \in lcbe_x$ do
7             $\varphi(x')(t) \leftarrow \varphi(x')(t) \cup \{b\}$
8     else "no solutions"
9 end
```

Figure 8. Procedure for determining inadmissible nodes.

associated to a node is a representative of its equivalence class. Function `New_Marking` yields the marking reached after the occurrence of a binding element from a given marking. The function `New_Arc` creates a new arc from a source node, a binding element, and a destination node. The function `From_Nodes` takes as argument a node x and returns the set of nodes in V from which x is directly reachable on occurrence of an uncontrollable binding element.

Each node has a status indicating whether or not its associated marking is *admissible*. A new node has its status set to *admissible*. The function `Inadmissible`, given in Figure 8, fixes the status of a node to *inadmissible* (line 2), determines those that become inadmissible among its predecessors (lines 3 and 4), and updates the feedback function by inserting, for each (x', t, b) in the *LCBE* of x, the binding b in the set of forbidden bindings for t of x' (lines 6 and 7). When a node becomes inadmissible, all its son nodes in V are removed (including their bound arcs) by using the procedure `Remove_Sons` (line 2). If the *LCBE* of x is empty, then there are no solutions (line 8).

The algorithm works as follows. Initially, the sets of nodes V and arcs A of the graph are both empty. The set of processed nodes is also empty. The node $v_0 = (M_0, \{\})$ is created and inserted into the set of unprocessed nodes (line 2). While there are unprocessed nodes, a node x is selected (line 4) and processed. The processing of a node starts with a test for an equivalence between the marking M_x and the marking M_y of an already processed node y (line 5). Only the first-picked node in each equivalence class is developed further. If such a node y exists, then the algorithm checks if it is an inadmissible node. If so, then the `Inadmissible` function is called on node x to disable the latest controllable binding elements on the path leading to x (line 21). Otherwise, the contents of $lcbe_x$ is inserted in $lcbe_y$ (line 21). If such a node y does not exist, every binding element (t, b) enabled in M_x is analyzed (line 6). The marking M reached after the occurrence of the binding element (t, b) is computed (line 7) and checked for an equivalence with some other marking M' included in the set of forbidden markings (line 8). If so, there are two cases: either t is controllable or not. If transition t is controllable, the binding b is inserted in the set of forbidden bindings of x (line 10). If t is uncontrollable, then the function `Inadmissible` is called on node x (line 12). If M is not equivalent to some forbidden marking in $\mathcal{M}_b$, then the algorithm checks if marking M is equivalent to a marking M_y where y is a son of x (line 14). If all the above conditions are not satisfied, it means that the marking M

is not equivalent to a marking of a son of x and not forbidden. In this case, a new node z is created with the following attributes: the marking M and $lcbe_z$ which is defined as $\{(x, t, b)\}$ if the transition t is controllable; otherwise, it is the $lcbe_x$ of its parent node x (lines 15 and 16). Furthermore, the set of unprocessed nodes is updated (line 17) and a new arc is created from node x to node z and added to A (line 18). At the end of the analysis of all binding elements (t, b) enabled in M_x, if the status of x is *admissible*, then x is included in V. Finally, the node x is included in the set of processed nodes and removed from the set of unprocessed nodes (lines 22 and 23).

In the implementation of the algorithm, testing $M \approx_M M'$ translates to finding a permutation symmetry $\phi \in \Phi$ such that $\phi(M) = M'$. In our example, the permutation symmetry specification is arbitrary permutation. Calculation of a permutation can be factorial in the worst case. Jensen (1995) proposes, however, techniques to efficiently calculate symmetries. For instance, several cases can be eliminated by first testing if M can be mapped to M' by verifying if they have the same multi-set of positive coefficients.

The following theorem shows that the set of nodes V generated by our algorithm represents the unique maximal set of markings that solves the forbidden state control problem with respect to a set of forbidden markings $\mathcal{M}_b$. The proof of the theorem is based on two lemmas. Let us first introduce some properties of Jensen's OS-graph (Jensen, 1995) that are used in the proof. Hereafter, a node is identified to its representative marking.

PROPOSITION 1 *Let Φ be a consistent symmetry specification and (W, A, N, v_0) an OS-graph. The following properties hold for all $M_1, \ldots, M_{n+1} \in [M_0\rangle$ and all $\phi \in \Phi$:*

1. $[M_0\rangle = [W]$
2. $M_1[(t_1, b_1)\rangle \ldots M_n[(t_n, b_n) > M_{n+1} \Leftrightarrow$
   $\phi(M_1)[(t_1, \phi(b_1))\rangle \ldots \phi(M_n)[(t_n, \phi(b_n))\rangle\phi(M_{n+1})$
3. $M \in [M_0\rangle \Leftrightarrow \phi(M) \in [M_0\rangle$

LEMMA 1 *Let Φ be a consistent symmetry specification. For all $M \in [M_0\rangle$ and all $\phi \in \Phi$, Inadmissible(M) $\Leftrightarrow$ Inadmissible$(\phi(M))$.*

Because of the definition of the predicate *Inadmissible,* there exists a sequence of uncontrollable binding elements $(t_1, b_1) \ldots (t_n, b_n)$ with $t_i \in T_u$ and a marking $M' \in \mathcal{M}$ such that $M[(t_1, b_1) \ldots (t_n, b_n)\rangle M'$ and $M' \in [\mathcal{M}_b]$. From Proposition 1, $\phi(M)[(t_1, \phi(b_1)) \ldots (t_n, \phi(b_n))\rangle\phi(M')$. By the definition of $\mathcal{M}_b$, $\phi(M') \in [\mathcal{M}_b]$. Therefore, *Inadmissible*$(\phi(M))$ = *true*. The proof of the converse is similar, using the function ϕ^{-1}. ■

LEMMA 2 *Let predicate Inadmissible be restricted to markings labeling nodes in the set $processed\_nodes$, $v \in V$ iff Inadmissible$([M_v])$ = false.*

Proof: We use induction on the number k of times the **repeat-until** loop is executed. Because of the initialization process in line 2, this loop will be executed at least one time.

Basis. ($k = 0$) Trivially true since $processed\_nodes$ (the set of markings reachable form M_0) is empty.

Induction hypothesis. $v \in V$ iff $Inadmissible(M_v) = false$.

Induction. ($k > 0$) We show that, if the hypothesis holds at the start of the loop, then it holds at the end. Let $x \in processed\_nodes$ at the end of the loop. There are two cases: whether $x \in processed\_nodes$ at the start of the loop or not.

Case 1. When $x \notin processed\_nodes$ at the start of the loop, suppose that x is selected from the set $unprocessed\_nodes$ by the current loop, yielding two subcases.

Subcase 1.1. The conditions of the **if** statement at line 5 is satisfied. The **for** loop at line 6 is entered, that is, all enabled markings at M_x are examined. This loop is finite since the color sets are finite. For each marking we have three subcases:

Subcase 1.1.1. The condition of the **if** statement at line 8 is satisfied and the **if** statement at line 9 is not satisfied. The procedure `Inadmissible` is called with the actual parameter x (line 12). The node x is not included in V (line 19) since its status has been set to *inadmissible* in procedure `Inadmissible` (line 2, Figure 8). These conditions are true *iff* there exists a binding element (t, b) such that $M_x[(t, b)\rangle M$, $t \in T_u$, and $M \in [M']$ for some $M' \in [\mathcal{M}_b]$. Then $Inadmissible(M_x) = true$ because of (3).

Subcase 1.1.2. The conditions of the **if** statement at line 8 and line 9 are both satisfied. Then $Inadmissible(M_x) = false$ because of (3) (with $n = 0$) and $x \in V$ because of the line 19.

Subcase 1.1.3. The condition of the **if** statement at line 8 is not satisfied, that is, M_x and its equivalents are not prohibited. Therefore, $Inadmissible(M_x) = false$ and $x \in V$ because of the line 19. The condition of the **if** statement at line 14 is used for implementation purpose only to avoid including enabled equivalent sons of M_x in the set of unprocessed nodes.

Subcase 1.2. The condition of the **if** statement at line 5 is not satisfied and the condition of **if** statement at line 21 is satisfied. The procedure `Inadmissible` is called with the actual parameter x (line 21) and its status is set to *inadmissible* (line 2, Figure 8). These conditions are true *iff* there exists an inadmissible node $y \in processed\_nodes$ with a marking that is equivalent to M_x ($\exists\phi \in \Phi$ such that $M_x = \phi(M_y)$). Therefore, $Inadmissible(M_x) = true$ because of Lemma 1. When the condition of the **if** statement at line 21 is not satisfied, M_x and its equivalents are not prohibited. Therefore, $Inadmissible(M_x) = false$, by Lemma 1.

Case 2. $x \in processed\_nodes$ at the start of the loop. At the start of the loop, either $x \in V$ or not. It can easily be checked that if $x \notin V$ at the start of the loop, then the algorithm never inserts a node with a marking equivalent to M_x in V (because of line 5). Let us consider the case where $x \in V$ at the start of the loop. The node x is removed from V at the end of loop *iff* a node x' from *unprocessed_nodes* has been selected and the procedure `Inadmissible` has been called with actual parameter x' (lines 12 or 21). Then, x'.*status* received the value *inadmissible* and the recursive calls to `Inadmissible` follow backward all the paths incoming to x', while transitions are uncontrollable. Furthermore, every encountered inadmissible node in paths leading to x' are removed, including x (line 2, Figure 8). The above is, however, possible *iff* x' has been selected, *Inadmissible*$(M_{x'}) =$ *true*, in accordance with Case 1, and there is a sequence of uncontrollable steps $Y_1 \ldots Y_n$ with $M_x[Y_1 \ldots Y_n\rangle M_{x'}$. These conditions are true *iff* *Inadmissible*$(M_x) =$ *true* at the end of the loop because of (3). ■

THEOREM 1 *The procedure* `Synthetize_Controller` *always terminates and, if* $M_0 \notin [\mathcal{M}_b]$ *and Inadmissible*$(M_0) =$ *false, then it returns a controller* $S = (G, \varphi)$ *that is the unique maximal solution to the forbidden state control problem with respect to the set of forbidden markings* $\mathcal{M}_b$.

Proof: Note that the procedure `Synthetize_Controller` always terminates since the color sets are finite and Pr satisfies boundedness properties.

Let us assume that $M_0 \notin [\mathcal{M}_b]$ and *Inadmissible*$(M_0) =$ *false*. Let $S = (G, \varphi)$ be the controller computed by `Synthetize_Controller`. We can show the following by using Lemma 2 and the definition of the predicate *Inadmissible* which expresses the converse of controllability. For every $v \in V$, $M_v \in [M_0\rangle$ and *Inadmissible*$(M_v) =$ *false*. Furthermore, for all $t \in T_u$ and $b \in B(t)$, if $M_v[(t, b)\rangle \in [M_0\rangle$ then *Inadmissible*$(M_v[(t, b)\rangle) =$ *false*. From Lemma 1, this property holds for all markings equivalent to M_v. Therefore, the controller is safe.

From Proposition 1, we know that a marking is reachable *iff* it belongs to $[W]$, where W is the set of nodes of the *OS-graph* generated from Pr. Remark that $V \subseteq$ *processed_nodes* $\subseteq W$. Let $w \in W$ such that *Inadmissible*$(M_w) =$ *false*, M_w is reachable under the control of S, and $w \notin V$. If $w \in$ *processed_nodes*, then, from Lemma 2, *Inadmissible*$(M_w) =$ *true*, a contradiction. If $w \notin$ *processed_nodes*, then for all sequences of steps $(t_1, b_1) \ldots (t_n, b_n)$ such that

$$M_0[(t_1, b_1)\rangle M_1 \ldots M_{n-1}[(t_n, b_n)\rangle M_w,$$

there exist $i \in \{1, \ldots, n-1\}$ and $v \in$ *processed_nodes*, *Inadmissible*$(M_v) =$ *true* (that is $v \notin V$), and $M_i \in [M_v]$. Therefore, $[W]$ are non reachable under the control of S, a contradiction. Hence, the controller is maximal. ■

6. An Example

Let us consider as an example, the network of Section 2.

Congestion Control Specification

Congestion arises in a switch when the rate at which cells arrive and queue up exceeds the rate at which cells can be transmitted. The queue size then grows without bound and cells are lost. We call such states congestion states. A rule of thumb states that when the link for which cells are queuing is 80% or more utilized, the queue length begins growing at an alarming rate (Stallings, 1997). We use this rule to specify the congestion states in the CP-net. That is, every CP-net marking in which at least one queue has four or five cells is considered as forbidden marking.

When the network reaches a congestion state, one of the following two strategies can be adopted: (i) a switch simply discards any incoming cell for which there is no available memory space; (ii) a congested switch exercises some sort of flow control over its neighbors and hosts so that the traffic flow remains manageable. In this paper, we adopt the second strategy. In the CP-net model of Figure 2, transitions *Produce*, *Route*, *Send*, and *Transmit* are controllable.

It is clear that switch behaviors are similar and there is little difference in the way they handle cells. For example, a marking in which all queues are empty except queue number 1 in place *Host* with two cells (cell numbered 2 and 3) is equivalent to a marking in which all queues are empty except queue number 2 in place *Host* with two cells (cell numbered 3 and 4). This equivalence is obtained by the permutation symmetry ϕ that maps *Queue* to ϕ_{Queue} defined as:

$$\phi_{Queue}(x) = \begin{cases} 2 & \text{if } x = 1 \\ 1 & \text{if } x = 2 \\ x & \text{otherwise} \end{cases}$$

and *Cell* to ϕ_{Cell} defined as:

$$\phi_{Cell}(x) = \begin{cases} 3 & \text{if } x = 2 \\ 4 & \text{if } x = 3 \\ x & \text{otherwise} \end{cases}$$

The congestion states are CP-net markings in which one place has at least one queue with four or five cells. It should be noted that the forbidden markings are not all given explicitly. We give one representative per equivalence class. Furthermore, in the implementation forbiddeness can be tested by an user provided function, making forbidden marking specification even simpler.

We compute the number of classes of forbidden markings as follows. First, let us compute the number of classes of unforbidden markings. For each place, we distinguish 56 different unforbidden states of five queues (this figure corresponds to the number of multi-sets of cardinal five that can be constructed from four different elements, elements 0 to 3). Since there are four places, we have a total of $56^4 \approx 10^{6.993}$ classes of unforbidden markings. Hence, we have $10^{9.606} - 10^{6.993} \approx 10^{9.605}$ classes of forbidden markings.

Limitations of the Model

Before showing how the derived controller can be combined with the network to prevent congestion, we first assess our model. This model deviates somewhat from the real world. We assume that cell transfer is not affected by communication delays and that a centralized controller can observe all network events. In the real world, cells are transmitted with delays and some network events are unobservable.

Despite these unrealistic modeling assumptions, the model remains a good theoretical first step toward application of supervisory design to the congestion control problem, for two reasons. First, the effect of communication delays can be counteracted by imposing additional constraints on the process model. In order words, we can, for example, model communication links with more than one place. Second, the centralized scheme may be approached by exchanging traffic statistics between switches. The original framework by Ramadge and Wonham can handle a decentralized and partial observation scheme (Lin and Wonham, 1988) and has also been extended to handle real-time constraints (Brandin and Wonham, 1994; Barbeau et al., 1998). Similar extensions must to be done in our CP-net framework.

Congestion Supervisory Control

Given a set of congestion states (CP-net forbidden markings) $\mathcal{M}_b$, a network (uncontrolled DES) modeled by a CP-net, and the network initial state (initial marking) M_0, the congestion control problem can be stated as the synthesis of a controller $S = (G, \varphi)$ that prevents the network from reaching any congestion state.

The state space of the CP-net model of the network with five switches is approximatly $10^{71.835}$. It is clear that an automaton-based approach is impracticable. Comparison given in Tables 1 and 2 show that our approach is very efficient relative to the automata approach. In addition to the congestion problem discussed in this paper, we give data for two other problems that we considered:

1. the circular railway example as the one introduced in (Makungu et al., 1994), but the track is divided in ten sections and transitions $T1$, $T3$, $T5$, $T7$, and $T9$ are controllable and

2. the flexible assembly problem introduced in (Desrochers and Al-Jaar, 1995; Makungu et al., 1996).

Tables 1 and 2 give the number of states (and transitions) in the process, the control specification, and the controller when the automata is the formalism used. These tables give also the number of places (and transitions) in the process, the number of forbidden markings in the control specification, and the number of states (and transitions) in the controller automaton when the colored Petri net is the formalism used in conjunction with a symmetry specification (a number between brackets represents the number of states generated).

The numbers in the tables were obtained by experimentation, in part using the tool SUCSEDES (Barbeau and St-Denis, 1995). The number of nodes in the graph of the con-

Table 1. Comparison of the automata and CP-nets approaches—Part 1.

	Process		Control specification	
	Automaton	CP-net	Automaton	$\mathcal{M}_b$
Trains				
2	10^2 (2×10^2)	10 (9)	70 (120)	20
3	10^3 (3×10^3)	10 (9)	150 (300)	180
4	10^4 (4×10^4)	10 (9)	100 (160)	1800
Robots				
3	14 (27)	4 (3)	13 (24)	1
Congestion				
5	$\approx 10^{72}$	4 (6)	—	$\approx 10^{9.605}$

Table 2. Comparison of the automata and CP-nets approaches—Part 2.

	Controller	
	Automaton	graph
Trains		
2	60 (100) [70]	30 (50) [54]
3	90 (150) [135]	30 (50) [64]
4	40 (40) [75]	10 (10) [17]
Robots		
3	13 (24) [13]	5 (7) [8]
Congestion		
2	—	$\approx 10^{6.798}$

troller for the congestion problem has been computed as follows. For place *Host*, queues may contain from zero to three elements. The contents of place *Host* can be kept below 80% by disabling its controllable transition *Produce*. There are therefore 56 classes of admissible equivalent markings for that place. The contents of place *Output Queue* can be kept below 80% by disabling its three input transitions (*Route*, *Send*, and *Transmit*). There are therefore 56 classes of admissible equivalent markings for that place. Transition *Receive* is uncontrollable. To keep below 80% the contents of place *Input Queue*, the contents of link queue $i \oplus 1$ ($i = 0, \ldots, 4$) must be controlled in accordance to the amount of empty space in input queue i. For instance, when the input queue i contains one cell, the link queue $i \oplus 1$ may have either zero, one or two cells. In other words, the total contents of link queue $i \oplus 1$ and input queue i must always be less than or equal to three. Maintenance of that condition is the task of the controller. There is a total of 2002 classes of admissible joint markings for places *Link* and *Input Queue*. Hence, there is a total of $56^2 \times 2002 \approx 10^{6.798}$ classes of admissible markings, a figure within the capabilities of current computers.

7. Conclusion

In this paper, we have presented a colored Petri-net approach to the control of DESs. One of the benefits of using CP-nets instead of equivalent PT-nets is the more compact and readable representation of the system. The algorithm developed for synthesizing the controller avoids an exhaustive search of the state space by the use of equivalence relations.

This work constitutes a first step towards the exploitation of the notion of symmetry with the aim of reducing the complexity of the controller synthesis procedurere. We deliberately unconstrained the structure of the Petri nets. At least two problems remain to be addressed. Firstly, how can we integrate our approach and approaches that do take advantage of the actual Petri net graphical structure, for example the approach in (Holloway and Krogh, 1990). Further complexity reduction could certainly be obtained. Secondly, how can we synthesize a CP-net model for the supervisor instead of an automaton. This would have the benefit of homogenizing the framework.

Acknowledgements

The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR).

References

Barbeau, M. and St-Denis, R. 1995. Verification of discrete event systems with the SUCSEDES tool. *Proc. of AMAST Workshop on Real-Time Systems (Models and Proofs)*, Bordeaux.

Barbeau, M., Kabanza, F., and St-Denis, R. 1997. An efficient algorithm for controller synthesis under full observation. *Journal of Algorithms* 25(1): 144–161.

Barbeau, M., Kabanza, F., and St-Denis, R. 1998. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Trans. on Automatic Control* 43(11): 1543–1559.

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

Brandin, B. A. and Wonham, W. M. 1994. Supervisory control of timed discrete-event systems. *IEEE Trans. on Automatic Control* 39(2): 329–342.

Clarke, E. M., Grumberg, O., and Long, D. E. 1994. Model checking and abstraction. *ACM Trans. Programming Languages and Systems* 16(5): 1512–1542.

Denham, M. J. 1988. A Petri net approach to the control of discrete-event systems. In *Advanced Computing Concepts and Techniques in Control Engineering*, pp. 191–214, Springer-Verlag, NATO ASI Series, vol. F47.

Desrochers, A. A. and Al-Jaar, R. Y. 1995. *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*, IEEE Press.

Giua, A. and DiCesare, F. 1991a. Supervisory design using Petri nets. *Proc. 30th IEEE Conf. on Decision and Control*, Brighton, England, pp. 92–97.

Giua, A. and DiCesare, F. 1991b. Blocking and controllability of Petri nets in supervisory control. *IEEE Trans. on Automatic Control* 39(4): 818–823.

Giua, A. and DiCesare, F. 1995. Decidability and closure properties of weak Petri net languages in supervisory control. *IEEE Trans. on Automatic Control* 40(5): 906–910.

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

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

Holloway, L. E., Krogh, B. H., and Giua, A. 1997. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications* 7(2): 151–190.

Ichikawa, A. and Hiraishi, K. 1988. Analysis and control of discrete event systems represented by Petri nets. *Proc. of Discrete Event Systems: Models and Applications 1987*, p. 115-134, Springer-Verlag, Lectures Notes in Control and Information Sciences, vol. 103.

Jensen, K. 1992. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, volume 1, Springer-Verlag.

Jensen, K. 1995. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, volume 2, Springer-Verlag.

Krogh, B. H. 1987. Controlled Petri nets and maximally permissive feedback logic. *Proc. 25th Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, pp. 317–326.

Kumar, R. and Holloway, L. E. 1992. Supervisory control of Petri net languages. *Proc. 31st IEEE Conf. on Decision and Control*, Tucson, AZ, pp. 1190–1195.

Li, Y. and Wonham, W. M. 1993. Control of vector discrete-event systems I—The base model. *IEEE Trans. on Automatic Control* 38(8): 1214–1227.

Lin, F. and Wonham, W. M. 1988. Decentralized supervisory control of discrete-event systems. *Information Sciences* 44(3): 199–224.

Makungu, M., Barbeau, M., and St-Denis, R. 1994. Synthesis of controllers with colored Petri nets. *Proc. 32th Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, pp. 709–718.

Makungu, M., St-Denis, R., and Barbeau, M. 1996. A colored Petri net-based approach to the design of controllers. *Proc. 35th IEEE Conf. on Decision and Control*, Kobe, Japan, pp. 4425–4432.

Moody, J. O. and Antsaklis, P. J. 1995. Petri net supervisors for DES in the presence of uncontrollable and unobservable transitions. *Proc. 33rd Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, pp. 176–185.

Park, Y. and Chong, E. K. P. 1995. Distributed inversion in timed discrete event system. *Discrete Event Dynamic Systems* 5(2/3): 219–241.

Ramadge, P. J. and Wonham, W. M. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization* 25(1): 206–230.

Ramadge, P. J. and Wonham, W. M. 1989. The control of discrete event systems. *Proc. IEEE* 77(1): 81–98.

Sreenivas, R. S. 1993. Deterministic λ-free Petri net languages and their application to the supervisory control of discrete event dynamic systems. *Proc. Midwest Circuits and Systems Conf.*, Detroit, MI.

Sreenivas, R. S. 1997a. On supervisory policies that enforce global fairness and bounded fairness in partially controlled Petri nets. *Discrete Event Dynamic Systems: Theory and Applications* 7(2): 1–18.

Sreenivas, R. S. 1997b. On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled Petri nets. *IEEE Trans. on Automatic Control* 42(7): 928-945.

Sreenivas, R. S. and Krogh, B. H. 1992. On Petri net models of infinite state supervisors. *IEEE Trans. on Automatic Control* 37(2): 274–277.

Sathaye, A. S. and Krogh, B. H. 1992. Logical analysis and control of time Petri nets. *Proc. 31st IEEE Conf. on Decision and Control*, Tucson, AZ, pp. 1198–1203.

Stallings. W. 1997. *Data and Computer Communications*, Fifth Edition, Prentice-Hall.

Ushio, T. 1990. Maximally permissive feedback and modular control synthesis in Petri nets with external inputs places. *IEEE Trans. on Automatic Control* 35(7): 844–848.

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