Optimal Liveness-Enforcing Control for a Class of Petri Nets Arising in Multithreaded Software

Hongwei Liao, *Student Member, IEEE*, Stéphane Lafortune, *Fellow, IEEE*, Spyros Reveliotis, *Senior Member, IEEE*, Yin Wang, *Member, IEEE*, and Scott Mahlke, *Member, IEEE*

***Abstract*—We investigate the synthesis of optimal liveness-enforcing control policies for Gadara nets, a special class of Petri nets that arises in the modeling of the execution of multithreaded computer programs for the purpose of deadlock avoidance. We consider maximal permissiveness as the notion of optimality. Deadlock-freeness of a multithreaded program corresponds to liveness of its Gadara net model. We present a new control synthesis algorithm for liveness enforcement of Gadara nets that need not be ordinary. The algorithm employs structural analysis of the net and synthesizes monitor places to prevent the formation of a special class of siphons, termed resource-induced deadly-marked siphons. The algorithm also accounts for uncontrollable transitions in the net in a minimally restrictive manner. The algorithm is generally an iterative process and converges in a finite number of iterations. It exploits a covering of the unsafe states that is updated at each iteration. The proposed algorithm is shown to be correct and maximally permissive with respect to the goal of liveness enforcement.**

***Index Terms*—Concurrent software, deadlock avoidance, liveness enforcement, optimal control, Petri nets.**

I. Introduction

LIVENESS-ENFORCING control is an important class of problems in the supervisory control of Petri nets. Petri nets have been employed to model resource allocation and concurrency of dynamic systems in many applications, and liveness is often an important property for these systems [22]. In this paper, we study a class of Petri nets that arises in modeling concurrent software. In this scenario, liveness of the Petri net model guarantees the complete absence of deadlocks in the corresponding program. Deadlock analysis based on Petri nets has been widely studied for flexible manufacturing systems and other technological applications involving a resource allocation function [15], [26]. It has also been applied to Ada programs [23]. Recently, supervisory control of Petri nets has been applied to concurrent program synthesis [9]. Our investigation in this paper is motivated by the recent multicore revolution in computer hardware. This trend is making parallel programming unavoidable but concurrency bugs are making it costly and error-prone. We have started a project, called Gadara [11], [16], [30], where we are interested in multithreaded programs with shared data. In this programming paradigm, *mutual exclusion locks* (or *mutexes*) are usually employed to protect shared data from inconsistent concurrent access. However, when mutexes are inappropriately used, an important class of failures, termed *circular-mutex-wait deadlocks*, can occur in the program when a set of threads are waiting for one another and none of them can proceed.

In [19], [32], we defined a special class of Petri nets, called Gadara nets, to systematically model multithreaded C programs with lock allocation and release operations. We formally established that a multithreaded program that can be modeled as a Gadara net is deadlock-free if and only if its associated Gadara net is *live* [19]. This correspondence motivates our study of *liveness-enforcing* control of Gadara nets. In addition to liveness, another important property desired in control synthesis is *maximal permissiveness*, so that the control logic will provably eliminate deadlocks while otherwise minimally constraining program behavior. Therefore, the main focus of the present paper is on the synthesis of *maximally-permissive liveness-enforcing (MPLE)* control policies for Gadara nets. By definition, an original Gadara net model of a concurrent program is ordinary (i.e., all its arc weights are equal to one), while a controlled Gadara net may no longer be ordinary due to the structure (new *monitor* places and arcs) added as a result of a control synthesis step. Such a step can be carried out prior to the control synthesis presented in this paper. In this step, users may enforce other properties (than liveness) on the net, or they may attempt to enforce liveness by using other methods. In either case, ordinariness of the resulting Gadara net is not guaranteed in general. This motivates our development of an MPLE control synthesis strategy for the *general* class of non-ordinary controlled Gadara nets that may arise from various applications. An MPLE control policy is often called an *optimal* liveness-enforcing control policy [13]. We employ the same terminology in this paper.

If the reachability graph of a Petri net is available, the problem of MPLE control can be solved by the Supervisory Control Theory for discrete event systems initiated by Ramadge and Wonham [3], [25]. The theory of regions (see, e.g., [5], [29]), which in some sense combines the modeling strength

Manuscript received May 02, 2011; revised January 25, 2012 and September 26, 2012; accepted November 17, 2012. Date of publication November 30, 2012; date of current version April 18, 2013. This work was supported in part by NSF grant CCF-0819882 and an award from HP Labs Innovation Research Program (University of Michigan) and by NSF grants CMMI-0619978 and CMMI-0928231 (Georgia Institute of Technology). Recommended by Associate Editor H. Marchand.

H. Liao, S. Lafortune, and S. Mahlke are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: hwliao@umich.edu; stephane@umich.edu; mahlke@umich.edu).

S. Reveliotis is with the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: spyros@isye.gatech.edu).

Y. Wang is with HP Labs, Palo Alto, CA 94304 USA (e-mail: yin.wang@hp.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2012.2230814

of Petri nets and the control strength of automata, synthesizes monitor places [21] back into the Petri net to avoid unsafe states in the reachability graph. But employing an automaton model (of the reachability graph) in controller synthesis suffers from the state explosion problem when modeling concurrent software, as it fails to capture the concurrency in the target parallel program. Moreover, the associated control decisions are made based on a centralized controller, which needs to be updated at every transition execution, and thus introduces a global bottleneck in the concurrent program. For these two reasons, we are investigating *structural* control techniques for Petri net models in the Gadara project. Petri net models can efficiently characterize system concurrency without enumerating the entire reachability space. Many approaches have been proposed for the synthesis of liveness-enforcing control logic for Petri nets. These approaches are typically sub-optimal, i.e., they sacrifice maximal permissiveness due to the complexity of the problem and the inherent limitation of monitor-based control. In the case of Gadara nets, we have demonstrated that MPLE control logic can always be implemented using monitor places [19], [32]. In this paper, we thoroughly exploit the *structural properties* of Gadara nets for the efficient synthesis of MPLE control policies. Our initial results in this regard were reported in our earlier work [31]. In this paper, we significantly extend and formalize MPLE control synthesis for controlled Gadara nets *that need not be ordinary*.

In general, the proposed MPLE control synthesis is an iterative process, because the synthesized control logic may introduce new potential deadlocks. That is, the added net structure, when coupled with the original net structure, may cause new potential deadlocks in the controlled net. This necessitates iterations on the controlled nets until no further deadlock is found. Few works address such an iterative process and its implications for MPLE control synthesis. A siphon-based iterative control synthesis method is proposed in [28] for the class of S^4PR nets. But this method is sub-optimal in general, i.e., it does not guarantee maximal permissiveness. In [8], the role of iterations in liveness-enforcing control synthesis is discussed and a net transformation technique is employed to transform non-ordinary nets into PT-ordinary nets during the iterations. This approach, however, may not guarantee convergence within a finite number of iterations. In fact, as pointed out in [12], it is not easy to establish a formal and satisfactory proof of finite convergence for this type of problem; moreover, achieving optimal control logic is very difficult. The key reason is that the Petri net modeling framework might not be able to express the MPLE property for general process-resource nets; as a result, the problem of MPLE control synthesis based on siphon analysis in *non-ordinary* nets has not been well-resolved yet [14]. In [1], the "max-controlled-siphon-property" is proposed; however, siphon-based control synthesis by enforcing this property is not maximally permissive in general.

This paper presents formal general results on MPLE control synthesis for the class of controlled Gadara nets. Further customized algorithms can be developed for particular concurrent software applications, which are beyond the scope of this paper and will be presented in another paper [20], along with experimental results on their performance. The main contributions of this paper can be summarized as follows: (i) We present a new iterative control synthesis scheme (called ICOG) for Gadara nets; this scheme is based on structural analysis and converges in finite iterations. (ii) We develop a new algorithm (called UCCOR) for controlling siphons in Gadara nets; this algorithm uses the notion of covering of unsafe states (markings) in order to achieve greater computational efficiency. (iii) The UCCOR Algorithm accounts for uncontrollable transitions in the net in a minimally restrictive manner using the technique of constraint transformation. (iv) We establish that the proposed ICOG Methodology and the associated UCCOR Algorithm synthesize a control policy that is correct and maximally permissive with respect to the goal of liveness enforcement.

This paper is organized as follows. In Section II, the definitions and properties of Gadara nets are reviewed and their implications for control synthesis are discussed. The development of the ICOG Methodology and the UCCOR Algorithm is presented in Section III, and their main properties are established in Section IV. We discuss the customization of the proposed algorithms in Section V. Finally, we conclude in Section VI. A preliminary and partial version of the results in Sections III and IV, without proofs, appears in [17].

II. Gadara Net Model and Its Main Properties

In this section, we review the class of Gadara nets and its main properties. We assume the readers are familiar with standard Petri net definitions and notations. The readers are referred to the Appendix for some necessary background and to [22] for a detailed tutorial on Petri nets. The Appendix also provides a brief introduction to monitor-based control of Petri nets.

A. Gadara Petri Nets

Gadara nets, a new class of Petri nets introduced in [19], [32], are formally defined to model multithreaded C programs with lock allocation and release operations.

Definition 1: [19], [32] Let $I_{\mathcal{N}} = \{1, 2, \ldots, m\}$ be a finite set of indices. A Gadara net is an ordinary, self-loop-free Petri net $\mathcal{N}_G = (P, T, A, M_0)$ where

1) $P = P_0 \cup P_S \cup P_R$ is a partition such that: a) $P_S = \bigcup_{i \in I_{\mathcal{N}}} P_{S_i}$, $P_{S_i} \neq \emptyset$, and $P_{S_i} \cap P_{S_j} = \emptyset$, for all $i \neq j$; b) $P_0 = \bigcup_{i \in I_{\mathcal{N}}} P_{0_i}$, where $P_{0_i} = \{p_{0_i}\}$; and c) $P_R = \{r_1, r_2, \ldots, r_n\}$, $n > 0$.
2) $T = \bigcup_{i \in I_{\mathcal{N}}} T_i, T_i \neq \emptyset, T_i \cap T_j = \emptyset$, for all $i \neq j$.
3) For all $i \in I_{\mathcal{N}}$, the subnet $\mathcal{N}_i$ generated by $P_{S_i} \cup \{p_{0_i}\} \cup T_i$ is a strongly connected state machine. There are no direct connections between the elements of $P_{S_i} \cup \{p_{0_i}\}$ and T_j for any pair (i, j) with $i \neq j$.
4) $\forall p \in P_S$, if $|p\bullet| > 1$, then $\forall t \in p\bullet, \bullet t \cap P_R = \emptyset$.
5) For each $r \in P_R$, there exists a unique minimal-support P-semiflow, Y_r, such that $\{r\} = \|Y_r\| \cap P_R$, $(\forall p \in \|Y_r\|)(Y_r(p) = 1)$, $P_0 \cap \|Y_r\| = \emptyset$, and $P_S \cap \|Y_r\| \neq \emptyset$.
6) $\forall r \in P_R, M_0(r) = 1$, $\forall p \in P_S, M_0(p) = 0$, and $\forall p_0 \in P_0, M_0(p_0) \geq 1$.
7) $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$.

Conditions 1 and 2 characterize a set of subnets $\mathcal{N}_i$ that define work processes (i.e., software threads), called *process subnets*. The *idle place* p_{0_i} is an artificial place added to facilitate the discussion of liveness and other properties. P_S is the set of *operation places*. P_R is the set of *resource places* that model mutex locks. The readers are referred to [19], [32] for further

discussion about the definition of Gadara nets. Here, we highlight the following: Conditions 5 and 6 characterize a distinct and crucial property of Gadara nets, which is stated as follows.

Property 1: For any resource place $r \in P_R$, and its associated Y_r, we have the following semiflow equation:

$$\sum_{p \in \|Y_r\| \cap P_S} M(p) + M(r) = 1 \tag{1}$$

Or, equivalently, at any marking of the net, only one place in $\|Y_r\|$ can have a token.

Given $\mathcal{N}_G$, we wish to augment the net by synthesizing monitor places that will control the firing of transitions for the purpose of deadlock avoidance in the program. In this regard, we partition T into two disjoint subsets: $T = T_c \cup T_{uc}$, where T_c is the set of controllable transitions (which can be disabled by a monitor place), and T_{uc} is the set of uncontrollable transitions (which cannot be disabled by a monitor place). A more detailed definition of this partitioning of the transition set T is provided in the next section.

B. Controlled Gadara Nets

When we use Supervision Based on Place Invariants (SBPI) [6], [8], [33] as the control technique on a Gadara net, we obtain an augmented net that we call a controlled Gadara net, which is defined in [19], [32].

Definition 2: [19], [32] Let $\mathcal{N}_G = (P, T, A, M_0)$ be a Gadara net. A controlled Gadara net $\mathcal{N}_G^c = (P \cup P_C, T, A \cup A_C, W^c, M_0^c)$ is a self-loop-free Petri net such that, in addition to all conditions in Definition 1 for $\mathcal{N}_G$, we have

8) For each $p_c \in P_C$, there exists a unique minimal-support P-semiflow, Y_{p_c}, such that $\{p_c\} = \|Y_{p_c}\| \cap P_C$, $P_0 \cap \|Y_{p_c}\| = \emptyset$, $P_R \cap \|Y_{p_c}\| = \emptyset$, $P_S \cap \|Y_{p_c}\| \neq \emptyset$, and $Y_{p_c}(p_c) = 1$.
9) For each $p_c \in P_C$, $M_0^c(p_c) \geq \max_{p \in P_S} Y_{p_c}(p)$.

Definition 2 indicates that the introduction of the monitor places $p_c \in P_C$ preserves the net structure that is implied by Definition 1. Furthermore, we observe that the monitor places possess similar structural properties with the resource places in $\mathcal{N}_G$, but have weaker constraints. More specifically, monitor places may have multiple initial tokens and non-unit arc weights associated with their input or output arcs. A monitor place in $\mathcal{N}_G^c$ can be considered as a *generalized* resource place, which preserves the conservative nature of resources in $\mathcal{N}_G$ and has the following property.

Property 2: For any monitor place $p_c \in P_C$, and its associated Y_{p_c}, we have the following semiflow equation:

$$Y_{p_c}^T M = M_0(p_c) \tag{2}$$

The readers should notice that the weights associated with the semiflows defined by Property 2 are not necessarily equal to 1, due to the possibility that a monitor place can introduce non-unit arc weights and multiple initial tokens.

Due to the similarity between the original resource places and the synthesized monitor places, we will use the term "*generalized resource place*" to refer to any place $p \in P_R \cup P_C$.

Remark 1: From Definitions 1 and 2, we observe that $\mathcal{N}_G$ is a special subclass of $\mathcal{N}_G^c$, where $P_C = \emptyset$ and $A_C = \emptyset$. Therefore, any property that we derive for $\mathcal{N}_G^c$ holds for $\mathcal{N}_G$ as well. In the following, for the sake of simplicity, we refer to $\mathcal{N}_G^c$ as a "Gadara net" (unless special mention is made). □

As discussed in Section II-A, in general, a Gadara net has both controllable and uncontrollable transitions. In view of this, a controlled Gadara net $\mathcal{N}_G^c$ is said to be *admissible* if $P_C \bullet \cap T_{uc} = \emptyset$. In the remainder of this paper, we only consider admissible $\mathcal{N}_G^c$.

Assumption 1: $\mathcal{N}_G^c$ is admissible.

According to the semantics of the program represented by Gadara nets, branching transitions are uncontrollable (this is why we separate branching transitions from lock acquisition transitions in Condition 4 of Definition 1, i.e., resource places do not connect to branching transitions). On the other hand, lock acquisition transitions are controllable so that we can avoid deadlocks. The rest of the transitions can be classified either way, representing the "upper bound" and the "lower bound" of T_{uc}, respectively. In practice, controlling only lock acquisition transitions often result in an equally permissive but much simpler control logic, in terms of the number of arcs connected between the monitor place and $\mathcal{N}_G$.

Assumption 2: $\{t \in T : (\exists p \in P_S), (|p \bullet| > 1) \wedge (t \in p\bullet)\} \subseteq T_{uc} \subseteq T \setminus (P_R \bullet)$

The development of the results presented in this paper *only* requires that T_{uc} contains all the branch selection transitions (i.e., the lower bound in Assumption 2); these results also extend to any other choice of T_{uc} that satisfies Assumption 2.

C. Liveness Properties and Implications for Control Synthesis

First, we present some definitions that are relevant to the main properties of Gadara nets. We use $R(\mathcal{N}, M)$ to denote the set of reachable markings of net $\mathcal{N}$ starting from M.

A Petri net $(\mathcal{N}, M_0)$ is *live* if $\forall t \in T$, and $\forall M \in R(\mathcal{N}, M_0)$, there is a marking $M' \in R(\mathcal{N}, M)$ such that t is enabled at M'. A Petri net $(\mathcal{N}, M_0)$ is said to be *reversible* if $M_0 \in R(\mathcal{N}, M)$, for all $M \in R(\mathcal{N}, M_0)$. Place p is said to be a *disabling place* at marking M if there exists $t \in p\bullet$, s.t. $M(p) < W(p, t)$. A nonempty set of places S is said to be a *siphon* if $\bullet S \subseteq S\bullet$.[1]

Definition 3: A siphon S of a Gadara net $\mathcal{N}_G^c$ is said to be a *Resource-Induced Deadly Marked (RIDM) siphon* [26] at marking M, if it satisfies the following conditions:

1) every $t \in \bullet S$ is disabled by some $p \in S$ at M;
2) $S \cap (P_R \cup P_C) \neq \emptyset$;
3) $\forall p \in S \cap (P_R \cup P_C)$, p is a disabling place at M.

From Definition 3, we know that a RIDM siphon S is specified by the set of places in S and its associated partial marking $M(S)$. In general, a siphon S that satisfies Condition 2 of Definition 3 above can be rendered a RIDM siphon under more than one partial marking $M(S)$. In the following discussion, whenever we refer to a RIDM siphon S, it means S with an associated $M(S)$.

Definition 4: Given $\mathcal{N}_G^c$ and $M \in R(\mathcal{N}_G^c, M_0^c)$, the *modified marking* $\overline{M}$ is defined by

$$\overline{M}(p) = \begin{cases} M(p), & \text{if } p \notin P_0; \\ 0, & \text{if } p \in P_0. \end{cases} \tag{3}$$

[1]The notation S, when used as a subscript in P_S, refers to the type of operation places. In all other cases, unless special mention is made, S refers to a siphon.

Modified markings essentially "erase" the tokens in idle places. The set of modified markings induced by the set of reachable markings is defined by $\overline{R}(\mathcal{N}_G^c, M_0^c) = \{\overline{M} | M \in R(\mathcal{N}_G^c, M_0^c)\}$. Note that the number of tokens in idle place p_{0_i} can always be uniquely recovered from the invariant implied by the (strongly connected state machine) structure of subnet $\mathcal{N}_i$. Therefore, we have the following property.

Property 3: There is a one-to-one mapping between the original marking and the modified marking, i.e., $M_1 = M_2$ if and only if $\overline{M}_1 = \overline{M}_2$.

When it comes to liveness, the main properties of Gadara nets are formally established in [19], [32], and they serve as the foundation for the control synthesis results in the present paper.

Theorem 1: [19], [32] (Liveness and reversibility of Gadara nets)

(a) $\mathcal{N}_G^c$ is live *iff* it is reversible.

(b) $\mathcal{N}_G$ is live *iff* there does not exist a marking $M \in R(\mathcal{N}_G, M_0)$ and a siphon S such that S is an empty siphon at M.

(c) $\mathcal{N}_G^c$ is live *iff* there does not exist a modified marking $\overline{M} \in \overline{R}(\mathcal{N}_G^c, M_0^c)$ and a siphon S such that S is a RIDM siphon at $\overline{M}$.

We have formally shown in [19] that a multithreaded program that can be modeled as a Gadara net $\mathcal{N}_G^c$ is deadlock-free *if and only if* $\mathcal{N}_G^c$ is reversible. According to Theorem 1(a), reversibility and liveness are equivalent in Gadara nets. Therefore, deadlock-freeness of the program corresponds to liveness of the Gadara net. This correspondence is the primary motivation for our study of the liveness-enforcing control of Gadara nets. As established in Theorem 1(b), the liveness of $\mathcal{N}_G$ is guaranteed when $\mathcal{N}_G$ cannot reach a marking M under which some siphon S is empty. Thus, in control synthesis, we need to prevent all the siphons in $\mathcal{N}_G$ from becoming empty by adding appropriate monitor places. As discussed in Section I, in general we need to iterate the control synthesis process. When $\mathcal{N}_G^c$ remains ordinary, we can carry out control synthesis in the way similar to $\mathcal{N}_G$.

When $\mathcal{N}_G^c$ becomes non-ordinary, we need Theorem 1(c) to guide control synthesis. Theorem 1(c) characterizes the liveness of $\mathcal{N}_G^c$ by a more general type of siphon, namely the RIDM siphon, under the modified markings. A RIDM siphon can be nonempty. An empty siphon is a special case of RIDM siphon. Fig. 1 shows an example of a nonempty RIDM siphon $S = \{p_{c_1}, p_{c_2}, p_{12}, p_{13}, p_{22}, p_{23}\}$. This example implies that simply preventing the siphons from becoming empty is *not* sufficient for the control synthesis in non-ordinary $\mathcal{N}_G^c$. Therefore, in non-ordinary $\mathcal{N}_G^c$, we need to consider all the RIDM siphons that are present in the modified markings of the net.

The above discussion implies that the problem of deadlock avoidance in a multithreaded program is *equivalent* to the problem of preventing any RIDM siphon (resp., empty siphon) from becoming reachable in the modified reachability space (resp., original reachability space) of its Gadara net model $\mathcal{N}_G^c$ (resp., $\mathcal{N}_G$). Since $\mathcal{N}_G^c$ represents the most general subclass of Gadara nets, we will focus on control synthesis for $\mathcal{N}_G^c$ in the next section; the derived results can also be applied to $\mathcal{N}_G$. We formally state our problem as follows.

Problem statement: Given a controlled Gadara net, find a monitor-based control policy such that the resulting controlled Gadara net is admissible, live, and maximally permissive with respect to the goal of liveness enforcement.

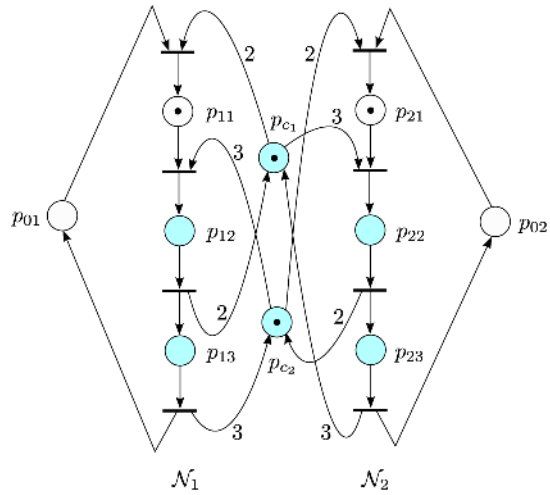


Fig. 1. Example of a nonempty RIDM siphon.

Remark 2: We briefly discuss the existence of a solution to the aforementioned problem. From the viewpoint of an automaton model, if we construct the reachability graph (i.e., an automaton model) of a Gadara net and only mark its initial state, then the coaccessible part of this automaton [3] will not be empty. This is because a single instance from any given process subnet can always execute to completion, in isolation. On the other hand, according to Theorem 1(a), if a Gadara net can always return to its initial marking, then it is live. Therefore, the simple control policy that executes all threads sequentially is necessarily live, thereby proving that a liveness-enforcing control policy always exists. This control policy is also admissible because it can be realized by connecting an outgoing arc of a monitor place, with one initial token, to the first lock acquisition transition of each process subnet (which is not an uncontrollable transition by Assumption 2), and returning this token to the monitor place only at the last transition of each process subnet. In Section III-C-3 we will also show that a *maximally permissive* control policy using monitor places always exists in Gadara nets. □

III. Control Synthesis for Gadara Nets: Algorithms

In this section, we present a new MPLE control synthesis methodology for general controlled Gadara nets that need not be ordinary. The proposed methodology exploits the structural properties of Gadara nets and enforces liveness by preventing RIDM siphons from being reachable. We will use a running example, depicted in Fig. 2, to facilitate our discussion. The net structure shown in solid lines is the original Gadara net before control; the net structure shown in dashed lines represents the monitor places that are synthesized using the algorithms to be presented next.

A. Motivation

We first briefly discuss the motivation of our investigation of the MPLE control of $\mathcal{N}_G^c$.

A non-ordinary $\mathcal{N}_G^c$ can arise from various reasons in applications. For example, a non-ordinary Gadara net may be the result of enforcing other properties on multithreaded programs, like

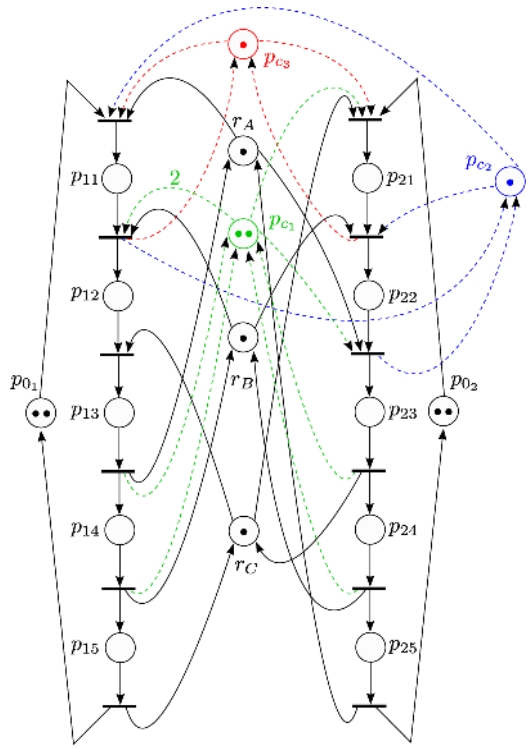


Fig. 2. Running example of control synthesis. The original Gadara net is shown with solid lines, and denoted as $\mathcal{N}_G$. The synthesized monitor places and their associated arcs are shown with dashed lines. We define three controlled Gadara nets of interest: (i) $\mathcal{N}_G^{c(1)}$ consists of $\mathcal{N}_G$ and p_{c1}; (ii) $\mathcal{N}_G^{c(2)}$ consists of $\mathcal{N}_G$, p_{c1}, and p_{c2}; and (iii) $\mathcal{N}_G^{c(3)}$ consists of $\mathcal{N}_G$, p_{c1}, p_{c2}, and p_{c3}.

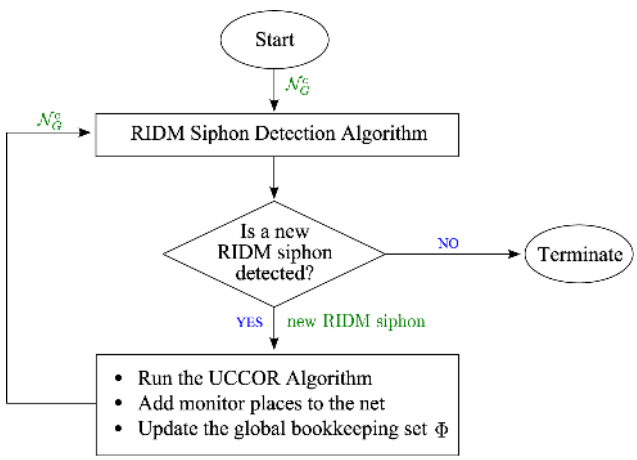


Fig. 3. Iterative control of controlled Gadara nets (ICOG).

atomicity [10], prior to the control synthesis presented in this paper, where more general types of specifications expressible as linear inequalities may be enforced upon the net. In general, the enforcement of such linear inequalities (e.g., by the SBPI technique) may result in monitor places that have non-unit arc weights.

Example 1: Consider the running example as shown in Fig. 2. The original Gadara net, denoted as $\mathcal{N}_G$, is shown in solid lines. Prior to ICOG, the following specification[2] was enforced upon $\mathcal{N}_G$ by using SBPI:

$$M(r_A)+M(r_B)+M(r_C)+M(p_{11})+M(p_{13})+M(p_{14}) +M(p_{15})+M(p_{22})+M(p_{23})+M(p_{24})+M(p_{25}) \geq 1 \quad (4)$$

The synthesized monitor place is denoted as p_{c1} and shown in dashed lines. The resulting net, which consists of $\mathcal{N}_G$, p_{c1}, and its associated arcs, is a controlled Gadara net, denoted as $\mathcal{N}_G^{c(1)}$. Note that $\mathcal{N}_G^{c(1)}$ is non-ordinary, due to the introduction of p_{c1}. Therefore, to fully resolve liveness enforcement in a maximally permissive manner for $\mathcal{N}_G^{c(1)}$, a general MPLE control synthesis methodology that works for non-ordinary Gadara nets is required. □

B. Overall Strategy—Iterative Control of Gadara Nets

We propose an Iterative Control Of Gadara nets (ICOG) Methodology, with a net in the class of $\mathcal{N}_G^c$ as the initial condition. The flowchart of ICOG is shown in Fig. 3. Given a controlled Gadara net, we first see if there is any new RIDM siphon under the modified markings of the net. If no RIDM siphon is detected, then, according to Theorem 1(c), the net is live and ICOG terminates. Otherwise, we synthesize control logic to prevent the detected RIDM siphon from becoming reachable, by using an algorithm, called UCCOR, to be presented next. The UCCOR Algorithm outputs a set of monitor places, which are added to the net. After UCCOR, we go back to the first step of ICOG and determine if there are any remaining or *new* RIDM siphons. One important feature of the proposed ICOG is that we maintain a "global bookkeeping set", denoted by Φ, throughout the iterations. The set Φ records all the control syntheses that have been carried out in terms of prevented unsafe coverings, which will be introduced shortly.

ICOG is an iterative process in general, because there may be some siphons that have not been identified in the previous iterations and need further consideration. Moreover, we explained above that the added monitor place can be considered as a generalized resource place, and may introduce new potential deadlocks.

Note that the ICOG Methodology is fully modular so that the detection of RIDM siphons is not associated with any specific algorithm. This can be done, for instance, by using a Mixed Integer Programming (MIP) based approach that finds a maximal RIDM siphon in the net [26]. In the case of an ordinary net, the MIP technique has also been employed to detect a maximal empty siphon in the net [4]. We have developed a set of customized and efficient MIP formulations for RIDM siphon detection in general Gadara nets and empty siphon detection in ordinary Gadara nets [18], [19]. Moreover, siphons can also be detected via structural analysis; a recent result on siphon detection in S^4PR nets using graph theory is presented in [2].

We emphasize that the RIDM siphon detection is carried out under the *modified markings*, due to Theorem 1(c). The detected RIDM siphon, say S, will be characterized by the set of places S, and an associated partial modified marking on S.

[2]The set of places involved in the left-hand-side of (4) consists of a maximal empty siphon, obtained from the siphon detection algorithm presented in [4]. The rationale of (4) was to attempt to address liveness enforcement by preventing this maximal empty siphon from being reachable.

C. Fundamentals of the UCCOR Algorithm

We propose a new algorithm, used as a module of ICOG, for preventing RIDM siphons in $\mathcal{N}_G^c$. We call it the *UCCOR Algorithm*, where UCCOR is short for "Unsafe-Covering-based Control Of RIDM siphons". (The notion of unsafe covering induced by a RIDM siphon will be introduced in Section III-C-2.)

1) Definitions and Partial-Marking Analysis: Similarly to the modified marking defined in Section II-C, we also define the P_S-marking to facilitate the discussion.

Definition 5: Given $\mathcal{N}_G^c$ and $M \in R(\mathcal{N}_G^c, M_0^c)$, the *$P_S$-marking* $\overline{\overline{M}}$ is defined by

$$\overline{\overline{M}}(p) = \begin{cases} M(p), & \text{if } p \in P_S; \\ 0, & \text{if } p \notin P_S. \end{cases} \tag{5}$$

P_S-markings essentially "erase" the tokens in idle places and generalized resource places, retaining only tokens in operation places. The P_S-marking does not introduce any ambiguity. More specifically, given the P_S-marking $\overline{\overline{M}}$ corresponding to the original marking M, the number of tokens in places P_R and P_C under M can be uniquely recovered by solving the equations given in Properties 1 and 2, respectively. Therefore, combining this result with Property 3 of the modified markings, we have the following property.

Property 4: There is a one-to-one mapping between the original marking and the P_S-marking, i.e., $M_1 = M_2$ if and only if $\overline{\overline{M}}_1 = \overline{\overline{M}}_2$.

As revealed by Properties 3 and 4, there is a one-to-one mapping among the original marking, modified marking, and P_S-marking. Thus, in the UCCOR Algorithm, when synthesizing linear inequality specifications for monitor-based control, we can focus our attention on $\overline{\overline{M}}$ only, and the coefficients in the linear inequalities corresponding to places P_0, P_R, and P_C are all zero, i.e., they are "don't care" terms in the linear inequalities. We observe that Conditions 5, 6, and 7 of Definition 1 imply that $\overline{\overline{M}}$ is always a *binary* vector. It is this property that motivates us to focus on $\overline{\overline{M}}$.

Through the UCCOR Algorithm, essentially we want to synthesize control logic that can prevent the net from reaching any unsafe marking with respect to RIDM siphons. The next definition concretizes this concept.

Definition 6: A marking M is said to be a *RIDM-unsafe* marking if there exists at least one RIDM siphon at the corresponding modified marking $\overline{M}$. Given a siphon S, a marking M is said to be a RIDM-unsafe marking *with respect to* S, if S is a RIDM siphon at marking $\overline{M}$.

From Definition 6 and Theorem 1(c), we immediately have:

Corollary 1: $\mathcal{N}_G^c$ is live *iff* it cannot reach a marking that is a RIDM-unsafe marking.

Example 2: Let us refer to the controlled Gadara net $\mathcal{N}_G^{c(1)}$ in Fig. 2, and consider the following two markings (for the sake of simplicity, we only specify the marked places; the unspecified places are empty by default): (i) M_{u1}, where p_{01}, p_{02}, p_{11}, and p_{22} each have one token, and (ii) M_{u2}, where p_{01}, p_{02}, p_{11}, and p_{21} each have one token. In this example, the marking M_{u1} is RIDM-unsafe, and the siphon $S_1 = \{r_A, r_B, r_C, p_{12}, p_{13}, p_{14}, p_{15}, p_{21}, p_{23}, p_{24}, p_{25}\}$ is a RIDM siphon at M_{u1}. The marking M_{u2} is not RIDM-unsafe, but starting from M_{u2}, the net cannot go back to the initial marking and can only go to a RIDM-unsafe marking. Therefore, both M_{u1} and M_{u2} should be prevented by control synthesis.

As we will see in the following discussion, the latter type of markings (such as M_{u2} in this example) will be eventually exposed as RIDM-unsafe markings as the iterations evolve. Thus, in the rest of this section, we can focus our attention on RIDM-unsafe markings. □

From the above discussion, for any given RIDM-unsafe marking M_u, it is the *partial modified marking* $\overline{M}_u(S)$ on the RIDM siphon S that is critical to the lack of safety. Here, $\overline{M}_u(S)$ is a column vector with $|S|$ entries corresponding to the places in S, and the subscript "u" denotes "RIDM-unsafe". In other words, if we know that S is a RIDM siphon, and an associated partial modified marking is $\overline{M}_u(S)$, then *any* (full) marking M, such that $\overline{M}(S) = \overline{M}_u(S)$, must also be a RIDM-unsafe marking with respect to S. This leads to the following result.

Proposition 1: Given a RIDM siphon S, and an associated partial modified marking $\overline{M}_u(S)$, any marking M such that $\overline{M}(S) = \overline{M}_u(S)$, is RIDM-unsafe with respect to S.

Thus, in the control synthesis, we want to prevent *any* marking M such that $\overline{M}(S) = \overline{M}_u(S)$. This is achieved by considering RIDM-unsafe *partial* markings in a way that each synthesized monitor place can prevent more than one RIDM-unsafe marking. As we mentioned, the control will be implemented on P_S-markings. From Proposition 1, we observe that the partial modified marking $\overline{M}_u(S)$ is sufficient to characterize the corresponding RIDM-unsafe markings with respect to S. However, this is not true for partial P_S-marking $\overline{\overline{M}}_u(S)$. Consider the siphon $S = \{p_{c_1}, p_{c_2}, p_{12}, p_{13}, p_{22}, p_{23}\}$ in Fig. 1 that we discussed earlier. Since S is a RIDM siphon, in this case we know that the current marking of the net, say M, is RIDM-unsafe with respect to S. On the other hand, Fig. 7 (without considering the dashed lines) shows the same net under its initial marking M_0. M_0 is not RIDM-unsafe by assumption.[3] But, we observe that $\overline{\overline{M}}(S) = \overline{\overline{M}}_0(S)$. This is because from the partial P_S-marking $\overline{\overline{M}}_u(S)$, one cannot tell the "status" of the resources (namely, tokens) in $S \cap (P_R \cup P_C)$. Intuitively, we want to consider more places under the partial P_S-marking. This deficiency can be made up by further considering the partial P_S-marking on the supports of minimal semiflows associated with $S \cap (P_R \cup P_C)$, which are introduced as follows.

The minimal-support P-semiflow for any generalized resource place is a well-defined concept in Petri nets [22] (see Appendix). This concept can be extended for any resource-induced siphon; for the sake of discussion, we introduce the notation, $\|\tilde{Y}_S\|$, as follows:

$$\|\tilde{Y}_S\| = \bigcup_{p \in S \cap (P_R \cup P_C)} \|Y_p\|$$

where, Y_p is the minimal-support P-semiflow of p.

[3]More specifically, this statement is true since no place in $P_R \cup P_C$ can be a disabling place at M_0.

Property 5: For any resource-induced siphon S, the corresponding $\|\tilde{Y}_S\|$ is unique.

Based on Properties 1 and 2, starting from a partial P_S-marking on $\|\tilde{Y}_S\|$, one can uniquely recover the tokens in $S \cap (P_R \cup P_C)$. This observation, together with Proposition 1, implies that the partial P_S-marking $\overline{\overline{M}}_u(S \cup \|\tilde{Y}_S\|)$ (or, equivalently, $\overline{\overline{M}}_u((S \cup \|\tilde{Y}_S\|) \cap P_S)$ since the P_S-marking only considers tokens in P_S), is sufficient to characterize the RIDM-unsafe markings with respect to S. For simplicity, we define $\Theta_S := (S \cup \|\tilde{Y}_S\|) \cap P_S$. This leads to our next result.

Proposition 2: Given a RIDM siphon S, and an associated partial modified marking $\overline{M}_u(\Theta_S)$, any marking M such that $\overline{\overline{M}}(\Theta_S) = \overline{\overline{M}}_u(\Theta_S)$, is RIDM-unsafe with respect to S.

Remark 3: Proposition 2 bridges the notion of partial *modified marking* on S, which is obtained in the RIDM siphon detection, and the notion of partial P_S*-marking* on S, which is used in the control synthesis. It also implies that the P_S-marking of any $p \notin \Theta_S$ is a "*don't care*" term in the control synthesis, i.e., the coefficient associated with it in the linear inequality that will prevent siphon S is 0. The partial P_S-marking analysis is further facilitated by the notion of covering, which is introduced next. □

2) Notion of Covering: We introduce the notation "χ" for the value of a P_S-marking component, where "χ" stands for "0 or 1".

Definition 7: In $\mathcal{N}_G^c$, a *covering* C is a generalized P_S-marking, whose components can be 0, 1, or χ.

For any place $p \in P_S$, $C(p)$ represents the covering component value on p. This notation can be extended to a set of places $Q \subseteq P_S$ in a natural way. Furthermore, we extend the notion of covering so that it encompasses any place $p \in P$ by setting $C(p) = \chi, \forall p \in P_0 \cup P_R \cup P_C$.

Given two coverings C_1 and C_2, we say that C_1 *covers* C_2, denoted as $C_1 \succeq C_2$, if $\forall p \in P_S$ such that $C_1(p) \neq C_2(p)$, $C_1(p) = \chi$. As a special case, if $C_1 = C_2$, then we have $C_1 \succeq C_2$ and $C_2 \succeq C_1$. The "cover" relationship between a covering and a P_S-marking, which have the same dimensions, is defined in a similar way. For example, for a binary marking vector $[p_1, p_2, p_3]^T$, $C = [1, \chi, 1]^T$ *covers* the P_S-markings $\overline{\overline{M}}_1 = [1, 0, 1]^T$ and $\overline{\overline{M}}_2 = [1, 1, 1]^T$. A covering C is said to be a *RIDM-unsafe* covering if for all P_S-markings $\overline{\overline{M}}$ it covers, the corresponding M is RIDM-unsafe.

Remark 4: As a result of Proposition 2 and the notion of covering, for any RIDM siphon S to be prevented, the control synthesis *only* needs to consider the set of places Θ_S, and the associated RIDM-unsafe covering, $C(\Theta_S)$, and $C(p) = \chi$, $\forall p \notin \Theta_S$. □

Remark 5: By Definition 7, a covering is a generalized P_S-marking. So the component values in a covering can only be 0, 1, or χ. In the context of control synthesis, χ is a "*don't care*" term, and the coefficient associated with it in the corresponding linear inequality will always be 0. □

3) Feasibility of Maximally Permissive Control: In [19], [32], we have established a "convexity-type" property of Gadara nets. This property is based on the *binary* nature of the P_S-markings and it states that, in these nets, any set of reachable markings can be separated from the rest through a set of linear inequalities, which are provided in the constructive proof of Theorem 6 in [32]. These linear inequalities can be subsequently enforced upon the original net through monitor places. Following Remarks 4 and 5, this property can be generalized to any set of RIDM-unsafe coverings with respect to some given RIDM siphon S.

Algorithm: UCCOR Algorithm
Input: $\mathcal{N}_G^c$, RIDM siphon S, and an associated partial modified marking on S
Output: A set of monitor place(s) to prevent S
Method:
1. Take the RIDM siphon S and the provided partial modified marking on S as the input to the *Unsafe Covering Construction Algorithm*, and obtain a set of RIDM-unsafe coverings with respect to S, denoted as $\mathcal{C}_u$.
2. Take $\mathcal{C}_u$ as the input to the *Unsafe Covering Generalization*, and obtain the output, denoted as $\mathcal{C}_u^{(1)}$.
3. Take $\mathcal{C}_u^{(1)}$ as the input to the *Inter-Iteration Coverability Check*, and obtain the output, denoted as $\mathcal{C}_u^{(2)}$.
4a. If $\mathcal{C}_u^{(2)} = \emptyset$, then output the empty set $\emptyset$ and terminate.
4b. If $\mathcal{C}_u^{(2)} \neq \emptyset$, then take $\mathcal{C}_u^{(2)}$ as the input to the *Monitor Place Synthesis Algorithm*, which will synthesize a monitor place for each element in $\mathcal{C}_u^{(2)}$.

Fig. 4. UCCOR Algorithm.

Theorem 2: In $\mathcal{N}_G^c$, for any RIDM siphon S, the set of all RIDM-unsafe coverings with respect to S can be separated by a finite set of linear inequality constraints $\Lambda = \{(l_1, b_1), (l_2, b_2), \ldots\}$ such that a covering C is RIDM-unsafe with respect to S *iff* $\exists (l_i, b_i) \in \Lambda, l_i^T C > b_i$.

Theorem 2 implies that it is feasible to implement maximally permissive control using monitor-based control in terms of RIDM-unsafe coverings. More specifically, for a given covering C we want to prevent, its associated linear inequality can be specified as: $l_C(p) = 1$, if $C(p) = 1$; $l_C(p) = -1$, if $C(p) = 0$; $l_C(p) = 0$, if $C(p) = \chi$; and, $b_C = \sum_{p:p\in\Theta_S \text{ and } C(p)=1} C(p) - 1$.

D. UCCOR Algorithm

We now formally present the UCCOR Algorithm. Our presentation is organized in a top-down manner. We first give the overall procedure of the UCCOR Algorithm in Fig. 4, and then explain the embedded modules in subsequent sections. We will apply the UCCOR Algorithm to $\mathcal{N}_G^{c(1)}$, which is the controlled Gadara net with the monitor place p_{c1} shown in Fig. 2, to illustrate the steps of UCCOR.

The input to the algorithm is $\mathcal{N}_G^c$, a RIDM siphon S, and an associated partial modified marking $\overline{M}_u(S)$. In Step 1, the Unsafe Covering Construction Algorithm is used to solve for a set of possible RIDM-unsafe coverings with respect to S, denoted as $\mathcal{C}_u$. As a result of Step 1 and Propositions 1 and 2, any RIDM-unsafe marking M with respect to S, such that $\overline{\overline{M}}(S) = \overline{\overline{M}}_u(S)$, is captured by $\mathcal{C}_u$. In Step 2, $\mathcal{C}_u$ is taken as the input to the Unsafe Covering Generalization. This step further generalizes the RIDM-unsafe coverings obtained from Step 1, by utilizing a certain type of monotonicity property of Gadara nets. It outputs a modified set of coverings, $\mathcal{C}_u^{(1)}$, which is taken as the input to the Inter-Iteration Coverability Check carried out in Step 3. In Step 3, the coverings that have already been controlled are removed from consideration. The output of this step

is a further modified set of coverings, $\mathcal{C}_u^{(2)}$. In Step 4, if $\mathcal{C}_u^{(2)}$ is an empty set, then the algorithm terminates; otherwise, control synthesis using SBPI is carried out. One monitor place will be synthesized for each covering in $\mathcal{C}_u^{(2)}$.

Define Φ to be the set of coverings that have already been prevented in the previous iterations. One can think of Φ as a global "bookkeeping set" in the control synthesis process, which records all the coverings that have been prevented so far. The set Φ helps us to determine the convergence of ICOG. Since Φ only needs to record a relatively *small* number of *coverings* to keep track of a potentially much *larger* number of *markings* that need to be prevented, the complexity of the bookkeeping process is greatly reduced—a saving on both time and space. The set Φ is updated by the UCCOR Algorithm during the Inter-Iteration Coverability Check in Step 3 discussed below. In addition, Φ is also updated after the termination of the UCCOR Algorithm, i.e., $\Phi = \Phi \cup \mathcal{C}_u^{(2)}$, to include the coverings that are prevented in this iteration.

E. Unsafe Covering Construction Algorithm

From the input of the UCCOR Algorithm, we know the RIDM siphon S and an associated partial modified marking $\overline{M}_u(S)$. As discussed above, we want to find the RIDM-unsafe coverings that cover any possible RIDM-unsafe marking M, such that $\overline{M}(S) = \overline{M}_u(S)$. The desired RIDM-unsafe coverings are obtained in the Unsafe Covering Construction Algorithm, which is described as follows.

First, for each generalized resource place in S, there is an associated P-semiflow equation. Denote the set of all such equations associated with $S \cap (P_R \cup P_C)$ as $\mathcal{V}$. Secondly, substitute the unknown variables in $\mathcal{V}$ corresponding to places $S \cap \|\tilde{Y}_S\|$ using the values specified by $\overline{M}_u(S)$. The set of updated equations is denoted as $\mathcal{V}'$. Thirdly, solve $\mathcal{V}'$, together with the constraint that $M(p) \in \{0,1\}$, $\forall p \in \|\tilde{Y}_S\| \setminus S$. The set of solutions of $\mathcal{V}'$ are denoted as $\mathcal{M}_u(\|\tilde{Y}_S\|)$, which is a set of partial markings on $\|\tilde{Y}_S\|$. Finally, construct the RIDM-unsafe coverings based on the obtained $\mathcal{M}_u(\|\tilde{Y}_S\|)$ and the given $\overline{M}_u(S)$. For each $M \in \mathcal{M}_u(\|\tilde{Y}_S\|)$, define the corresponding covering C with a dimension of $|P| \times 1$ as follows: (i) $C(\|\tilde{Y}_S\| \cap P_S) = \overline{\overline{M}}(\|\tilde{Y}_S\| \cap P_S)$; (ii) $C((S \setminus \|\tilde{Y}_S\|) \cap P_S) = \overline{M}_u((S \setminus \|\tilde{Y}_S\|) \cap P_S)$; and, (iii) $C(p) = \chi$, $\forall p \notin \Theta_S$. The resulting set of coverings is the output of this algorithm, denoted as $\mathcal{C}_u$.

Remark 6: Observe that for any $C \in \mathcal{C}_u$, C is a RIDM-unsafe covering with respect to S. Thus, for any P_S-marking $\overline{\overline{M}}$ that is covered by C, the corresponding original marking M is also RIDM-unsafe with respect to S. Moreover, C only specifies binary values for the places in Θ_S, and the other places not in Θ_S are irrelevant to the analysis of the RIDM siphon S under the notion of covering. □

Example 3: Consider the net $\mathcal{N}_G^{c(1)}$ in Fig. 2. We use $\mathcal{N}_G^{c(1)}$ as the initial condition of ICOG. The first iteration of ICOG detects a RIDM siphon:

$$S_1 = \{r_A, r_B, r_C, p_{12}, p_{13}, p_{14}, p_{15}, p_{21}, p_{23}, p_{24}, p_{25}\} \quad (6)$$

at the marking M_{u1}, as described in Example 2. For this example, Step 1 of UCCOR solves the set of semiflow equations $\mathcal{V}$ that contains three equations: $M(r_A) + M(p_{11}) + M(p_{12}) + M(p_{13}) + M(p_{23}) + M(p_{24}) + M(p_{25}) = 1$, $M(r_B) + M(p_{12}) + M(p_{13}) + M(p_{14}) + M(p_{22}) + M(p_{23}) + M(p_{24}) = 1$, and $M(r_C) + M(p_{13}) + M(p_{14}) + M(p_{15}) + M(p_{21}) + M(p_{22}) + M(p_{23}) = 1$. Using the information from $\overline{M}_{u1}(S_1)$, the updated set of equations $\mathcal{V}'$ is: $M(p_{11}) = 1$, and $M(p_{22}) = 1$. Thus, Step 1 finally outputs the set $\mathcal{C}_u$ that contains one RIDM-unsafe covering C, where $C(p_{11}) = C(p_{22}) = 1$, $C(p_{1i}) = C(p_{2j}) = 0$, for $i = 2, 3, 4, 5$ and $j = 1, 3, 4, 5$, and $C(p_{01}) = C(p_{02}) = C(r_A) = C(r_B) = C(r_C) = \chi$. □

F. Unsafe Covering Generalization

Given the set of possible RIDM-unsafe coverings $\mathcal{C}_u$ with respect to S, the Unsafe Covering Generalization *generalizes* $\mathcal{C}_u$ and outputs a modified set of coverings $\mathcal{C}_u^{(1)}$.

Given two markings M_1 and M_2, we say that "M_1 *dominates* M_2", denoted by $M_1 >_d M_2$, if the following two conditions are satisfied: (i) $M_1(p) \geq M_2(p)$, for all $p \in P$, and (ii) $M_1(q) > M_2(q)$, for at least some $q \in P$. The dominance relationship between two coverings C_1 and C_2 can be defined in a similar way by substituting "M" above by "C". Note that "χ", as a covering component, stands for "0 or 1". So, we have: $1 \geq \chi \geq 0$. Moreover, if $C_1 >_d C_2$, then Condition (ii) above can only be satisfied by the case when $C_1(q) = 1$ and $C_2(q) = 0$.

The following theorem is closely related to the *monotonicity property* of state safety in resource allocation systems [27].

Theorem 3: Consider a Gadara net $\mathcal{N}_G^c$, and a marking M of it that satisfies the net semiflow equations (1) and (2) but cannot reach M_0^c. Then, any marking M' that satisfies all the semiflow equations (1) and (2) and $\overline{\overline{M'}} >_d \overline{\overline{M}}$, cannot reach M_0^c either.

Proof: We prove the contra-positive proposition, i.e., we prove that if M' can reach M_0^c and satisfies all the semiflow equations (1) and (2), then any marking M that satisfies all the semiflow equations (1) and (2) and $\overline{\overline{M'}} >_d \overline{\overline{M}}$, can also reach M_0^c.

By assumption, starting from M', there exists a feasible firing transition sequence σ', which will lead the net from M' to M_0^c. Furthermore, since both markings M and M' satisfy all the semiflow equations (1) and (2) and $\overline{\overline{M'}} >_d \overline{\overline{M}}$, Properties 1 and 2 imply that $M(r) \geq M'(r)$, $\forall r \in P_R \cup P_C$. That is, the net under M contains only a subset of the processes that are active in M', and it is "resource richer". Thus, starting from M, there also exists a feasible firing transition sequence σ, which will lead the net from M to M_0^c; such a sequence σ can be obtained from σ' by "erasing" the set of transitions that are fired by the extra tokens in P_S under M' as compared to M, and the feasibility of σ under M can be formally established by an induction on the length of the sequence. ■

An immediate corollary of Theorem 3 is as follows:

Corollary 2: Consider a Gadara net $\mathcal{N}_G^c$, and a marking M that is RIDM-unsafe and satisfies all the semiflow equations (1) and (2). Then, any marking M' that satisfies all the semiflow equations (1) and (2) and $\overline{\overline{M'}} >_d \overline{\overline{M}}$, cannot reach M_0^c.

Remark 7: From Proposition 2 and its associated discussion, we know that only the set of places $S \cup \|\tilde{Y}_S\|$ is relevant to the analysis of siphon S (or equivalently, under the notion of P_S-marking, only the set of places Θ_S is relevant). Note that

$(S \cup \|\tilde{Y}_S\|) \cap (P_R \cup P_C) = S \cap (P_R \cup P_C)$. This implies that Corollary 2 still holds if we replace the condition "satisfies all the semiflow equations (1) and (2)" on M and M', by the condition "satisfies all the semiflow equations associated with $S \cap (P_R \cup P_C)$." □

In Step 1 of UCCOR, we obtain the set of RIDM-unsafe coverings $\mathcal{C}_u$ with respect to S. According to Remark 6, for *any* $C_1 \in \mathcal{C}_u$, and *any* M_1, such that $C_1 \succeq \overline{\overline{M}}_1$, M_1 is RIDM-unsafe with respect to S. Due to the construction of $\mathcal{C}_u$ in Step 1 of UCCOR, M_1 satisfies all the semiflow equations associated with $S \cap (P_R \cup P_C)$. Consider the partial marking $M_1(\Theta_S)$. If there exists at least one "0" component in $M_1(\Theta_S)$, we replace *any* subset of the "0" components in $M_1(\Theta_S)$ by "1", and leave the other components in M_1 unchanged. The resulting marking is denoted as M_2, and it is obvious that $\overline{\overline{M}}_2 >_d \overline{\overline{M}}_1$. Therefore, M_2 either does not satisfy the semiflow equations associated with $S \cap (P_R \cup P_C)$ (and hence is not reachable), or satisfies the semiflow equations associated with $S \cap (P_R \cup P_C)$ and cannot reach M_0^c (based on Corollary 2 and Remark 7).

As a consequence, for a given covering $C \in \mathcal{C}_u$ that needs to be prevented, any covering C', such that $C' >_d C$, can also be prevented. Therefore, all the 0 components in C can be replaced by χ, and the resulting covering is denoted as C', where $C' \succeq C$. In the control synthesis, we can prevent C' instead of C.

In the Unsafe Covering Generalization, we "generalize" each $C \in \mathcal{C}_u$ by replacing all the 0 components in C by χ, and obtain a corresponding modified covering $C^{(1)}$. The resulting set of modified coverings is denoted as $\mathcal{C}_u^{(1)'}$. Consequently, the elements in $\mathcal{C}_u$ and those in $\mathcal{C}_u^{(1)'}$ are in one-to-one correspondence. Observe that any corresponding pair $(C, C^{(1)})$, where $C \in \mathcal{C}_u$ and $C^{(1)} \in \mathcal{C}_u^{(1)'}$, satisfies: $C^{(1)} \succeq C$. Therefore, by considering the set of modified coverings $\mathcal{C}_u^{(1)'}$ afterwards in the UCCOR Algorithm, we will not "miss" preventing any element in $\mathcal{C}_u$ due to this coverability relationship. Moreover, the property of maximal permissiveness is still preserved, i.e., we only prevent reachable markings that cannot reach M_0^c, or markings that are not reachable, due to the above discussion.

Furthermore, we determine if there exists a pair of coverings (C_1, C_2), such that $C_1, C_2 \in \mathcal{C}_u^{(1)'}$ and $C_1 \succeq C_2$. (i) If such a pair is detected, then we perform $\mathcal{C}_u^{(1)'} = \mathcal{C}_u^{(1)'} \setminus \{C_2\}$, and repeat the process in the updated $\mathcal{C}_u^{(1)'}$. (ii) If no pair is detected, then we output the set $\mathcal{C}_u^{(1)} := \mathcal{C}_u^{(1)'}$. Note that $\mathcal{C}_u^{(1)}$ and $\mathcal{C}_u^{(1)'}$ have the same power of coverability, because the operations performed above simply remove the "redundant" coverings in the set $\mathcal{C}_u^{(1)'}$.

Example 4: Let us continue the example of applying UCCOR to the net $\mathcal{N}_G^{c(1)}$ shown in Fig. 2. In Step 2 of UCCOR, for this example, the set $\mathcal{C}_u^{(1)}$ contains one covering C_1, where $C_1(p_{11}) = C_1(p_{22}) = 1$ and $C_1(p) = \chi$, for any $p \in P \setminus \{p_{11}, p_{22}\}$. □

Clearly, $\mathcal{C}_u^{(1)}$ will cover, in general, a larger set of markings than $\mathcal{C}_u$ does. Thus, by considering $\mathcal{C}_u^{(1)}$ in the UCCOR Algorithm, the synthesized monitor places are more efficient, in terms of the number of markings that they can prevent. As we mentioned, some markings covered by $\mathcal{C}_u^{(1)}$ may not be reachable, however, the property of maximal permissiveness is not compromised because of this.

G. Inter-Iteration Coverability Check

In the Inter-Iteration Coverability Check, each pair of coverings $(C_1, C_2) \in \{(C_1, C_2) : C_1 \in \mathcal{C}_u^{(1)} \text{ and } C_2 \in \Phi\}$ is tested. (i) If $C_1 \preceq C_2$, then the existing monitor place associated with $C_2 \in \Phi$ already prevents C_1, and we perform: $\mathcal{C}_u^{(1)} = \mathcal{C}_u^{(1)} \setminus \{C_1\}$. (ii) If $C_1 \succeq C_2$ and $C_1 \neq C_2$, then by synthesizing a new monitor place in the current iteration that prevents C_1, this monitor place will also prevent $C_2 \in \Phi$. That is, the existing monitor place associated with C_2 will become redundant after the current iteration. In this case, we perform: $\Phi = \Phi \setminus \{C_2\}$, and remove the existing monitor place (and its incoming and outgoing arcs) associated with C_2 from the net. (iii) If C_1 and C_2 are incomparable, then no action is performed. The algorithm finally outputs a modified set of coverings corresponding to $\mathcal{C}_u^{(1)}$, denoted as $\mathcal{C}_u^{(2)}$, and updates Φ.

Example 5: We continue the discussion on the running example. The set Φ is initialized as an empty set before the first iteration of ICOG. Thus, in the first iteration of ICOG, no action is needed in Step 3 of UCCOR. Ater this step of UCCOR, we have: $\mathcal{C}_u^{(2)} = \{C_1\}$ and $\Phi = \emptyset$. □

H. Monitor Place Synthesis Algorithm

In Step 4 of UCCOR, if the set $\mathcal{C}_u^{(2)}$ is empty, then we terminate the algorithm and start the next iteration of ICOG. If the set $\mathcal{C}_u^{(2)}$ is not empty, then for each covering in $\mathcal{C}_u^{(2)}$, a monitor place is synthesized. The key of the Monitor Place Synthesis Algorithm is to find an appropriate linear inequality constraint in the form of (19) for each element $C_u \in \mathcal{C}_u^{(2)}$, so that we can employ SBPI to synthesize a monitor place to prevent C_u, and finally obtain an *admissible* controlled Gadara net. In general, for any given $C_u \in \mathcal{C}_u^{(2)}$, we can find an associated linear inequality constraint in two stages.

In Stage 1, we specify a linear inequality constraint in the form of (19) for C_u, according to the discussion following Theorem 2. From the above discussion of UCCOR, we know that C_u contains only "1" or "χ" components. So the parameters of the constraint associated with C_u are:

$$l_{C_u}(p) = \begin{cases} 1, & \text{if } C_u(p) = 1; \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

$$b_{C_u} = \sum_{p: p \in \Theta_S \text{ and } C_u(p) = 1} C_u(p) - 1 \tag{8}$$

Note that this constraint *only* prevents C_u according to Theorem 2. SBPI can be employed to synthesize a monitor place based on this constraint. If the resulting $\mathcal{N}_G^c$ is admissible, then Stage 2 is not necessary for this C_u and we can continue with the next element (if any) in $\mathcal{C}_u^{(2)}$; otherwise, we need to proceed to Stage 2, where constraint transformation is carried out to deal with the partial controllability and ensure the admissibility of $\mathcal{N}_G^c$.

Example 6: Before moving on to Stage 2, let us first illustrate Stage 1 by the running example. T_{uc} is chosen to be the lower bound specified in Assumption 2, which is $\emptyset$ in this example. From Step 3 of UCCOR, we know that $\mathcal{C}_u^{(2)}$ contains one covering C_1. According to (7) and (8), we specify the following linear inequality constraint in the form of (19) to prevent C_1:

$$M(p_{11}) + M(p_{22}) \leq 1 \tag{9}$$

Algorithm: Constraint Transformation
Input: A linear inequality constraint, e.g., (10)
Output: A set of places C
Method:
1. add $p_1, ..., p_n$ in (10) to stack S, and to set C
2. while S is not empty
3. $p = S.pop()$
4. for each uncontrollable t in $\bullet p$, if $\bullet t$ is not in C, add $\bullet t$ to S and C
5. end while

Fig. 5. The constraint transformation technique used in Stage 2 of the Monitor Place Synthesis Algorithm.

The monitor place p_{c2}, which enforces (9), is synthesized by SBPI and shown in Fig. 2. The controlled net obtained in the first iteration of ICOG, which consists of $\mathcal{N}_G, p_{c1}, p_{c2}$, and their associated arcs, is denoted as $\mathcal{N}_G^{c(2)}$. At the end of the first iteration, we update the global bookkeeping set as: $\Phi = \Phi \cup \mathcal{C}_u^{(2)} = \{C_1\}$. □

The constraint transformation technique in Stage 2 is presented as follows. For the sake of discussion, the constraint obtained in Stage 1 can be rewritten as:

$$M(p_1) + M(p_2) + \ldots + M(p_n) \leq n - 1 \tag{10}$$

We apply constraint transformation to (10) to handle partial controllability, adapted and much simplified from the corresponding procedure in [21], due to the special structure of Gadara nets. The core idea is the following. If place p_i in (10) can gain tokens through a sequence of uncontrollable transitions, places along the sequence of uncontrollable transitions must be included to the left-hand-side of (10) as we cannot prevent these transitions from firing and populating tokens into p_i. We make two remarks for the above statement: (i) The set of places corresponding to a given sequence of uncontrollable transitions is unique due to the state-machine structure of the process subnet. (ii) The uncontrollable transitions in this sequence are not blocked by any generalized resource place, otherwise they would be controllable. The pseudo-code that implements the constraint transformation for (10) is given in Fig. 5. Based on the set of places C obtained above, the new, transformed constraint is:

$$\sum_{p \in C} M(p) \leq n - 1 \tag{11}$$

Without any confusion, in the following discussion we will refer to (10) as the original constraint, and refer to (11) as the new constraint.

Proposition 3: Using SBPI, all the outgoing arcs of the monitor place synthesized for the new constraint do not connect to any uncontrollable transition, i.e., the resulting $\mathcal{N}_G^c$ is admissible.

Proof: We prove the result of Proposition 3 by contradiction. It can be shown that by applying SBPI to a constraint of the form (10) or (11), the outgoing arcs of the synthesized monitor place connect to "entry" transitions only, i.e., transitions whose input places (in the process subnet) are not in the constraint, and whose output places (in the process subnet) are in the constraint. This follows from the fact that SBPI enforces a P-invariant based on the constraint via a monitor place. If an "entry" transition is uncontrollable, the constraint transformation technique must have included its input place into the new constraint. Therefore, it is not an "entry" transition anymore. ■

Proposition 4: (a) Any marking prevented by the original constraint is also prevented by the new constraint. (b) Any reachable marking that is prevented by the new constraint but not by the original constraint, can reach a marking prevented by the original constraint via a sequence of uncontrollable transitions.

Proof:

(a) This is a direct result of the construction of the new constraint. Any marking that violates the original constraint will also violate the new one.

(b) The new constraint simply adds more places to the left-hand-side of the original constraint. By construction, any token in these new places may reach one of the places in the original constraint through a sequence of uncontrollable transitions. If a reachable marking M satisfies the original constraint but not the new one, then at M there must be extra tokens in the set of places added. These tokens can "leak" into the set of places in the original constraint through a sequence of uncontrollable transitions. Thus, in the reachability graph, there must be a sequence of uncontrollable transitions connecting from M to a marking that violates the original constraint.

■

Example 7: The Gadara net model of a deadlock case in the OpenLDAP software is shown in Fig. 6. In this example, T_{uc} is chosen to be the upper bound as specified in Assumption 2; thus, only t_1, t_2, and t_8 are controllable transitions. When we apply UCCOR to this example, both stages in Step 4b are required. In Stage 1 of Step 4b, the original constraint is $M(p_1) + M(p_5) \leq 1$; in Stage 2 of Step 4b, the new, transformed constraint is $\sum_{i=1}^{6} M(p_i) \leq 1$, where the synthesized monitor place p_c is shown in dashed lines in Fig. 6. The resulting controlled Gadara net is admissible. □

Example 8: Let us return to the running example of Fig. 2. In the second iteration of ICOG, we further input $\mathcal{N}_G^{c(2)}$ to ICOG, and detect a new RIDM siphon:

$$S_2 = \{p_{c1}, p_{c2}, p_{12}, p_{13}, p_{14}, p_{15}, p_{22}, p_{23}, p_{24}, p_{25}\} \tag{12}$$

at the marking M_{u2}, where the places p_{01}, p_{02}, p_{11}, and p_{21} each have one token, and all the other places are empty. We apply UCCOR to this RIDM siphon in the second iteration of ICOG. After Step 3 of UCCOR, the set $\mathcal{C}_u^{(2)}$ contains one covering C_2, where $C_2(p_{11}) = C_2(p_{21}) = 1$ and $C_2(p) = \chi$, for any $p \in P \setminus \{p_{11}, p_{21}\}$. In Stage 1 of Step 4b, the monitor place p_{c3} shown in Fig. 2 is synthesized to prevent C_2. The resulting controlled net is denoted as $\mathcal{N}_G^{c(3)}$, and it is admissible. Thus, Stage 2 of Step 4b is not necessary. At the end of the second iteration, we update the global bookkeeping set as: $\Phi = \Phi \cup \mathcal{C}_u^{(2)} = \{C_1, C_2\}$. Next, we input $\mathcal{N}_G^{c(3)}$ to ICOG, and no new RIDM siphon is detected. Therefore, ICOG converges after the second iteration. □

Two important observations can be made from the above example. (i) In the second iteration of ICOG, we notice that the new RIDM siphon S_2 is induced by the monitor places p_{c1} and p_{c2}. This is an example of the scenario we discussed in Section III-B, where monitor places can introduce new potential

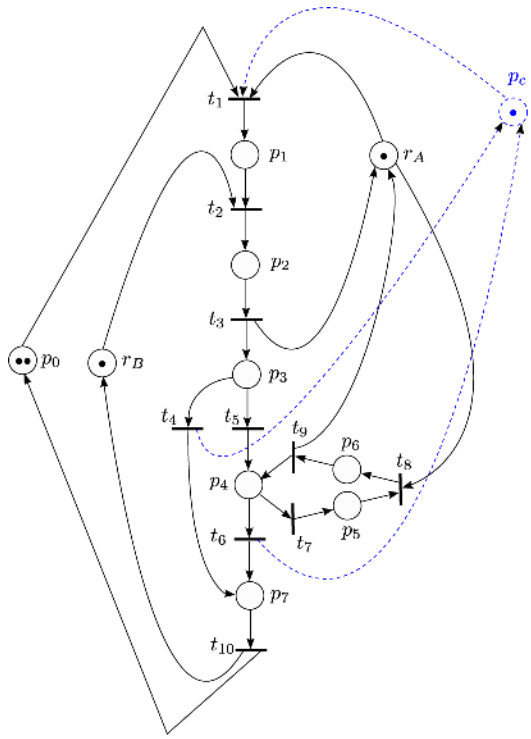


Fig. 6. Gadara net model of a deadlock example in the OpenLDAP software.

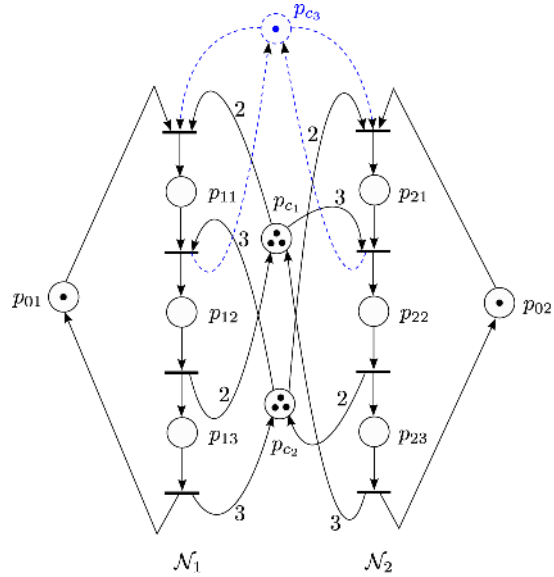


Fig. 7. A simple example of UCCOR.

deadlocks and thus force further iterations. (ii) As discussed in Section III-C-1, in the initial net $\mathcal{N}_G^{c(1)}$, the marking M_{u2} is not RIDM-unsafe but cannot reach the initial marking. However, in the controlled net $\mathcal{N}_G^{c(2)}$, M_{u2} is RIDM-unsafe. More specifically, it is RIDM-unsafe with respect to the RIDM siphon S_2. In other words, as the control iterations evolve, the added control logic *exposes* the marking M_{u2}, which was not RIDM-unsafe but could not reach the initial marking, to a marking that is RIDM-unsafe. Therefore, M_{u2} can eventually be captured by its associated RIDM siphon in ICOG and prevented by UCCOR.

Example 9: In Fig. 1, we gave a controlled Gadara net that contains a RIDM siphon. The monitor place, which is synthesized by UCCOR and prevents this RIDM siphon, is shown in Fig. 7. The controlled net after this iteration is admissible for any choice of T_{uc} satisfying Assumption 2. ICOG converges after this iteration. □

By the definition of covering, we know that the relation "$\succeq$" is a partial order on the set Φ, and Φ is a partially ordered set. Steps 2 and 3 of the UCCOR Algorithm imply that after ICOG converges, any two distinct elements of Φ are *incomparable*. Thus, the final controlled Gadara net does not contain any redundant monitor place.

The performance of the ICOG Methodology has been systematically investigated in [18], in terms of execution time, number of iterations, and scalability. Our experimental results reveal that the RIDM siphon detection step is the bottleneck of ICOG, while the UCCOR Algorithm and the maintenance of the bookkeeping set Φ only account for a negligible portion of the computational overhead.

IV. Control Synthesis for Gadara Nets: Properties

In Section III-B, we presented the global flowchart of the ICOG Methodology. Here, we present its main properties. In this section, when we say that ICOG is "correct with respect to the goal of liveness enforcement", it will mean that the resulting controlled net is admissible and live. We will carry out the proofs in two steps: we first prove the properties of UCCOR (employed in each iteration of ICOG), then we prove the properties maintained by ICOG throughout the entire set of the performed iterations.

Theorem 4: In $\mathcal{N}_G^c$, the control logic synthesized for any RIDM siphon S based on the UCCOR Algorithm is correct and maximally permissive with respect to the goal of preventing S from becoming a RIDM siphon and the given set of uncontrollable transitions.

Proof: First, we prove the correctness. We are going to show that the UCCOR Algorithm does not miss preventing any RIDM-unsafe marking with respect to S.

For any RIDM siphon S with its associated $\overline{M}_u(S)$, which needs to be prevented, Step 1 of the UCCOR Algorithm finds the set of RIDM-unsafe coverings $\mathcal{C}_u$ that covers *all* the possible RIDM-unsafe markings M, such that $\overline{M}(S) = \overline{M}_u(S)$. This set of RIDM-unsafe markings is denoted as $\mathcal{M}_u(S)$. According to the Unsafe Covering Generalization Algorithm and the property of "covering", the set $\mathcal{C}_u^{(1)}$ obtained in Step 2 covers $\mathcal{C}_u$, which covers $\mathcal{M}_u(S)$. Moreover, Step 3 only removes the coverings that are already prevented in previous control synthesis iterations. Thus, $\mathcal{C}_u^{(2)}$, together with the prevented coverings Φ, covers $\mathcal{M}_u(S)$. From Step 4 as well as Propositions 2 and 4(a), we know that any marking in the set $\mathcal{M}_u(S)$ will be prevented in this step, or this marking has already been prevented in previous iterations.

The above discussion applies to any RIDM siphon S in the net. Thus, UCCOR does not miss preventing any RIDM-unsafe marking with respect to any RIDM siphon S. This, together with Assumption 1 and Proposition 3, implies that UCCOR is correct with respect to the goal of preventing S from becoming a RIDM siphon and the given set of uncontrollable transitions.

Next, we prove the maximal permissiveness. Recall from Theorem 1(a) that $\mathcal{N}_G^c$ is live *iff* it is reversible. Thus, we are

going to show that the UCCOR Algorithm *only* prevents markings that cannot reach the initial marking M_0^c, or markings that can reach the aforementioned type of markings via a sequence of uncontrollable transitions.

According to Proposition 2, it follows from the Unsafe Covering Construction Algorithm that the set $\mathcal{C}_u$ obtained in Step 1 of the UCCOR Algorithm *only* covers RIDM-unsafe markings with respect to S. Moreover, Corollary 2 and Remark 7 imply that any marking, covered by the refined set $\mathcal{C}_u^{(1)}$ obtained in Step 2, cannot reach M_0^c. In Step 3, the obtained set $\mathcal{C}_u^{(2)}$ is a subset of $\mathcal{C}_u^{(1)}$. Hence, any marking covered by $\mathcal{C}_u^{(2)}$ cannot reach M_0^c either. Moreover, Stage 1 of Step 4b only prevents the coverings in $\mathcal{C}_u^{(2)}$; Stage 2 of Step 4b only prevents the coverings in $\mathcal{C}_u^{(2)}$, and the markings that can reach some marking covered by $\mathcal{C}_u^{(2)}$, through a sequence of uncontrollable transitions (see Proposition 4(b)). Such coverings and markings must be removed. Therefore, UCCOR is maximally permissive with respect to the goal of preventing S from becoming a RIDM siphon and the given set of uncontrollable transitions. ■

From Definition 5, we know that any marking in a Gadara net can be uniquely characterized by the corresponding P_S-marking without any ambiguity. That is, $M_1 = M_2$ if and only if $\overline{\overline{M}}_1 = \overline{\overline{M}}_2$. The set of reachable P_S-markings induced by the set of reachable markings is defined as $\overline{\overline{R}}(\mathcal{N}_G^c, M_0^c) = \{\overline{\overline{M}} | M \in R(\mathcal{N}_G^c, M_0^c)\}$, which is denoted as $\overline{\overline{R}}$ for simplicity. We immediately have the following result.

Lemma 1: The reachability graph associated with $R(\mathcal{N}_G^c, M_0^c)$ and the reachability graph associated with $\overline{\overline{R}}(\mathcal{N}_G^c, M_0^c)$ are isomorphic.

Lemma 1 subsequently enables us to prove the following theorem.

Theorem 5: ICOG terminates in a finite number of iterations.

Proof: Due to Lemma 1, in the following proof, we can restrict our attention to $\overline{\overline{R}}$. By Definition 1, the number of operation places in a Gadara net is finite. Further, for any $p \in P_S$, $M(p)$ is always binary. As a result, the set of reachable P_S-markings $\overline{\overline{R}}$ has finite cardinality. The set of reachable P_S-markings that need to be prevented, which is a subset of $\overline{\overline{R}}$, also has finite cardinality.

For the sake of discussion, we use $\overline{\overline{N}}$ to denote the set of non-reachable P_S-markings, each of which is a binary vector. According to the above discussion, the cardinality of $\overline{\overline{R}} \cup \overline{\overline{N}}$ is at most $2^{|P_S|}$. Note that the aforementioned $\overline{\overline{R}}$ and $\overline{\overline{N}}$ are associated with the input Gadara net before the first iteration of ICOG is applied.

In each iteration of ICOG, we only expand the set P_C by adding monitor places while leaving P_S unchanged. So, ICOG does not expand $\overline{\overline{R}}$. In other words, the set of reachable P_S-markings that need to be prevented has a finite upper bound $\overline{\overline{R}}$; another looser but also finite upper bound for this set is $\overline{\overline{R}} \cup \overline{\overline{N}}$, whose cardinality is $2^{|P_S|}$ at most. In every iteration of ICOG, the synthesized monitor place eliminates at least one marking from $\overline{\overline{R}} \cup \overline{\overline{N}}$, which is not prevented in the previous iterations of ICOG. Therefore, the proposed ICOG will terminate in a finite number of iterations. ■

Theorem 6: ICOG is correct and maximally permissive with respect to the goal of liveness enforcement and the given set of uncontrollable transitions.

Proof: First, we prove the correctness of ICOG.

In each iteration of ICOG, a new RIDM siphon in the net is detected. According to Theorem 4, for any detected RIDM siphon S with its associated $\overline{M}_u(S)$, the UCCOR Algorithm ensures that *any* possible RIDM-unsafe marking M, such that $\overline{M}(S) = \overline{M}_u(S)$, will be prevented. And the detected RIDM siphon S will not become reachable under the synthesized control logic. ICOG terminates when no further new RIDM siphons can be detected. By using the UCCOR Algorithm for all detected RIDM siphons in all the iterations, any RIDM-unsafe marking associated with any RIDM siphon will be prevented, and no siphon will become a RIDM siphon. Furthermore, in each iteration of ICOG, UCCOR always synthesizes an admissible Gadara net, according to Assumption 1 and Proposition 3. This, together with Theorems 1(c) and 5, implies that the proposed ICOG ensures admissibility and liveness of the final controlled Gadara net, i.e., it is correct with respect to the goal of liveness enforcement and the given set of uncontrollable transitions.

Next, we prove the maximal permissiveness of ICOG. This is an immediate consequence of the maximal permissiveness of UCCOR on a single iteration basis as established in Theorem 4. In each iteration of ICOG, UCCOR is employed to prevent markings in the net. Since UCCOR only prevents markings that cannot reach the initial marking, or markings that can reach the aforementioned type of markings via a sequence of uncontrollable transitions, so does ICOG. Therefore, ICOG is maximally permissive with respect to the goal of liveness enforcement and the given set of uncontrollable transitions. ■

Remark 8: We interpret the effect of ICOG and UCCOR from the viewpoint of the Supervisory Control Theory. Let $\mathcal{N}_G^{c,\text{final}}$ be the final controlled Gadara net when ICOG terminates. Let G and G_{final} be the automata models of the reachability graphs associated with $\mathcal{N}_G$ and $\mathcal{N}_G^{c,\text{final}}$, respectively. The language generated by G is denoted as $\mathcal{L}(G)$. In G and G_{final}, only the initial states are marked. The languages marked by G and G_{final} are denoted as $\mathcal{L}_m(G)$ and $\mathcal{L}_m(G_{\text{final}})$, respectively. The live (equivalently, reversible) part of $\mathcal{N}_G$ corresponds to the trim of automaton G and it is captured by the marked language $\mathcal{L}_m(G)$. However, this language need not be *controllable* (as defined in [25]) with respect to $\mathcal{L}(G)$ and E_{uc}, where E_{uc} is the set of uncontrollable events corresponding to the set T_{uc} in the Gadara net. ICOG and UCCOR control $\mathcal{N}_G$ and finally obtain $\mathcal{N}_G^{c,\text{final}}$, so that $\mathcal{L}_m(G_{\text{final}})$ is equal to the *supremal controllable sublanguage* [25] of $\mathcal{L}_m(G)$ with respect to $\mathcal{L}(G)$ and E_{uc}. Throughout the iterations of ICOG, the cumulative effect of the constraint transformation in Stage 2 of Step 4b of UCCOR corresponds to the elimination of the states that violate the controllability condition in the supremal controllable sublanguage algorithm; the cumulative effect of the remaining operations in UCCOR corresponds to the removal of the blocking states in that algorithm. □

Our last result is the following interesting property of the UCCOR Algorithm.

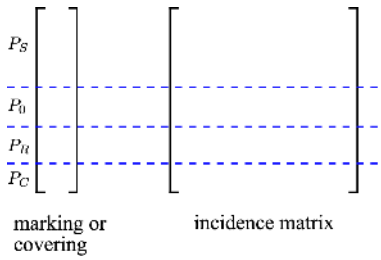


Fig. 8. Illustration of the rearranged order of rows in a marking, covering, and incidence matrix.

Theorem 7: In $\mathcal{N}_G^c$, for any monitor place synthesized by the UCCOR Algorithm, all its incoming and outgoing arcs have *unit* arc weights.

Proof: From Step 4b of the UCCOR Algorithm, we know that for any synthesized monitor place p_c, the arc weights of its associated incoming and outgoing arcs are determined by the nonzero components in the row vector D_{p_c}, which is calculated as

$$D_{p_c} = -l_{C_u}^T D. \tag{13}$$

In (13)

$$l_{C_u}(p) = \begin{cases} 1, & \text{if } C_u(p) = 1; \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

is a column vector that has the same dimension with C_u, and D is the incidence matrix of the net.

For the sake of discussion and without loss of generality, we can always rearrange the order of rows in a marking, covering, and incidence matrix such that row 1 to row $|P_S|$ correspond to the set of all operation places P_S, row $|P_S|+1$ to row $|P_S|+|P_0|$ correspond to the set of all idle places P_0, row $|P_S|+|P_0|+1$ to row $|P_S|+|P_0|+|P_R|$ correspond to the set of all resource places P_R, and row $|P_S|+|P_0|+|P_R|+1$ to $|P_S|+|P_0|+|P_R|+|P_C|$ correspond to the set of all monitor places P_C. The rearranged order is shown in Fig. 8.

In this way, any covering can be logically divided into four blocks corresponding to the four types of places. Then, any covering C_u can be rewritten as

$$C_u = \left(C_{u,S}^T, C_{u,0}^T, C_{u,R}^T, C_{u,C}^T\right)^T \tag{15}$$

where $C_{u,S}$, $C_{u,0}$, $C_{u,R}$, and $C_{u,C}$ are the partial coverings on P_S, P_0, P_R, and P_C, respectively. Similarly, the aforementioned column vector l_{C_u} in (14) can be rewritten as

$$l_{C_u} = \left(l_{C_u,S}^T, l_{C_u,0}^T, l_{C_u,R}^T, l_{C_u,C}^T\right)^T \tag{16}$$

and the incidence matrix D can be rewritten as

$$D = \left(D_S^T, D_0^T, D_R^T, D_C^T\right)^T \tag{17}$$

The blocks are self-explanatory by their subscripts. From Definition 7 and its discussion thereafter, we know that for any covering C_u written as in (15), any component in $C_{u,0}$, $C_{u,R}$, and $C_{u,C}$ is always χ. As a result of this and (14), for any column vector l_{C_u} written as in (16), any component in $l_{C_u,0}$, $l_{C_u,R}$, and $l_{C_u,C}$ is always 0. Therefore, (13) can be simplified as:

$$\begin{aligned} D_{p_c} &= -\begin{pmatrix} l_{C_u,S} \\ l_{C_u,0} \\ l_{C_u,R} \\ l_{C_u,C} \end{pmatrix}^T \begin{pmatrix} D_S \\ D_0 \\ D_R \\ D_C \end{pmatrix} = -\begin{pmatrix} l_{C_u,S} \\ 0 \\ 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} D_S \\ D_0 \\ D_R \\ D_C \end{pmatrix} \\ &= -l_{C_u,S}^T D_S \end{aligned} \tag{18}$$

Note that D_S is the part of the incidence matrix of $\mathcal{N}_G^c$ that corresponds to P_S, which describes the connectivity between operation places and transitions. Since D_S is also a part of the incidence matrix of $\mathcal{N}_G$ and $\mathcal{N}_G$ is ordinary, any component in D_S can only be -1, 1, or 0. Moreover, according to Condition 3 of Definition 1, we know that each transition in $\mathcal{N}_G^c$ has at most one input operation place and at most one output operation place. That is, any column in D_S contains at most one "-1" and at most one "1", with all other components being zeros. On the other hand, we know from (14) that any component in $l_{C_u,S}$ is either 0 or 1. Consequently, any component in the row vector D_{p_c} calculated in (18) can only be -1, 1, or 0. ■

The implication of Theorem 7 will be discussed in the next section.

V. Discussion

Theorem 7 implies that the UCCOR Algorithm will never introduce any non-ordinariness to the Gadara net. If ICOG starts with a controlled Gadara net that is ordinary, then the resulting controlled Gadara nets will remain ordinary throughout the iterations. Therefore, if the objective of our control synthesis is *strictly* liveness enforcement and the initial condition is an ordinary controlled Gadara net (including $\mathcal{N}_G$), then the general methodology of ICOG and UCCOR can be customized for this special case. More specifically, in the customized control synthesis, we could focus on preventing empty siphons that are induced by resources, rather than RIDM siphons. Such customization preserves all the properties of ICOG and UCCOR presented in this paper, because the former type of siphons is a special case of RIDM siphons. In addition, some steps in ICOG and UCCOR, such as bookkeeping, can also be simplified as a result of the customization.

Furthermore, observing that RIDM siphon detection is the computational bottleneck of ICOG, as mentioned in Section III, we have developed a set of customized MIP formulations for RIDM siphon detection in general Gadara nets and empty siphon detection in ordinary Gadara nets. We have shown via experiments that the proposed MIP formulations, when used as a module in ICOG, perform consistently better than the similar MIP formulations available in the literature for broader classes of Petri nets. Further, our stress tests indicate that ICOG is scalable to very large nets that are typical of the size of real-world software. We have also shown that the customized ICOG, together with the customized MIP formulation, never synthesizes redundant control logic throughout the iterations. On the other hand, the number of monitor places synthesized by ICOG need not be minimal in general. In this regard, we refer the readers to the recent work [24], developed in parallel by members of our team, which addresses the minimization of the number of monitor places in control problems for a class of resource allocation systems using classification theory.

The detailed description of the above customizations and experiments is beyond the scope of this paper and is reported in a follow-up applications paper [20].

VI. Conclusion

We have presented an iterative methodology, called ICOG, for the synthesis of optimal liveness-enforcing control policies

for the class of controlled Gadara nets based on siphon analysis. As a module in ICOG, a new algorithm, called UCCOR, is proposed to prevent any RIDM siphon from becoming reachable. Using the notion of covering, each monitor place synthesized by the UCCOR Algorithm can prevent more than one undesirable state. In addition, the net uncontrollability is accounted for in a minimally restrictive manner. ICOG applies the UCCOR Algorithm until all RIDM siphons are prevented by at least one monitor place; this convergence is achieved in a finite number of iterations. We formally show that both ICOG and UCCOR are correct and maximally permissive with respect to the goal of liveness enforcement. The proposed ICOG and UCCOR provide a general methodology for the liveness enforcement of Gadara nets that can be further customized for specific applications.

APPENDIX

A. Petri Net Preliminaries

Definition 8: A Petri net dynamic system $\mathcal{N} = (P, T, A, W, M_0)$ is a bipartite graph (P, T, A, W) with an initial number of tokens. Specifically, $P = \{p_1, p_2, \ldots, p_n\}$ is the set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is the set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, $W : A \to \{0, 1, 2, \ldots\}$ is the arc weight function, and for $p \in P$, $M_0(p)$ is the initial number of tokens in p.

The *marking* (or *state*) of a Petri net $\mathcal{N}$ is a column vector M of n entries corresponding to the n places. As defined above, M_0 is the initial marking. We use $M(p)$ to denote the (partial) marking on a place p (a scalar) and use $M(Q)$ to denote the (partial) marking on a set of places Q, which is a $|Q| \times 1$ column vector. The notation $\bullet p$ denotes the set of input transitions of place $p : \bullet p = \{t | (t, p) \in A\}$. Similarly, $p\bullet$ denotes the set of output transitions of p. The sets of input and output places of transition t are similarly defined by $\bullet t$ and $t\bullet$. This notation is extended to sets of places or transitions in a natural way. A pair (p, t) is called a *self-loop* if p is both an input and output place of t. We consider only *self-loop-free* Petri nets in this paper. Our Petri net models of multithreaded programs have unit arc weights. Such Petri nets are called *ordinary*. However, the imposition of further control structure upon these nets may render them *non-ordinary*. A transition t is *enabled* or *fireable* at a marking M, if $\forall p \in \bullet t$, $M(p) \geq W(p, t)$. The *reachable state space* $R(\mathcal{N}, M_0)$ of $\mathcal{N}$ is the set of all markings reachable by transition firing sequences starting from M_0. The *incidence matrix* D of a Petri net is an integer matrix $D \in \mathbb{Z}^{n \times m}$, where $D_{ij} = W(t_j, p_i) - W(p_i, t_j)$ represents the net change in the number of tokens in place p_i when transition t_j fires. A *state machine* is an ordinary Petri net such that each transition t has exactly one input place and exactly one output place, i.e., $\forall t \in T, |\bullet t| = |t \bullet| = 1$.

Definition 9: Let D be the incidence matrix of a Petri net $\mathcal{N}$. Any non-zero integer vector y such that $D^T y = 0$, is called a *P-invariant* of $\mathcal{N}$. Further, P-invariant y is called a *P-semiflow* if all the elements of y are non-negative.

A straightforward property of P-invariants is given by the following well-known result [22]: If vector y is a P-invariant of $\mathcal{N} = (P, T, A, M_0)$, then $M^T y = M_0^T y$ for any $M \in R(\mathcal{N}, M_0)$. The *support* of P-semiflow y, denoted by $\|y\|$, is defined to be the set of places that correspond to nonzero entries in y. A support $\|y\|$ is said to be *minimal* if there does not exist another nonempty support $\|y'\|$, for some other P-semiflow y', such that $\|y'\| \subset \|y\|$. A P-semiflow y is said to be *minimal* if there does not exist another P-semiflow y' such that $y'(p) \leq y(p), \forall p$. For a given minimal support of a P-semiflow, there exists a unique minimal P-semiflow, which we call the *minimal-support P-semiflow* [22].

B. Control Synthesis for Petri Nets

Supervision Based on Place Invariants (SBPI) [6]–[8], [33] provides an efficient algebraic technique for control logic synthesis by introducing a monitor place, which essentially enforces a P-invariant so as to achieve a given linear inequality constraint of the following form

$$l^T M \leq b \tag{19}$$

where M is the marking vector of the net under control, l is a weight (column) vector, and b is a scalar. All entries of l and b are integers. The main result of SBPI is as follows.

Theorem 8: [8] Consider a Petri net $\mathcal{N}$, with incidence matrix D and initial marking M_0. If it satisfies $b - l^T M_0 \geq 0$, then a monitor place, p_c, with incidence matrix $D_{p_c} = -l^T D$, and initial marking $M_0(p_c) = b - l^T M_0$, enforces the constraint $l^T M \leq b$ upon the net marking. This supervision is maximally permissive.

The property of maximal permissiveness stated in the above theorem implies that a transition in the net is disabled by the monitor place only if its firing leads to a marking where the linear constraint in (19) is violated.

REFERENCES

[1] K. Barkaoui and J.-F. Pradat-Peyre, "On liveness and controlled siphons in Petri nets," in *Proc. 17th Int. Conf. Appl. Theory Petri Nets*, 1996, pp. 57–72.

[2] E. E. Cano, C. A. Rovetto, and J.-M. Colom, "An algorithm to compute the minimal siphons in S^4PR nets," in *Proc. Int. Workshop Discrete Event Syst.*, 2010, pp. 18–23.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Boston, MA: Springer, 2008.

[4] F. Chu and X.-L. Xie, "Deadlock analysis of Petri nets using siphons and mathematical programming," *IEEE Trans. Robot. Autom.*, vol. 13, no. 6, pp. 793–804, Dec. 1997.

[5] A. Ghaffari, N. Rezg, and X. Xie, "Design of a live and maximally permissive Petri net controller using the theory of regions," *IEEE Trans. Robot. Autom.*, vol. 19, no. 1, pp. 137–142, Jan. 2003.

[6] A. Giua, "Petri Nets as Discrete Event Models for Supervisory Control," Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1992.

[7] A. Giua, F. DiCesare, and M. Silva, "Generalized mutual exclusion constraints on nets with uncontrollable transitions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1992, pp. 974–979.

[8] M. V. Iordache and P. J. Antsaklis, *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Boston, MA: Birkhäuser, 2006.

[9] M. V. Iordache and P. J. Antsaklis, "Concurrent program synthesis based on supervisory control," in *Proc. American Control Conf.*, 2010, pp. 3378–3383.

[10] G. Jin, L. Song, W. Zhang, S. Lu, and B. Liblit, "Automated atomicity-violation fixing," in *Proc. ACM SIGPLAN 2011 Conf. Programming Language Design Implementation*, 2011, pp. 389–400.

[11] T. Kelly, Y. Wang, S. Lafortune, and S. Mahlke, "Eliminating concurrency bugs with control engineering," *IEEE Computer*, vol. 42, no. 12, pp. 52–60, Dec. 2009.

[12] Z. Li, N. Wu, and M. Zhou, "Deadlock Control of Automated Manufacturing Systems Based on Petri Nets—A Literature Review," Xidian University, Shaanxi, China, Tech. Rep., 2010.
[13] Z. Li and M. Zhou, *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. London, U.K.: Springer-Verlag, 2009.
[14] Z. Li and M. Zhou, *Modeling, Analysis, and Deadlock Control of Automated Manufacturing Systems*. Beijing, China: Science Press, 2009.
[15] Z. Li, M. Zhou, and N. Wu, "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern. C*, vol. 38, no. 2, pp. 173–188, Mar. 2008.
[16] H. Liao, "Modeling, Analysis, and Control of a Class of Resource Allocation Systems Arising in Concurrent Software," Ph.D. dissertation, Univ. Michigan, Ann Arbor, 2012.
[17] H. Liao, S. Lafortune, S. Reveliotis, Y. Wang, and S. Mahlke, "Synthesis of maximally-permissive liveness-enforcing control policies for Gadara Petri nets," in *Proc. 49th IEEE Conf. Decision Control*, 2010, pp. 2797–2804.
[18] H. Liao, J. Stanley, Y. Wang, S. Lafortune, S. Reveliotis, and S. Mahlke, "Deadlock-avoidance control of multithreaded software: An efficient siphon-based algorithm for Gadara Petri nets," in *Proc. 50th IEEE Conf. Decision Control*, 2011, pp. 1142–1148.
[19] H. Liao, Y. Wang, H. K. Cho, J. Stanley, T. Kelly, S. Lafortune, S. Mahlke, and S. Reveliotis, "Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets," *J. Discrete Event Dynamic Syst.*, to be published.
[20] H. Liao, Y. Wang, J. Stanley, S. Lafortune, S. Reveliotis, T. Kelly, and S. Mahlke, "Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control," *IEEE Trans. Control Syst. Technol.*, to be published.
[21] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Boston, MA: Kluwer, 1998.
[22] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[23] T. Murata, B. Shenker, and S. M. Shatz, "Detection of Ada static deadlocks using Petri net invariants," *IEEE Trans. Software Eng.*, vol. 15, no. 3, pp. 314–326, Mar. 1989.
[24] A. Nazeem, S. Reveliotis, Y. Wang, and S. Lafortune, "Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: The linear case," *IEEE Trans. Autom. Control*, vol. 56, no. 8, pp. 1818–1833, Aug. 2011.
[25] P. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.
[26] S. A. Reveliotis, *Real-Time Management of Resource Allocation Systems: A Discrete-Event Systems Approach*. New York: Springer, 2005.
[27] S. A. Reveliotis and P. M. Ferreira, "Deadlock avoidance policies for automated manufacturing cells," *IEEE Trans. Robot. Autom.*, vol. 12, pp. 845–857, 1996.
[28] F. Tricas, F. Garcia-Valles, J. Colom, and J. Ezpeleta, "A Petri net structure-based deadlock prevention solution for sequential resource allocation systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 271–277.
[29] M. Uzam, "An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions," *Int. J. Adv. Manufact. Technol.*, vol. 19, no. 3, pp. 192–208, 2002.
[30] Y. Wang, H. K. Cho, H. Liao, A. Nazeem, T. P. Kelly, S. Lafortune, S. Mahlke, and S. Reveliotis, "Supervisory control of software execution for failure avoidance: Experience from the Gadara project," in *Proc. Int. Workshop Discrete Event Syst.*, 2010, pp. 259–266.
[31] Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke, "The theory of deadlock avoidance via discrete control," in *Proc. 36th Annu. ACM SIGPLAN-SIGACT Symp. Principles Programming Languages*, 2009, pp. 252–263.
[32] Y. Wang, H. Liao, S. Reveliotis, T. Kelly, S. Mahlke, and S. Lafortune, "Gadara nets: Modeling and analyzing lock allocation for deadlock avoidance in multithreaded software," in *Proc. 48th IEEE Conf. Decision Control*, 2009, pp. 4971–4976.
[33] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of Petri nets based on place invariants," *Automatica*, vol. 32, no. 1, pp. 15–28, Jan. 1996.

Hongwei Liao (S'10) received the B.Eng. degree in electrical engineering and Dual B.Mgt. degree in business administration (with honors) from Shanghai Jiao Tong University, Shanghai, China, in 2007, and the M.Sc. degree in electrical engineering-systems, the M.S.E. degree in industrial and operations engineering, and the Ph.D. degree in electrical engineering-systems from the University of Michigan, Ann Arbor, in 2009, 2011, and 2012, respectively.

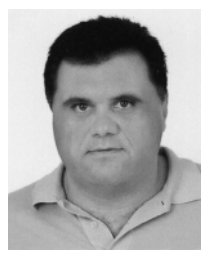
He joined the Operations Research Group at US Airways, Tempe, AZ, in 2012, where he is an Analyst and Technologies Lead. He was an Intern at General Electric Global Research, Niskayuna, NY, in summer 2010, and an Intern at General Motors Global Research and Development, Warren, MI, in summer 2011. His research interests include discrete event systems, operations research, and wireless communications.

Dr. Liao received the Rackham Predoctoral Fellowship Award (2011) and the College of Engineering Distinguished Achievement Award (2011) from the University of Michigan, Ann Arbor. He is a member of Phi Kappa Phi and Tau Beta Pi.

Stéphane Lafortune (F'99) received the B. Eng degree from Ecole Polytechnique de Montréal Montréal, QC, Canada, in 1980, the M. Eng. degree from McGill University, Montréal, Montréal, QC, in 1982, and the Ph.D. degree from the University of California at Berkeley in 1986, all in electrical engineering.

Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. He is the lead developer of the software package UMDES and co-developer of DESUMA with L. Ricker. He co-authored the textbook *Introduction to Discrete Event Systems—Second Edition* (Springer, 2008). He is a member of the editorial boards of the *Journal of Discrete Event Dynamic Systems: Theory and Applications* and of the *International Journal of Control*. His research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer systems.

Dr. Lafortune received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994, and in 2001.



Spyros Reveliotis (SM'03) received the Diploma in electrical engineering from the National Technical University of Athens, Athens, Greece, the M.Sc. degree in computer systems engineering from Northeastern University, Boston, MA, and the Ph.D. degree in industrial engineering from the University of Illinois at Urbana-Champaign.

He is a Professor in the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta. His research interests are in the area of Discrete Event Systems theory and its applications.

Dr. Reveliotis received the 1998 IEEE Intl. Conf. on Robotics and Automation Kayamori Best Paper Award. He serves as Associate Editor for the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and as Department Editor for the IIE TRANSACTIONS. He has also been an Associate Editor for the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION and the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, a Senior Editor in the Conference Editorial Board for the IEEE International Conference on Robotics and Automation. He is a member of INFORMS. In 2009, he was the Program Chair for the IEEE Conference on Automation Science and Engineering, and currently he serves as a member of the Steering Commitee for this conference.

Yin Wang (M'09) received the B.S. and M.S. degrees from the Department of Automation, Shanghai Jiao Tong University, Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree from the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, in 2009.

He joined Hewlett-Packard Laboratories in early 2009. He interned at Microsoft Shanghai, IBM Almaden Research Center, and HP Labs.

Scott Mahlke (M'90) received the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, in 1997.

He is a Professor in the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, where he leads the Compilers Creating Custom Processors Group. The CCCP Group delivers technologies in the areas of compilers for multicore processors, application-specific processors for mobile computing, and reliable system design.

Dr. Mahlke was named the Morris Wellman Assistant Professor in 2004 and received the Most Influential Paper Award from the International Symposium on Computer Architecture in 2007. He is a member of the IEEE Computer Society and the ACM.