Synthesis of Maximally-Permissive Liveness-Enforcing Control Policies for Gadara Petri Nets

Hongwei Liao, Stéphane Lafortune, Spyros Reveliotis, Yin Wang, and Scott Mahlke

Abstract— **This paper studies the synthesis of maximally-permissive liveness-enforcing control policies for Gadara nets. Gadara nets are a special class of Petri nets that model allocation of locks in multithreaded computer programs for the purpose of deadlock avoidance. We propose a new control synthesis algorithm that can be used for liveness enforcement of Gadara nets. The algorithm employs structural analysis of the net and synthesizes monitor places to control a special class of siphons, termed resource-induced deadly-marked siphons. We present an iterative control methodology based on this algorithm that converges in a finite number of iterations. The methodology exploits a covering of the unsafe states that is updated at each iteration. Both the proposed algorithm and the associated iterative control methodology are shown to be correct and maximally permissive with respect to the goal of liveness enforcement.**

I. Introduction

The popularity of multicore architectures in computer hardware leads to the prevalence of parallel programming in software development. In our on-going project, called Gadara [6], we are interested in multithreaded programs with shared data, where *mutual exclusion locks (mutexes)* are usually employed by programmers to protect shared data and prevent data races. When mutexes are inappropriately used, *circular-mutex-wait deadlocks* can occur in the program, where a set of threads are waiting for one another and none of them can proceed. Deadlock analysis based on Petri nets has been widely studied for flexible manufacturing systems [8]. It has also been applied to Ada programs [11]. Recently, supervisory control of Petri nets has been applied to concurrent program synthesis [5].

The goal of the Gadara project is to implement control synthesis techniques from Petri nets for deadlock avoidance in general-purpose concurrent software. In [16], we defined a special class of Petri nets, called Gadara nets, to systematically model lock allocations and releases in multithreaded C programs. Deadlock-freeness of the program corresponds to liveness of the Gadara net [15]. The main focus of the present paper is on the synthesis of maximally-permissive liveness-enforcing (MPLE) control policies for (controlled) Gadara nets. By definition, an original Gadara net is ordinary, while a controlled Gadara net may be non-ordinary due to the added structure as a result of the control synthesis. (A net is ordinary if all its arc weights are equal to one.) With reachability graph calculated, the problem of MPLE control can be solved by the Ramadge and Wonham Supervisory Control Theory. But automaton based controllers are serial in nature and therefore impair the performance of the target concurrent program. Theory of regions (see, e.g., [14], [3]) addresses this issue by synthesizing monitor places [9] back in the Petri net to avoid unsafe states in the reachability graph. Our recent work [12] further minimizes the number of monitor places added. However, all these approaches may pose scalability challenges due to the need to explore the entire set of reachable states. In the context of Petri net models, many approaches have been considered for the synthesis of liveness-enforcing control strategies. These approaches typically sacrifice maximal permissiveness due to the complexity of the problem and the inherent limitation of monitor-based control. In [16], we have demonstrated that for Gadara nets, MPLE control logic can be implemented using monitor places. Our objective in this paper is to exploit *structural properties* of Gadara nets thoroughly for the efficient synthesis of MPLE control policies. Comparing with our recent work [12], the number of monitor places added by this structure based approach is not minimal in general. However, we have observed that in practice, many real-world programs have huge state spaces, but relatively simple deadlock patterns resulting in small sets of unsafe states [15]. These systems may favor the structure based approach as it enumerates only the *unsafe states*, not the *entire reachable state space*. Our initial results in this regard were reported in our earlier work [15]. 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. It is possible that newly 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. Few works address such an iterative process and its implications for control synthesis. In [4], the role of iterations in LE 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. The problem of MPLE control synthesis based on siphon analysis in *non-ordinary* nets has not been well-resolved yet [7]. In [1], the "max-controlled-siphon-property" is proposed; however, siphon-

H. Liao, S. Lafortune and S. Mahlke are with the Department of EECS, The University of Michigan, Ann Arbor, MI 48109, USA, {hwliao, stephane, mahlke}@eecs.umich.edu. S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, spyros@isye.gatech.edu. Y. Wang is with HP Labs, Palo Alto, CA 94303, USA, yin.wang@hp.com. This work was partially supported by NSF grant CCF-0819882 and an award from HP Labs Innovation Research Program (Michigan) and by NSF grants CMMI-0619978 and CMMI-0928231 (Georgia Tech).

based control synthesis by enforcing this property is not maximally permissive in general. In the Gadara project, maximal permissiveness is an important requirement for the control policy as it asserts that the behavior of the original program is least impacted. Another important requirement in this application area is ease of implementation of the control policy, in order to minimize the run-time overhead. In this regard, monitor-based control is very attractive [9].

The main contributions of this paper address the above requirements and are 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) 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. Section II reviews results about Petri nets. In Section III, the definition and properties of Gadara nets are reviewed and their implications for control synthesis are discussed. Section IV presents the development of the ICOG Methodology and UCCOR Algorithm. To make the paper self-contained while not exceeding page limitations, we were not able to include proofs of our results and had to summarize some of the specifics of the UCCOR Algorithm. (These are presented in a comprehensive report available from the authors.) For the same reason, only a simple example is presented.

II. Petri net preliminaries

A. Standard definitions

Definition 1: 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, $P = \{p_1, p_2, ..., p_n\}$ is the set of places, $T = \{t_1, t_2, ..., t_m\}$ is the set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, $W : A \rightarrow \{0, 1, 2, ...\}$ is the arc weight function, and for each $p \in P$, $M_0(p)$ is the initial number of tokens in p.

The *marking* 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, which is 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, addition of monitor (control) places 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.

Definition 2: 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.

Definition 3: 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 4: 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.

By definition, P-semiflow is a special case of P-invariant. A straightforward property of P-invariants is given by the following well known result [10]: If a vector y is a P-invariant of Petri net $\mathcal{N} = (P, T, A, M_0)$, then we have $M^T y = M_0^T y$ for any reachable marking $M \in R(\mathcal{N}, M_0)$. The *support* of P-semiflow y, denoted as $\| y \|$, 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* [10].

B. Control synthesis for Petri nets

Supervision Based on Place Invariants (SBPI) [4] 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

$$l^T M \leq b \tag{1}$$

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 1: [4] 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 $l^T M \leq b$ when included in the closed-loop system. This supervision is maximally permissive.

A very important property of SBPI is that the synthesized control logic is guaranteed to be *maximally permissive* with respect to the given linear inequality specification, which is proved in [9]. Therefore, a transition in the net is disabled by the monitor only if its firing leads to a state (marking) where the linear constraint in (1) is violated.

III. The Gadara Net Model and Its Main Properties

A. Gadara Petri nets

Gadara nets, a new class of Petri nets introduced in [16], are formally defined to model lock allocations and releases of multithreaded C programs.

Definition 5: [16] Let $I_{\mathcal{N}} = \{1, 2, ..., m\}$ be a finite set of process subnet 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) $P_R = \{r_1, r_2, ..., r_n\}$, $n > 0$.
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.
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) $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$.

A Gadara net is defined to be an ordinary Petri net, because it models mutex locks. Conditions 1 and 2 characterize a set of subnets $\mathcal{N}_i$ that define work processes (i.e., 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. Further discussion about Definition 5 is presented in [16]. 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:

$$\sum_{p \in \|Y_r\| \cap P_S} M(p) + M(r) = 1 \tag{2}$$

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

B. Controlled Gadara nets

When we use SBPI as the control technique on a Gadara net, we obtain an augmented net that we call a controlled Gadara net, which is defined in [16].

Definition 6: [16] 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 5 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 6 indicates that the introduction of the monitor places $p_c \in P_C$ preserves the net structure that is implied by Definition 5. 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 Property 2 as follows. However, the weights associated with the semiflows are not necessarily 1, due to the possibility that a monitor place can introduce non-unit arc weights and multiple initial tokens.

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

$$M^T Y_{p_c} = M_0(p_c) \tag{3}$$

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

Observe that $\mathcal{N}_G$ is a special subclass of $\mathcal{N}_G^c$ where $P_C = \emptyset$. Therefore, any property or algorithm that we derive for $\mathcal{N}_G^c$ applies to $\mathcal{N}_G$ as well. For simplicity, we refer to $\mathcal{N}_G^c$ as a Gadara net hereafter.

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.

Definition 7: 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'.

Definition 8: Place p is said to be a *disabling place* at marking M if there exists $t \in p\bullet$, s.t. $M(p) < W(p, t)$.

Definition 9: A nonempty set of places S is said to be a *siphon* if $\bullet S \subseteq S\bullet$.

Definition 10: A siphon S of a Gadara net $\mathcal{N}_G^c$ is said to be a *resource-induced siphon* if $S \cap (P_R \cup P_C) \neq \emptyset$.

Definition 11: A siphon S of a Gadara net $\mathcal{N}_G^c$ is said to be a *Resource-Induced Deadly Marked (RIDM) siphon* [13] 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.

Definition 12: Given a Gadara net $\mathcal{N}_G^c$ and a marking $M \in R(\mathcal{N}_G^c, M_0^c)$, the *modified marking* $\overline{M}$ is defined by

$$\overline{M}(p) = \begin{cases} M(p), & \text{if } p \notin P_0; \\ 0, & \text{if } p \in P_0. \end{cases} \tag{4}$$

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 $\overline{M}_1 = \overline{M}_2$.

The main properties of Gadara net are formally established in [16], and they serve as the foundation for the control synthesis in the present paper.

Theorem 2: [16] (1) $\mathcal{N}_G$ 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. (2) If $\mathcal{N}_G^c$ is ordinary, then $\mathcal{N}_G^c$ is live *iff* there does not exist a marking $M \in R(\mathcal{N}_G^c, M_0^c)$ and a siphon S such that S is an empty siphon at M.

We want to prevent any potential deadlock in a multi-threaded program that is modeled by $\mathcal{N}_G$. Deadlock-freeness of the program corresponds to liveness of the Gadara net. As

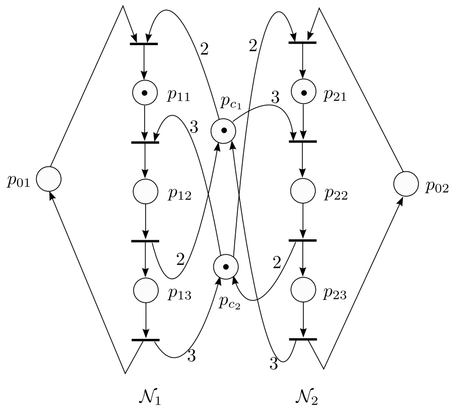

Fig. 1. Example of a nonempty RIDM siphon

established in Theorem 2, 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. Note that the synthesized monitor places, considered as generalized resource places, may introduce new potential deadlocks into $\mathcal{N}_G^c$, which have not been considered in the previous control synthesis. As a result, we may need to iterate the process of control synthesis for $\mathcal{N}_G^c$. When $\mathcal{N}_G^c$ remains ordinary, we can carry out control synthesis in the way similar to $\mathcal{N}_G$. There are many existing results for maximally permissive liveness-enforcing control synthesis of ordinary Petri nets. However, few results exist for non-ordinary Petri nets. When $\mathcal{N}_G^c$ becomes non-ordinary, we need Theorem 3 from [16] to guide control synthesis.

Theorem 3: [16] $\mathcal{N}_G^c$ 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 $\overline{M}$.

Theorem 3 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. Figure 1 shows an example of a nonempty RIDM siphon $S = \{p_{c_1}, p_{c_2}, p_{12}, p_{13}, p_{22}, p_{23}\}$. Therefore, in non-ordinary $\mathcal{N}_G^c$, instead of considering only empty siphons, we need to consider all the RIDM siphons that are present in the modified markings of the net. The above example implies that simply preventing the siphons from becoming empty is *not* sufficient for the control synthesis in non-ordinary $\mathcal{N}_G^c$. Control synthesis for RIDM siphons in non-ordinary $\mathcal{N}_G^c$ is the subject of the next section.

IV. Control synthesis of Gadara nets

In this section, we propose a new algorithm for controlling RIDM siphons in non-ordinary $\mathcal{N}_G^c$. 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. Note that since $\mathcal{N}_G$ is a special subclass of $\mathcal{N}_G^c$ and the empty siphons arising in $\mathcal{N}_G$ are essentially RIDM siphons, the UCCOR Algorithm works for both $\mathcal{N}_G^c$ and $\mathcal{N}_G$.

Similar to the modified marking defined in Section III-C, we further define the P_S-marking to facilitate the discussion.

Definition 13: Given a Gadara net $\mathcal{N}_G^c$ and a marking $M \in R(\mathcal{N}_G^c, M_0^c)$, the P_S*-marking* $\overline{\overline{M}}$ is defined by

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

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 $\overline{\overline{M}}_1 = \overline{\overline{M}}_2$.

Due to Property 4, 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 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 5 implies that $\overline{\overline{M}}$ is always a *binary* vector. It is this property that motivates us to focus on $\overline{\overline{M}}$.

A. Motivation for the UCCOR Algorithm

1) Control synthesis for ordinary Gadara nets: In [15], we presented a control synthesis algorithm for ordinary Gadara nets. We call it the Empty-Siphon-Based Control (ESBC) Algorithm in the following discussion. In the ESBC Algorithm, for each *place-minimal siphon* [2] S with respect to a set of generalized resource places, a linear inequality constraint is specified so that the sum of tokens in S is greater than 0. Then, SBPI is employed to enforce the constraint through a monitor place. Under the monitor places synthesized by the ESBC Algorithm, all the siphons in the ordinary Gadara nets will never become empty.

2) The need for iteration: Given the original Gadara net model $\mathcal{N}_G$, according to Theorem 2, we can use the ESBC Algorithm (or the UCCOR Algorithm to be presented) for ordinary nets to synthesize the control logic and obtain a set of monitor places with associated arcs. The resulting net is an augmented Gadara net, $\mathcal{N}_G^c$. We explained above that the added monitor place can be considered as a generalized resource place, and may introduce new potential deadlocks. Thus, in general we need to iterate the process of control synthesis. When $\mathcal{N}_G^c$ remains ordinary, we can continue to use the ESBC Algorithm. However, when $\mathcal{N}_G^c$ becomes non-ordinary, algorithms for controlling RIDM siphons are needed, according to Theorem 3. The UCCOR Algorithm, presented next, can be used for this purpose.

3) Overall strategy: We propose an Iterative Control Of Gadara nets (ICOG) Methodology based on the UCCOR Algorithm that, for the sake of generality, can take as input a potentially non-ordinary $\mathcal{N}_G^c$. The flowchart of ICOG is shown in Figure 2. Given a (controlled) Gadara net, we first see if there are any new RIDM siphons under the modified markings of the net. This can be done, for instance, by

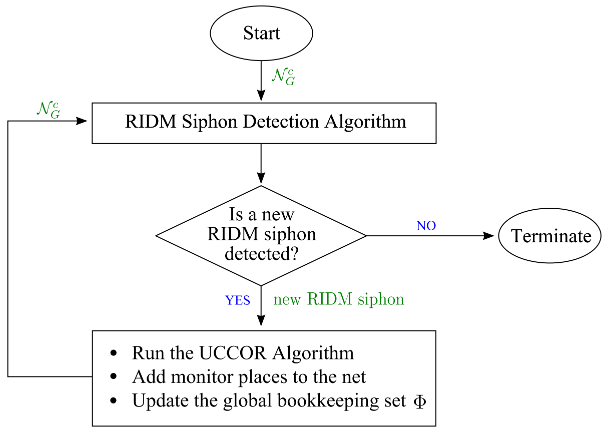
Start
$\mathcal{N}_G^c$
$\mathcal{N}_G^c$
RIDM Siphon Detection Algorithm
Is a new RIDM siphon detected?
NO
Terminate
YES
new RIDM siphon
Run the UCCOR Algorithm
Add monitor places to the net
Update the global bookkeeping set Φ

Fig. 2. Iterative Control of (controlled) Gadara nets (ICOG)

using a Mixed Integer Programming based approach that finds the maximal RIDM siphon in the net [13]. If no RIDM siphon is detected, then, according to Theorem 3, the net is live and ICOG terminates. Otherwise, we use the UCCOR Algorithm to *control* the detected RIDM siphon, i.e., to prevent it from becoming reachable. The algorithm outputs a set of monitor places, which are added to the net. After the UCCOR Algorithm, we go back to the first step of ICOG and determine if there are any *new* RIDM siphons remaining. 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.

We emphasize that the RIDM siphon detection is carried out under the *modified markings*, due to Theorem 3. The detected RIDM siphon, say S, will be characterized by the set of places S, and an associated modified marking $\overline{M}$. 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, we can always find the P_S-marking $\overline{\overline{M}}$ that corresponds to $\overline{M}$. Therefore, the UCCOR Algorithm can be implemented under the *P_S-markings* to control the detected RIDM siphon.

B. Fundamentals of the UCCOR Algorithm

1) Definitions and partial-marking analysis: In the UCCOR Algorithm, we want to synthesize control logic that can prevent the net from reaching any unsafe marking. The next set of definitions concretize this concept.

Definition 14: 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 resource-induced 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 14 and Theorem 3, we immediately have:

Corollary 1: A Gadara net is live *iff* it cannot reach a marking that is a RIDM-unsafe marking.

Moreover, it is possible for the net to reach a marking M, such that M is not a RIDM-unsafe marking, but starting from M, the net will unavoidably lead to a RIDM-unsafe marking. In this case, we call M an unsafe marking.

Definition 15: M is said to be an *unsafe* marking if M is either a RIDM-unsafe marking or a marking from which the net will unavoidably lead to a RIDM-unsafe marking.

Remark 1: For any unsafe marking that is not RIDM-unsafe, either it will be directly controlled, due to Step 2 of the UCCOR Algorithm (which we will discuss shortly), or it will eventually become RIDM-unsafe in future iterations, which will then be controlled by the UCCOR Algorithm. Thus, in the rest of this section, we can focus our attention to RIDM-unsafe markings based on structural analysis.

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 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. Thus, in the control synthesis, we want to prevent any marking M, such that $\overline{M}(S) = \overline{M}_u(S)$. This leads to:

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

In the control synthesis, we want to consider RIDM-unsafe *partial* markings so that each synthesized monitor place can prevent more than one RIDM-unsafe markings. 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 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 Figure 1 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, Figure 5 (without considering the dashed line) shows the same net under its initial marking M_0. M_0 is not RIDM-unsafe by assumption. 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 has been defined in Section II. The definition can be extended for any resource-induced siphon as follows.

Definition 16: Given a resource-induced siphon S, the *S-induced support of minimal semiflow*, denoted by $\|\tilde{Y}_S\|$, is

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

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

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, $\overline{\overline{M}}_u\big((S \cup \|\tilde{Y}_S\|) \cap P_S\big)$ 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, let us define $\Theta_S := (S \cup \|\tilde{Y}_S\|) \cap P_S$. This leads to:

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

Remark 2: 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 constraint 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 17: 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 $p \in P$, $C(p)$ represents the covering component value on p. This notation can be extended to a set of places $Q \subseteq P$ in a natural way. As we can restrict our attention to the set of P_S-markings, we assume, from now on, that $C(p) = \chi$, $\forall p \in P_0 \cup P_R \cup P_C$.

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. Similarly, a covering C is said to be an *unsafe* covering if for all P_S-markings $\overline{\overline{M}}$ it covers, the corresponding M is unsafe.

Remark 3: As a result of Proposition 2 and the notion of covering, for any RIDM siphon S to be controlled, 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 4: By Definition 17, a covering is a generalized P_S-marking, thus 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 linear inequality constraint is always 0.

3) Feasibility of maximally permissive control: In [16], we have established a "convexity-type" property of Gadara nets. It is shown that for any set of reachable markings in $\mathcal{N}_G^c$, the set is always convex so that the control synthesis can separate this set from the rest of the state space of the net, by enforcing a finite set of linear inequality constraints through monitor places. Following Remarks 3 and 4, this property can also be generalized to any set of RIDM-unsafe coverings with respect to some given RIDM siphon S.

Theorem 4: 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 constraints $\Lambda = \{(l_1, b_1), (l_2, b_2), ...\}$ such that C is RIDM-unsafe with respect to S *iff* $\exists (l_i, b_i) \in \Lambda$, $l_i^T C > b_i$.

Note that we assume that the condition of Remark 4 is enforced in Theorem 4. Theorem 4 implies that it is feasible to implement maximally permissive control using monitor-based control in terms of unsafe coverings.

C. UCCOR Algorithm

We can now formally present the UCCOR Algorithm. Our presentation will be carried out in a top-down manner. We will first give the overall procedures of the UCCOR Algorithm in Figure 3, and then explain the embedded modules in subsequent sections.

```
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) with respect to $S$
Method:
  1. Take the RIDM siphon $S$ 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 for each unsafe covering $C_u \in \mathcal{C}_u^{(2)}$,
      synthesize a monitor place $p_c$ as follows.
      Connectivity: $D_{p_c} = -l_{C_u}^T D$
      Initial Marking: $M_0(p_c) = b_{C_u} - l_{C_u}^T M_0 = b_{C_u}$
      where, $l_{C_u}(p) = \begin{cases} 1, & \text{if } C_u(p) = 1; \\ 0, & \text{otherwise.} \end{cases}$
             $b_{C_u} = \sum_{p: p \in \Theta_S \text{ and } C_u(p)=1} C_u(p) - 1$
```

Fig. 3. UCCOR Algorithm

Figure 4 is the flowchart of the UCCOR Algorithm. 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 $\overline{M}(S) = \overline{M}_u(S)$, are 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 unsafe coverings obtained form Step 1, by utilizing a certain type of monotonicity property of Gadara nets. It outputs a modifed set of unsafe 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 unsafe 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 unsafe covering in $\mathcal{C}_u^{(2)}$.

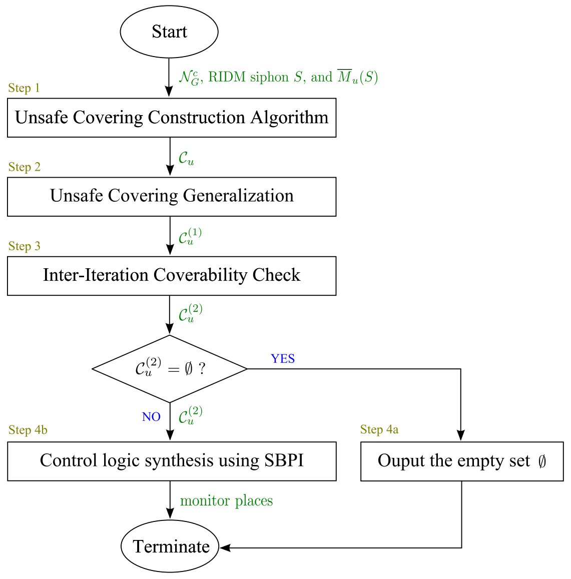
Start
$\mathcal{N}_G^c$, RIDM siphon S, and $\overline{M}_u(S)$
Step 1
Unsafe Covering Construction Algorithm
$\mathcal{C}_u$
Step 2
Unsafe Covering Generalization
$\mathcal{C}_u^{(1)}$
Step 3
Inter-Iteration Coverability Check
$\mathcal{C}_u^{(2)}$
$\mathcal{C}_u^{(2)} = \emptyset$?
YES
NO
$\mathcal{C}_u^{(2)}$
Step 4b
Control logic synthesis using SBPI
Step 4a
Ouput the empty set $\emptyset$
monitor places
Terminate

Fig. 4. Flowchart of the UCCOR Algorithm

For example, if $[1, \chi, \chi, 1] \in \mathcal{C}_u^{(2)}$ is an unsafe covering corresponding to the set of places $\{p_1, p_2, p_3, p_4\}$, then, according to Step 4b, we can specify a linear inequality constraint in the form of Equation (1) to prevent this unsafe covering: $1\cdot M(p_1)+0\cdot M(p_2)+0\cdot M(p_3)+1\cdot M(p_4) \leq 2-1$, which is in turn enforced by a monitor place.

The following interesting property of the UCCOR Algorithm can be demonstrated[1]:

Proposition 3: In $\mathcal{N}_G^c$, for any monitor place synthesized by the UCCOR Algorithm, its associated incoming and outgoing arcs all have unit arc weights.

Define Φ to be the set of coverings that are unsafe and have already been controlled in the previous iterations. One can think of Φ as a global "bookkeeping set" in the control synthesis process, which records all the unsafe coverings that have been controlled so far. The set Φ helps us to determine the convergence of ICOG. Since Φ only needs to record a relatively *small* number of unsafe coverings to keep track of a relatively *large* number of unsafe markings, the complexity of the bookkeeping process is greatly reduced – a saving on both time and space. The set Φ is updated 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 unsafe coverings that are controlled in this iteration.

D. 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 briefly described as follows.

Firstly, 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, we 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) $C\big((S \setminus \|\tilde{Y}_S\|) \cap P_S\big) = \overline{M}_u\big((S \setminus \|\tilde{Y}_S\|) \cap P_S\big)$; and, (iii) $C(p) = \chi$, $\forall p \notin \Theta_S$. The resulting set of coverings is the output of this algorithm, denoted as $\mathcal{C}_u$.

E. 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 unsafe 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 unsafe markings, a well-known result for Petri nets that model resource allocation systems.

Theorem 5: In $\mathcal{N}_G^c$, if M is an unsafe marking and M satisfies all the semiflow equations, then for any marking M' such that $\overline{\overline{M'}} >_d \overline{\overline{M}}$ and M' satisfies all the semiflow equations, M' is also an unsafe marking.

Remark 5: As a consequence of Theorem 5, for a given unsafe covering $C_u \in \mathcal{C}_u$ that needs to be controlled, any covering C', such that $C' >_d C_u$, satisfies the following: for any P_S-marking $\overline{\overline{M}}$ covered by C', the corresponding marking M is either reachable and unsafe, or not reachable. Therefore, any 0 component in C_u can be replaced by χ, and the resulting covering is denoted as C', where $C' >_d C_u$. In the control synthesis, we can prevent C' instead of C_u.

In the Unsafe Covering Generalization, we "generalize" each $C \in \mathcal{C}_u$ by replacing any 0 component in C by χ, and obtain a corresponding modified unsafe covering $C^{(1)}$. The resulting set of modified coverings is the output, 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 unsafe 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 unsafe markings, or markings that are not reachable, due to Remark 5.

Intuitively, $\mathcal{C}_u^{(1)}$, in general, covers a larger set of unsafe

[1]The implications of this property will be addressed in follow-up work.

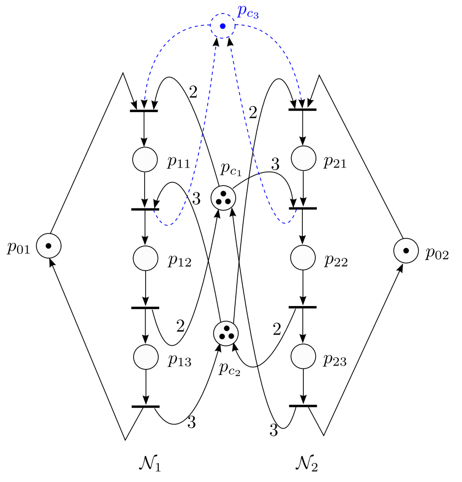

Fig. 5. A simple example of the UCCOR Algorithm

markings than $\mathcal{C}_u$ does. Thus, by considering $\mathcal{C}_u^{(1)}$ in the UCCOR Algorithm afterwards, the synthesized monitor places are more efficient, in terms of number of unsafe markings 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.

F. Inter-Iteration Coverability Check

In the Inter-Iteration Coverability Check, each pair of coverings $(C_1, C_2) \in \{(C_1, C_2) : C_1 \in \mathcal{C}_u^{(1)}$ 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 ingoing 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 unsafe coverings corresponding to $\mathcal{C}_u^{(1)}$, denoted as $\mathcal{C}_u^{(2)}$, and updates Φ.

G. A simple example of the UCCOR Algorithm

Example 1: In Figure 1, we gave a controlled Gadara net that contains a RIDM siphon. The monitor place, which is synthesized by the UCCOR Algorithm and prevents the RIDM-unsafe marking with respect to this RIDM siphon, is shown in dashed line in Figure 5.

H. Correctness and maximal permissiveness

In Section IV-A.2, we presented the global flowchart of the ICOG Methodology. Here, we present its main properties.

Theorem 6: 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 liveness enforcement.

Theorem 7: ICOG converges in a finite number of iterations.

Theorem 8: ICOG is correct and maximally permissive with respect to the goal of liveness enforcement.

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.

V. Conclusion

We presented the ICOG Methodology and the associated UCCOR Algorithm for the synthesis of MPLE control policies for Gadara nets. Taking any RIDM siphon as input, the UCCOR Algorithm synthesizes monitor places that prevent all possible unsafe states, with respect to the given RIDM siphon, from becoming reachable. Using the notion of covering of unsafe states, each monitor place synthesized by the UCCOR Algorithm can prevent more than one unsafe states and can control more than one RIDM siphons. ICOG applies the UCCOR Algorithm until all RIDM siphons are controlled by at least one monitor place; this convergence is achieved in a finite number of iterations using bookkeeping of coverage of unsafe states, and preserves the property of maximal permissiveness.

References

[1] K. Barkaoui and J.-F. Pradat-Peyre. On liveness and controlled siphons in Petri nets. *Proc. the 17th International Conference on Applications and Theory of Petri Nets*, pages 57–72, 1996.

[2] E. R. Boer and T. Murata. Generating basis siphons and traps of Petri nets using the sign incidence matrix. *IEEE Trans. on Circuits and Systems—I*, 41(4):266–271, Apr. 1994.

[3] A. Ghaffari, N. Rezg, and X. Xie. Design of a live and maximally permissive Petri net controller using the theory of regions. *IEEE Transactions on Robotics and Automation*, 19(1):137–142, 2003.

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

[5] M. V. Iordache and P. J. Antsaklis. Concurrent program synthesis based on supervisory control. In *ACC '10*, pages 3378–3383, 2010.

[6] T. Kelly, Y. Wang, S. Lafortune, and S. Mahlke. Eliminating concurrency bugs with control engineering. *IEEE Computer*, 42(12):52–60, December 2009.

[7] Z. Li and M. Zhou. *Modeling, Analysis, and Deadlock Control of Automated Manufacturing Systems*. Science Press, Beijing, 2009.

[8] Z. Li, M. Zhou, and N. Wu. A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems. *IEEE Trans. on Systems, Man, and Cybernetics—Part C*, 38(2):173–188, Mar. 2008.

[9] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, Boston, MA, 1998.

[10] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

[11] T. Murata, B. Shenker, and S. M. Shatz. Detection of Ada static deadlocks using Petri net invariants. *IEEE Trans. on Software Engineering*, 15(3):314–326, Mar. 1989.

[12] A. Nazeem, S. Reveliotis, Y. Wang, and S. Lafortune. Optimal deadlock avoidance for complex resource allocation systems through classification theory. In *WODES '10*, 2010.

[13] S. A. Reveliotis. *Real-Time Management of Resource Allocation Systems: A Discrete-Event Systems Approach*. Springer, New York, NY, 2005.

[14] M. Uzam. An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions. *International Journal of Advanced Manufacturing Technology*, 19(3):192–208, 2002.

[15] Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke. The theory of deadlock avoidance via discrete control. In *POPL '09*.

[16] 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 *CDC '09*, pages 4971–4976.