2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Deadlock-Avoidance Control of Multithreaded Software: An Efficient Siphon-based Algorithm for Gadara Petri Nets

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

Abstract— **This paper presents an efficient implementation of an iterative control algorithm for the synthesis of maximally-permissive liveness-enforcing control policies for Gadara nets presented in earlier work. Gadara nets are a special class of Petri nets arising when modeling multithreaded software for the purpose of deadlock analysis and resolution. The considered control synthesis algorithm is based on structural analysis of Gadara nets in terms of a certain type of siphons, called resource-induced deadly-marked siphons. We propose a new customized mixed integer programming formulation to detect these siphons in Gadara nets. We then compare the performance of our customized algorithm with that of a generic siphon detection algorithm for process-resource nets in the context of the iterative control algorithm. Finally, we investigate the scalability of the overall algorithm to large program models.**

I. Introduction

Petri nets, one of the most popular modeling formalisms for Discrete Event Systems (DES), are an efficient tool to capture the concurrency of a dynamic system [17]. Deadlock analysis of Petri nets has been extensively investigated in the context of flexible manufacturing systems and other technological applications involving a resource allocation function [14], [21]. Recently, there has also been a growing interest in the application of DES to computer systems [10], [1], [6]. In our on-going Gadara project [11], we use Petri nets to systematically model, analyze, and control shared-memory multithreaded C programs with lock allocation and release operations, for the purpose of deadlock avoidance. Petri nets provide a compact representation of the program dynamics without enumerating the entire reachable state space. This feature of Petri nets is crucial for the scalability of our method to real-world software. Multithreaded programming is a common paradigm for concurrent software, which aims to exploit the full potential of multicore hardware architectures. Compared to the traditional serial programming, reasoning about concurrency in the multithreaded programming paradigm is much more challenging for human programmers. Lock primitives, such as *mutual exclusion locks (mutexes)*, are often employed to protect shared data and prevent data races. Inappropriate use of mutexes can lead to *circular-mutex-wait (CMW) deadlocks* in the program, where a set of threads are waiting indefinitely for one another and none of them can proceed. In the remainder of this paper, deadlock freeness of a program will refer to the absence of CMW deadlocks.

In [24], [16], we proposed a new class of Petri nets, called Gadara nets, to systematically model multithreaded C programs with lock allocation and release operations; the main properties of Gadara nets that are important to deadlock analysis were also established. Gadara nets share features with classes of Petri nets that arise in the modeling of manufacturing systems, such as S^3PR and S^4PR, even though they are distinctly different from these classes. The properties of Gadara nets reveal that a multithreaded program that can be modeled as a Gadara net is CMW-deadlock-free *if and only if* its associated Gadara net is *live* [16]. This correspondence motivated our investigation of a structural approach to maximally-permissive liveness-enforcing (MPLE) control of Gadara nets in [15]. (In parallel, we are also pursuing an alternative approach based on state space expansion and classification theory [18].) The control synthesis presented in [15] is based on structural analysis of Gadara nets in terms of a certain type of siphons, called resource-induced deadly-marked (RIDM) siphons. There have been results reported in the literature on deadlock prevention policies using siphon analysis; see, e.g., [5], [12], [8], [19]. But these results do not exploit the special structure of Gadara nets, and thus they are not well suited for our pursued MPLE control policy. In [22], an iterative liveness-enforcing control based on siphon analysis has been studied for the class of S^4PR nets. But this method is sub-optimal in general, i.e., it does not guarantee maximal permissiveness. In [9], 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, the problem of MPLE control synthesis based on siphon analysis in non-ordinary Petri nets has not been fully resolved yet. Compared to existing results on structure-based liveness-enforcing control, the main contribution of the control synthesis technique proposed in [15] is that, under certain conditions, it achieves two metrics of *effectiveness* at the same time: *correctness* and *maximal permissiveness* with respect to the goal of liveness enforcement. These two metrics of effectiveness are independent of the method used to detect the RIDM siphons in the net. On the other hand, we also noticed that the RIDM siphon detection algorithm

H. Liao, J. Stanley, S. Lafortune, and S. Mahlke are with the Department of EECS, The University of Michigan, Ann Arbor, MI 48109, USA, {hwliao, jasonsta, stephane, mahlke}@eecs.umich.edu.
Y. Wang is with HP Labs, Palo Alto, CA 94303, USA, yin.wang@hp.com.
S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, spyros@isye.gatech.edu.
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).

does play a significant role in the *efficiency* of the iterative control strategy; *this is the main topic of the present paper*.

In Petri nets, siphon is a well-defined *structural* construct that is important to the analysis of certain *behavioral* properties of the net, such as deadlock-freeness and liveness [20], [21]. Many approaches have been proposed for siphon detection in Petri nets. In [2], a basis siphon generation algorithm is presented using the sign incidence matrix derived from the original incidence matrix of the net. In the Gadara project, an efficient siphon detection algorithm using the so-called *lock dependency graph* is reported in [23], where the correspondence between place-minimal siphons and strongly connected components in the graph is studied. Recently, a similar method of siphon detection using graph theory has been applied to the class of S^4PR nets [3].

In contrast to the above explicit siphon generation approaches, a Mixed Integer Programming (MIP) formulation is proposed in [4] to detect the maximal empty siphon that is reachable in a structurally bounded ordinary net. Based on the detected maximal empty siphon, strict minimal siphons can be derived. This method has been employed for the synthesis of more permissive liveness-enforcing supervisors in the class of S^3PR nets [13]. The notion of RIDM siphon was introduced in [21]. RIDM siphons serve as a generic structural construct to characterize the liveness of general process-resource nets that need not be ordinary. In an ordinary net, an empty siphon that involves resource places is a special case of a RIDM siphon. For Gadara nets, liveness of the net can also be characterized in terms of RIDM siphons [24], [16]. In [21], an MIP formulation is proposed to detect maximal RIDM siphons in general process-resource nets.

In the control synthesis methodology presented in [15], we know that if the iterative control strategy starts with a Gadara net that is ordinary, then the resulting controlled Gadara nets will remain ordinary throughout the iterations. In this paper, we exploit this property and propose a customized RIDM siphon detection algorithm that focuses on the case of ordinary Gadara nets. At each iteration, a RIDM siphon is detected by solving a customized MIP; then, the detected RIDM siphon is prevented from becoming reachable by synthesizing MPLE monitor place(s). This iterative process is guaranteed to converge in a finite number of iterations as it is a special case of the general methodology investigated in [15]. In summary, reference [15] focuses on the general methodology of MPLE control synthesis of Gadara nets, while the present paper concentrates on its efficient implementation, performance evaluation, and scalability analysis.

The main contributions of this paper are summarized as follows. (i) We present an efficient MIP-based RIDM siphon detection algorithm for the iterative control synthesis of Gadara nets, where the problem of siphon detection is mapped to the problem of finding a "total-deadlock modified-marking." (ii) We report on the experimental results of a comparative analysis between the performance of the iterative control strategy based on the proposed RIDM siphon detection algorithm with that of the original RIDM siphon detection algorithm in [21] for general process-resource nets. The proposed method detects a RIDM siphon significantly more quickly at each iteration; moreover, it requires fewer iterations to converge. (iii) We present a sample of experimental results that demonstrate the scalability of the proposed iterative control strategy.

This paper is organized as follows. Section II reviews the definitions and control synthesis results pertaining to Gadara nets. In Section III, we present the customized MIP-based RIDM siphon detection algorithm for the iterative control synthesis of Gadara nets. The experimental results on the comparative analysis and the scalability study of the iterative control process are reported in Section IV. We conclude the paper in Section V. Due to space limitations, proofs and examples are omitted. (These are presented in a comprehensive report available from the authors.)

II. Gadara Petri Nets

Gadara nets are a special class of Petri nets that are employed to systematically model multithreaded C programs with lock allocation and release operations, for the purpose of deadlock analysis and resolution. In this section, we review important results on Gadara nets.

A. Definitions of Gadara nets

We assume readers are familiar with standard Petri net definitions; see Section II of [15] for necessary background and [17] for a comprehensive treatment. The class of Gadara nets [24], [16] is defined as follows.

Definition 1: [24], [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) $P_R = \{r_1, r_2, ..., r_n\}$, $n > 0$. P_0 is the set of idle (or initial) places, P_S the set of *operation* places of the process subnets, and P_R the set of *resource* places that are shared among the process subnets.
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) $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$.

Supervision Based on Place Invariants (SBPI) [9] is a common monitor-based control technique for Petri nets. This approach enforces any given specification that is in the form of a linear inequality on the net markings by synthesizing a monitor place. When we use SBPI as the control technique on a Gadara net, the original net will be augmented by the synthesized monitor places with their associated arcs. The resulting augmented net is called a *controlled* Gadara net [24], [16], and it is formally defined as follows.

Definition 2: [24], [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 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 $p_c \in P_C$, $M_0^c(p_c) \geq \max_{p \in P_S} Y_{p_c}(p)$.

From Definition 1, we know that an original Gadara net $\mathcal{N}_G$ is ordinary, due to the nature of the multithreaded programs it models. On the other hand, from Definition 2, a controlled Gadara net $\mathcal{N}_G^c$ need not be ordinary *in general*, because the added monitor places may introduce associated arcs with weights greater than one. Condition 4 of Definition 1 implies that any transition that models a branch selection in the program (e.g., `if/else`) should not be constrained by any place in P_R, due to the inherent nature of the program. In the case of $\mathcal{N}_G^c$, it might be desirable to extend the scope of that condition to include monitor places in P_C in addition to resources places in P_R. Specifically, let Condition 4' be the equivalent to Condition 4 with P_R replaced by P_C. Condition 4' implies that all the branching transitions are *uncontrollable*, which is consistent with the requirement that the control logic for the software program should not interfere with its branching behavior. If a controlled net $\mathcal{N}_G^c$ satisfies Condition 4', then it is said to be *admissible*; otherwise, it is said to be *inadmissible*. The notion of admissibility is defined for monitor places in a similar way. In the remainder of this paper, we shall only consider admissible $\mathcal{N}_G^c$ and admissible monitor places, and therefore we have the following assumption:

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

B. Control synthesis for Gadara nets

We first present some relevant definitions.

The *reachable state space* $R(\mathcal{N}, M_0)$ of net $\mathcal{N}$ is the set of all markings reachable by transition firing sequences starting from M_0. The set of reachable markings of net $\mathcal{N}$ starting from M is denoted as $R(\mathcal{N}, M)$.

Definition 3: 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 4: 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 5: A nonempty set of places S is said to be a *siphon* if $\bullet S \subseteq S\bullet$.

Definition 6: A siphon S in a Gadara net $\mathcal{N}_G^c$ is said to be a *Resource-Induced Deadly Marked (RIDM) siphon* [21] at marking M, if it satisfies the following conditions:

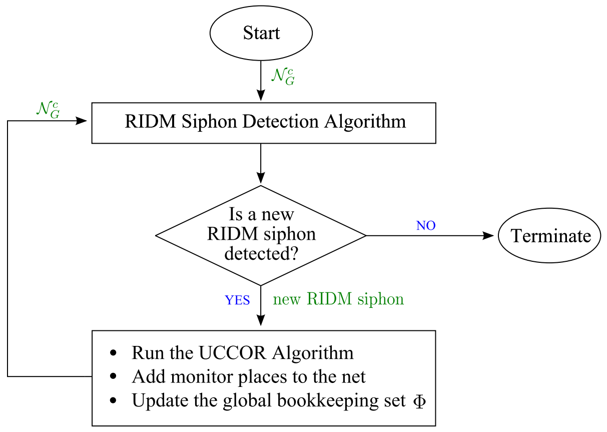
1) every $t \in \bullet S$ is disabled by some $p \in S$ at M;
2) $S \cap (P_R \cup P_C) \neq \emptyset$;
3) $\forall p \in S \cap (P_R \cup P_C)$, p is a disabling place at M.

In order to characterize liveness in Gadara nets, a *behavioral* property, in terms of RIDM siphons, a *structural* property, we need the notion of modified marking [21].

Definition 7: 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

$$\overline{M}(p) = \begin{cases} M(p), & \text{if } p \notin P_0; \\ 0, & \text{if } p \in P_0. \end{cases} \tag{1}$$

Next, we define the notion of *unsafe* marking. M is said to be a *RIDM-unsafe* marking if there exists at least one RIDM siphon at the corresponding modified marking $\overline{M}$. 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.



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. 1. Iterative Control of (controlled) Gadara nets (ICOG)

In order to enforce liveness for Gadara nets, we want to prevent all unsafe markings from becoming reachable in the controlled Gadara nets through control synthesis.

We now review the Iterative Control Of Gadara nets (ICOG) Methodology proposed in [15], whose flowchart is shown in Figure 1. ICOG takes as input a potentially non-live Gadara net $\mathcal{N}_G^c$. ICOG first runs the RIDM Siphon Detection Algorithm to find a RIDM siphon that is potentially reachable in $\mathcal{N}_G^c$. According to the properties of Gadara nets established in [24], [16], if $\mathcal{N}_G^c$ is not live, then the algorithm will always detect a new RIDM siphon that has not been considered in previous iterations. On the other hand, if the algorithm does not detect any new RIDM siphon, ICOG terminates and outputs a live Gadara net. If a new RIDM siphon S is detected, ICOG runs the UCCOR Algorithm, which takes S as input and synthesizes MPLE monitor place(s) that prevent S from becoming reachable in the augmented controlled net. At the same time, a "global bookkeeping set" is maintained and updated; this set records all the unsafe states that have been prevented so far. At the end of each iteration, the augmented $\mathcal{N}_G^c$ is fed into the RIDM Siphon Detection Algorithm, which signifies the start of a new iteration.

We briefly describe the UCCOR Algorithm as follows; the detailed development of UCCOR is presented in [15]. UCCOR is a new algorithm for controlling RIDM siphons in Gadara nets, where UCCOR is short for "Unsafe-Covering-based Control Of RIDM siphons". A *covering* C (refer to Definition 17 of [15]) is a generalized marking on the set of operation places, whose components can be $0, 1$, or χ. Here, "χ" stands for "0 or 1". The notions of RIDM-unsafe marking and unsafe marking can also be generalized in terms of coverings. Figure 2 is the flowchart of UCCOR. In Step 1, the set of possible RIDM-unsafe coverings with respect to the input RIDM siphon is solved. Step 2 further generalizes the unsafe coverings obtained from Step 1, by utilizing a certain type of monotonicity property of Gadara nets. In Step 3, the coverings that have already been controlled are removed from consideration. The output of this step is a further modified set

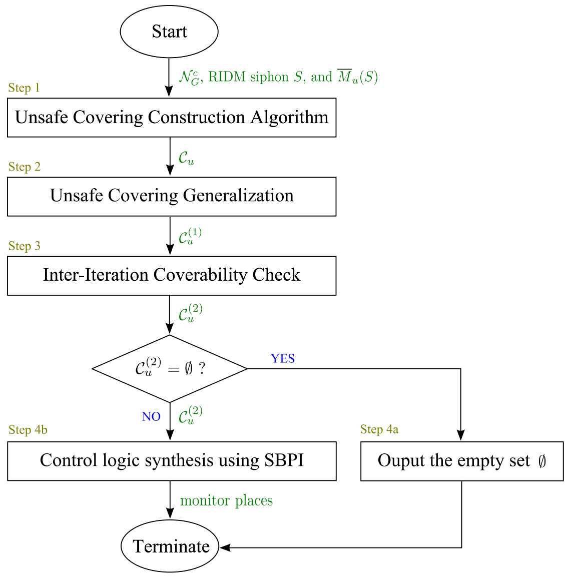

Start
$\mathcal{N}_G^c$, RIDM siphon S, and $\overline{M}_u(S)$
Step 1
Unsafe Covering Construction Algorithm
$\mathcal{C}_u$
Step 2
Unsafe Covering Generalization
$\mathcal{C}_u^{(1)}$
Step 3
Inter-Iteration Coverability Check
$\mathcal{C}_u^{(2)}$
$\mathcal{C}_u^{(2)} = \emptyset$?
YES
NO
$\mathcal{C}_u^{(2)}$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

Fig. 2. Flowchart of the UCCOR Algorithm

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 [9] is carried out. One MPLE monitor place will be synthesized for each unsafe covering in $\mathcal{C}_u^{(2)}$.[1]

We highlight the following important properties:

Proposition 1: [15] 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.

Corollary 1: If ICOG starts with a Gadara net that is ordinary, then the resulting controlled Gadara nets will remain ordinary throughout the iterations.

Theorem 1: [15] (a) 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. (b) ICOG is correct and maximally permissive with respect to the goal of liveness enforcement. (c) ICOG converges in a finite number of iterations.

III. Efficient RIDM Siphon Detection in Gadara Nets Using Mathematical Programming

The results of Section II-B show that both the ICOG Methodology and the associated UCCOR Algorithm are correct and maximally permissive with respect to the goal of liveness enforcement, *independent of the method used to detect the RIDM siphons*. However, the RIDM siphon detection algorithm used does play an important role in the efficiency of control synthesis; it is in fact the computational bottleneck of ICOG (see experimental results in Section IV). Mixed integer programming has been employed to detect maximal RIDM siphons in general process-resource nets [21]; we refer to this more general MIP formulation, stated on pp. 139-140 of [21] and not repeated here due to space constraints, as G-MIP hereafter. According to Corollary 1, we know that if ICOG takes as input an ordinary $\mathcal{N}_G^c$, then the resulting controlled Gadara nets will *remain ordinary* throughout the iterations. We exploit this nice property in this section and present a customized MIP formulation for efficient RIDM siphon detection in ordinary $\mathcal{N}_G^c$; we refer to this new Customized MIP formulation as C-MIP hereafter.

A. Key properties

We first present some properties of Gadara nets that are relevant to the development of our algorithm.

Definition 8: A Petri net is in a *total deadlock* if all the transitions in the net are disabled.

The set of modified markings (Definition 7) induced by the set of reachable markings is defined by $\overline{R}(\mathcal{N}_G^c, M_0^c) = \{\overline{M} | M \in R(\mathcal{N}_G^c, M_0^c)\}$. From the properties of Gadara nets established in [24], [16], we have the following result.

Proposition 2: If $\mathcal{N}_G^c$ is not live, then $\mathcal{N}_G^c$ will reach a modified marking $\overline{M} \in \overline{R}(\mathcal{N}_G^c, M_0^c)$, such that $M \neq \overline{M_0^c}$ and $\mathcal{N}_G^c$ is in a total deadlock at the modified marking $\overline{M}$.

The intuition of this proposition is explained as follows. If $\mathcal{N}_G^c$ is not live, then $\mathcal{N}_G^c$ can reach a marking, such that there exists at least one process subnet "half-way" in execution, where no transition in it can be fired, and the process subnet (as well as $\mathcal{N}_G^c$) cannot return to its initial marking. For all other process subnets that can successfully complete their executions, their tokens in the operation places will eventually return to the idle places. After all these process subnets complete their executions, their returned tokens in the idle places are erased under the notion of modified marking. Now, all transitions in $\mathcal{N}_G^c$ are disabled under the modified marking. Thus, we have a total-deadlock modified-marking that is different from the modified initial marking.

Based on the definitions of Gadara nets and RIDM siphons we can demonstrate the following two results. We use the term "generalized resource place" to refer to any place $p \in P_R \cup P_C$.

Proposition 3: Given a modified marking $\overline{M} \in \overline{R}(\mathcal{N}_G^c, M_0^c)$ and a set of places S, if $\mathcal{N}_G^c$ is in a total deadlock at the modified marking $\overline{M}$, and S contains P_0, P_S, and all the disabling generalized resource places at $\overline{M}$, then S is a RIDM siphon at marking $\overline{M}$.

Proposition 4: In an ordinary $\mathcal{N}_G^c$, any RIDM siphon contains at least two generalized resource places.

B. The customized MIP formulation for Gadara nets

Let $I_{\overline{M}}(p)$ be an indicator variable associated with place p at modified marking $\overline{M}$. The value of $I_{\overline{M}}(p)$ is defined as:

$$I_{\overline{M}}(p) = \begin{cases} 1, & \text{if place } p \text{ contains at least one token at } \overline{M}; \\ 0, & \text{if place } p \text{ does not contain any token at } \overline{M}. \end{cases} \tag{2}$$

Remark 1: Since $\mathcal{N}_G^c$ remains ordinary throughout the control synthesis, we know that: (i) $I_{\overline{M}}(p) = 1$ *iff* p enables all of its output transitions at $\overline{M}$, and (ii) $I_{\overline{M}}(p) = 0$ *iff* p disables all of its output transitions at $\overline{M}$.

Define $SB(p)$ to be a *structural bound* of place p. In Gadara nets, we can set: $SB(p) = M_0^c(p)$, $\forall p \in P_0 \cup P_C$, and $SB(p) = 1$, $\forall p \in P_S \cup P_R$.

According to Proposition 2, if $\mathcal{N}_G^c$ is not live, then we know *a priori* that the net will reach a total deadlock at modified marking $\overline{M}$, where all the transitions in the net are

[1]In [15] it was implicitly assumed that the synthesized monitor places satisfy Assumption 1. A more detailed discussion of the necessary modifications of the SBPI method so that it guarantees the satisfaction of Assumption 1 in the considered application context, is provided in an extended journal version of the results of [15], that is currently under review.

disabled. Moreover, once $\overline{M}$ is reached, we know *a priori* from Proposition 3 that there exists a RIDM siphon S under $\overline{M}$. This RIDM siphon is constructed as follows:

$$S = \{p \in P : (p \in P_0 \cup P_S) \vee (I_{\overline{M}}(p) = 0)\} \tag{3}$$

This implies that we can find a RIDM siphon in a non-live $\mathcal{N}_G^c$ in a very efficient way by proceeding in two steps:

1) Find a total-deadlock modified-marking $\overline{M}$;
2) Based on $\overline{M}$, construct a RIDM siphon using (3).

So the problem of detecting a RIDM siphon in a non-live $\mathcal{N}_G^c$ can be mapped to the problem of finding a total-deadlock modified-marking in the modified reachability space. The latter one can be solved by the formulation C-MIP below:

$$\min \quad \sum_{p \in P_S} I_{\overline{M}}(p) \tag{4}$$

$$s.t. \quad M = M_0^c + D\sigma \tag{5}$$

$$\overline{M}(p) = M(p), \forall p \notin P_0; \overline{M}(p) = 0, \forall p \in P_0 \tag{6}$$

$$I_{\overline{M}}(p) = 0, \forall p \in P_0 \tag{7}$$

$$I_{\overline{M}}(p) = 0, \forall p \in Q, \text{where}, \tag{8}$$

$$Q = \{q \in P : (\exists t \in T), (\bullet t = \{q\}) \wedge (q \in P_S)\}$$

$$\sum_{p \in \bullet t} I_{\overline{M}}(p) - |\bullet t| + 1 \leq 0, \forall t \text{ s.t. } |\bullet t| > 1 \tag{9}$$

$$\overline{M}(p) \geq I_{\overline{M}}(p) \geq \frac{\overline{M}(p)}{SB(p)}, \forall p \in P_S \cup P_R \cup P_C \tag{10}$$

$$\sum_{p \in P_R \cup P_C} I_{\overline{M}}(p) \leq |P_R \cup P_C| - 2 \tag{11}$$

$$\sum_{p \in P_S} I_{\overline{M}}(p) \geq 2 \tag{12}$$

$$M \geq 0; \sigma \in \mathbb{Z}_0^+; I_{\overline{M}}(p) \in \{0, 1\}, \forall p \in P \tag{13}$$

Before explaining the above formulation, we briefly review some results derived in [15] that are relevant to the development of this formulation. From Conditions 5, 6, and 7 of Definition 1, we know that the marking of any operation place can only be 0 or 1. Further, we have shown in [15] that Gadara nets have a so-called *monotonicity property*, which can be illustrated by the following simple example. Consider an unsafe marking M_1, whose partial marking on the set of operation places P_S is $[1, 1, 0, 0]^T$. Then, any reachable marking M_2, such that its partial marking on P_S is $[1, 1, 0, 1]^T$, $[1, 1, 1, 0]^T$, or $[1, 1, 1, 1]^T$ (which dominates $[1, 1, 0, 0]^T$), is also unsafe. Using the notion of covering, discussed in Section II-B, any reachable marking, whose partial marking on P_S is covered by $[1, 1, \chi, \chi]^T$, is unsafe.

From the above discussion, we observe that for any given unsafe marking associated with a detected RIDM siphon, we can apply the monotonicity property and consider its partial marking on P_S. Any "0" component in this partial marking can be generalized (i.e., replaced) by "χ"; and, any "1" component in this partial marking remains unchanged. The resulting generalized marking is called an *unsafe covering*, such that all markings it covers are unsafe. This process is carried out in Step 2 of the UCCOR Algorithm.

We are now ready to explain the C-MIP formulation presented in (4) – (13). In the objective function (4), we want to minimize the number of marked operation places in the detected RIDM siphon. In this way, the unsafe covering that is controlled in Step 4b of the UCCOR Algorithm contains the smallest possible number of 1's, or equivalently, the largest possible number of χ's. The intuition is that the more χ's an unsafe covering contains, the more unsafe markings it can potentially cover. The RIDM siphon detected by this algorithm involves the smallest number of activated threads that can be entangled in a deadlock. This attribute of the detected RIDMs when combined with the covering effect of the UCCOR Algorithm mentioned above, tends to reduce the total number of iterations needed in ICOG. This effect is highlighted in the experiments presented in Section IV.

Constraint (5) is the state equation of the net. Constraint (6) connects an original marking with its associated modified marking based on Definition 7. Constraint (7) specifies that the value of the indicator variable of any idle place is 0, since our RIDM siphon detection is carried out in the set of modified markings. From above, we want to detect a RIDM siphon by finding a total-deadlock modified-marking $\overline{M}$. Constraints (7), (8), and (9) enforce this condition. According to Remark 1, for a place to be a disabling place in an ordinary $\mathcal{N}_G^c$, its associated indicator variable must take the value of 0. In a total deadlocked Gadara net, consider the set of transitions that have only one input place: this single input place can be either a disabling idle place, which is addressed above in (7), or a disabling operation place, which is addressed in (8). Moreover, for the set of transitions that have more than one input place, Constraint (9) enforces that at least one input place must be a disabling place. Constraint (10) bounds the value of the indicator variable associated with any operation place or generalized resource place based on (2). Constraint (11) follows from Proposition 4. Constraint (12) follows from the fact that at least two threads must be involved in a deadlock incurred by lack of tokens in some generalized resource places, and that each process stage cannot have more than one active process instance, due to Condition 6 of Definition 1. Constraint (13) specifies the bounds of the variables. We see that the variable M can be nonnegative rational, while the other variables must be integers; thus, this is an MIP problem.

The solution of this C-MIP problem is a total-deadlock modified-marking $\overline{M}$ and its associated $I_{\overline{M}}$, based on which we can construct a RIDM siphon using (3). The correctness of the C-MIP formulation is established as a result of Propositions 2, 3, and 4, and the preceding discussion.

The G-MIP formulation for maximal RIDM siphon detection in general process-resource nets is $O(|P||T|)$ in terms of variables and constraints, because it includes binary variables and constraints that relate to the arcs of the net [21]. On the other hand, the number of variables and constraints used by C-MIP is $O(|P|+|T|)$; in particular, the formulation involves $2|P|$ non-negative real variables, $|P|$ binary variables, and $|T|$ non-negative integer variables.

IV. Experimental Evaluation of ICOG

In this section, we present a summary of experimental results from a comparative analysis between the performance of ICOG based on C-MIP, denoted as ICOG-C, and that of ICOG based on G-MIP, denoted as ICOG-G. We also report a sample of experimental results that demonstrate the scalability of ICOG-C. The experiments in this section were completed on a Mac OS X laptop with a 2.4 GHz Intel Core2Duo processor and 2 GB of RAM. Both ICOG-C and ICOG-G are entirely implemented in C++ and compiled under the GNU gcc compiler. The MIP formulations are solved using Gurobi 3.0.1 [7]. Random Gadara nets for these experiments are generated by a random-walk-style algorithm. At each step, the program randomly decides either to grab a lock or to release one already held; the number of steps is specified by the input parameters described below. Additional logic was applied to ensure valid behavior. The random Gadara net generator is based on our experience modeling real concurrent programs [23].

Table I presents a summary of the experimental results of the comparative analysis between the performance of ICOG-C and that of ICOG-G. For each set of parameters (each row in the table), over 90 samples of random Gadara nets are generated. The generated nets with no unsafe states are removed from the samples. The left half of each row shows the set of parameters of the sample random Gadara nets. The right half of each row is further divided into two sub-rows: the top sub-row is the performance of ICOG-C, and the bottom italicized sub-row is the performance of ICOG-G. We set a time-out threshold of 15 seconds for the stage of RIDM siphon detection in ICOG. A net times out if it cannot be solved by either C-MIP or G-MIP in less than 15 seconds. All statistical data in this table, such as means μ and standard deviations σ (except TLE in the tenth column), are calculated over only the sample nets where both ICOG-C and ICOG-G did not time out.

The first four columns correspond to the parameters used to generate sample random Gadara nets. The first column (l) is the number of resources (locks) in the original Gadara net. The second (s) and third (a) columns are the number of process subnets and the number of resource acquisitions per subnet. The fourth column (r) is the probability of acquiring a new resource before releasing one already held. The fifth (P) and sixth (T) columns correspond to the average number of places and transitions in the original Gadara nets. The seventh (SS) and eighth (US) columns describe the state space complexity, i.e., the average number of safe and unsafe states that are reachable by the original nets. *Note that ICOG does not construct this state space, since it exploits structural properties of Gadara nets; these numbers were generated separately for the sake of scalability assessment.* ICOG only needs to bookkeep the unsafe coverings of the net, the number of which is equal to the number of monitor places synthesized. The ninth column (n) is the number of generated Gadara nets, where both ICOG-C and ICOG-G did not ever time out throughout the iterations. The tenth column (TLE) shows the ratio of sample nets that timed out in any iteration of ICOG-C and ICOG-G. The eleventh column (time (s)) is the average time (in seconds) the entire ICOG-C and ICOG-G processes took until they converged. The twelfth column (iterations) is the average number of iterations for ICOG-C and ICOG-G to converge. The thirteenth column (monitors) is the average number of monitor places synthesized by ICOG-C and ICOG-G. The last column (time/iterations) is the average time per iteration of ICOG-C and ICOG-G.

In the experiments, we observed that the majority of time spent by ICOG is on the stage of RIDM siphon detection. This is precisely why we developed a customized MIP formulation for RIDM siphon detection in Gadara nets. Since C-MIP is customized for Gadara nets, while G-MIP is formulated for general process-resource nets, we of course expected ICOG-C to be more efficient than ICOG-G in the context of the Gadara nets. The data in Table I show that ICOG-C is in fact *significantly* faster than ICOG-G – the improvement in time ranges from 32 to 525 times faster. In addition, ICOG-C takes an average of 2.5 to 3.3 times fewer iterations than ICOG-G. ICOG-C timed out on much fewer nets; in fact, ICOG-C never timed out on a net that ICOG-G solved within time. Finally, we see from the data that ICOG-C does not remove previously synthesized monitor places throughout the iteration process (the number of iterations is one more because of the final siphon detection to determine that the net is live); we are investigating further this interesting property. We note that ICOG does not guarantee that the number of monitor places is minimal. The alternative method studied in [18] does guarantee a minimal number of monitor places, but at the price of having to generate the reachable state space, so that techniques from classification theory can be applied.

Table II presents a sample of experimental results that highlight the scalability of ICOG-C. The first (SS) and second (US) columns are the number of safe and unsafe states. (Again, ICOG does not expand these states; these numbers were generated separately.) The third column (time (s)) is the total time (in seconds) for ICOG-C to converge. The fourth column (iters) is the number of iterations until convergence, and the fifth column (monits) is the number of monitor places synthesized. We set a time-out threshold of 6000 seconds for these experiments. If ICOG-C took longer than 6000 seconds, it was forced to terminate (see the second row of the table). Table II shows that ICOG-C is very scalable even on a modest computer set up.

V. Conclusion

Motivated by the fact that the bottleneck of the ICOG Methodology for the synthesis of MPLE control policies for Gadara nets is the RIDM siphon detection step, we presented an efficient MIP-based RIDM siphon detection algorithm that is customized for Gadara nets. In the proposed C-MIP, the problem of siphon detection is mapped to the problem of finding a total-deadlock modified-marking. Based on the detected total-deadlock modified-marking, we can construct a RIDM siphon using *a priori* knowledge derived from the properties of Gadara nets. The experimental results in Section IV show that ICOG-C is *significantly* faster than ICOG-G and has scalability properties that make it practical for a large class of multithreaded concurrent programs, the application area driving these investigations. In future work, we will further compare our approach with other siphon detection algorithms widely used in the literature, and evaluate their performance in the context of ICOG.

TABLE I

EXPERIMENTAL RESULTS OF COMPARATIVE ANALYSIS BETWEEN ICOG-C AND ICOG-G

l	**s**	**a**	**r**	**P**	**T**	**SS**	**US**	**n**	**TLE**	**time** (s)		**iterations**		**monitors**		$\frac{\text{time}}{\text{iteration}}$
										μ	σ	μ	σ	μ	σ	
10	6	6	0.3	63.2	54.2	26,033	1,518	26	0.00	0.04	0.02	3.35	2.00	2.35	2.00	0.01
									0.10	*1.20*	*1.59*	*8.23*	*8.19*	*2.35*	*2.00*	*0.15*
10	6	6	0.5	68.4	59.1	38,863	2,623	39	0.05	0.59	3.14	5.74	4.40	4.74	4.40	0.10
									0.39	*28.70*	*79.35*	*18.80*	*26.53*	*5.41*	*5.66*	*1.53*
10	6	7	0.3	72.4	62.9	77,641	5,074	28	0.00	0.05	0.03	3.96	2.24	2.96	2.24	0.01
									0.15	*5.59*	*13.55*	*10.39*	*10.13*	*3.14*	*2.46*	*0.54*
10	6	7	0.5	75.5	66.1	63,827	3,539	31	0.11	0.09	0.08	6.03	3.95	5.03	3.95	0.01
									0.62	*13.56*	*27.80*	*14.84*	*12.87*	*5.23*	*4.29*	*0.91*
10	7	6	0.3	71.5	62.3	79,322	3,531	32	0.00	0.05	0.05	3.50	2.27	2.50	2.27	0.01
									0.16	*2.33*	*4.63*	*9.03*	*9.83*	*2.69*	*2.80*	*0.26*
10	7	6	0.5	77.2	67.8	102,063	5,808	33	0.07	0.08	0.11	5.33	3.88	4.33	3.88	0.02
									0.54	*7.99*	*18.36*	*13.79*	*13.43*	*4.45*	*3.93*	*0.58*
10	7	7	0.3	82.7	72.9	248,289	11,936	26	0.00	0.05	0.03	4.08	2.23	3.08	2.23	0.01
									0.30	*28.75*	*114.27*	*13.46*	*22.83*	*4.00*	*4.89*	*2.14*
10	7	7	0.5	86.4	76.8	173,024	9,330	25	0.17	0.10	0.09	5.88	3.97	4.88	3.97	0.02
									0.69	*19.82*	*37.38*	*18.88*	*19.14*	*5.48*	*4.57*	*1.05*

TABLE II

EXPERIMENTAL RESULTS OF SCALABILITY STUDY OF ICOG-C

SS	**US**	**time** (s)	**iters**	**monits**
8,696,502	71,677	0.11	6	5
5,696,776	1,165,958	6000.00	121*	120*
5,501,728	1,321,928	1.10	24	23
4,532,269	1,237,469	2.54	40	39
3,981,880	109,164	119.92	18	17
3,580,234	438,753	0.17	10	9
2,940,540	1,107,696	419.41	78	77
2,598,863	351,324	0.50	16	15
2,041,645	109,164	119.24	18	17
1,488,093	118,016	0.19	10	9
1,109,390	152,908	61.00	29	28
1,022,542	307,176	2.14	40	39
812,089	133,944	0.82	23	22
656,917	134,260	446.32	85	84
555,392	164,358	3.08	51	50

REFERENCES

[1] A. Auer, J. Dingel, and K. Rudie. Concurrency control generation for dynamic threads using discrete-event systems. In *Proc. Allerton Conference on Communication, Control and Computing*, 2009.

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

[3] E. E. Cano, C. A. Rovetto, and J.-M. Colom. An algorithm to compute the minimal siphons in S^4PR nets. In *Proc. International Workshop on Discrete Event Systems*, pages 18–23, 2010.

[4] F. Chu and X.-L. Xie. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, 13(6):793–804, December 1997.

[5] J. Ezpeleta, J. M. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(2):173–184, April 1995.

[6] A. Gamatie, H. Yu, G. Delaval, and E. Rutten. A case study on controller synthesis for data-intensive embedded system. In *Proc. International Conference on Embedded Software and Systems*, 2009.

[7] Gurobi. Gurobi Optimizer, 2010.

[8] Y.-S. Huang, M. Jeng, X. Xie, and D.-H. Chung. Siphon-based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(6):1248 –1256, November 2006.

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

[10] M. V. Iordache and P. J. Antsaklis. Petri nets and programming: A survey. In *Proc. 2009 American Control Conference*, pages 4994–4999, 2009.

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

[12] Z. Li and M. Zhou. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):38 – 51, January 2004.

[13] Z. Li and M. Zhou. Two-stage method for synthesizing liveness-enforcing supervisors for flexible manufacturing systems using Petri nets. *IEEE Transactions on Industrial Informatics*, 2(4):313–323, November 2006.

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

[15] 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. the 49th IEEE Conference and Decision and Control*, 2010.

[16] 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. *Submitted for publication*, 2011.

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

[18] A. Nazeem, S. Reveliotis, Y. Wang, and S. Lafortune. Optimal deadlock avoidance for complex resource allocation systems through classification theory. In *Proc. the 10th International Workshop on Discrete Event Systems*, 2010.

[19] L. Piroddi, R. Cordone, and I. Fumagalli. Selective siphon control for deadlock prevention in Petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 38(6):1337 – 1348, November 2008.

[20] W. Reisig. *Petri Nets: An Introduction*. Springer-Verlag, 1985.

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

[22] 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 International Conference on Robotics and Automation*, pages 271 – 277, 2005.

[23] Y. Wang. *Software Failure Avoidance Using Discrete Control Theory*. PhD thesis, University of Michigan, 2009.

[24] 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. the 48th IEEE Conference and Decision and Control*, pages 4971–4976, 2009.