Eliminating Concurrency Bugs in Multithreaded Software: A New Approach Based on Discrete-Event Control

Hongwei Liao, Student Member, IEEE, Yin Wang, Member, IEEE, Jason Stanley, Stéphane Lafortune, Fellow, IEEE, Spyros Reveliotis, Senior Member, IEEE, Terence Kelly, Senior Member, IEEE, and Scott Mahlke, Member, IEEE

Abstract—Computer hardware is moving from uniprocessor to multicore architectures. One problem arising in this evolution is that only parallel software can exploit the full performance potential of multicore architectures, and parallel software is far harder to write than conventional serial software. One important class of failures arising in parallel software is circular-wait deadlock in multithreaded programs. In our ongoing Gadara project, we use a special class of Petri nets, called Gadara nets, to systematically model multithreaded programs with lock allocation and release operations. In this paper, we propose an efficient optimal control synthesis methodology for ordinary Gadara nets that exploits the structural properties of Gadara nets via siphon analysis. Optimality in this context refers to the elimination of deadlocks in the program with minimally restrictive control logic. We formally establish a set of important properties of the proposed control synthesis methodology, and show that our algorithms never synthesize redundant control logic. We conduct experiments to evaluate the efficiency and scalability of the proposed methodology, and discuss the application of our results to real-world concurrent software.

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

I. INTRODUCTION

A FUNDAMENTAL revolution has taken place in the computer industry in the past decade. Mainstream computer CPUs formerly had only a single processor core capable of executing a single task at a time. CPU speeds have doubled roughly every 18 months according to Moore’s law. Processor core speed cannot increase indefinitely, however, because faster cores would generate excessive heat. Successive CPU generations, therefore, now provide more processor cores rather than a faster single core and can execute several tasks at once. The problem is that only parallel software can exploit the full performance potential of multicore architectures, and parallel software is far harder to write than conventional serial software. Choreographing a productive and harmonious interplay among concurrent tasks is a very difficult task because reasoning about concurrency is very challenging for human programmers. Multicore architectures are making parallel programming unavoidable but concurrency bugs are making it costly and error prone. Significant effort has been spent to eliminate several types of concurrency bugs, see [1]–[4].

In our ongoing Gadara project [5], we are interested in shared-memory multithreaded software, a very common computing paradigm in which concurrent tasks share access to a pool of computer memory. Mutual exclusion locks (or “mutexes”) prevent tasks from accessing the same memory concurrently, thus allowing tasks to update shared memory in an orderly way, because at most one task may hold a given lock at any moment. However, it is easy for situations to arise in which, e.g., task 1 has acquired lock A and needs lock B, while task 2 holds B but requires A; these tasks are deadlocked and neither can perform useful work. This type of deadlock is called a circular-mutex-wait (CMW) deadlock in the literature, where a set of threads are waiting indefinitely for one another and none of them can proceed. In this paper, we focus on CMW deadlocks, an important class of concurrency bugs. Variants of the Banker’s algorithm [6] provide a principled approach to dynamic deadlock avoidance for concurrent software. The algorithm, however, requires a central controller that can potentially impose a global serialization bottleneck on the software it governs. Deadlock “Healing” [2] addresses potential deadlocks by adding “gate locks” that prevent out-of-order lock acquisitions from causing deadlocks. At runtime, actual deadlocks are detected and remedied by adding further gate locks, gradually eliminating deadlocks from programs. Healing is more practical than the Banker’s algorithm because its runtime checks are efficient and because it does not introduce a global serialization point into the software that it controls.
In the Gadara project, we adopt a model-based approach to systematically model, analyze, and control multithreaded software for the purpose of deadlock avoidance [7]. Our results on the first two steps, namely modeling and analysis, have been reported in [8]; the third step, control, is the focus of this paper. More specifically, we employ modeling and control techniques from discrete-event systems (DES), which have discrete state spaces and event-driven dynamics. While classical control theory, which focuses on time-driven systems, has been successfully applied to computer systems [9], the application of DES to computer systems is more recent; see [10]–[17]. Concurrent software is a typical example of a DES. Petri nets, a commonly used modeling formalism in DES, are employed in our project to model multithreaded programs. Iordache and Antsaklis [18] provide a review of the application of Petri nets to computer programming. There are at least three advantages of using Petri nets in this application context: 1) Petri nets provide a compact, graphical representation of a concurrent program’s inherent dynamics, without explicitly enumerating its state space; 2) the Petri net models enable formal analysis and verification of important properties of their associated programs via efficient structural analysis; and 3) the models make possible the synthesis of provably correct and optimal control logic that can be instrumented in the original programs for deadlock avoidance at run-time. In this regard, we defined a special class of Petri nets, called Gadara nets, to systematically model multithreaded C programs with lock allocation and release operations [8].

The special features of Gadara nets enable the mapping of the desired property of programs (e.g., deadlock-freeness) to some structural properties of their corresponding Gadara net models. More specifically, we have formally established that a multithreaded program that can be exactly modeled as a Gadara net is deadlock-free (a behavioral property) if and only if a certain type of siphon (a structural property) cannot be reached in its associated Gadara net [8]. Therefore, once we have obtained a Gadara net model of the program, we can focus on detecting the aforementioned siphons in the net. If no such siphon is detected, then this verifies that the underlying program is deadlock-free; otherwise, we synthesize control logic to prevent the above siphons from becoming reachable, thereby avoiding their associated deadlocks. As we will discuss in Section II-B, when it is not possible to build an exact Gadara net model of a program due to modeling constraints, a conservatively-built Gadara net model is needed, which is certain to include all possible execution paths of the program (and possibly some infeasible paths as well). In this case, the absence of the aforementioned siphons is a sufficient condition for CMW-deadlock-freeness of the program; the rest of the discussion in this paper still applies for the conservative model.

In control synthesis, we employ a common control technique for Petri nets, called supervision based on place invariants (SBPI) [19]–[22]. The control logic synthesized by SBPI is in the form of monitor places that augment the original net. An original Gadara net model of a concurrent program is ordinary by definition, i.e., all the arcs in the net have unit weights. However, after a net is augmented by monitor places and their associated arcs, the resulting controlled Gadara net is not necessarily ordinary in general. Moreover, Petri net models, for some other applications, may also belong to the class of controlled Gadara nets and contain arcs with nonunit weights. In [23], we have developed a general methodology of optimal control synthesis for controlled Gadara nets that need not be ordinary. Technically, this proposed control methodology is also called a maximally-permissive liveness-enforcing control policy, since the synthesized control logic will provably eliminate deadlocks while otherwise minimally constraining program behavior, and the resulting controlled Gadara net is live.

The control synthesis algorithm proposed in [23] prevents a special type of siphons, termed resource-induced deadly marked (RIDM) siphons [24], from becoming reachable in the net. This algorithm possesses a very nice property that for any monitor place synthesized by the algorithm, its associated arcs always have unit weights. In other words, the algorithm will never introduce additional nonordinariness to a controlled Gadara net. As a result, if our control synthesis starts with a Gadara net model of a concurrent program, then the original net is ordinary, and the subsequent controlled nets will remain ordinary as well. This motivates us to investigate, in this paper, the customization of the general algorithm in [23], and concentrate on the ordinary case of controlled Gadara nets, where only resource-induced (RI) empty siphons need to be considered. An RI empty siphon is a special case of an RIDM siphon. Thus, all the properties of the general algorithm will be preserved in the customized algorithm.

The main contributions of this paper are as follows: 1) we propose an iterative control synthesis methodology for ordinary Gadara nets, which customizes the general methodology presented in [23] and preserves the properties of correctness and maximal permissiveness; 2) we formally establish a set of important properties of the proposed control synthesis methodology, and show that our algorithms never synthesize redundant control logic; and 3) we conduct experiments to evaluate the efficiency and scalability of the proposed methodology.

This paper is organized as follows. Section II reviews the class of Gadara nets and its relevant properties. Section III provides an overview of the proposed methodology. The customized methodology for the optimal control of ordinary Gadara nets is presented in Section IV. We investigate some important properties of the proposed methodology in Section V, and report on the results of its experimental evaluation in Section VI. We discuss the application of our results to real-world software in Section VII and conclude in Section VIII.

II. GADARA PROJECT AND GADARA PETRI NETS

The objective of the Gadara project [5], [25] is to develop a software tool that takes as input a deadlock-prone multithreaded C program and outputs a modified version of the program that is guaranteed to run deadlock-free without affecting any of the functionalities of the program. The system architecture of the Gadara project is shown in Fig. 1,
which includes four stages: 1) we extract from the program source code a standard graphical representation, called control flow graph (CFG), which captures the execution paths of the program; 2) the CFG is translated into a Petri net model of the program, formally defined as a Gadara net, based on which potential deadlocks in the program can be mapped to structural features in the net; 3) optimal control logic is synthesized for the Gadara net using the method presented in this paper; and 4) the control logic is used to instrument the source code and manage lock allocation and release at run-time to avoid deadlocks.

Our publications in computer science venues [26], [27] addressed Steps 1 and 4 in detail. Moreover, the details about Step 2, i.e., the modeling and analysis of multithreaded programs using Gadara nets, are systematically studied in [8]. The results from [8] lay the foundation for the development of the results in this paper where we focus on Step 3. In this section, we review the definitions and properties of Gadara nets. We assume that readers are familiar with standard Petri net definitions; see the appendix and [28] for necessary background.

A. Definitions of Gadara 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. The class of Gadara nets is formally defined in [8]; we review the relevant results for the sake of completeness.

Definition 1 [8]: Let $I_N = \{1, 2, \ldots, m\}$ 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) $P_S = \bigcup_{i \in I_N} P_i$, $P_i \neq \emptyset$, and $P_S \cap P_i = \emptyset$, for all $i \neq j$; b) $P_0 = \bigcup_{i \in I_N} P_{0i}$, where $P_{0i} = \{p_0\}$; and c) $P_R = \{r_1, r_2, \ldots, r_n\}, n > 0$;
2) $T = \bigcup_{i \in I_N} T_i$, $T_i \neq \emptyset$, $T_i \cap T_j = \emptyset$, for all $i \neq j$;
3) for all $i \in I_N$, the subnet $\mathcal{N}_i$ generated by $P_S \cup \{p_0\} \cup T_i$ is a strongly connected state machine. There are no direct connections between the elements of $P_S \cup \{p_0\}$ and $T_i$ for any pair $(i, j)$ with $i \neq j$;
4) $\forall p \in P_S$, if $|p \bullet | > 1$, then $\forall t \in p \bullet, t \cap P_R = \emptyset$;
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) $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$.

$\mathcal{N}_G$ is defined to be an ordinary Petri net that consists of three types of places: 1) $P_0$ is the set of idle places that model the operational “boundary” of the net processes and, from a more technical standpoint, are used to facilitate the discussion of liveness and other properties; 2) $P_S$ is the set of operation places, each of which models a basic block of the program in its critical section; and 3) $P_R$ is the set of resource places that model mutexes and are shared among the threads. Each resource, modeled by a place in $P_R$, must satisfy the semiflow requirement as specified in Condition 5, which implies that a lock acquired by a thread will always be released later. A detailed discussion about Definition 1 is presented in [8].

Example 1: The Gadara net model $\mathcal{N}_G$ of a deadlock bug in version 2.5.62 of the Linux kernel is shown in Fig. 2. This model, together with the source code involved in the deadlock, is presented in [8] (without control). The model and our analysis to be presented later in this paper capture the real deadlock bug of the underlying program that is studied in [29]. We will use it as a running example, and demonstrate the relevant control synthesis throughout this paper. □

Given a Gadara net model of a deadlock-prone program, we employ SBPI to synthesize control logic and augment the original Gadara net by the obtained monitor places. The augmented net is called a controlled Gadara net, denoted as $\mathcal{N}_G^C$; the definition of the class of $\mathcal{N}_G^C$ is given in [8]. In general, a controlled Gadara net is not necessarily ordinary, due to the addition of monitor places and their associated arcs. As a special case, if all the arcs associated with the monitor places in the net have unit weights, then the resulting controlled Gadara net is an ordinary net. Next, we define this special subclass of controlled Gadara nets, which, as discussed...
in Section 1, is the focus of the control synthesis presented in this paper.

**Definition 2** [8]: Let \( \mathcal{N}_G = (P, T, A, M_0) \) be a Gadara net. A controlled Gadara net \( \mathcal{N}^c_{G1} = (P \cup PC, T, A \cup AC, M_0) \) is an ordinary, self-loop-free Petri net such that, in addition to all conditions in Definition 1 for \( \mathcal{N}_G \), we have:

8) for each \( p_c \in PC \), there exists a unique minimal-support \( P \)-semiflow, \( Y_{p_c} \), such that \( \{p_c\} = \|Y_{p_c}\| \cap PC \), \( P_0 \cap \|Y_{p_c}\| = \emptyset \), \( P_R \cap \|Y_{p_c}\| = \emptyset \), \( PS \cap \|Y_{p_c}\| \neq \emptyset \), and \( Y_{p_c}(p_c) = 1 \);

9) for each \( p_c \in PC \), \( M^{0}_{0}(p_c) \geq \max_{p \in PS} Y_{p_c}(p) \).

From Definition 2, we can see that \( \mathcal{N}^c_{G1} \) preserves the net structure of \( \mathcal{N}_G \). A monitor place \( p_c \in PC \) can be considered as a generalized resource place, i.e., \( p_c \) must also satisfy a semiflow requirement that is specified in Condition 8, but is weaker than that in Condition 5. Due to the similarity between the original resources places and the synthesized monitor places, we will use the term “generalized resource place” to refer to any place \( p \in PR \cup PC \). By Definition 2, a monitor place can have multiple initial tokens.

As we mentioned in Section 1 and will further elaborate later, the control synthesis algorithm to be presented next guarantees that for any monitor place synthesized by this algorithm, its associated arcs always have unit weights. In the particular application of concurrent software, we always start with a Gadara net model \( \mathcal{N}_G \) of the software, which is ordinary. Thus, by applying the aforementioned control synthesis algorithm, the resulting controlled Gadara nets will remain within the class of \( \mathcal{N}^c_{G1} \). Consequently, we can restrict our attention to \( \mathcal{N}_G \) and \( \mathcal{N}^c_{G1} \) in the following development of the control synthesis algorithm.

**Remark 1:** We observe that \( \mathcal{N}_G \) is a special subclass of \( \mathcal{N}^c_{G1} \), where \( PC = \emptyset \) and \( AC = \emptyset \). Therefore, any property that we derive for \( \mathcal{N}^c_{G1} \) holds for \( \mathcal{N}_G \) as well.

According to the semantics of the program modeled by Gadara nets, branching transitions; such as those corresponding to if/else, should not be constrained by any resource place, which is stated in Condition 4 of Definition 1. Any monitor place, as a generalized resource place, is also desired to satisfy a similar condition, so that the corresponding control logic can be properly instrumented in the program. Technically, the branching transitions in Gadara nets are said to be uncontrollable, as they cannot be disabled by any generalized resource place; if a controlled Gadara net satisfies that no monitor place in the net will attempt to disable any uncontrollable transition, then the net is said to be admissible.

In the remainder of this paper, we only consider admissible \( \mathcal{N}^c_{G1} \).

**Assumption 1:** \( \mathcal{N}^c_{G1} \) is admissible.

The control synthesis algorithm to be presented guarantees the satisfaction of Assumption 1 in the considered application context.

---

1 The optimal liveness-enforcing control synthesis for \( \mathcal{N}^c_{G1} \) is treated in detail in [23].

2 The set of branching transitions is formally defined as: \( \{t \in T : (\exists p \in P_S) ((p \bullet) > 1) \wedge (t \in p\bullet) \} \).

---

**B. Properties of Gadara Nets**

The main properties of Gadara nets are formally established in [8]. We introduce some relevant definitions, and discuss the properties that will be used in the following control synthesis.

A Petri net is live if for any transition \( t \) in the net and any reachable marking \( M \), there exists another marking \( M' \) that is reachable from \( M \), such that \( t \) is enabled under \( M' \). In other words, in a live Petri net, starting from any reachable marking, any given transition can always be enabled in some future reachable marking.

Perfect static analysis of program behavior is undecidable. A conservative model is almost needed due to the control flow obtained with limited static analysis. We do our best to build an accurate model but we err on the side of being conservative when uncertain, e.g., some paths in the model may not be feasible in the program. When there is uncertainty with static analysis, we conservatively approximate the model in order to capture the CMW deadlock [26]. Based on the above discussion, and Proposition 2 and Theorem 3 established in [8], we have the following proposition that bridges a multithreaded program and its associated model.

**Proposition 1:** Given a conservatively-built Gadara net model of a multithreaded program, the program is CMW-deadlock-free if the Gadara net model is live.

**Example 2:** Referring to Fig. 2, let us consider the reachable marking \( M_{a1} \), where there is one token in \( p_{14} \), one in \( p_{22} \), and one in \( p_{03} \), while all other places are empty. At marking \( M_{a1} \), all the transitions in the net are disabled, i.e., the net is in a total deadlock.

Proposition 1 implies that the goal of deadlock-avoidance of a program can be achieved by liveness-enforcing control of its corresponding Gadara net model. Our main intention for modeling a program via a Gadara net is to enable the analysis (and control) of the behavioral properties of the program through the analysis of the structural properties of the corresponding Gadara net. Here, the behavioral property of interest is deadlock-freeness of the program (and liveness of its associated Gadara net). Yet, as we will show in the following, we only need to focus on the structure of the Gadara net to detect the program’s potential deadlocks, without exhaustively enumerating all the possible behaviors of the program. In this regard, we introduce the notion of siphon, which is a well-defined structural construct in Petri nets.

**Definition 3:** A siphon is a nonempty set of places \( S \), such that \( \bullet S \subseteq S \bullet \).

**Definition 4:** In Gadara nets, a siphon \( S \) is said to be a resource-induced (RI) siphon, if \( S \) contains at least one generalized resource place, i.e., \( S \cap (PR \cup PC) \neq \emptyset \).

We further introduce the notion of modified marking [24] to facilitate our discussion.

**Definition 5:** Given \( \mathcal{N}^c_{G1} \) and a reachable marking \( M \), the modified marking \( \bar{M} \) is defined by

\[
\bar{M}(p) = \begin{cases} 
M(p), & \text{if } p \notin P_0 \\
0, & \text{if } p \in P_0.
\end{cases}
\]

Modified markings essentially erase the tokens in idle places. The number of tokens in idle place \( p_0 \) can always be
uniquely recovered from the invariant implied by the structure of sub-net \( N_i \), i.e., \( M_1 = M_2 \) if and only if \( \overline{M}_1 = \overline{M}_2 \). We use \( R(N_{G1}, M) \) to denote the set of reachable markings of \( N_{G1} \) starting from \( M \). The set of modified markings induced by the set of reachable markings is defined by \( R(N_{G1}, M_0) = \{ \overline{M} | M \in R(N_{G1}, M_0) \} \).

The following theorem relates the liveness property of a Gadara net to its structural properties in terms of siphons. This theorem is a direct result of Theorem 1 presented in [8]. According to Remark 1, this theorem also holds for \( N_G \).

**Theorem 1:** (Liveness of Gadara nets) \( N_{G1} \) is live iff there does not exist a modified marking \( \overline{M} \in R(N_{G1}, M_0) \) and a siphon \( S \) such that \( S \) is an RI empty siphon at \( \overline{M} \).

**Example 3:** We know from Example 2 that the net \( N_G \) shown in Fig. 2 is not live. Let \( \overline{M}_{a1} \) be the modified marking that is induced by the marking \( M_{a1} \) defined in Example 2. At \( \overline{M}_{a1} \), there is one token in \( p_{14} \) and one in \( p_{22} \), while all other places are empty. Let \( S_1 \) be the set of all empty places in the net at \( \overline{M}_{a1} \). Then, \( S_1 \) is an RI empty siphon at \( \overline{M}_{a1} \).

Proposition 1 and Theorem 1 together imply that the goal of deadlock-avoidance of a program can be achieved by preventing RI empty siphons from becoming reachable in its associated Gadara net model. They serve as a foundation for the control synthesis to be carried out next.

III. OVERALL METHODOLOGY

In this section, we first present an overview of our control synthesis methodology based on RI empty siphons. Then, we review an efficient method for detecting RI empty siphons using mathematical programming, which is discussed in detail in [8].

A. Iterative Control of Gadara Nets

Our overall strategy for control synthesis is shown in Fig. 3 and described as follows. Given a multithreaded program and its associated Gadara net model \( N_G \), we first detect if there is a potential RI empty siphon that can be reached under the modified markings of \( N_G \). For the detected RI empty siphon, we synthesize control logic to prevent it from becoming reachable, and obtain a controlled Gadara net \( N_{G1} \). Then, we detect again, over the modified markings of \( N_{G1} \), if there is a new RI empty siphon; and synthesize control logic to prevent it, if any. The above process continues, until there is no new RI empty siphon being detected. According to Proposition 1 and Theorem 1, upon termination, the resulting Gadara net is live, and its corresponding program is deadlock-free.

We see that the proposed methodology is an iterative process, because: 1) there may be some RI empty siphons that have not been identified in the previous iterations and need further consideration, and 2) the synthesized monitor places are generalized resource places, so that they may introduce new potential RI empty siphons in the

3Liveness of \( N_{G1} \) is also equivalent to the absence of any empty siphon in the original reachable markings of the net. But we have opted to use the result of Theorem 1 in order to stay close to the developments of the results in [23].

controlled net. We refer to the above process as the iterative control of (controlled) Gadara nets: ordinary case (ICOG-O) methodology. The general ICOG methodology that works for both ordinary and nonordinary cases is presented in [23]. While ICOG in [23] is based on exploiting RIDM siphons [8], [24] (which can be considered as a generalization of the notion of RI empty siphons for nonordinary nets), ICOG-O presented in this paper customizes ICOG and only considers RI empty siphons, resulting in lower analytical complexity and some interesting properties. In particular, bookkeeping of prevented states, which is required in ICOG in [23], is no longer necessary in ICOG-O.

The main features of the proposed ICOG-O methodology to be presented are summarized as follows: 1) ICOG-O is based on structural analysis (using RI empty siphons), and does not require the construction of the reachability space of the net; 2) ICOG-O is correct and maximally permissive with respect to the goal of liveness enforcement; and 3) ICOG-O is guaranteed to terminate in a finite number of iterations.

There are two major tasks in ICOG-O: detecting RI empty siphons and rendering them unreachable. For the first task, the potential RI empty siphon is detected in each iteration by a mixed integer programming (MIP) formulation we proposed in [8]. We will briefly review this formulation in Section III-B. For the second task, the detected RI empty siphon is prevented by the unsafe-covering-based control of RIDM siphons: ordinary case (UCCOR-O) algorithm, which will be presented in Section IV.

B. Detection of RI Empty Siphons [8]

In [8], we have developed a customized and efficient MIP formulation for the detection of RI empty siphons in \( N_G \) and \( N_{G1} \). The formulation exploits the following important property of Gadara nets. If a Gadara net is not live, then the net will always reach a total-deadlock modified-marking \( \overline{M} \) (with \( M \) being different from the initial marking), i.e., a modified marking \( \overline{M} \) where all the transitions in the net are disabled. Moreover, if we let \( S \) be the set of all empty places at \( \overline{M} \), then \( S \) is an RI empty siphon [8]. Using this property, the problem of detecting an RI empty siphon in \( N_G \)

![Fig. 3. Iterative control of (controlled) Gadara nets: ordinary case (ICOG-O).](image-url)

- Run the UCCOR-O Algorithm
- Add monitor places to the net

\[ N_G \rightarrow \text{RI Empty Siphon Detection by MIP-N_G} \]

\[ \text{Is a new RI empty siphon detected?} \]

\[ \text{YES} \rightarrow \text{new RI empty siphon} \]

\[ \text{NO} \rightarrow \text{Terminate} \]
In particular, (10) specifies that of the indicator variables associated with any monitor place. The latter one can be solved by the MIP formulation (2)–(10), denoted as MIP-$N_{G1}^c$, which is briefly reviewed as follows.

In the formulation MIP-$N_{G1}^c$, $\overline{M}(p)$ is a binary indicator variable associated with any place $p \in P$, such that if $p$ is not an empty place at $\overline{M}$, then $\overline{M}(p) = 1$; otherwise, $\overline{M}(p) = 0$. In fact, for any $p \in P_0 \cup P_S \cup P_R$, $\overline{M}(p)$ represents both its associated binary indicator variable and its modified marking; however, for any $p \in P_C$, $\overline{M}(p)$ only represents its associated binary indicator variable, but not necessarily its modified marking (a slight abuse of notation)

\begin{equation}
\text{MIP--}N_{G1}^c: \min \sum_{p \in P_S} \overline{M}(p) \tag{2}
\end{equation}

s.t. $M = M_0 + \sigma$

\begin{equation}
\overline{M}(p) = M(p) \quad \forall p \in P_S \cup P_R \tag{3}
\end{equation}

\begin{equation}
\overline{M}(p) = 0 \quad \forall p \in P_0 \tag{4}
\end{equation}

\begin{equation}
\overline{M}(p) = 0 \quad \forall p \in Q \tag{5}
\end{equation}

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

\begin{equation}
\sum_{p \in \bullet t} \overline{M}(p) - | \bullet t | + 1 \leq 0 \tag{6}
\end{equation}

\begin{equation}
\sum_{p \in P_S} \overline{M}(p) \geq 2 \tag{7}
\end{equation}

\begin{equation}
M \geq 0; \quad \sigma \in \mathbb{Z}_0^+ \tag{8}
\end{equation}

\begin{equation}
M(p) \geq \overline{M}(p) \geq \frac{M(p)}{SB(p)} \quad \forall p \in P_C \tag{9}
\end{equation}

\begin{equation}
\overline{M}(p) \in \{0,1\} \quad \forall p \in P_C. \tag{10}
\end{equation}

The objective function (2) seeks to minimize the number of marked operation places in the detected total-deadlock modified-marking. The selection of such an objective function will produce siphons that are efficient for control synthesis using ICOG-O. Some resulting interesting properties will be presented in Section V.

We briefly explain (3)–(10); see [8] for a detailed discussion. Constraint (3) is the state equation of the net. Constraint (4) connects an original marking with its associated modified marking based on Definition 5. Constraints (4)–(6) enforce that all the transitions in the net are disabled at $\overline{M}$. Constraint (7) follows from the fact that at least two threads must be involved in a CMW deadlock. Constraint (8) specifies the sign restrictions for the variables $M$ and $\sigma$. Constraints (9) and (10) are specifically formulated for $N_{G1}^c$, i.e., the siphon detection in $N_G$ will not involve these two constraints since $P_C$ is empty in that case. These two constraints set the values of the indicator variables associated with any monitor place. In particular, (10) specifies that $\overline{M}(p)$ is used as an indicator variable in the context of this formulation (and not as the modified marking of the corresponding monitor place $p$). On the other hand, the parameter SB($p$) that appears in Constraint (9) denotes a structural bound for the marking of place $p$. In Gadara nets, we can set: SB($p$) = $M_0^p$, $\forall p \in P_0 \cup P_C$, and SB($p$) = 1, $\forall p \in P_S \cup P_R$.

IV. OPTIMAL CONTROL ALGORITHM BASED ON RI EMPTY SIPHONS

Once an RI empty siphon is detected, we input it to the control synthesis algorithm, called unsafe-covering-based control of RIDM siphons: ordinary case (UCCOR-O), which customizes the general algorithm UCCOR presented in [23]. In UCCOR-O, we focus on a special type of RIDM siphons in ordinary nets, namely RI empty siphons. UCCOR-O synthesizes control logic based on the notion of unsafe covering, which is introduced next.

Similar to the modified-marking defined in Definition 5, we further define the notion of $P_S$-marking to facilitate the discussion.

Definition 6: Given $N_{G1}^c$ and $M \in \mathcal{R}(N_{G1}^c, M_0^c)$, the $P_S$-marking $\overline{M}$ is defined by

\begin{equation}
\overline{M}(p) = \begin{cases} M(p), & \text{if } p \in P_S \\ 0, & \text{if } p \notin P_S. \end{cases} \tag{11}
\end{equation}

$P_S$-markings essentially erase the tokens in idle places and generalized resource places, retaining only tokens in operation places. Given the $P_S$-marking $\overline{M}$ corresponding to the original marking $M$, the number of tokens in places $P_0$ and $P_C$ under $M$ can be uniquely recovered from their associated semiflows; the number of tokens in places $P_0$ can also be uniquely recovered similar to the case of modified marking. In other words, $P_S$-markings do not introduce any ambiguity, i.e., there is a one-to-one mapping between the original marking and the $P_S$-marking, such that $M_1 = M_2$ if and only if $\overline{M_1} = \overline{M_2}$. Therefore, we can restrict our attention to $P_S$-markings in the following discussion, which greatly facilitates the control synthesis. Furthermore, from Condition 6 of Definition 1, we know that a $P_S$-marking is always a binary vector, i.e., any component of a $P_S$-marking is either 0 or 1.

In view of the above discussion, when focusing on $P_S$-markings, we don’t care about the number of tokens in $P_0 \cup P_R \cup P_C$. We introduce the notation $\chi$ for the value of a $P_S$-marking component, where $\chi$ stands for “0 or 1.” The notion of covering is introduced below.

Definition 7: In Gadara nets, a covering $C$ is a generalized $P_S$-marking, whose components can be 0, 1, or $\chi$.

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

Given two coverings $C_1$ and $C_2$, we say that $C_1$ covers $C_2$, if $\forall p \in P_S$ such that $C_1(p) \neq C_2(p), C_1(p) = \chi$. The “cover” relationship between a covering and a $P_S$-marking is defined in a similar way. Note that as ICOG-O evolves, new monitor places will be added to the net throughout the iterations. In the rest of this paper, when comparing two coverings (or, a covering and a $P_S$-marking) with different dimensions, and the difference is due to the synthesized monitor places, we assume the one with a lower dimension is padded by $\chi$’s for those monitor places.
Definition 8: In $N_{G1}^c$, a marking $M$ is said to be an RIE-unsafe marking, if at its associated modified marking $\overline{M}$, there exists at least one RI empty siphon.

Definition 9: A covering $C$ is said to be an RIE-unsafe covering, if for all $P_S$-markings $\overline{M}$ it covers, the corresponding $M$ is an RIE-unsafe marking.

### A. UCCOR-O Algorithm: Overview

We are now ready to present the UCCOR-O algorithm. We organize our presentation in a top-down manner. We first overview the procedure of UCCOR-O as illustrated in Fig. 4, and then explain the three steps of UCCOR-O in subsequent sections. We will apply UCCOR-O to the running example throughout our discussion.

The input to UCCOR-O is $N_{G1}^c$, an RI empty siphon $S$, and the associated total-deadlock modified marking $\overline{M}$ obtained from MIP-$N_{G1}^c$. The output of UCCOR-O is a monitor place that prevents the RI empty siphon $S$ from becoming reachable. The UCCOR-O algorithm contains three steps. In Step 1, an RIE-unsafe covering is generated based on the input to the algorithm. This covering captures the RI empty siphon we want to prevent. In Step 2, the obtained RIE-unsafe covering is generalized into a new covering, by exploiting a monotonicity property of Gadara nets. This generalization step enhances the efficiency of the algorithm, in terms of the number of undesirable markings that can be prevented by the final monitor place. In Step 3, a monitor place is synthesized to prevent the covering obtained in Step 2. Step 3 contains two stages in general. Stage 2 is necessary only when the controlled Gadara net obtained in Stage 1 is not admissible. We discuss these three steps in further detail below.

### B. Unsafe Covering Generation

Step 1 of UCCOR-O generates an RIE-unsafe covering, denoted as $C_{u1}$, based on the input to the algorithm. We consider the following set of places:

\[
\Lambda_S = \bigcup_{p \in S^\prime (P_R \cup P_C)} \|Y_p\| \cup S.
\]  

Intuitively, $\Lambda_S$ contains the set of all places that are relevant to the siphon $S$. In particular, $\Lambda_S$ complements $S$ with all those operation places that utilize the generalized resources appearing in $S$.

Therefore, we can specify the values for the components of $C_{u1}$ that are associated with $\Lambda_S$ as: $C_{u1}(\Lambda_S) = \overline{M}(\Lambda_S)$; and set $C_{u1}(p) = \chi$, $\forall p \notin \Lambda_S$, since these places are irrelevant to the considered siphon. Moreover, we know from the definition of covering that we can further set $C_{u1}(p) = \chi$, $\forall p \in P_0 \cup P_R \cup P_C$. The resulting $C_{u1}$ is input to Step 2 of UCCOR-O.

**Example 4:** We continue our discussion on the example in Fig. 2. Let $N_G$, $\overline{M}_{u1}$, and $S_1$, described in Example 3, be the input to UCCOR-O. After Step 1 of UCCOR-O, the RIE-unsafe covering $C_{u1}$ is specified as follows. $C_{u1}(P_{14}) = C_{u1}(P_{22}) = 1$; $C_{u1}(p) = 0$, $\forall p \in P_S \setminus \{P_{14}, P_{22}\}$; and $C_{u1}(p_{01}) = C_{u1}(p_{02}) = C_{u1}(r_A) = C_{u1}(r_B) = C_{u1}(r_C) = \chi$.

### C. Unsafe Covering Generalization

Step 2 of UCCOR-O generalizes the RIE-unsafe covering obtained from Step 1, by exploiting a monotonicity property of Gadara nets, which is formally proved in [23]. The monotonicity property is explained as follows. Let $M$ and $M'$ be two markings of a Gadara net, which satisfy: $M(p) \geq M'(p)$, for all $p \in P_S$, and $M(p) > M'(p)$, for at least some $p \in P_S$. If $M'$ is a marking that needs to be prevented, then $M$ also needs to be prevented. The intuition is that loading a program, which is already in a deadlock or will unavoidably enter a deadlock, with even more active threads will only worsen the deadlock situation, but not cure it.

Based on the above property, for the RIE-unsafe covering $C_{u1}$ obtained in Step 1, if we replace any of its 0 components (associated with operation places) by 1, the resulting covering will only cover reachable $P_S$-markings that need to be prevented, or nonreachable $P_S$-markings. Therefore, $C_{u1}$ can be generalized by replacing all of its 0 components by $\chi$, and the resulting covering is denoted as $C_{u2}$, which is input to Step 3 of UCCOR-O.

By construction, the generalized covering $C_{u2}$ will not “miss” covering any $P_S$-markings that are covered by $C_{u1}$. In general, $C_{u2}$ will cover a larger set of $P_S$-markings than $C_{u1}$, because the former contains more $\chi$ components. So instead of preventing $C_{u2}$, a monitor place that prevents $C_{u2}$ is more efficient, in the sense that it will prevent a larger set of markings in the controlled net. More importantly, the property of maximal permissiveness is still preserved, i.e., we only prevent reachable markings that need to be prevented, or markings that are not reachable, due to the above discussion.
Example 5: Given the RIE-unsafe covering $C_{u_1}$ described in Example 4, Step 2 of UCCOR-O generalizes $C_{u_1}$ and obtains $C_{u_2}$, which is specified as follows. $C_{u_2}(p_{14}) = C_{u_2}(p_{22}) = 1$ and $C_{u_2}(p) = \chi$, $\forall p \in P \setminus \{p_{14}, p_{22}\}$. \hfill $\square$

D. Monitor Place Synthesis Algorithm

Step 3 of UCCOR-O aims to find an appropriate linear inequality constraint in the form

$$M \leq b$$

so that SBPI can be employed to synthesize a monitor place, to prevent $C_{u_2}$ that is obtained in Step 2; the constraint should also guarantee that the resulting controlled Gadara net is admissible. Generally, Step 3 consists of two stages. Stage 2 is necessary only when the controlled Gadara net obtained in Stage 1 is not admissible. For the sake of simplicity and without any confusion, we let $C_u \equiv C_{u_2}$ and will use the notation $C_u$ in the following discussion. (Step 3 of UCCOR-O in this paper is similar to the corresponding step of UCCOR that is presented in [23], for which no customization is necessary; we include it here for the sake of completeness. Also note that in the general UCCOR algorithm, there is a step called “Inter-Iteration Coverability Check,” which is eliminated in the customized UCCOR-O algorithm. The reason of this customization will become clear when we present Theorem 3 in Section V.)

In Stage 1, we specify a linear inequality constraint in the form of (13) for $C_u$. From the first two steps of UCCOR-O, we know that $C_u$ contains only “1” or “$\chi$” components. The parameters of the constraint associated with $C_u$ are

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

(14)

$$b_{C_u} = \left( \sum_{p: p \in \chi \text{ and } C_u(p) = 1} C_u(p) \right) - 1.$$  

(15)

According to [23, Th. 3], this constraint only prevents $C_u$ (i.e., any $P_S$-marking or covering that is covered by $C_u$). Thus, the corresponding control logic synthesized based on this constraint is maximally permissive. The synthesis of a monitor place based on this constraint can be achieved by SBPI. If the resulting $N_{G}^{R}$ is admissible, then Stage 2 is not necessary and we can continue with the next iteration of ICOG-O; otherwise, we need to proceed to Stage 2, where constraint transformation is carried out to deal with the partial controllability and ensure the admissibility of $N_{G}^{R}$.

Example 6: We illustrate Stage 1 by continuing our discussion on the running example. Given the covering described in Example 5, we specify the following linear inequality constraint according to (14) and (15):

$$M(p_{14}) + M(p_{22}) \leq 1.$$  

(16)

The monitor place $p_c$, which enforces (16), is synthesized by SBPI and shown in Fig. 5. We see that $p_c$ has two outgoing arcs, both of which connect to branching transitions. In this running example, we define that only the lock acquisition transitions are controllable; and all the other transitions (i.e., those corresponding to branching and lock releases) are uncontrollable. Thus, the controlled net that contains $p_c$ is not admissible. We resolve this problem in Stage 2 of Step 3. \hfill $\square$

In Stage 2, the original constraint specified by (14) and (15) is transformed, so that the new constraint, when applied to SBPI, will render a monitor place that leads to an admissible controlled net. For the sake of discussion, the constraint obtained in Stage 1 can be rewritten as

$$M(p_1) + M(p_2) + \cdots + M(p_n) \leq n - 1.$$  

(17)

The key idea of the proposed constraint transformation is the following. If place $p_i$ in (17) can gain tokens through a sequence of uncontrollable transitions, places along the sequence of uncontrollable transitions must be included to the left-hand side of (17) as we cannot prevent these transitions from firing and populating tokens into $p_i$. We make two remarks for the above statement: 1) the set of places corresponding to a given sequence of uncontrollable transitions is unique due to the state-machine structure of the process subnet and 2) the uncontrollable transitions in this sequence are not blocked by any generalized resource place, otherwise they would be controllable. The pseudo-code that implements the constraint transformation for (17) is given in Algorithm 1. Based on the set of places $C$ obtained above, the new transformed constraint is

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

(18)

The important properties of the proposed constraint transformation technique are summarized as follows; see [23] for a formal treatment: 1) the constraint transformation technique guarantees that the resulting controlled Gadara net is admissible; 2) any marking prevented by the original constraint is also prevented by the new constraint; and 3) any reachable place in the controlled Gadara net is controllable.
Algorithm 1 Constraint Transformation Technique Used in Stage 2 of the Monitor Place Synthesis Algorithm

**Input:** A linear inequality constraint, e.g., (17)

**Output:** A set of places $C$

**Method:**
1) add $p_1, \ldots, p_n$ in (17) to stack $S$, and to set $C$
2) while $S$ is not empty
3) $p = S$.pop()
4) for each uncontrollable $t$ in $\bullet p$, if $\bullet t$ is not in $C$
   add $t$ to $S$ and $C$
5) end while

marking that is prevented by the new constraint but not by the original constraint, can reach a marking prevented by the original constraint via a sequence of uncontrollable transitions.

**Example 7:** We apply the proposed constraint transformation technique to (16), which is obtained from Stage 1 in Example 6. After Stage 2, the new transformed constraint is

\[
[M(p_{14}) + M(p_{13}) + M(p_{15})] + [M(p_{22}) + M(p_{21}) + M(p_{23})] \leq 1. \tag{19}
\]

The monitor place $p_{14}$, which enforces (19), is synthesized by SBPI and shown in Fig. 5. We see that $p_{14}$ has two out-going arcs, both of which connect to uncontrollable transitions. Thus, the controlled net that contains $p_{14}$ is admissible. We denote the resulting controlled Gadara net as $N_{G1}^{c(1)}$, which consists of $N_G$ and $p_{14}$. \hfill $\Box$

Example 7 completes the first iteration of ICOG-O on the running example. We continue our discussion on the second iteration in Example 8.

**Example 8:** In the second iteration of ICOG-O, we first input the net $N_{G1}^{c(1)}$ obtained from Example 7 into MIP-$N_{G1}^{c}$ for the detection of RI empty siphons. MIP-$N_{G1}^{c}$ finds a total-deadlock modified-marking $\overline{M}_{a2}$, where there is one token in $p_{12}$ and one in $p_{22}$, while all other places are empty. Note that this corresponds to a circular-wait deadlock induced by $r_{a}$ and $p_{14}$. Let $S_2$ be the set of all empty places in the net at $\overline{M}_{a2}$. Then, $S_2$ is an RI empty siphon at $\overline{M}_{a2}$.

We input $N_{G1}^{c(1)}$, $S_2$, and $\overline{M}_{a2}$ to UCCOR-O. Step 1 of UCCOR-O generates the covering $Cu_1$, which is specified as: $Cu_1(p_{12}) = Cu_1(p_{22}) = 1$; $Cu_1(p) = 0$, $\forall p \in P_s \setminus \{p_{12}, p_{22}\}$; and $Cu_1(p) = \chi$, $\forall p \in P_0 \cup P_R \cup P_C$. Step 2 of UCCOR-O further generalizes $Cu_1$ and obtains the covering $Cu_2$, which is specified as: $Cu_2(p_{12}) = Cu_2(p_{22}) = 1$ and $Cu_2(p) = \chi$, $\forall p \in P \setminus \{p_{12}, p_{22}\}$. Based on $Cu_2$, Stage 1 of Step 3 of UCCOR-O constructs the following constraint:

\[
M(p_{12}) + M(p_{22}) \leq 1. \tag{20}
\]

Similar to the situation encountered in Example 6, the monitor place that is synthesized by SBPI and enforces (20), will attempt to disable uncontrollable transitions. Thus, the resulting controlled net would not be admissible, which necessitates Stage 2 of Step 3.

In Stage 2, the original constraint in (20) is transformed into

\[
[M(p_{12}) + M(p_{11})] + [M(p_{22}) + M(p_{21}) + M(p_{23})] \leq 1. \tag{21}
\]

The monitor place $p_{12}$, which enforces (21), is synthesized by SBPI and shown in Fig. 5. We denote the resulting controlled net as $N_{G1}^{c(2)}$, which consists of $N_G$, $p_{14}$, and $p_{22}$. The controlled Gadara net $N_{G1}^{c(2)}$ is admissible.

In the third iteration of ICOG-O, we input $N_{G1}^{c(2)}$ into MIP-$N_{G1}^{c}$, and no solution is found. Therefore, no new RI empty siphon can be detected in $N_{G1}^{c(2)}$, and ICOG-O terminates. \hfill $\Box$

V. PROPERTIES OF THE PROPOSED CONTROL SYNTHESIS METHODOLOGY

The general ICOG methodology and UCCOR algorithm proposed in [23] are shown to be both correct and maximally permissive, with respect to the goal of liveness enforcement of Gadara nets via siphon-based control. Moreover, ICOG is guaranteed to terminate in a finite number of iterations. The general ICOG methodology is developed independent of the method used to detect siphons. Thus, ICOG-O and UCCOR-O presented in this paper, which are customized versions of ICOG and UCCOR, respectively, still preserve the aforementioned properties. Moreover, the customization possesses some new properties that are formally established below.

A. Properties of the UCCOR-O Algorithm

**Theorem 2:** In $N_{G1}^{c}$, for any monitor place $p_c \in P_C$ synthesized by UCCOR-O, any process subnet will never have two consecutive resource acquisitions from $p_c$ without a resource release to $p_c$ in between. Also, any process subnet will never have two consecutive resource releases to $p_c$ without a resource acquisition from $p_c$ in between. Moreover, all the arcs associated with $p_c$ have unit arc weights.\footnote{This theorem also applies to UCCOR developed for the control synthesis for $N_{G}^{c}$, as presented in [23]. However, this result was not presented in [23].}

**Proof:** Since any covering $Cu$ considered in UCCOR-O is a generalized $P_R$-marking, the linear constraint generated in Step 3 of UCCOR-O will involve only operation places. That is, for any $p$ such that $l_{Cu}(p) = 1$, $p$ must be an operation place; further, $l_{Cu}(p) = 0$, $\forall p \in P_0 \cup P_R \cup P_C$. Given $Cu$, let $p_c$ be the corresponding monitor place synthesized by UCCOR-O using SBPI. Also, let $Q$ be the set of places that are involved in the linear constraint, i.e., $Q = \{p \in P_S : l_{Cu}(p) = 1\}$.

Consider an arbitrary transition $t \in T$. We discuss the connectivity of the monitor place $p_c$ to $t$ in four cases, as shown in Fig. 6, where the places that belong to $Q$ are highlighted. Due to the aforementioned property of $l_{Cu}$, we can focus on process subnets (since the generalized resource places will not affect the connectivity of $p_c$ to $t$ in terms of $l_{Cu}$). Recall Condition 3 of Definition 1 that, in the process subnet, $t$ has only one input place, denoted as $p_{11}$, and one output place, denoted as $p_{12}$.

**Case 1:** $p_{11} \in Q$ and $p_{12} \notin Q$, as shown in Fig. 6(a). In this case, we have: $l_{Cu}(p_{11}) = 1$ and $l_{Cu}(p_{12}) = 0$. The state
machine structure of the process subnets leads to the following feature of the incidence matrix \( D \) of \( N_{G1} \): if we only consider the rows associated with the places in all the process subnets, then in the column corresponding to \( t \), there are only two nonzero entries, i.e., \( D_{p11,t} = -1 \) and \( D_{p12,t} = 1 \). Therefore, the algebraic calculation of SBPI [22] will result in one arc connecting \( t \) to \( p_c \), whose weight equals to 1.

Case 2: \( p_{11} \notin Q \) and \( p_{12} \in Q \), as shown in Fig. 6(b). In this case, we have: \( l_{c_u}(p_{11}) = 0 \) and \( l_{c_u}(p_{12}) = 1 \). Similar to the analysis in Case 1, the calculation results in one arc connecting \( p_c \) to \( t \), whose weight equals to 1.

Case 3: \( p_{11} \in Q \) and \( p_{12} \notin Q \), as shown in Fig. 6(c). In this case, we have: \( l_{c_u}(p_{11}) = 1 \) and \( l_{c_u}(p_{12}) = 0 \). According to the calculation of SBPI, no arc will be synthesized between \( p_c \) and \( t \).

Case 4: \( p_{11} \notin Q \) and \( p_{12} \notin Q \), as shown in Fig. 6(d). In this case, we have: \( l_{c_u}(p_{11}) = l_{c_u}(p_{12}) = 0 \). Similar to Case 3, no arc will be synthesized between \( p_c \) and \( t \).

Note that Cases 1 and 4 also apply to the situation when \( t \) is a terminating transition of the process subnet and \( p_{12} \) is an idle place. Thus, the above four cases cover all the possibilities of the connectivity of \( p_c \) to an arbitrary transition \( t \).

As a result, if we traverse from the upstream to the downstream of a process subnet, it is impossible for the subnet to have two consecutive resource acquisitions from (or resource releases to) \( p_c \).

As a consequence of Theorem 2, we have the following corollary, which can be considered as a special case of Condition 8 of Definition 2 when UCCOR-O is employed to synthesize monitor places.

**Corollary 1:** In Gadara nets, for each \( p_c \in P_C \) synthesized by the UCCOR-O algorithm, there exists a unique minimal-support P-semiflow, \( Y_{p_c} \), such that \( \{p_c\} = \|Y_{p_c}\| \cap P_C \), \((\forall p \in \|Y_{p_c}\|)(Y_{p_c}(p) = 1)\), \( P_0 \cap \|Y_{p_c}\| = \emptyset \), \( P_R \cap \|Y_{p_c}\| = \emptyset \), and \( P_S \cap \|Y_{p_c}\| \neq \emptyset \).

**B. Properties of the ICOG-O Methodology**

Define \( C^{(i)}_u \) to be the covering input to Step 3 of UCCOR-O in the \( i \)th iteration of ICOG-O; and define

\[
K^{(i)} = \sum_{p:p \in \Lambda_S \text{ and } C^{(i)}_u(p) = 1} C^{(i)}_u(p)
\]

namely, \( K^{(i)} \) is the total number of 1s in \( C^{(i)}_u \) that is induced by the siphon \( S \) under consideration.

**Lemma 1:** In ICOG-O, \( K^{(i)} \) is nondecreasing with respect to \( i \). That is, the total number of 1s in the covering considered in Step 3 of UCCOR-O is nondecreasing, throughout the iterations of ICOG-O.

**Proof:** Consider an arbitrary \( i \geq 1 \), and let \( p_c \) be the monitor place synthesized in the \( i \)th iteration of ICOG-O that prevents \( C^{(0)}_u \). According to Step 3 of UCCOR-O, the initial marking of \( p_c \) is \( M_0(p_c) = K^{(i)} - 1 \).

We mentioned above that a monitor place is essentially a generalized resource place and may introduce new potential deadlocks in the controlled net. More specifically, the monitor place \( p_c \) can directly induce a new circular-wait deadlock, if in the controlled net: 1) there exists a total-deadlock modifying-marking \( M \) (\( M \neq M_0 \)), such that \( p_c \) is empty at \( M \), and 2) \( p_c \) blocks at least one thread that is involved in a circular-wait deadlock at \( M \), i.e., the thread is waiting for the resource from \( p_c \) while holding some other resources involved in the deadlock.

Let \( C^{(i+1)}_u \) be the covering that corresponds to the optimal solution of MIP-\( N_{G1}^c \) in the \((i + 1)\)-st iteration of ICOG-O. We consider the following two cases.

**Case 1:** \( p_c \) does not directly induce the deadlock involved in \( C^{(i+1)}_u \), i.e., \( p_c \) is not part of the deadlock. In this case, the optimal solution of MIP-\( N_{G1}^c \) in the \((i + 1)\)-st iteration of ICOG-O must also be a feasible solution in the \( i \)th iteration, because, by the assumption of Case 1, this optimal solution is not a new feasible solution induced by \( p_c \). Therefore, MIP-\( N_{G1}^c \) guarantees that the number of 1s contained in \( C^{(i+1)}_u \) will be greater than or equal to that in \( C^{(i)}_u \); otherwise, \( C^{(i+1)}_u \) would have been exploited in earlier iterations.

![Fig. 6. Cases considered in the proof. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4.](image-url)
Case 2: $p_c$ directly induces the deadlock involved in $C_u^{(l+1)}$, i.e., $p_c$ is part of the deadlock. In this case, we show that at least $K^{(l)}$ operation places must be marked at $C_u^{(l+1)}$. Since $p_c$ directly induces the deadlock, $p_c$ is empty at $C_u^{(l+1)}$ [Condition 1] mentioned above]. Thus, according to Theorem 2, there must be $M_0(p_c) = K^{(l)} - 1$ different operation places in $|Y_{p_c}|$ that are marked at $C_u^{(l+1)}$ in order to empty $p_c$. Moreover, we know that $p_c$ blocks at least one thread that is involved in the deadlock at $C_u^{(l+1)}$ [Condition 2] mentioned above]. Then, there exists an output transition $t$ of $p_c$, such that the (unique) input operation place of $t$ (denoted as $q_1$) is marked at $C_u^{(l+1)}$, which corresponds to a thread blocked by $p_c$. We argue that $q_1 \notin Y_{p_c}$. If $q_1 \in Y_{p_c}$, then the (unique) output operation place of $t$ (denoted as $q_2$), which belongs to $|Y_{p_c}|$ by definition, must satisfy $Y_{p_c}(q_2) > 1$. This contradicts Corollary 1. Thus, the marked operation place $q_1$ is different from the aforementioned $K^{(l)} - 1$ marked operation places in $|Y_{p_c}|$. As a result, at least $K^{(l)}$ operation places are marked at $C_u^{(l+1)}$, and the number of 1s contained in $C_u^{(l+1)}$, $K^{(l+1)}$, is at least $K^{(l)}$.

We conclude this section with an important property of ICOG-O.

**Theorem 3:** ICOG-O will not synthesize redundant monitor places. That is, there does not exist a pair of monitor places $p_{ci}$ and $p_{cj}$ synthesized by ICOG-O, such that the covering prevented by $p_{ci}$ covers the covering prevented by $p_{cj}$.

**Proof:** For the sake of discussion, let $p_{ci}$ and $p_{cj}$ be the monitor places synthesized in the $i$th and $j$th iterations of ICOG-O, respectively. Correspondingly, let $C_u^{(i)}$ and $C_u^{(j)}$ be the coverings considered in Step 3 of UCCOR-O in the $i$th and $j$th iterations of ICOG-O, respectively. That is, $p_{ci}$ is synthesized to prevent $C_u^{(i)}$ and $p_{cj}$ is synthesized to prevent $C_u^{(j)}$.

If $i > j$, then according to Lemma 1 and the fact that $C_u^{(i)} \neq C_u^{(j)}$, we know that $C_u^{(i)}$ cannot cover $C_u^{(j)}$.

If $i < j$, we want to show that $C_u^{(i)}$ cannot cover $C_u^{(j)}$ either. In the $i$th iteration of ICOG-O, $p_{ci}$ is synthesized to prevent $C_u^{(j)}$; hence, any marking covered by $C_u^{(j)}$ will not be reachable in the net considered in the $j$th iteration of ICOG-O. As a result, in the $j$th iteration, any marking covered by $C_u^{(i)}$ will not be a feasible solution to the state equation of the net, and hence will not be a feasible solution to MIP-$N_{G1}^c$. In other words, in the $j$th iteration, the solution of MIP-$N_{G1}^c$ and the corresponding $C_u^{(j)}$ cannot be covered by $C_u^{(i)}$. 

### VI. Experimental Evaluation

We discussed above that the development of the control synthesis methodology and the validity of the associated properties are independent of the method used to detect RI empty siphons. However, we observe that the RI empty siphon detection algorithm plays an important role in the efficiency of control synthesis; it is in fact the computational bottleneck of ICOG-O. This motivated us to develop the customized formulation, MIP-$N_{G1}^c$, for efficient siphon detection in Gadara nets, which we reviewed in Section III-B. While MIP-$N_{G1}^c$ is specifically designed for Gadara nets, siphon detection algorithms for more general classes of Petri nets have been extensively studied in the literature. A generic MIP formulation is presented in [30] for the detection of maximal empty siphons in ordinary, structurally bounded Petri nets, and it is one of the most widely used empty siphon detection algorithms in the literature; we refer to this formulation as MIP-ES hereafter. Our customized algorithm, MIP-$N_{G1}^c$, is inspired by MIP-ES, and further incorporates the special properties of Gadara nets.

#### A. Objective and Setup of the Experiments

In this section, we investigate the performance of two versions of ICOG-O: 1) the original ICOG-O that uses MIP-$N_{G1}^c$, for siphon detection and 2) a modified version of ICOG-O, denoted as ICOG-O-ES, that uses MIP-ES for siphon detection. Since MIP-$N_{G1}^c$ is customized for Gadara nets, while MIP-ES is formulated for general, ordinary bounded Petri nets, we, of course, expect ICOG-O to be more efficient than ICOG-O-ES in the context of the Gadara nets. Thus, in the following experiments, we use ICOG-O-ES as the baseline for assessing and concretizing this attained efficiency by ICOG-O. We also report a sample of experimental results that demonstrate the scalability of ICOG-O.

Our experiments were completed on a Mac OS X laptop with a 2.4-GHz Intel Core2Duo processor and 2 GB of RAM. Both ICOG-O and ICOG-O-ES are implemented in C++ and compiled under the GNU gcc compiler. The MIP formulations are solved using Gurobi 3.0.1 [31]. 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, according to the input parameters. Additional logic was applied to ensure valid behavior. The random Gadara net generator (available at http://gadara.eecs.umich.edu/software.html) is based on our experience modeling real concurrent programs [32]. The input parameters of the generator are further explained in Section VI-B and Table I.

In our experiments, for each set of parameters (each row in Table I), 150 samples of random Gadara nets are generated. The generated nets with no unsafe states\(^5\) are removed from the samples. We set a time-out threshold of 10 sec for the stage of RI empty siphon detection in ICOG-O and ICOG-O-ES. A net times out if it cannot be solved by either MIP-$N_{G1}^c$ or MIP-ES in less than 10 seconds. Unless otherwise specified, all statistical results reported below are calculated over the sample nets where both ICOG-O and ICOG-O-ES did not time out.

#### B. Comparative Analysis of ICOG-O and ICOG-O-ES

Fig. 7 shows the time to converge (TTC) of ICOG-O and ICOG-O-ES. Fig. 7(a) shows the normalized cumulative frequency (NCF, a.k.a. the empirical cumulative distribution function). The $x$-axis is the TTC (in seconds), and the $y$-axis is the NCF, which is the cumulative number of samples

---

\(^5\)A state is said to be unsafe if: 1) at this state, there exists a deadlock in the corresponding program, or 2) starting from this state, the net will unavoidably or uncontrollably reach a state, where there exists a deadlock in the corresponding program; otherwise it is said to be safe.
TABLE I

<table>
<thead>
<tr>
<th>s</th>
<th>a</th>
<th>TLE</th>
<th>SS1</th>
<th>US1</th>
<th>n</th>
<th>P</th>
<th>T</th>
<th>SS2</th>
<th>US2</th>
<th>Time (s)</th>
<th>Iterations</th>
<th>Time Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>0.00</td>
<td>4202</td>
<td>969</td>
<td>16</td>
<td>35.62</td>
<td>29.00</td>
<td>1441</td>
<td>200</td>
<td>0.10</td>
<td>0.21</td>
<td>5.25</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.00</td>
<td>1441</td>
<td>200</td>
<td>27</td>
<td>41.46</td>
<td>34.69</td>
<td>1612</td>
<td>251</td>
<td>0.69</td>
<td>0.82</td>
<td>2.73</td>
</tr>
</tbody>
</table>

normalized by the sample size. A point \((x, y)\) on the graph means that a fraction of \(y\) samples have a TTC that is less than \(x\) seconds. From Fig. 7(a), we observe that using ICOG-O, 64% of the samples can be completed within 0.1 sec, while using ICOG-O-ES, 18% of the samples can be completed within 0.1 sec. Moreover, using ICOG-O, 89% of the samples can be completed within 1 sec, while using ICOG-O-ES, 43% of the samples can be completed within 1 sec. Fig. 7(b) is the empirical probability distribution function obtained by kernel density estimation. The \(x\)-axis is the TTC, and the \(y\)-axis is the probability. We see that using ICOG-O, the majority of the samples can be completed between 0.01 sec and 0.1 sec, while using ICOG-O-ES, the majority of the sample completion times span a wider range from 0.1 sec to 100 sec.

Fig. 7(c) is the NCF graph for the difference of the number of iterations of ICOG-O-ES and ICOG-O. The \(x\)-axis is the extra number of iterations required by ICOG-O-ES as compared to ICOG-O. The \(y\)-axis is the NCF. Note that for all the samples we tested, ICOG-O always requires fewer or equal number of iterations than ICOG-O-ES; and correspondingly, ICOG-O always synthesizes fewer or equal number of monitor places than ICOG-O-ES. From Fig. 7(c), we see that ICOG-O requires fewer iterations (and synthesizes fewer monitor places) than ICOG-O-ES for 43% of the samples.

Table I presents a summary of the experimental results of the comparative analysis between the performance of ICOG-O and that of ICOG-O-ES. For each row of the table, the sub-row with italics corresponds to the performance of ICOG-O-ES, and the counterpart without italics corresponds to the performance of ICOG-O. The first two columns correspond to the parameters used to generate the random sample Gadara nets. The first (s) and second (a) columns are the number of process subnets and the number of resource acquisitions per subnet. In generating the random nets, the
number of resources (locks) in the original Gadara net is set to be 11, the probability of acquiring a new resource before releasing one already held is 0.2, and the branching probability is 0.1. The third column (TLE) shows the ratio of sample nets that timed out in any iteration of ICOG-O and ICOG-O-ES. The fourth (SS) and fifth (US) columns describe the state space complexity. The sub-row without italics (resp., with italics) shows the average number of safe and unsafe states that are reachable by the original nets, where ICOG-O (resp., ICOG-O-ES) did not ever time out. Note that ICOG-O and ICOG-O-ES do not construct the state space, since they exploit structural properties of Gadara nets; these numbers were generated separately for the sake of scalability assessment. The sixth column (n) is the number of generated Gadara nets, where both ICOG-O and ICOG-O-ES did not ever time out throughout the iterations. The seventh (P) and eighth (T) columns correspond to the average number of places and transitions in the original Gadara nets. The ninth (SS2) and tenth (US2) columns show the average number of safe and unsafe states that are reachable by the original nets, where both ICOG-O and ICOG-O-ES did not ever time out. The eleventh column [time (s)] shows the average and standard deviation of the time (in seconds), the entire ICOG-O and ICOG-O-ES processes took until they converged. The twelfth column (iterations) shows the average and standard deviation of the number of iterations for ICOG-O and ICOG-O-ES to converge. Since for any sample net, the number of synthesized monitor places is always one less than the number of total iterations, we have not included the number of monitor places in the table. The last column (time/iteration) is the average time per iteration of ICOG-O and ICOG-O-ES.

In the experiments, we observed that the majority of time spent by ICOG-O or ICOG-O-ES is on the stage of RI empty siphon detection. This is precisely why we developed a customized MIP formulation for RI empty siphon detection in Gadara nets. Compared to the baseline performance of ICOG-O-ES, the data above show the efficiency attained by ICOG-O – the improvement in average time ranges from 17 to 404 times faster. In addition, the average number of iterations of ICOG-O is smaller than that of ICOG-O-ES for all the cases. From the second to fourth columns, we see that ICOG-O timed out on much fewer nets; and, on average, ICOG-O is able to handle much larger nets than ICOG-O-ES.

### C. Scalability Study of ICOG-O

Table II presents a sample of experimental results that highlight the scalability of ICOG-O. The first (SS) and second (US) columns are the number of safe and unsafe states. (Again, ICOG-O does not expand these states; these numbers were generated separately.) The third column [time (s)] is the total time (in seconds) for ICOG-O to converge. The fourth column (iters) is the number of iterations until convergence. We set a time-out threshold of 6000 sec for these experiments. Table II shows that ICOG-O is very scalable even on a modest computer set up.

### VII. DISCUSSION

In the analysis of multithreaded programs, our approach fully exploits the structural properties of the proposed Petri net models, without explicitly constructing the reachability space of the programs [8]. Our choice of Petri nets is also supported by the implementation of control logic. The overhead of controlling software can be generally attributed to two aspects: 1) control logic runtime decisions and 2) transitions blocked...
as a result of the control decisions. With an automaton model, the control decision is based on the global state of the program. In contrast, the control logic in a Petri net model is expressed as a set of decentralized monitor places, which only locally intervene the critical regions that are involved in the potential deadlocks, thus avoiding a global bottleneck for control decisions. A synthesized monitor place is essentially interpreted as a set of decentralized monitor places, which model locks) implies that the synthesized control logic can be implemented with primitives supplied by standard multithreading libraries, e.g., libthread. The framework of Gadara nets enables the synthesis of correct and maximally permissive control logic that provably prevents all the potential CMW-deadlocks in the program and will delay a lock acquisition only when necessary [23]. The customized methodology developed in this paper further guarantees that no redundant control logic is synthesized throughout the iteration process (Theorem 3).

VIII. CONCLUSION

We proposed an iterative control synthesis methodology for ordinary Gadara nets, called ICOG-O, based on structural analysis in terms of siphons. The control logic synthesized by ICOG-O enforces liveness in Gadara nets and provably eliminates all the potential CMW-deadlocks in the corresponding multithreaded programs. ICOG-O customizes the general control synthesis algorithm, ICOG, presented in [23], from which the properties of correctness and maximal permissiveness are preserved. In addition, we formally established a set of important properties of the proposed methodology, and showed that ICOG-O never synthesizes redundant control logic. Compared to the general ICOG and UCCOR [23], the customized ICOG-O and UCCOR-O presented in this paper focus on ordinary Gadara nets, and thus enable us to implement control synthesis based on a type of empty siphons. The customization permits a conceptually simpler process for the control of ordinary Gadara nets. It also simplified some steps in the general ICOG and UCCOR. Due to Theorem 3, in the control of ordinary Gadara nets, ICOG-O requires a fewer number of iterations than ICOG in general, and ICOG-O never synthesizes redundant control logic even without the bookkeeping of prevented states (that is required in ICOG). Our experimental results showed that ICOG-O is very efficient in terms of time and the number of synthesized monitor places. The results also demonstrated the scalability of our approach to large-scale real-world software. From a more general perspective, the results in the Gadara project illustrated that software failure avoidance is a fertile application area for discrete-event control, and moreover, special features from this application area are motivating further theoretical developments on the control of discrete-event systems.

APPENDIX

Definition 10: A Petri net dynamic system \( N = (P, T, A, W, M_0) \) is a bipartite graph \((P, T, A, W)\) with an initial number of tokens. Specifically, \( P = \{p_1, p_2, \ldots, p_n\} \) is the set of places, \( T = \{t_1, t_2, \ldots, t_m\} \) is the set of transitions, \( A \subseteq (P \times T) \cup (T \times P) \) is the set of arcs, \( W : A \to \{0, 1, 2, \ldots, \} \) is the arc weight function, and for each \( p \in P, M_0(p) \) is the initial number of tokens in \( p \).

The marking of a Petri net \( N \) is a column vector \( M \) of \( n \) entries corresponding to the \( n \) places. \( M_0 \) is the initial marking. We use \( M(p) \) to denote the (partial) marking on a place \( p \), which is a scalar. The notation \( \bullet p \) denotes the set of input transitions of place \( p \). \( \bullet p = \{t | (t, p) \in A\} \). Similarly, \( \bullet \) 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 transition \( t \) is enabled or fireable at a marking \( M \), if \( \forall p \in \bullet t, M(p) \geq W(p, t) \). 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 places may render them nonordinary. The incidence matrix \( D \) of a Petri net is an integer matrix \( D \in \mathbb{Z}^{n \times m} \), where \( D_{ij} = W(t_j, p_i) - W(p_i, t_j) \) represents the net change in the number of tokens in place \( p_i \) when transition \( t_j \) fires. A state machine is an ordinary Petri net such that each transition \( t \) has exactly one input place and exactly one output place, i.e., \( \forall t \in T, |\bullet t| = |t \bullet| = 1 \).

Let \( D \) be the incidence matrix of a Petri net \( N \). Any nonzero integer vector \( y \) such that \( D^T y = 0 \), is called a P-invariant of \( N \). Further, P-invariant \( y \) is called a P-semiflow if all the elements of \( y \) are nonnegative. 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 [28]. If a vector \( y \) is a P-invariant of Petri net \( N = (P, T, A, M_0) \), then we have \( M^T y = M_0^T y \) for any reachable marking \( M \in R(N, M_0) \). 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 \( \|y'\|\subseteq\|y\| \). 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 [28].

SBPI [22] 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 form: \( l^T M \leq b \), 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 4** [22], [33]: Consider a Petri net \( \mathcal{N} \), with incidence matrix \( D \) and initial marking \( M_0 \). If \( M_0 \) 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, i.e., a transition in the net is disabled by the monitor place only if its firing leads to a marking where the given linear constraint \( l^T M \leq b \) is violated.

**REFERENCES**


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

He joined Hewlett-Packard Laboratories (HP Labs), Palo Alto, CA, in 2009. He was an Intern with Microsoft Shanghai, IBM Almaden Research Center, and HP Labs.

Jason Stanley received the B.Eng. degree in computer science from the University of Michigan, Ann Arbor, in 2012.

He was a Research Assistant with the Electrical Engineering and Computer Science Department, University of Michigan. He is currently with Citadel Investment Group, Chicago, IL. His current research interests include computer science theory and machine learning.

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

He has been with the University of Michigan, Ann Arbor, since 1986, where he is currently a Professor of electrical engineering and computer science. He has co-authored the book entitled Introduction to Discrete Event Systems, Second Edition (Springer, 2008). His current research interests include discrete event systems and multiple problem domains, such as modeling, diagnosis, control, optimization, and applications to computer systems.

Dr. Lafortune is a member of the Editorial Boards of the Journal of Discrete Event Dynamic Systems: Theory and Applications and the International Journal of Control. He was a recipient of the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 and 2001, for co-authored papers. He is the lead developer of the software package UMDES and a co-developer of DESUMA with L. Ricker.

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

He is currently a Professor with the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta. His current research interests include discrete event systems theory and its applications.

Dr. Reveliotis is a member of INFORMS. He is currently an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and a Department Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION and the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and a Senior Editor on the Conference Editorial Board of the IEEE International Conference on Robotics and Automation. In 2009, he was the Program Chair of the IEEE Conference on Automation Science and Engineering, for which he is currently a member of the Steering Committee. He was a recipient of several awards, including the Kayamori Best Paper Award at 1998 IEEE International Conference on Robotics and Automation.

Terence Kelly (SM’09) received the Ph.D. degree in computer science from the University of Michigan, Ann Arbor.

He is currently a Senior Researcher with the Intelligent Infrastructure Laboratory, Hewlett-Packard Laboratories. His current research interests include discrete control theory to failure avoidance and elimination in computing systems.

Dr. Kelly is a Senior Member of the ACM.

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

He is currently a Professor with the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, where he leads the Compilers Creating Custom Processors Group (http://cccp.eecs.umich.edu) that is involved in research on technologies in compilers for multicore processors, application-specific processors for mobile computing, and reliable system design.

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