IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INTELLIGENT INFORMATION SERVICES

Received December 20, 2019, accepted January 2, 2020, date of publication January 6, 2020, date of current version January 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2964312

A Deadlock Prevention Policy for a Class of Multithreaded Software

WENLI DUO, XIAONING JIANG, OUSSAMA KAROUI, XIN GUO, DAN YOU, (Student Member, IEEE), SHOUGUANG WANG, (Senior Member, IEEE), AND YUAN RUAN

School of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou 310018, China

Corresponding author: Xiaoning Jiang (jiangxiaoning@zjgsu.edu.cn)

This work was supported in the part by the Zhejiang Provincial Key R&D Program of China under Grant 2018C01084, in part by the Zhejiang Natural Science Foundation under Grant LQ20F020009, and in part by the Zhejiang Gongshang University, Zhejiang Provincial Key Laboratory of New Network Standards and Technologies under Grant 2013E10012.

ABSTRACT Deadlock is an undesired situation in multithreaded software since it can lead to the stoppage of software. This paper studies the problem of deadlock control of multithreaded software based on Gadara nets, which are well studied for modelling concurrent programs. In particular, an iterative deadlock prevention policy based on siphons is proposed for a class of ordinary Gadara nets where the initial marking of each idle place is one. At each iteration, we compute emptiable siphons containing the smallest number of resource places. Then, bad markings are computed based on these siphons. On the basis of the bad markings, a constraint is constructed that forbids not only bad markings that empty one of the siphons but also some other bad markings. The algorithm is carried out until no emptiable siphon exists in the net. Compared with the existing methods, the resultant net derived from the proposed method is live and maximally permissive with a simpler supervisor. Finally, two examples are provided to illustrate the proposed deadlock prevention policy.

INDEX TERMS Petri nets, multithreaded software, deadlock prevention.

I. INTRODUCTION

The transformation of computer hardware from a single processor core to multiple processor cores has promoted the emergence of multi-threaded parallel programming systems (MPPS). The development and usage of multithreaded software are gradually coming into the mainstream [30], [31]. A major feature of this type of software is the sharing of data among threads, which however makes multithreaded software vulnerable to concurrency errors. Hence, it usually requires lock primitives to protect the shared data in parallel programming paradigm. One of the most typical lock primitives is mutual exclusion locks (mutexes). However, circular-mutex-wait (CMW) deadlocks will occur if mutexes are improperly used [12], [14]–[17]. In this case, a set of threads wait infinitely for resources held by each other and none of them can continue to execute. Such deadlocks are always deemed as undesirable situations since they can lead to the stoppage of the software or even cause catastrophic consequences. Therefore, it is necessary to implement effective deadlock control methods to ensure that deadlocks never occur.

Many efforts have been made over the years to deal with the deadlock problem in multithreaded software, resulting in many methods such as static deadlock prevention and detection [8], [26], [32] dynamic deadlock detection and avoidance [12], [14]. Static deadlock prevention prevents deadlocks by executing a strict order to acquire mutexes. Such a method is straightforward in principle, but difficult to apply in practice. Static deadlock detection is mainly based on program analysis, which usually suffers from spurious warning. Dynamic deadlock detection utilizes automated rollback technique to debug the behavior of programs online [11], however it cannot detect all potential deadlocks. As for dynamic deadlock avoidance, Banker's algorithm [1], [2] is often used to avoid deadlocks. Unfortunately, it requires a large amount of data and comes with high computational cost.

To solve the above problems, researchers gradually turn their attention to model-based deadlock control methods. One of the most common tools for modeling multithreaded software is Petri nets [3], [6], [9], [10], [13]. Petri nets are a powerful tool for investigating the supervisory control

The associate editor coordinating the review of this manuscript and approving it for publication was Guanjun Liu.

methods of discrete event systems [22]. They can describe the inherent dynamics of the systems accurately and intuitively, while avoid state enumeration. Wang *et al*. [12] propose a subclass of Petri nets named *Gadara nets*. Compared with other subclasses of Petri nets, it better models concurrent programs with lock acquisition and release operations. In order to establish a systematic modelling formalism, Liao *et al*. [15] formally define the class of Gadara nets as well as controlled Gadara nets. They show that deadlock-freeness of a program is related to the liveness of its Gadara model. Furthermore, they present a necessary and sufficient condition for the liveness of Gadara nets, i.e., a Gadara net is live if and only if there exists no resource-induced deadly marked siphon in its modified marking space. Motivated by the work of [15], Liao *et al*. propose an optimal control policy called ICOG for Gadara nets [16]. An original Gadara net is ordinary by definition. However, the controlled Gadara net may be non-ordinary since the added monitor places may possess weighted arcs. Thus, the control policy studied in [16] aims at the generalized Gadara nets. Subsequently, an improved ICOG method called ICOG-O is presented to achieve optimal control for ordinary Gadara nets [17]. In the above methods, mixed integer programming (MIP) formulations are used to detect siphons in Gadara nets that lead to a computational bottleneck in the algorithm. Stanley *et al*. [18] propose a boolean satisfiability formulation to detect siphons in Gadara nets. Moreover, they combine this paradigm with ICOG scheme to obtain a deadlock detection and avoidance approach for Gadara nets.

In general, there are three criteria to evaluate a deadlock control policy: computational complexity, structural complexity, and behavioral complexity. Although ICOG scheme is maximally permissive and its computational complexity depends on the algorithm of siphon detection only, the controlled net synthesized by ICOG may face the problem of adding multiple monitor places. In this paper, we proposes an iterative deadlock prevention policy for a class of Gadara nets to ensure that the controlled net is not only live and maximally permissive but also with low structural complexity.

The iterative deadlock control policy proposed in the paper mainly focuses on the minimal resource-number emptiable siphon (MRES), i.e., the minimal emptiable siphon containing the smallest number of resource places. At each iteration, all MRESs in a Gadara net are computed by a function named *ComputeMRESs*. If the output of the function is not empty, a function named *FindBMs* is called to compute bad markings. Then, monitor places are computed by the technique of Supervision Based on Place Invariants (SBPI) [20], [21], which prevent the computed bad markings from being reached. The resultant net is input to the Function *ComputeMRESs* to compute MRESs and the above steps are repeated until there is no MRES in the controlled net. Finally, the obtained controlled net is live and maximally permissive. At each iteration, the algorithm takes into account all the bad markings that can empty an MRES so that each synthesized monitor can prevent as many bad markings as possible. Thus, the number of monitor places that need to be added to the Gadara net is decreased, which reduces the structural complexity of the final controlled net. In addition, uncontrollable transitions are taken into consideration so that the proposed method can be applied to Gadara nets with uncontrollable transitions.

The rest of the paper is organized as follows. Section II reviews basic definitions and properties of Petri nets and recalls the notions of Gadara nets. Section III studies the method to compute bad markings based on siphons and reachability analysis for a class of Gadara nets. An iterative deadlock prevention policy for a class of Gadara nets is proposed in Section IV. Some examples and comparisons with existing methods are provided in Section V to show the performance of the proposed policy. Finally, Section VI concludes this paper.

II. PRELIMINARIES

In this section, we recall the formalism used in the paper, namely Petri nets and Gadara nets. For more details, we refer the readers to [4] and [15].

A. PETRI NETS

A *Petri net* is a four-tuple $N = (P, T, F, W)$, where P is the set of *places* and T is the set of *transitions*. Both the two sets are finite and non-empty sets with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$. $F \subseteq (P \times T) \cup (T \times P)$ describes the *flow relation* of the net, represented by arcs with arrows from places to transitions or from transitions to places. $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a mapping that assigns a weight to an arc such that $W(x, y) > 0$ iff $(x, y) \in F$, and $W(x, y) = 0$, otherwise, where $x, y \in P \cup T$ and $\mathbb{N} = \{0, 1, 2, \ldots\}$. If $\forall (x, y) \in F, W(x, y) = 1, N$ is said to be *ordinary* and denoted as $N = (P, T, F)$. Given a node $x \in P \cup T$, the *preset* of x is denoted as ${}^\bullet x = \{y \in P \cup T | (y, x) \in F\}$ and the *postset* of x is $x^\bullet = \{y \in P \cup T | (x, y) \in F\}$. A *marking* of N is a mapping $M: P \to \mathbb{N}$. Usually, a marking M is denoted by the multi-set notation $\sum_{p \in P} M(p)p$, where $M(p)$ is the number of *tokens* in place p at M. *Incidence matrix* $[N]$ of a Petri net is a $|P| \times |T|$ integer matrix with $[N](p, t) = W(t, p) + W(p, t)$.

A transition t is *enabled* at M if $\forall p \in^\bullet t, M(p) \geq W(p, t)$, which is denoted as $M[t\rangle$. A transition t can *fire* if t is enabled at M. Once t fires, M reaches another marking M', denoted as $M[t\rangle M'$, where $M'(p) = M(p) - W(p, t) + W(t, p), \forall p \in P$. M' is said to be *reachable* from M if there is a transition sequence $\sigma = t_1 t_2 \ldots t_n$ and markings $M_1, M_2, \ldots$, and M_{n-1} such that $M[t_1\rangle M_1[t_2\rangle M_2 \ldots M_{n-1}[t_n\rangle M'$ holds. We use $M[\sigma\rangle M'$ to denote that M' is reachable from M via a transition sequence σ. For a Petri net $N, R(N, M_0)$ is often used to denote the set of all markings that reachable from initial marking M_0. The set of all markings that are reachable from M is denoted as $R(N, M)$. Petri net N is said to be *reversible* if $\forall M \in R(N, M_0), M_0 \in R(N, M)$.

Given a Petri net (N, M_0), a transition $t \in T$ is *live* at M_0 if $\forall M \in R(N, M_0), \exists M' \in R(N, M), M'[t\rangle$. (N, M_0) is *live* if $\forall t \in T, t$ is live at M_0. A Petri net N is said to be *dead*

at M_0 if $\nexists t \in T, M_0[t\rangle$. (N, M_0) is *deadlock-free* if $\forall M \in R(N, M_0), \exists t \in T, M[t\rangle$.

A *path* of Petri net N is a string $\pi = x_1x_2\ldots x_n$ that $\forall i \in \mathbb{N}_{n-1}, x_{i+1} \in x_i^\bullet$, where $\forall x \in \{x_1, x_2, \ldots, x_n\}, x \in P \cup T$. If $x_1 = x_n$, then the path π is called a *circuit*. $P(\pi)$ is used to denote the set of places in π and the set of transitions in π is denoted as $T(\pi)$.

Let (N, M_0) be a Petri net, a nonempty set of places S is said to be a *siphon* if ${}^\bullet S \subseteq S^\bullet$.

Let $[N]$ denotes the incidence matrix of a Petri net N. A *P-vector* is a column vector I: $P \to \mathbb{Z}$ indexed by P, where $\mathbb{Z}$ is the set of integers. I is a *P-invariant* if $I \neq 0$ and $I^T \bullet [N] = 0^T$ holds. P-invariant I is a *semiflow* if every element of I is non-negative. The *support* of a P-semiflow I is denoted by $||I|| = \{p \in P | I(p) \neq 0\}$.

B. GADARA NETS

Gadara nets are composed of a set of process subnets and resource places, where the former corresponds to thread entry points in the program and the latter to the locks.

Definition 1 [15]: 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 $N_G = (P, T, F, M_0)$ where:

1. $P = P_0 \cup P_A \cup P_R$ is a partition such that:

a) $P = \cup_{i \in IN} P_{Ai}$, $P_{Ai} \neq \emptyset$, and $P_{Ai} \cap P_{Aj} = \emptyset$, for all $i \neq j$;

b) $P_0 = \cup_{i \in IN} P_{0\,i}$, where $P_{0\,i} = \{p0\ i\}$; and

c) $P_R = \{r_1, r_2, \ldots, r_k\}, k > 0$.

2. $T = \cup_{i \in IN} T_i$, $T_i \neq \emptyset$, $T_i \cap T_j = \emptyset$, for all $i \neq j$.

3. For all $i \in I_N$, the subnet N_i generated by $P_{Ai} \cup \{p_{0\,i}\} \cup T_i$ is a strongly connected state machine.

4. $\forall p \in P_A$, 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 $P_A \cap ||Y_r|| \neq \emptyset$.

6. $\forall r \in P_R, M_0(r) = 1$, $\forall p \in P_A, M_0(p) = 0$, and $\forall p_0 \in P_0, M_0(p_0) \geq 1$.

7. $P_A = \cup_{r \in PR}(||Y_r|| \backslash \{r\})$.

Given a Gadara net, SBPI [21] is often employed to enforce the control logic on the net. The resultant net is augmented with a set of monitor places that is defined as follows.

Definition 2 [15]: Let $N_G = (P, T, F, M_0)$ be a Gadara net. A controlled Gadara net $N_G^c = (P \cup P_C, T, F \cup F_C, W^c, M_0^c)$ is a self-loop-free Petri net such that, in addition to all conditions in Definition 1 for N_G, it holds that:

8. For each $p_c \in P_C$, there exists a unique minimal-support P-semiflow, Y_{pc}, such that $\{p_c\} = ||Y_{pc}|| \cap P_C$, $P_0 \cap ||Y_{pc}|| = \emptyset$, $P_R \cap ||Y_{pc}|| = \emptyset$, $P_A \cap ||Y_{pc}|| \neq \emptyset$, and $Y_{pc}(p_c) = 1$.

9. For each $p_c \in P_C$, $M_0^c(p_c) \geq \max_{p \in P_A} Y_{pc}(p)$.

The structural properties of monitor places $p_c \in P_C$ in N_G^c are similar to those of resource places. Moreover, $\forall p_c \in P_C$, the initial marking of p_c can be greater than one and the weights of arcs associated with p_c can be non-unit. Hence, the monitor places are often considered as generalized resource places, i.e.,$\forall r, r \in P_R \cup P_C$. Thus, N_G^c preserves the net structure of N_G. The controlled Gadara net N_G^c is a generalization of Gadara net N_G, namely, N_G is a subclass of N_G^c. N_G is ordinary while N_G^c can be non-ordinary. Furthermore, we note that the controlled net is ordinary if the arcs associated with monitor places in the net possess unit weights.

The Gadara nets considered in this paper actually belong to a class of ordinary controlled Gadara nets N_G^c where $\forall p_0 \in P_0, M_0(p_0) = 1$. For the sake of simplicity, we use $N_G = (P, T, F, M_0)$ to denote the Gadara nets considered in this paper unless special mention is made.

III. COMPUTATION OF BAD MARKINGS BASED ON SIPHONS

This section provides a method to compute bad markings based on siphons and reachability analysis.

Deadlocks are closely related to emptiable siphons in Gadara nets. Specifically, a net system is non-live once a siphon is emptied. According to the work of Liao *et al.* [15], we have the following theorem that reveals the relationship between emptiable siphons and ordinary Gadara nets. It shows that the absence of emptiable siphons is a necessary and sufficient condition for the liveness of an ordinary Gadara net.

Theorem 1 [15]: An ordinary Gadara net is live *iff* there exists no emptiable siphon in the net.

The deadlock prevention policy proposed in this paper focuses on a class of minimal emptiable siphons defined as follows. In simple words, they are minimal emptiable siphons containing the smallest number of resource places.

Definition 3: Given a Gadara net $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$, a siphon S of N_G is called a *Minimal Resource-number Emptiable Siphon* (MRES) if:

1) S is an minimal emptiable siphon; and
2) $\nexists S' \in N_G$ such that S' is an minimal emptiable siphon and $|S'_R| < |S_R|$, where $S_R = S \cap P_R$.

Definition 4: Let $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$ be a Gadara net and $r \in P_R$. The set of the holders of r is defined as $H(r) = ||Y_r|| \cap P_A$, where $||Y_r||$ is the unique minimal-support P-semiflow of r. Given a siphon S, the set $[S] = (\cup_{r \in S} H(r)) \backslash S$ is called the *complementary set* of siphon S.

Consider a Gadara net in Fig. 1. There exists an MRES in the net, that is $S = \{p_{11} - p_{13}, p_{15} - p_{18}, p_{21}, p_{23}, p_{34}, R_1, R_3\}$. R_1 and R_3 are both resource places with $||Y_{R1}|| = \{p_{11} - p_{18}, p_{23}, R_1\}$ and $||Y_{R3}|| = \{p_{15}, p_{16}, p_{21} - p_{23}, p_{34}, R_3\}$. According to Definition 4, $H(R_1) = \{p_{11} - p_{18}, p_{23}\}$ and $H(R_1) = \{p_{15}, p_{16}, p_{21} - p_{23}, p_{34}\}$. As a result, the complementary set of S is $[S] = (\cup_{r \in S} H(r)) \backslash S = \{p_{14}, p_{22}\}$.

We note that, if we control siphons randomly in Gadara nets, it could happen that the number of siphons to be controlled is very large, which leads to high computational and structural complexity. Moreover, it could happen that the weights of some arcs related to monitor places are non-unit. In this case, bad siphons possibly emerge whose control is much more complex than that of emptiable siphons.

Whether a siphon can be emptied is directly related to its complementary set. Given an MRES S, all the tokens

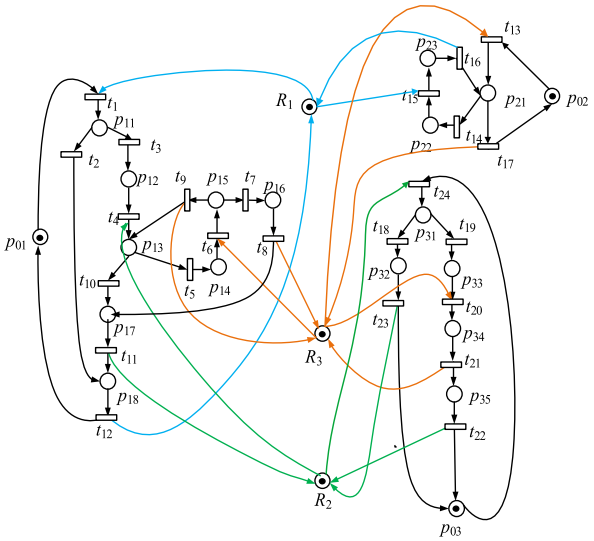


FIGURE 1. A gadara net model for deadlock in linux kernel.

in S flow into its complementary set $[S]$ if S is empty. In other words, S can be emptied when certain places of $[S]$ are marked. Thus, we construct a constraint to control the complementary set, guaranteeing that not all tokens in the siphon can flow into its complementary set. The constraint is

$$\sum_{p\in[S]} M(p) \leq M_0(S) - 1 \tag{1}$$

where $M_0(S) = \sum_{p\in S} M_0(p)$.

In the following, σ or σ' is used to denote a transition sequence.

Definition 5: Given a Petri net (N, M_0) and a reachable marking M of the net, M can be divided into three categories:

1) M is a *live marking* if $\exists\sigma$ such that $M[\sigma\rangle M_0$;
2) M is a *bad marking* if $\exists\sigma$ such that $M[\sigma\rangle M'$ where $M' \in R(N, M_0)$, but $\nexists\sigma'$ such that $M[\sigma'\rangle M_0$. Or M is bad if there is no control logic to make M reach M_0;
3) M is a *deadlock marking* if all transitions are disabled at M.

Note that condition 2 of Definition 5 shows that a marking $M \in R(N_G, M_0)$ is bad if there is no control logic to make M reach M_0. It is determined by the structure of Gadara net. The condition 4 of Definition 1 shows that $\forall p \in P_A$, if $|p^\bullet| > 1$, then $\forall t \in p^\bullet, {}^\bullet t \cap P_R = \emptyset$, which means branching transitions cannot request any resource. However, monitor places are usually considered as a class of resource places. Thus, the output arcs of monitor places cannot connect to any branching transition, i.e., branching transitions cannot be disabled by monitors. Therefore, a marking is bad if it can reach deadlock markings via a branching transition since it cannot be controlled by monitors.

Branching transitions in Gadara net are similar to uncontrollable transitions. Hence, branching transitions in Gadara net are often regarded as uncontrollable.

Definition 6: Given a Gadara net $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$, $\forall p \in P_A$, if $|p^\bullet| > 1$, then $\forall t \in p^\bullet, t$ is uncontrollable.

The control logic defined by constraint (1) can forbid a set of deadlock markings and bad markings. However, it may be inefficient since it can only remove a small number of bad markings. In addition, new emptiable siphons may emerge in the net after monitor places are added. In other words, new deadlock markings may arise. To solve these problems, we provide a method that considers bad markings in a class of Gadara nets. We need the following definitions before presenting the method.

Definition 7: Given a Gadara net (N_G, M_0) and an emptiable siphon S in N_G, the set of reachable markings at which S is empty is denoted as $A(S)$, i.e, $A(S) = \{M \in R(N_G, M_0)|M(S) = 0\}$ where $M(S) = \sum_{p\in S} M(p)$.

Definition 8: Let S be an emptiable siphon in a Gadara net N_G and $[S]$ be the complementary set of S. $C(S)$ is defined as $C(S) = \{P' \subseteq [S]|M(S) = 0, \forall p \in P', M(p) = 1\}$.

Take the Gadara net shown in Fig. 1 as an example. There is an MRES S in the net, that is, $S = \{p_{11}-p_{13}, p_{15}-p_{18}, p_{21}, p_{23}, p_{34}, R_1, R_3\}$. Its complementary set is $[S] = \{p_{14}, p_{22}\}$ by Definition 4. Siphon S is emptied if p_{14} and p_{22} are both marked. Clearly, $C(S) = \{\{p_{14}, p_{22}\}\}$.

Definition 9: Given a Gadara net $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$ and a set of place $P' \subseteq P_A$, the set of reachable markings at which all places of P' are marked is defined as $B(P') = \{M \in R(N_G, M_0)|\forall p \in P', M(p) = 1\}$.

Let Ω be an element of $C(S)$, where S is an emptiable siphon. Clearly, S is empty if all the places of Ω are marked, i.e., S is empty at the marking $M \in B(\Omega)$. Hence, we have the following lemma.

Lemma 1: Given a Gadara net N_G, an emptiable siphon S of the net and an element Ω of $C(S)$, it holds that

$$A(S) = \cup_{\Omega\in C(S)} B(\Omega).$$

Proof: The proof is trivial. ♣

Consider again the Gadara net in Fig. 1 and the siphon $S = \{p_{11}-p_{13}, p_{15}-p_{18}, p_{21}, p_{23}, p_{34}, R_1, R_3\}$, where $C(S) = \{\{p_{14}, p_{22}\}\}$. Let $\Omega = \{p_{14}, p_{22}\}$ and we have $B(\Omega) = \{M \in R(N, M_0)|M(p_{14}) = M(p_{22}) = 1\}$. There is only one element in $C(S)$. Hence, we have

$$A(S) = B(\Omega) = \{M \in R(N, M_0)|M(p_{14}) = M(p_{22}) = 1\}.$$

$A(S)$ contains all the markings at which S is empty. Obviously, constraint (1) can only remove the markings of $A(S)$ but cannot remove other bad markings that may reach markings in $A(S)$. In our work, $A(S)$ can be equivalent to many sets of $B(\Omega)$. By introducing $B(\Omega)$, we can look for bad markings based on certain operation places rather than considering all the reachable markings.

Definition 10: Given a Gadara net $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$, and a set of places $P' \subseteq P_A$, a place p' is called the *downstream* (*upstream*) place of p if $p' \in P_A$ and there

exists a path π from p to p' (from p' to p) such that $P(\pi) \cap P_0 = \emptyset$, $P(\pi) \cap P_R = \emptyset$ and $T(\pi) \neq \emptyset$. The set of all downstream (upstream) places of p is denoted as $P_d(p)$ ($P_u(p)$). Given a set of places P', the set of downstream (upstream) places of all the places in P' is denoted as $P_d(P')(P_u(P'))$.

Consider the Gadara net in Fig. 1 and a place set $P' = \{p_{14}, p_{22}\}$. By Definition 10, the set of all downstream places of p_{14} is $P_d(p_{14}) = \{p_{13}, p_{15}, p_{16}, p_{17}, p_{18}\}$ and the set of all upstream places of p_{14} is $P_u(p_{14}) = \{p_{11}, p_{12}, p_{13}, p_{15}\}$. In addition, the sets of downstream and upstream places of all places in P' are $P_d(P') = \{p_{13}, p_{15}, p_{16}, p_{17}, p_{18}, p_{21}, p_{23}\}$ and $P_u(P') = \{p_{11}, p_{12}, p_{13}, p_{15}, p_{21}, p_{23}\}$.

Definition 11: Given a Gadara net $N_G = (P_0 \cup P_A \cup P_R, T, F, M_0)$ and a set of place $P' \subseteq P_A$, the set of reachable markings at which k places of P' are marked is defined as $B_k(P') = \{M \in R(N_G, M_0) | \Sigma_{p \in P'} M(p) = k \wedge M(p) \leq 1\}$, where $k \in \{1, 2, \ldots, |P'|\}$.

Suppose that $P' = \{p_1, p_2, p_3\}$ and any two places of P' can be marked at the same. Thus, $B_2(P') = B(\{p_1, p_2\}) \cup B(\{p_1, p_3\}) \cup B(\{p_2, p_3\})$. Obviously, $|B_k(P')| \leq |B(P')|$. Thus, k can be ignored if $k = |P'|$.

As $B(\Omega)$ is a set of bad markings, places of Ω cannot be marked simultaneously; otherwise, the system can reach a bad marking. Hence, we have the following proposition.

Proposition 1: Given a Gadara net N_G, an MRES S of the net and an element Ω of $C(S)$. If N_G^c is a live controlled net of N_G, then the following constraint holds in N_G^c:

$$\sum\nolimits_{p \in \Omega} M(p) \leq |\Omega| - 1. \tag{2}$$

Proof: By contradiction, suppose that (2) does not hold in N_G^c. It implies there is a marking $M \in (N_G^c, M_0)$ at which $\forall p \in \Omega, M(p) = 1$. Since $\Omega \in C(S)$, S is empty at M. Based Theorem 1, N_G^c is not live, which contradicts N_G^c is live. Thus, (2) holds in N_G^c if N_G^c is a live controlled net of N_G. ♣

On the basis of the above definitions and analysis, Function *FindBMs* is developed to compute bad markings. Given a Gadara net N_G and an MRNE S of N_G, $C(S)$ can be computed by Definition 8. Let $\Omega \in C(S)$ where $B(\Omega)$ is a set of bad markings, at which S is empty. Then, a set Π of MRESs is given. The basic idea of Function *FindBMs* is as follows:

First, according to Definition 9 and Lemma 1, a set of bad markings Ψ can be computed based on the MRESs in steps 1 to 6.

Second, let $\Phi = \Omega$ and $k = |\Omega|$. We compute bad markings by considering the upstream places of Φ. More specifically, consider a place $p \in^{\bullet\bullet} \Phi \cap P_u(\Phi)$ and $P_u(\Phi) = \emptyset$. As we know, N_G can reach a bad marking once k places of Φ are marked. To obtain more bad markings, we want to determine whether the markings are bad where p and k-1 places of Φ are marked, i.e., whether $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings. To facilitate the understanding, we explain $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ as follows. The Gadara net considered in our work satisfies that $\forall p_0 \in P_0, M(p_0) = 1$, i.e., at most one operation place can be marked on each subnet. If $B_k(\{p\} \cup (\Phi \backslash P_d(p))) \neq \emptyset$, p must be marked and only k-1 places can be marked in $\Phi \backslash P_d(p)$. Thus, $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of markings where p and k-1 places of Φ are marked.

Third, steps 10 to 20 of Function *FindBMs* are used to determine whether $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings. In steps 10 to 13, if $\forall M \in B_k(\{p\} \cup (\Phi \backslash P_d(p)))$, M always reaches the marking of Ψ, then $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings since any marking of Ψ is bad. Furthermore, Definition 5 shows that a marking is bad if it can reach a bad marking via uncontrollable transitions. Hence, steps 14 to 17 are used to determine whether the markings of $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ can reach bad markings via uncontrollable transitions. Once $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings, p is added to Φ. The function terminates when all the upstream places of Φ are considered.

Finally, a set of places Φ is obtained where $B_k(\Phi)$ is a set of bad markings.

Theorem 2: Given a Gadara net N_G, an MRES S of the net, an element Ω of $C(S)$ and a set Π of MRESs, $B_{|\Omega|}(\Phi)$ is a set of bad markings that can empty siphon S, where $\Phi = \mathit{FindBMs}(\Omega, \Pi)$.

Proof: Consider the following cases:

1) If no place is added to Φ, then $\Phi = \Omega$. Let $k = |\Omega|$, $B_k(\Phi) = B(\Phi) = B(\Omega)$ is a set bad markings that can empty siphon S;

2) If a place p is added to Φ, then p must satisfy at least one of the following conditions:

a) $\forall M \in B_k(\{p\} \cup (\Phi \backslash P_d(p)))$, $\forall M' \in R(N_G, M)$ such that $R(N_G, M') \cap \Psi \neq \emptyset$. It implies M can always reach markings of Ψ. By contradiction, suppose that there is a marking $M \in B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ cannot always reach markings of Ψ. Thus, $\exists M' \in R(N_G, M), \forall M_\Psi \in \Psi, \nexists \sigma$ such that $M'[\sigma\rangle M_\Psi$, i.e., $R(N_G, M') \cap \Psi = \emptyset$. Clearly, it contradicts that $\forall M' \in R(N_G, M)$ such that $R(N_G, M') \cap \Psi \neq \emptyset$. Hence, given a marking M, if $\forall M' \in R(N_G, M)$ such that $R(N_G, M') \cap \Psi \neq \emptyset$, M can always reach markings of Ψ.

Ψ is known to be a set of bad markings, and M is also bad if M can always reach the markings of Ψ. By contradiction, suppose that M is a live marking so that there must exist a transition sequence σ' such that $M[\sigma'\rangle M_0$. We know that M must reach a marking of Ψ before it reaches M_0. As any marking of Ψ is bad, there exists no transition sequence σ'' such that $M[\sigma''\rangle M_0$, which contradicts that there is a transition sequence σ' such that $M[\sigma'\rangle M_0$. Hence, $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings if all the markings of $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ can always reach the markings of Ψ;

b) $|p^\bullet| > 1$. It implies that the output transitions of p are branching transitions. Definition 6 shows that branching transitions in Gadara net are uncontrollable so that they are not disabled by any monitors. Due to $p \in^{\bullet\bullet} \Phi$, there is at least one transition sequence such that the markings of $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ can reach the markings of $B(\Omega)$ via uncontrollable transitions. On the basis of Definition 5, $B_k(\{p\} \cup (\Phi \backslash P_d(p)))$ is a set of bad markings;

Function $\Phi = FindBMs\ (\Omega, \Pi)$

Input: A set $\Omega \in C(S)$ and a set Π of MRESs in a Gadara net (N_G, M_0).
Output: A set of places Φ.
1. Let $\Psi := B(\Omega)$;
2. **for** each $S' \in \Pi$ **do**
3. compute $C(S')$ based on Definition 8;
4. compute $A(S')$ based on Lemma 1; /*$A(S') = \cup_{\Omega \in C(S')} B(\Omega)$*/
5. $\Psi := \Psi \cup A(S')$;
6. **end for**/* compute a set of bad markings based on the known siphons */
7. let $\Phi := \Omega$ and $k := |\Omega|$; /*k is the number of places in Ω*/
8. **while** $(P_u(\Phi) \neq \emptyset)$ **do**
9. **if** $\exists p \in^{\bullet\bullet} \Phi \cap P_u(\Phi)$ **then**
10. **if** $\forall M \in B_k(\{p\} \cup (\Phi \backslash P_d(p)))$, $\forall M' \in R(N_G, M)$ such that $R(N_G, M') \cap \Psi \neq \emptyset$**then** /*$M$ will always reach markings of Ψ*/
11. $\Phi := \Phi \cup \{p\}$;
12. $\Psi := \Psi \cup B_k(\{p\} \cup (\Phi \backslash P_d(p)))$;
13. **end if**
14. **if** $|p^{\bullet}| > 1$ **then**/* if $|p^{\bullet}| > 1$, then $\forall t \in p^{\bullet}, t$ is uncontrollable according to Definition 6*/
15. $\Phi := \Phi \cup \{p\}$;
16. $\Psi := \Psi \cup B_k(\{p\} \cup (\Phi \backslash P_d(p)))$;
17. **end if**
18. $P_u(\Phi) := P_u(\Phi) - p$;
19. **else**
20. $P_u(\Phi) = \emptyset$;
21. **end if**
22. **end while**
23. output: Φ;
24. **end**

Based on the above analysis, $B_k(\Phi)$ is a set of bad markings at the initial situation. A place will be added to Φ only if at least one of the above conditions is satisfied. Hence, $B_k(\Phi)$ is still a set of bad markings after adding these places. In addition, all the added places satisfy $p \in P_u(\Omega)$ and the Gadara net considered in our work is subject to $\forall p \in P_0, M_0(p_0) = 1$, i.e., only one operation place can be marked on each subnet. Thus, $\forall M \in B_k(\Phi)$, there exists at least one transition sequence σ such that $M[\sigma\rangle M'$, where $M' \in B(\Omega)$. Therefore, $B_k(\Phi)$ is a set of bad markings that can empty siphon S. ♣

Theorem 2 shows that we can find some bad markings based on the MRESs. Furthermore, these markings can eventually empty a siphon. Obviously, any $|\Omega|$ places of Φ cannot be marked at the same time, since otherwise, the net can inevitably reach a deadlock state. Hence, it is trivial to obtain the following proposition.

Proposition 2: Given a Gadara net N_G, an MRES S of the net, an element Ω of $C(S)$ and a set Π of MRESs, it holds that $\sum_{p \in \Phi} M(p) \leq |\Omega| - 1$ in N_G^c where $\Phi = FindBMs(\Omega, \Pi)$, if N_G^c is live controlled net of N_G.

Consider the Gadara net in Fig. 1. There is an MRES in the net that is $S_1 = \{p_{11} - p_{13}, p_{15} - p_{18}, p_{21}, p_{23}, p_{34}, R_1, R_3\}$. The complementary set of S_1 is $[S_1] = \{p_{14}, p_{22}\}$ and $C(S_1)$ can be computed based on Definition 8, i.e., $C(S_1) = \{\{p_{14}, p_{22}\}\}$. There is no other MRES in the net. Let $\Omega = \{p_{14}, p_{22}\}$ and $\Pi = \{S_1\}$.

Next, Function *FindBMs* is called. First, we have $\Phi = \Omega = \{p_{14}, p_{22}\}$ and $\Psi = B(\Omega) = \{M \in R(N, M_0) | M(p_{14}) = M(p_{22}) = 1\}$. Then, each place of $P_u(\Omega) = \{p_{11}, p_{12}, p_{13}, p_{15}, p_{21}, p_{23}\}$ is taken into account. Consider the markings of $B(\{p_{13}, p_{22}\})$ since $p_{13} \in^{\bullet\bullet} p_{14}, t_{15}$ is disabled and t_5, t_{10} are enabled at the markings of $B(\{p_{13}, p_{22}\})$. However, t_5 and t_{10} are both branching transitions, which implies that p_{14} can obtain a token from p_{13} via an uncontrollable transition. Thus, we have a set of bad markings where $B(\{p_{13}, p_{22}\}) = \{M \in R(N, M_0) | M(p_{13}) = M(p_{22}) = 1\}$. p_{13} should be added to Φ and $\Psi = B(\Omega) \cup B(\{p_{13}, p_{22}\})$. Next, consider place p_{12}, i.e., consider the markings of $B(\{p_{12}, p_{22}\}) = \{M \in R(N, M_0) | M(p_{12}) = M(p_{22}) = 1\}$. Clearly, as t_{15} is disabled at these markings, these markings can reach the markings of $B(\{p_{13}, p_{22}\})$ once t_4 is enabled. Thus, $B(\{p_{12}, p_{22}\})$ and p_{12} should be added to Ψ and Φ, respectively. Similarly, $B(\{p_{11}, p_{22}\}) = \{M \in R(N, M_0) | M(p_{11}) = M(p_{22}) = 1\}$ should be added to Ψ, and p_{11} should be added to Φ. As for p_{15}, there is only one place in p_{15} and p_{22} can be marked since they both require resource P_{R3}. Thus, remove p_{15} from $P_u(\Omega)$.

Now, we have $\Phi = \{p_{11}, p_{12}, p_{13}, p_{14}, p_{22}\}$ and $P_u(\Omega) = \{p_{21}, p_{23}\}$. Consider the markings of $B_2(p_{21} \cup \{\Phi \backslash p_{22}\})$, i.e., consider the following markings:

$B(\{p_{14}, p_{21}\}) = \{M \in R(N, M_0) | M(p_{14}) = M(p_{21}) = 1\}$,
$B(\{p_{13}, p_{21}\}) = \{M \in R(N, M_0) | M(p_{13}) = M(p_{21}) = 1\}$,
$B(\{p_{12}, p_{21}\}) = \{M \in R(N, M_0) | M(p_{12}) = M(p_{21}) = 1\}$,
$B(\{p_{11}, p_{21}\}) = \{M \in R(N, M_0) | M(p_{11}) = M(p_{21}) = 1\}$.

Obviously, all the above markings can reach markings of Ψ via uncontrollable transitions. Hence, p_{21} is added to Φ and $B_2(p_{21} \cup \Phi \backslash \{p_{22}\})$ is added to Ψ.

Consider place p_{23}. Only one of the places $p_{11}, p_{12}, p_{13}, p_{14}, p_{15}$ and p_{23} can be marked at any time since they all use the same resource P_{R1}. Hence, they cannot be marked at the same time. p_{23} is thus not added to Φ. As a result, $P_u(\Omega) = \emptyset$ and Function *FindBMs* terminates. Φ is updated to $\Phi = \{p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}\}$. As a result, we have a set of bad markings where $B_2(\Phi) = B(\Omega) \cup B(\{p_{11}, p_{22}\}) \cup B(\{p_{12}, p_{22}\}) \cup B(\{p_{13}, p_{22}\}) \cup B(\{p_{11}, p_{21}\}) \cup B(\{p_{12}, p_{21}\}) \cup B(\{p_{13}, p_{21}\}) \cup B(\{p_{14}, p_{21}\})$.

The method of finding bad markings based on Function *FindBMs* is simple but may be inefficient when the places of Ω refer to many process subnets. Given a set $\Omega \in C(S)$ and let $\Phi = \Omega$, $B(\Phi)$ is a set of bad markings. Let $p \in \Phi$, we want to determine whether the markings of $B(\{^{\bullet\bullet}p\} \cup (\Phi \backslash p))$ are also bad. It often requires enumerating all or parts of the reachable markings that reduces the computational efficiency. To solve this problem, the emptiable siphon considered in our work is

minimal and contains the smallest number of resource places and the initial marking of each idle place is equal to one. Due to the structural characteristics of the considered Gadara net, the efficiency of computing bad markings can be enhanced.

IV. ITERATIVE DEADLOCK PREVENTION POLICY

A. COMPUTATION OF ALL MRNES

In our work, the set of all MRNEs in a Gadara net is required to be computed so that bad markings can be computed by *FindBMs*. It is not the focus of our work. Thus, we briefly introduce a method to look for all MRNEs of a Gadara net. The mixed integer programming (MIP) formulation employed in the following function is proposed in [32] called MMIP-1. MRESs in a Gadara net can be computed by MMIP-1. More details are referred to [32].

First, MMIP-1 is employed to compute an MRES in a Gadara net. If there is an MRES S in the net, S is added to Π and let $n = |S \cap P_R|$. Monitor place is designed based on the method in [32] to prevent S from being empty. After obtaining a controlled net, two constraints are added to MMIP-1: 1) the objective siphon can only possess n resource places; 2) for any objective siphon S', $S' \cap V_S = \emptyset$, where V_S is the set of monitor places computed in Function *ComputeMRESs*. The new MIP formulation is denoted as MIP*. Then, MIP* is used to detect MRES in the controlled net. Once an MRES is detected, the siphon will be added to Π and then be controlled. The controlled net is input to MIP* again to detect MRES and the above steps are repeated. The function terminates when there is no feasible solution for MIP*.

Proposition 3: Given a Gadara net (N_G, M_0), $\Pi = ComputeMRESs(N_G, M_0)$ is the set of all MRESs of N_G.

Proof: Let S be an MRES detected by MMIP-1 in Gadara net N_G. Let $n = |S \cap P_R|$ and $V_S = \{p_c\}$, where p_c is the monitor place to prevent S from being empty. Two constraints are added to MMIP-1 to obtain MIP*, where $\Sigma_{r \in PR}\ (1\text{-}v_r) = n$ and $\forall p \in V_S, v_p = 1$. It implies that for an MRNEs S' detected by MIP*, $|S' \cap P_R| = n$ and $|S' \cap V_S| = \emptyset$. MIP* only computes MRESs of the original Gadara net. Thus, Π is a set of MRESs.

Then, we prove that Π is the set of all MRESs of N_G. Π is the output of Function *ComputeMRESs* when the function terminates. By contradiction, suppose that Π is not the set of all MRESs of N_G. It means that there is an MRES S'' such that $S'' \notin \Pi$. According to Function *ComputeMRESs*, S'' will be added to Π and then be controlled, which contradicts that Function *ComputeMRESs* terminates. Therefore, Π is the set of all MRESs of N_G. ♣

The Gadara net (N_G, M_0) in Fig. 5 is used to illustrate Function *ComputeMRESs*. Let $(N_{G1}, M_{01}) = (N_G, M_0)$. First, an MRES is detected by MMIP-1, which is $S_1 = \{p_5, p_{10}, R_3, R_4\}$. Add S_1 to the set Π and let $n = |S_1 \cap P_R| = 2$. Next, a monitor pc_1 is designed to prevent S_1 from being empty according to [32], i.e., the monitor is computed for $\sum_{p \in [S]} M(p) \leq M_0(S) - 1$ based on SBPI. pc_1 is added to V_S and N_{G1}. Subsequently, two constraints are added to MMIP-1: $\Sigma_{r \in PR}\ (1\text{-}v_r) = 2$ and $\forall p \in V_S, v_p = 1$. The new MIP formulation is called MIP*. Clearly, any MRES in N_G detected by MIP* can only possess 2 resource places. Then, $S_2 = \{p_6, p_9, R_4, R_5\}$ is detected by MIP* and p_{c2} is added to N_{G1} to prevent S_2 from being empty. Π and V_S are updated to $\Pi = \{S_1, S_2\}$ and $V_S = \{pc_1, pc_2\}$, respectively. $S_3 = \{p_4, p_{11}, R_2, R_3\}$ is detected by MIP* in N_{G1} augmented with pc_1 and pc_2. N_{G1} is updated by augmenting with pc_3. Thus, $\Pi = \{S_1, S_2, S_3\}$ and $V_S = \{pc_1, pc_2, pc_3\}$. Similarly, $S_4 = \{p_3, p_{12}, R_1, R_2\}$ is detected in the updated net and we have $\Pi = \{S_1, S_2, S_3, S_4\}$, $V_S = \{pc_1, pc_2, pc_3, pc_4\}$, where pc_4 is designed to control S_4. There is no feasible solution for MIP* after updating N_{G1} with pc_4. Hence, *ComputeMRESs* terminates. As a result, the set of all MRESs in the Gadara net of Fig. 5 is $\Pi = \{S_1, S_2, S_3, S_4\}$.

Function $\Pi = ComputeMRESs(N_G, M_0)$

Input: A Gadara net (N_G, M_0).
Output: The set of all MRESs Π.
1. Let $(N_{G1}, M_{01}) := (N_G, M_0)$ and $\Pi := \emptyset$;
2. **if** there is an MRES S in (N_{G1}, M_{01}) computed by the MIP formulation in [32] **do**
3. $\Pi := \Pi \cup S$;
4. let $n := |S \cap P_R|$;
5. add a monitor p_c to (N_{G1}, M_{01}) to prevent S from being empty according to [32];
6. the resultant net is denoted as (N_{G1}, M_{01});
7. $V_S := V_S \cup \{p_c\}$;
8. add two constraints to the MIP formulation in [32]: $\Sigma_{r \in P_R}\ (1\text{-}v_r) = n$ and $\forall p \in V_S, v_p = 1$, the new MIP formulation is denoted as MIP*; /* In the MIP formulation of [32], $v_p = 1 \Rightarrow p \notin S$, $v_p = 0 \Rightarrow p \in S$. */
9. **while** (there exists an MRES S' in (N_{G1}, M_{01}) computed by MIP* and $S' \notin \Pi$) **do**
10. $\Pi := \Pi \cup S'$;
11. add a monitor p_c to (N_{G1}, M_{01}) to prevent S' from being empty according to [32];
12. the resultant net is denoted as (N_{G1}, M_{01});
13. $V_S := V_S \cup \{p_c\}$;
14. **end while**
15. **end if**
16. **end**

B. DEADLOCK PREVENTION POLICY

In this section, we propose an iterative deadlock prevention policy to obtain a live Gadara net.

Definition 12: Let N_G be a Gadara net and S be an emptiable siphon of N_G. Let $\Omega_i \subseteq C(S)$, Ω_i is called the *redundant set* of $C(S)$ if $\exists \Omega_j \subseteq C(S), \forall p \in \Omega_i, p \in \Omega_j$ or $p \in P_u(\Omega_j)$, where $i \neq j$.

Definition 13: Let N_G be a Gadara net and S be an emptiable siphon of N_G. $C(S)_{max}$ is defined as $C(S)_{max} \subseteq C(S)$ and there is no redundant set in $C(S)_{max}$.

Suppose that $C(S) = \{\{p_1, p_2\},\{p_1, p_3\}\}$ where $p_1 \in N_i, p_2, p_3 \in N_j$ and $p_2 \in^{\bullet\bullet} p_3, i \neq j$. According to Definition 12, we know $\{p_1, p_2\}$ is the redundant set of $C(S)$. Hence, $C(S)_{max} = \{\{p_1, p_3\}\}$. Let $\Omega_1 = \{p_1, p_2\}$ and $\Omega_2 = \{p_1, p_3\}$, *FindBMs* is called to compute bad markings that can reach $B(\Omega_1)$ and $B(\Omega_2)$. Assume that the results are $B(\Phi_1)$ and $B(\Phi_2)$, respectively. As Function *FindBMs* only considers the upstream places of Ω_1, Ω_2 and $P_u(\Omega_1) \subseteq P_u(\Omega_2)$, we have $B(\Phi_1) \subseteq B(\Phi_2)$ if $\Phi_1 \subseteq \Phi_2$. Then, two monitors C_1 and C_2 are designed to forbid the markings of $B(\Phi_1)$ and $B(\Phi_2)$, respectively. If $B(\Phi_1) \subseteq B(\Phi_2)$, once the markings of $B(\Phi_2)$ are forbidden by C_2, the markings of $B(\Phi_1)$ are also prevented from being reachable without adding monitor place C_1. Thus, C_1 is a redundant monitor.

Based on the above analysis, the redundant set of $C(S)$ may result in redundant monitors. Therefore, the proposed deadlock prevention policy in the following only considers $C(S)_{max}$.

Algorithm 1 An Iterative Deadlock Prevention Policy

Input: A Gadara net (N_G, M_0)
Output: A live controlled Gadara net (N_G^c, M_0^c).
1 Let $(N_G^c, M_0^c) = (N_G, M_0)$.
2 **while** $(\Pi = ComputeMRESs(N_G^c, M_0^c) \neq \emptyset)$ **do**
3 let $S \in \Pi$;
4 compute the complementary set $[S]$ of S based on Definition 4;
5 compute $C(S)_{max}$ based on Definition 13;
6 **for** each $\Omega \in C(S)_{max}$ **do**
7 $\Phi := FindBMs(\Omega, \Pi)$;
8 construct a constraint as $\sum_{p\in\Phi} M(p) \leq |\Omega|$-1 and add a monitor place p_c to (N_G^c, M_0^c) based on SBPI;
9 **end for**
10 the resultant net is denoted by (N', M_0');
11 let $(N_G^c, M_0^c) := (N', M_0')$;
12 **end while**
13 **end**

Algorithm 1 works as follows. Steps 2 to 12 represent the iteration cycle. At each iteration, if $\Pi = ComputeMRESs(N_G^c, M_0^c) \neq \emptyset$, let $S \in \Pi$, steps 4 to 5 compute the complementary set $[S]$ and the set $C(S)_{max}$ for an MRES S of Π. For each set $\Omega \in C(S)_{max}$, let $\Phi = FindBMs(\Omega, \Pi)$ and $B_{|\Omega|}(\Phi)$ be a set of bad markings that can empty S. Thus, Step 8 constructs the constraint $\sum_{p\in\Phi} M(p) \leq |\Omega|$-1 and SBPI is employed to compute the monitor places. Every element of $C(S)_{max}$ is taken into account so that a large amount of bad markings that can empty the detected siphon are removed. Finally, the added monitor places are considered as resource places and the controlled net is input to *ComputeMRESs* to compute all MRESs. The algorithm executes until there is no MRES in the Gadara net. Consequently, all emptiable siphons are prevented from being emptied.

Theorem 3: Let (N_G^c, M_0^c) be the net obtained from (N_G, M_0) by Algorithm 1. (N_G^c, M_0^c) is ordinary.

Proof: The constraint established in Algorithm 1 have the form as $\sum_{p\in\Phi} M(p) \leq |\Omega| - 1$. We can rewrite it as

$$\sum_{i=1}^{n} l_i M(p_i) \leq |\Omega| - 1 \tag{3}$$

where n is the number of places in the Gadara net. Let $L = [l_1, l_2, \ldots, l_n]$, (2) is equivalent to $LM(p) \leq |\Omega|$-1, where $L(p_i) = 1$ if $p_i \in \Phi$, otherwise, $L(p_i) = 0$. According to SBPI, the monitor place is computed as $[N_c] = -L\cdot[N_G] = -L_\Phi\cdot[N_G]_\Phi$, where L_Φ is the set of non-zero components of L and $[N_G]_\Phi$ is a part of the incidence matrix $[N_G]$ that corresponds to the places of Φ. The element of $[N_G]_\Phi$ can only be 0, 1 and -1 since all the places of Φ are operation places. Furthermore, the definition of Gadara net shows that each subnet is a strongly connected state machine. Hence, there is at most one output operation place and at most one input operation place for each transition. Thus, there is at most one 1 and one -1 in each column of $[N_G]_\Phi$ while other elements are 0. Moreover, all the components of L_Φ are 1. As a result, the element of $[N_c]$ can only be 0, 1 and -1, i.e., the weights of the arcs associated with the monitor places are all one. Therefore, the controlled Gadara net is still ordinary. ♣

Theorem 4: Given a Gadara net (N_G, M_0) where $\forall p \in P_0, M_0(p_0) = 1$, and the net (N_G^c, M_0^c) obtained from (N_G, M_0) by Algorithm 1. (N_G^c, M_0^c) is live and maximally permissive.

Proof: At each iteration, if $\Pi = ComputeMRESs(N_G^c, M_0^c) \neq \emptyset$, it implies that there exists an MRES S, the algorithm will compute the bad markings when S is empty and all the bad markings that can eventually empty S. Monitor places will be designed to forbid all the bad markings and deadlock markings that can empty S. Therefore, S can be prevented from being empty under the control logic. Algorithm 1 is carried out if $\Pi = ComputeMRESs(N_G^c, M_0^c) = \emptyset$, i.e., there is no MRES in the net. Theorem 3 shows that the controlled net is still ordinary. Hence, the liveness of the controlled is related to emptiable siphon only. According to Theorem 1, the final controlled Gadara net (N_G^c, M_0^c) is live since it has no any emptiable siphon.

As for the maximal permissiveness, at each iteration, the proposed method will establish a constraint based on the output of Function *FindBMs*. For an MRES $S \in \Pi$, $\Phi = FindBMs(\Omega, \Pi)$ where $\Omega \in C(S)$. Theorem 2 shows that $B|_\Omega|(\Phi)$ is a set of bad markings. The constraint established in step 8 only forbids bad markings of $B_{|\Omega|}(\Phi)$. After Algorithm 1 terminate, all bad markings that lead to emptiable siphons are forbidden and all live markings are reachable after the iterative process terminates. Thus, the final controlled Gadara net (N_G^c, M_0^c) obtained by Algorithm 1 is maximally permissive. ♣

Remark 1: Since Definition 5 shows that a marking is bad if there is no control logic to make it go back to the initial marking, the markings associated with uncontrollable transitions are considered as bad. Thus, the controlled net obtained by

Algorithm 1 can be regarded as maximally permissive even if the markings associated with branching transitions are forbidden. Moreover, the controlled net obtained by Algorithm 1 is admissible, i.e., the outgoing arcs of monitor places never connect to branching transitions.

Remark 2: The computational complexity of Algorithm 1 mainly depends on Function *FindBMs*. Hence, the computational complexity of Algorithm 1 is exponential since Function *FindBMs* requires the reachability analysis. Note that it is mentioned in Section III that the computational efficiency of *FindBMs* is improved since we add some constraints on the considered Gadara net. Thus, the computational efficiency of Algorithm 1 can also be improved.

Remark 3: As analyzed in Section III, the constraint (1) or (2) can only forbid a small number of bad markings, which may lead to the problem of adding multiple monitors to the controlled net. Thus, Algorithm 1 employs Function *FindBMs* to compute all the bad markings that can empty an MRES. Then, a monitor place is designed in Algorithm 1 to forbid as many bad markings as possible so as to simplify the control structure. Although Algorithm 1 cannot guarantee that the control structure is the simplest, it can obtain a supervisor with simpler structure compared with the existing methods.

The Gadara net model in Fig. 1 is used to demonstrate the proposed algorithm. The model corresponds to a deadlock bug in version 2.5.62 of the Linux kernel. This model is constructed in the paper [16], where $P_0 = \{p_{01}, p_{02}, p_{03}\}$, $P_A = \{p_{11}-p_{18}, p_{21}-p_{23}, p_{31}-p_{35}\}$, and $P_R = \{R_1, R_2, R_3\}$. There are 80 reachable markings in the net in which 2 markings are dead and 78 markings are live. However, 22 of 78 markings can reach two dead markings via uncontrollable transitions. Hence, these markings are considered as bad.

According to our iterative control scheme, the set of all MRESs is computed as $\Pi = \{S_1\}$, where

$$S_1 = \{p_{11}-p_{13}, p_{15}-p_{18}, p_{21}, p_{23}, p_{34}, R_1, R_3\}.$$

The complementary set of S_1 is $[S_1] = \{p_{14}, p_{22}\}$. Based on Definition 7 and Definition 13, we have

$$C(S_1)_{max} = C(S_1) = \{\{p_{14}, p_{22}\}\}.$$

Hence, let $\Omega = \{p_{14}, p_{22}\}$, Function *FindBMs* is called.

The result of Function *FindBMs* in this example is computed in Section III. Namely, $\Phi = FindBMs(\Omega, \Pi) = \{p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}\}$. Clearly, any two places in Φ cannot be marked at the same time, since otherwise siphon S_1 is empty. Hence, a constraint is constructed as

$$M(p_{11})+M(p_{12})+M(p_{13})+M(p_{14})+M(p_{21})+M(p_{22}) \leq 1.$$

Then, a monitor place p_{c1} can be computed on the basis of SBPI. As a result, we have $M_0(p_c1) = 1, {}^\bullet p_{c1} = \{t_2, t_6, t_{10}, t_{15}, t_{17}\}$ and $p_{c1}^\bullet = \{t_1, t_9, t_{13}, t_{16}\}$ after p_{c1} is added to the Gadara net. The controlled net is shown in Fig. 2 and we can see that the controlled net is admissible.

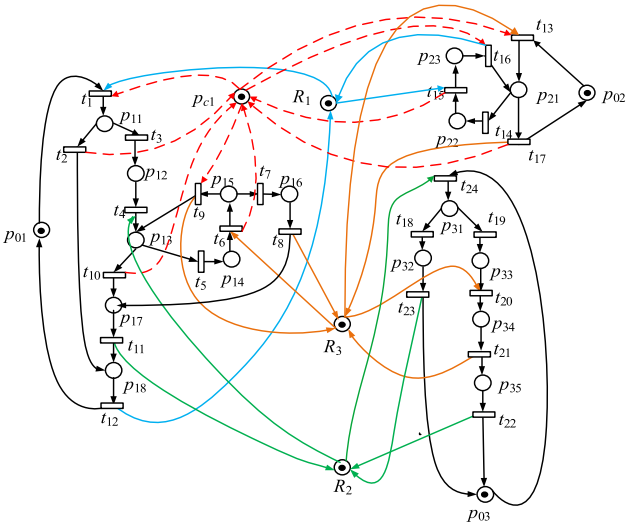


FIGURE 2. Controlled net of the gadara net shown in fig. 1 by algorithm 1.

At the second iteration, the added monitor place p_{c1} is regarded as a resource place and the controlled net in Fig. 2 is input to Function *ComputeMRESs*. As a result, $\Pi = \emptyset$. Hence, Algorithm 1 terminates and only one monitor place is needed to guarantee the liveness of the considered net. The resultant net is shown in Fig. 2. Adding the monitor place to the original net model, we obtain a live controlled Gadara net with 56 reachable markings. The obtained result proves that Algorithm 1 prevents all the deadlock markings and handles uncontrollable transitions in a minimal restrictive manner. Furthermore, all the incoming and outgoing arcs of the synthesized monitor place have unit arc weight, i.e., the resultant net remains ordinary.

Note that the proposed method is only applicable to the ordinary Gadara nets where $\forall p_0 \in P_0, M_0(p_0) = 1$, i.e., there is always one token in each process subnet so that the operation places of a same subnet cannot be marked at the same time. Hence, the constraints involved in Algorithm 1 only prevent the deadlock markings and bad markings but do not prevent any live marking. Unfortunately, the method may fail to guarantee the maximal permissiveness if $\exists p_0 \in P_0, M_0(p_0) > 1$. In this case, the constraints in the method may prevent live markings since the operation places of a same subnet can be marked at the same time.

V. EXAMPLES

This section further shows the performance of the proposed policy by applying it to two Gadara nets in the literature.

The Gadara net model of a multithreaded software is shown in Fig. 3, which has been studied in several papers (see [15]–[17]). The places of Fig. 3 can be divided as: $P_0 = \{p_1, p_7\}$, $P_R = \{R_1 - R_3\}$, and $P_A = \{p_2 - p_6, p_8 - p_{12}\}$. The net has 16 reachable markings in which two markings are dead and 11 markings are live. Our deadlock prevention policy is applied to prevent the two dead markings from being reachable. At the first iteration, the set of all MRESs is

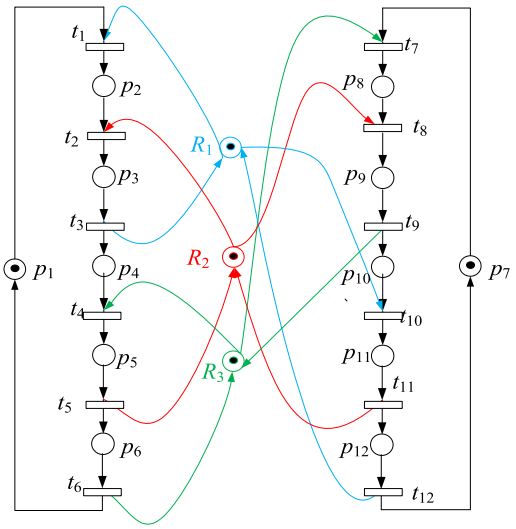


FIGURE 3. Gadara net model in [15].

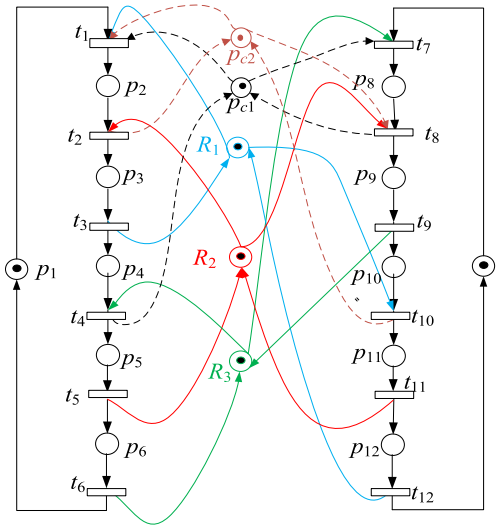


FIGURE 4. Controlled net of the gadara net in fig. 3.

$\Pi_1 = \{S_1, S_2\}$, where $S_1 = \{p_5, p_6, p_9, p_{10}, p_{11}, R_2, R_3\}$ and $S_2 = \{p_3, p_4, p_5, p_{11}, p_{12}, R_1, R_2\}$. The complementary set of S_1 is $[S_1] = \{p_3, p_4, p_8\}$ and $C(S_1)_{max} = \{\{p_4, p_8\}\}$.Thus, let $\Omega_1 = \{p_4, p_8\}$, Function *FindBMs* is applied to Ω_1 and Π_1. As a result, we have $\Phi_1 = FindBMs(\Omega_1, \Pi_1) = \{p_2, p_3, p_4, p_8\}$. Since $|\Omega_1| = 2$, a constraint is constructed as

$$M(p_2) + M(p_3) + M(p_4) + M(p_8) \leq 1.$$

Thus, a monitor place p_{c1} is designed to forbid these bad markings. It is $M_0(p_{c1}) = 1, {}^\bullet p_{c1} = \{t_4, t_8\}$ and $p_{c1}^\bullet = \{t_1, t_7\}$. The resultant net is shown in Fig. 4.

At the second iteration, $\Pi_1 = \{S_2\}$ where $S_2 = \{p_3, p_4, p_5, p_{11}, p_{12}, R_1, R_2\}$. Similar to the first iteration, Function *FindBMs* is applied to Ω_2 and Π_2, where $\Omega_2 = \{p_2, p_{10}\}$. The result of Function *FindBMs* is $\Phi_2 = \{p_2, p_9, p_{10}\}$. By our deadlock prevention policy, a monitor place p_{c2} is designed for

$$M(p_2) + M(p_9) + M(p_{10}) \leq 1.$$

As a result, we have $M_0(p_{c2}) = 1, {}^\bullet p_{c2} = \{t_2, t_{10}\}$ and $p_{c2}^\bullet = \{t_1, t_8\}$. The resultant net is shown in Fig. 4.

At the third iteration, the net augmented with p_{c1} and p_{c2} is input to Function *ComputeMRESs* and $\Pi = \emptyset$.Thus, Algorithm 1 terminates. In the resultant net, there are two monitor places with eight arcs. After adding the two monitor places to the original Gadara net, the controlled net is live with 11 markings. It shows that the proposed method guarantees the liveness of Gadara nets with maximal permissiveness.

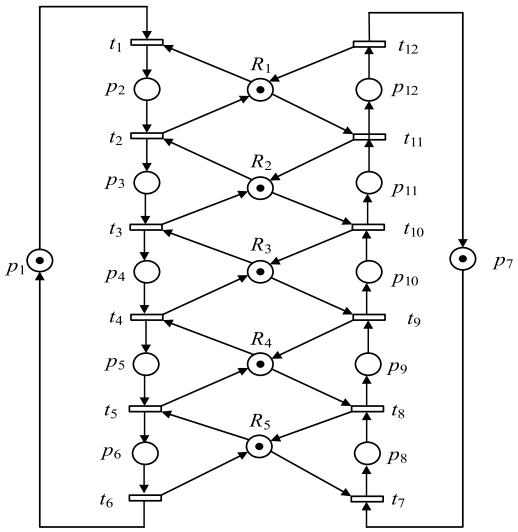


FIGURE 5. A Gadara net.

Next, we consider the Gadara net shown in Fig. 5, where $P_0 = \{p_1, p_7\}, P_R = \{R_1 - R_5\}$, and $P_A = \{p_2 - p_6, p_8 - p_{12}\}$. There are 21 reachable markings in the net with 11 live markings and four dead markings, respectively. By using the deadlock prevention policy, four dead markings and six bad markings can be forbidden. The application of our deadlock prevention policy is shown in Table 1, where the first column is the iteration number, the second is the output of Function *FindBMs*, the third column shows the constraints at each iteration, $M_0(p_{ci0}), {}^\bullet p_{ci}$, and $p_{ci}^\bullet$ shown in the fourth column to sixth column are the initial marking, preset and postset of the computed monitor place p_{ci}, respectively, and the last four columns are the numbers of reachable markings, bad markings, dead markings and live markings, respectively.

For this example, there are four iterations and four constraints are constructed by the algorithm and each of them corresponds to a monitor place. After adding these four monitor places to the original Gadara net, the controlled net is live with 11 live markings. Furthermore, the results listed in the last four columns of the table show that the final controlled net is maximally permissive. It demonstrates that the proposed method is maximally permissive since the added monitor places prevent all the deadlock markings and bad markings while forbid none of live markings.

TABLE 1. Monitor places computed for the net in Fig. 5 by Algorithm 1.

Iterations	Φ=FindBMs(Ω, Π)	Constraints	$M_0(p_{ci0})$	${}^\bullet p_{ci}$	$p_{ci}^\bullet$	RM	BM	DM	LM
1	$\Phi_1=\{p_2, p_3, p_4, p_8, p_9\}$	$M(p_2)+M(p_3)+M(p_4)+M(p_8)+M(p_9)\leq 1$	1	t_4, t_9	t_1, t_7	15	1	3	11
2	$\Phi_2=\{p_5, p_8\}$	$M(p_5)+M(p_8)\leq 1$	1	t_5, t_8	t_4, t_7	14	1	2	11
3	$\Phi_3=\{p_2, p_3, p_{10}\}$	$M(p_2)+M(p_3)+M(p_{10})\leq 1$	1	t_3, t_{10}	t_1, t_9	12	1	1	11
4	$\Phi_4=\{p_2, p_{11}\}$	$M(p_2)+M(p_{11})\leq 1$	1	t_2, t_{11}	t_1, t_{10}	11	0	0	11

TABLE 2. Monitor places computed for the net in Fig. 3 by ICOG-O.

Iterations	Constraints	$M_0(p_{ci0})$	${}^\bullet p_{ci}$	$p_{ci}^\bullet$	RM	BM	DM	LM
1	$M(p_4)+M(p_8)\leq 1$	1	t_4, t_8	t_3, t_7	15	2	2	11
2	$M(p_3)+M(p_8)\leq 1$	1	t_3, t_8	t_2, t_7	14	2	1	11
3	$M(p_2)+M(p_{10})\leq 1$	1	t_2, t_{10}	t_1, t_9	13	1	1	11
4	$M(p_2)+M(p_9)\leq 1$	1	t_2, t_9	t_1, t_8	12	0	1	11
5	$M(p_2)+M(p_8)\leq 1$	1	t_2, t_8	t_1, t_7	11	0	0	11

TABLE 3. Monitor places computed for the net in Fig.5 by ICOG-O.

Iterations	Constraints	$M_0(p_{ci0})$	${}^\bullet p_{ci}$	$p_{ci}^\bullet$	RM	BM	DM	LM
1	$M(p_4)+M(p_9)\leq 1$	1	t_4, t_9	t_3, t_8	20	6	3	11
2	$M(p_3)+M(p_{10})\leq 1$	1	t_3, t_{10}	t_2, t_9	19	5	3	11
3	$M(p_3)+M(p_9)\leq 1$	1	t_3, t_9	t_2, t_8	18	5	2	11
4	$M(p_5)+M(p_8)\leq 1$	1	t_5, t_8	t_4, t_7	18	4	2	11
5	$M(p_2)+M(p_{11})\leq 1$	1	t_2, t_{11}	t_1, t_{10}	16	3	2	11
6	$M(p_2)+M(p_{10})\leq 1$	1	t_2, t_{10}	t_1, t_9	15	2	2	11
7	$M(p_2)+M(p_9)\leq 1$	1	t_2, t_9	t_1, t_8	14	2	1	11
8	$M(p_4)+M(p_8)\leq 1$	1	t_4, t_8	t_7, t_7	13	1	1	11
9	$M(p_3)+M(p_8)\leq 1$	1	t_3, t_8	t_2, t_7	12	0	1	11
10	$M(p_2)+M(p_8)\leq 1$	1	t_2, t_8	t_1, t_7	11	0	0	11

In order to provide a comparison between our deadlock prevention policy and other approach, the ICOG-O method [17] is applied to the two examples. The application of ICOG-O in the two examples is shown in Table 2 and Table 3, respectively.

For the first example, it requires five iterations to terminate the ICOG-O algorithm. Accordingly, five constraints are constructed by the algorithm. The monitor places corresponding to the five constraints are shown in Table 2. It can be seen that the ICOG-O is maximally permissive and it requires five monitor places to guarantee the liveness of the net. In contrast, by using our method, we find a simpler supervisor that can keep the Gadara net live with only two monitor places. As for the second example, ten monitor places are designed by ICOG-O to forbid deadlock markings and bad markings in the net. We notice that three of the ten monitor places are redundant, which are $p_{c18} - p_{c20}$. Compared with the results in Table 1, the structure of the controlled net obtained by our method is simpler. It only requires four monitor places to forbid deadlock markings and bad markings and none of them is redundant. From the two examples, we have the following conclusions:

1) As behavior permissiveness is concerned, both of our method and ICOG-O can obtain an optimal supervisor since they prevent the bad and deadlock markings only while do not forbid any live markings.

2) From the point of view of structural complexity, the number of monitor places synthesized by our method is rather fewer than ICOG-O. Furthermore, there is no redundant monitor place in the controlled nets obtained by our method. Besides, it also implies that the monitors synthesized by our method are more effective since each of them can prevent more bad and deadlock markings.

3) Consider the computational efficiency, our method is of exponential complexity while ICOG-O is NP-hard. However, it is worth nothing that our method does not require computing the covering and there are obviously fewer iterations in our method.

VI. CONCLUSION

In this paper, the problem of deadlock prevention in a class of ordinary Gadara nets is studied. An iterative control scheme based on siphons is proposed to obtain a maximally permissive supervisor with a small number of monitor places.

Moreover, we provide a method to compute bad markings on the basis of siphons. All bad markings that lead to a minimal emptiable siphon are computed and forbidden at each iteration so that a live controlled Gadara net is finally obtained. The experimental results show that the proposed method has the following advantages: first, the controlled Gadara net obtained by the proposed algorithm is maximally permissive with a simple control structure; second, although the proposed algorithm is iterative, it requires only a few iterations to terminate; third, the final controlled net is still ordinary.

The computational complexity of the proposed approach is in theory exponential since it requires enumerating all or part of reachable markings. Another limitation is that the approach is applicable to a class of ordinary Gadara nets only, where each idle place contains exactly one token at the initial marking. Therefore, our future work aims to avoid enumerating reachable markings and extend our method to a broader class of Gadara nets.

REFERENCES

[1] J. Ezpeleta, F. Tricas, F. Garcia-Valles, and J. Colom, "A Banker's solution for deadlock avoidance in FMS with flexible routing and multiresource states," *IEEE Trans. Robot. Autom.*, vol. 18, no. 4, pp. 621–625, Aug. 2002.

[2] E. W. Dijkstra, "The mathematics behind the Banker's algorithm," in *Selected Writings on Computing: A personal Perspective*. New York, NY, USA: Springer, 1982, pp. 308–312.

[3] T. Murata, B. Shenker, and S. Shatz, "Detection of ada static deadlocks using Petri net invariants," *IEEE Trans. Softw. Eng.*, vol. 15, no. 3, pp. 314–326, Mar. 1989.

[4] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[5] M. Zhou and F. Dicesare, "Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources," *IEEE Trans. Robot. Autom.*, vol. 7, no. 4, pp. 515–527, Aug. 1991.

[6] K. Barkaoui and J.-F. Pradat-Peyre, "Verification in concurrent programming with Petri nets structural techniques," in *Proc. 3rd IEEE Int. High-Assurance Syst. Eng. Symp.*, Nov. 1998, pp. 124–133.

[7] G. Liu, "Complexity of the deadlock problem for Petri nets modeling resource allocation systems," *Inf. Sci.*, vol. 363, pp. 190–197, Oct. 2016.

[8] G. J. Liu, C. Jiang, and M. Zhou, "Two simple deadlock prevention policies for S^3PR based on key-resource/operation-place Pairs," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 4, pp. 945–957, Oct. 2010.

[9] K. M. Kavi, A. Moshtaghi, and D.-J. Chen, "Modeling multithreaded applications using Petri nets," *Int. J. Parallel Program.*, vol. 30, no. 5, pp. 353–371, Oct. 2002.

[10] J. Cheng, "Run-time detection of tasking deadlocks in real-time systems with the Ada 95 annex of real-time systems," in *Proc. 11th Int. Conf. Reliable Softw. Technol.*, Porto, Portugal, Jun. 2006, pp. 167–178.

[11] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: Treating bugs as allergies—A safe method to survive software failures," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 235–248, 2005.

[12] 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. 48h IEEE Conf. Decision Control (CDC) Held Jointly 28th Chin. Control Conf.*, Shanghai, China, Dec. 2009, pp. 4971–4976.

[13] Y. Wang, "Software failure avoidance using discrete control theory," Elect. Eng. Comput. Sci. Dept., Univ. Michigan, Ann Arbor, MI, USA, Tech. Rep., 2009.

[14] H. Liao, H. Zhou, and S. Lafortune, "Simulation analysis of multithreaded programs under deadlock-avoidance control," in *Proc. Winter Simul. Conf. (WSC)*, Phoenix, AZ, USA, Dec. 2011, pp. 703–715.

[15] 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," *Discrete Event Dyn. Syst.*, vol. 23, no. 2, pp. 157–195, Jun. 2013.

[16] H. Liao, S. Lafortune, S. Reveliotis, Y. Wang, and S. Mahlke, "Optimal liveness-enforcing control for a class of Petri nets arising in multithreaded software," *IEEE Trans. Automat. Contr.*, vol. 58, no. 5, pp. 1123–1138, May 2013.

[17] H. Liao, Y. Wang, J. Stanley, S. Lafortune, S. Reveliotis, T. Kelly, and S. Mahlke, "Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 6, pp. 2067–2082, Nov. 2013.

[18] J. Stanley, H. Liao, and S. Lafortune, "SAT-based control of concurrent software for deadlock avoidance," *IEEE Trans. Automat. Contr.*, vol. 60, no. 12, pp. 3269–3274, Dec. 2015.

[19] P. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–89, Jan. 1989.

[20] A. Giua, "Petri nets as discrete event models for supervisory control," Ph.D. dissertation, Rensselaer Polytech. Inst., Troy, NY, USA, 1992.

[21] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of Petri nets based on place invariants," *Automatica*, vol. 32, no. 1, pp. 15–28, Jan. 1996.

[22] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39974–39982, 2019.

[23] Y. Chen, Z. Li, and M. Zhou, "Behaviorally optimal and structurally simple liveness-enforcing supervisors of flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 3, pp. 615–629, May 2012.

[24] H. Hu, Y. Liu, and L. Yuan, "Supervisor simplification in FMSs: Comparative studies and new results using Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 1, pp. 81–95, Jan. 2016.

[25] B. Huang, M. Zhou, Y. Huang, and Y. Yang, "Supervisor synthesis for FMS based on critical activity places," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 5, pp. 881–890, May 2019.

[26] Q. Zhuang, W. Dai, S. Wang, and F. Ning, "Deadlock prevention policy for S^4PR nets based on siphon," *IEEE Access*, vol. 6, pp. 50648–50658, 2018.

[27] S. Wang, C. Wang, and Y. Yu, "Design of liveness-enforcing supervisors for S^3PR based on complementary places," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1, pp. 1–18, Jan. 2013.

[28] S. Wang, D. You, and C. Seatzu, "A novel approach for constraint transformation in Petri nets with uncontrollable transitions," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 8, pp. 1403–1410, Aug. 2018.

[29] S. Wang, C. Wang, M. Zhou, and Z. Li, "A method to compute strict minimal siphons in a class of Petri nets based on loop resource subsets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 1, pp. 226–237, Jan. 2012.

[30] S. Afshar, N. Khalilzad, F. Nemati, and T. Nolte, "Resource sharing among prioritized real-time applications on multiprocessors," *SIGBED Rev.*, vol. 12, no. 1, pp. 46–55, Mar. 2015.

[31] A. Giri and P. Patil, "Design of a parallel multi-threaded programming model for multicore architecture with resource sharing," in *Proc. NCRIET*, 2015, pp. 1–4.

[32] X. Guo, S. Wang, D. You, Z. Li, and X. Jiang, "A siphon-based deadlock prevention strategy for S^3PR," *IEEE Access*, vol. 7, pp. 86863–86873, 2019.

[33] H. Dou, K. Barkaoui, H. Boucheneb, X. Jiang, and S. Wang, "Maximal good step graph methods for reducing the generation of the state space," *IEEE Access*, vol. 7, pp. 155805–155817, 2019.

[34] S. Wang, D. You, and M. Zhou, "A necessary and sufficient condition for a resource subset to generate a strict minimal siphon in S^4PR," *IEEE Trans. Automat. Contr.*, vol. 62, no. 8, pp. 4173–4179, Aug. 2017.

[35] X. Xiao, W. Zheng, Y. Xia, X. Sun, Q. Peng, and Y. Guo, "A workload-aware VM consolidation method based on coalitional game for energy-saving in cloud," *IEEE Access*, vol. 7, pp. 80421–80430, 2019.

[36] W. Li, K. Liao, Q. He, and Y. Xia, "Performance-aware cost-effective resource provisioning for future grid IoT-cloud system," *J. Energy Eng.*, vol. 145, no. 5, Oct. 2019, Art. no. 04019016.

[37] G. Liu, C. Jiang, and M. Zhou, "Improved sufficient condition for the controllability of dependent siphons in system of simple sequential processes with resources," *IET Control Theory Appl.*, vol. 5, no. 9, pp. 1059–1068, Jun. 2011.

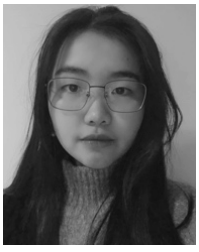
[38] L. Wang, Y. Du, and L. Qi, "Efficient deviation detection between a process model and event logs," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 6, pp. 1352–1364, Nov. 2019.

[39] N. Ran, J. Hao, S. Wang, Z. Dong, Z. He, Z. Liu, and Y. Ruan, "K-codiagnosability verification of labeled Petri nets," *IEEE Access*, to be published, doi: 10.1109/access.2019.2959904.

WENLI DUO received the B.S. degree from the School of Information and Electronic Electronic Engineering, Zhejiang Gongshang University, China, in 2017, where she is currently pursuing the M.S. degree with the School of Information and Electronic Engineering. Her main interests include supervisory control of discrete event systems, and Petri net theory and application.

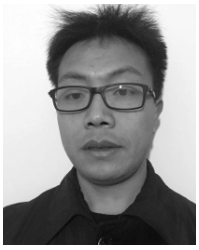
XIAONING JIANG received the M.E. degree in electronic engineering from Hangzhou Dianzi University, Hangzhou, China, in 1993, and the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2000. He is currently an Associate Professor and a Senior Engineer with Zhejiang Gongshang University and also the Vice Dean of the IOT Research Institute, Zhejiang Gongshang University. He has published more than 30 research articles and 10 invention patents. His research interests include applied information systems, network and information security, the industrial IOT, visual analytic, and Fin-tech.

OUSSAMA KAROUI received the B.S. and M.S. degrees in computer networks and telecommunications from the National Institute of Applied Sciences and Technology, Tunis, Tunisia, in 2012 and 2015, respectively, and the Ph.D. degree in computer technology and application from the Macau University of Science and Technology, Macau, China, in 2018. He joined Zhejiang Gongshang University, Hangzhou, China, in 2019, where he is currently a Professor with the School of Information and Electronic Engineering. His currently interests include modeling, control and scheduling of discrete event systems, Petri nets, automated manufacturing systems, the Internet of Things, mobile robotics, wireless ad hoc, and sensor networks.



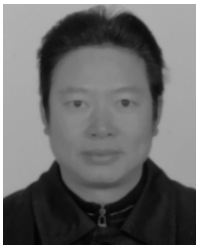
XIN GUO received the B.S. degree in measuring and control technology and instrumentations from Hangzhou Dianzi University, Hangzhou, China, in 2017. She is currently pursuing the M.S. degree with the School of Information and Electronic Engineering, Zhejiang Gongshang University. Her research interests include Petri net theory and application and supervisory control of discrete event systems.

DAN YOU (Student Member, IEEE) received the B.S. and M.S. degrees from the School of Information and Electronic Engineering, Zhejiang Gongshang University, China, in 2014 and 2017, respectively. Her research interests include supervisory control of discrete event systems, fault prediction, and deadlock control and siphon computation in Petri nets.



SHOUGUANG WANG (Senior Member, IEEE) received the B.S. degree in computer science from the Changsha University of Science and Technology, Changsha, China, in 2000, and the Ph.D. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2005.

He was a Visiting Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA, from 2011 to 2012. He was a Visiting Professor with the Electrical and Electronic Engineering Department, University of Cagliari, Cagliari, Italy, from 2014 to 2015. He was the Dean of the Department of Measuring and Control Technology and Instrument from 2011 to 2014. He joined Zhejiang Gongshang University, in 2005, where he is currently a Professor with the School of Information and Electronic Engineering, the Director of the Discrete-Event Systems Group, and the Dean of the System modeling and Control Research Institute, Zhejiang Gongshang University. He is currently an Associate Editor of IEEE Access and the IEEE/CAA Journal of Automatica Sinica.



YUAN RUAN received the B.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 1989, and the M.A. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 1992. He joined Zhejiang Gongshang University, in 1992, where he is currently an Associate Professor with the School of Information and Electronic Engineering. His current research interests include embedded systems.