**Noname manuscript No.**
(will be inserted by the editor)

# Concurrency Bugs in Multithreaded Software: Modeling and Analysis Using Petri Nets

**Hongwei Liao · Yin Wang · Hyoun Kyu Cho · Jason Stanley · Terence Kelly · Stéphane Lafortune · Scott Mahlke · Spyros Reveliotis**

Received: date / Accepted: date

**Abstract** In this paper, we apply discrete-event system techniques to model and analyze the execution of concurrent software. The problem of interest is deadlock avoidance in shared-memory multithreaded programs. We employ Petri nets to systematically model multithreaded programs with lock acquisition and release operations. We define a new class of Petri nets, called Gadara nets, that arises from this modeling process. We establish a set of important properties of Gadara nets, such as liveness, reversibility, and linear separability. We propose efficient algorithms for the verification of liveness of Gadara nets, and report experimental results on their performance. We also present modeling examples of real-world programs. The results in this paper lay the foundations for the development of effective control synthesis algorithms for Gadara nets.

**Keywords** Concurrent software · Deadlock analysis · Modeling · Petri nets

This work was partially supported by NSF grant CCF-0819882 and an award from HP Labs Innovation Research Program (University of Michigan) and by NSF grants CMMI-0619978 and CMMI-0928231 (Georgia Institute of Technology).

H. Liao, H. Cho, J. Stanley, S. Lafortune, and S. Mahlke
Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor, MI 48109, USA
E-mail: {hwliao, netforce, jasonsta, stephane, mahlke}@eecs.umich.edu

Y. Wang and T. Kelly
HP Labs, Palo Alto, CA 94303, USA
E-mail: {yin.wang, terence.p.kelly}@hp.com

S. Reveliotis
School of Industrial & Systems Engineering,
Georgia Institute of Technology, Atlanta, GA 30332, USA
E-mail: spyros@isye.gatech.edu

## 1 Introduction

In the past decade, computer hardware has undergone a true revolution, moving from uniprocessor architectures to multiprocessor architectures. In order to exploit the full potential of multicore hardware, there is an unprecedented interest in parallelizing computing tasks that were previously conducted in series. This trend forces parallel programming upon the average programmer. Parallel programming is fundamentally more challenging than serial programming because of the complexity of reasoning about concurrency. Lock primitives, such as *mutual exclusion locks (mutexes)*, are often employed to protect shared data and prevent data races. Inappropriate use of mutexes can lead to *circular-mutex-wait (CMW) deadlocks* in the program, where a set of threads are waiting indefinitely for one another and none of them can proceed. Significant effort has to be spent to detect and fix intricate deadlock bugs.

Development of highly reliable and robust software is a very active research area in the software and operating systems communities. Some recent work includes (Qin et al, 2005; Nir-Buchbinder et al, 2008; Novark et al, 2007, 2008; Musuvathi et al, 2008; Park et al, 2009). There is an emerging need for systematic methodologies that will enable programmers to characterize, analyze, and resolve software failures, such as deadlocks. Decades of study have yielded numerous approaches to deadlock, but none is a panacea. Static deadlock prevention via strict global lock-acquisition ordering is straightforward in principle but can be remarkably difficult to apply in practice. Static deadlock detection via program analysis has made impressive strides in recent years (Flanagan et al, 2002; Engler and Ashcraft, 2003), but spurious warnings can be numerous and the cost of manually repairing genuine deadlock bugs remains high. Dynamic deadlock detection may identify the problem too late, when recovery is awkward or impossible; automated rollback and re-execution as in (Qin et al, 2005) can help, but irrevocable actions such as I/O can preclude rollback. Variants of the Banker's Algorithm (Dijkstra, 1982) provide dynamic deadlock avoidance, but require more resource demand information than is often available and involve expensive runtime calculations.

Our on-going Gadara project (Kelly et al, 2009) is a multidisciplinary effort 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. In view of the event-driven nature of program dynamics and the logical control specification of deadlock avoidance, we approach this problem from a discrete-event systems (DES) angle (Cassandras and Lafortune, 2008). We build a formal model of the program, analyze its properties, and synthesize control logic to enforce deadlock freeness. The focus of this paper is on the first two steps: modeling and analysis. Our control synthesis results will be presented in subsequent papers.

Finite state automata and Petri nets are the two most popular modeling formalisms for DES. We chose Petri nets as the modeling formalism in the Gadara project, because they are efficient at capturing the concurrency of a dynamic system while avoiding enumerating its state space. Deadlock analysis based on Petri nets has been widely studied for flexible manufacturing systems and other technological applications involving a resource allocation function (Li et al, 2008; Reveliotis, 2005). Various special classes of Petri nets have been proposed to model and analyze manufacturing systems (Li et al, 2008). Recently, there has also been a growing interest in the application of DES to software systems and embedded systems; see, e.g., (Liu et al, 2006; Dragert et al, 2008; Auer et al, 2009; Gamatie et al, 2009; Iordache and Antsaklis, 2010; Delaval et al, 2010). A comprehensive review of the application of Petri nets to computer programming is presented in (Iordache and Antsaklis, 2009). Modeling thread creation/termination and mutex lock/unlock operations are in fact classical applications of Petri nets (Murata, 1989); in particular, Petri nets were used in (Murata et al, 1989) to analyze deadlocks in Ada programs. In the case of the popular Pthread library for C/C++ programs, Petri nets have also been employed to model multithreaded synchronization primitives (Kavi et al, 2002).

We discovered that the existing special classes of Petri nets that have been proposed in the literature do not exactly match the specific features of Petri nets that arise when modeling the locking behavior of multithreaded programs. Therefore, we propose a new class of Petri nets, called Gadara nets, that explicitly characterizes the models of multithreaded programs with lock acquisition and release operations. With the class of Gadara nets formally defined, we can efficiently analyze program deadlocks via formal models, and synthesize deadlock avoidance control policies that can in turn be instrumented in the underlying programs. By establishing a set of important properties of Gadara nets (e.g., liveness, reversibility, and linear separability), the deadlock-freeness – a *behavioral* property – of the program can be analyzed via the program's corresponding Gadara net model by exploiting the *structural* properties of the net. This correspondence is crucial to the effectiveness and efficiency of control synthesis of Gadara nets for the purpose of deadlock avoidance in the programs. By "effectiveness" we mean that the control logic synthesized from the Gadara net will *provably* avoid potential (CMW) deadlocks at run-time. By "efficiency" we mean that the run-time computational overhead is minimized and the control logic has the property of *maximal permissiveness* in the sense that it will restrict concurrency only if some future execution path will unavoidably lead to deadlock.

The main contributions of this paper are summarized as follows. (i) We formally define the classes of Gadara nets and controlled Gadara nets; the latter one is defined in anticipation of the effect of control on Gadara nets. (ii) We establish several important properties of Gadara nets, such as liveness, reversibility, and linear separability, which provide the necessary foundations for the future synthesis of maximally-permissive liveness-enforcing (MPLE) control policies for Gadara nets. (iii) We present efficient algorithms for the verification of liveness of Gadara nets using mathematical programming, and report experimental results on the performance of the algorithms. We use examples of deadlock from two real-world programs, BIND and the Linux kernel, to illustrate our results.

As a technical note, the two terminologies, "deadlock freeness" and "deadlock avoidance", have slightly different meanings in the software engineering and control engineering communities. We make the following remarks for the sake of clarity.

*Remark 1* Unless otherwise specified, the notion of "deadlock freeness" in the context of this paper refers to the case where a program is free from CMW deadlocks; in the Petri net literature, it usually refers to the case where at least one transition in the net is enabled (see Definition 12 in Section 4.1).

*Remark 2* The strategy in Gadara corresponds to what is termed "deadlock avoidance" in computer systems (Silberschatz et al, 2008); in the Petri net literature, such strategies are usually classified as "deadlock prevention" (Li et al, 2008).

This paper is organized as follows. Section 2 overviews the Gadara project and describes the modeling of multithreaded programs by Petri nets. In Section 3, we formally define Gadara nets and controlled Gadara nets. The liveness and reversibility properties of Gadara nets are established in Section 4; the algorithms for the verification of liveness of Gadara nets are presented in Section 5. In Section 6, we establish the linear separability of the safe region of Gadara nets. We present some examples of deadlocks from real-world software in Section 7, and finally conclude in Section 8. A preliminary version of some of the results in Sections 3, 4, and 6 appears in (Wang et al, 2009b); a preliminary version of some of the results in Sections 5.2 and 5.3 appears in (Liao et al, 2011).

## 2 Modeling of Multithreaded Software

### 2.1 Overview of the Gadara project

The architecture of Gadara is shown in Fig. 1 and comprises the following four steps.

1. The C program source code is converted into a Control Flow Graph (CFG) by compiler techniques. A CFG is a high-level graphical representation of all code execution paths that might be traversed by the program. The CFG is augmented with additional information about lock variables and lock functions. The enhanced CFG is a directed graph.
2. The enhanced CFG is translated into a Petri net model, namely, a Gadara net.
3. Based on the obtained Gadara net model of the program, the goal of deadlock freeness of the program is mapped to an appropriate *necessary and sufficient* condition that must be satisfied by the Gadara net model. Control synthesis is further carried out on the Gadara net to enforce this condition. The output of this step is a controlled Gadara net, augmented with

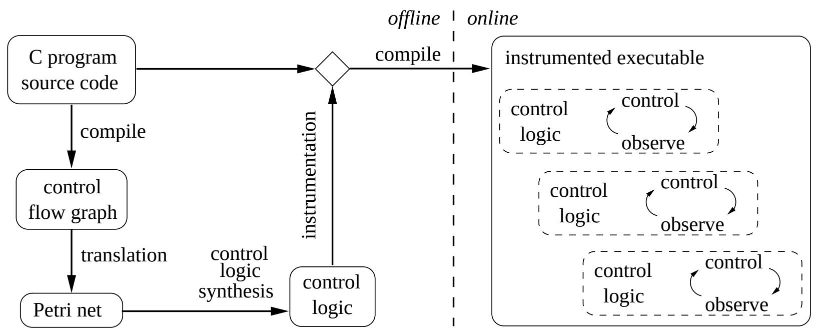


**Fig. 1** Gadara architecture

*monitor* (a.k.a. *control*) places, which corresponds to a deadlock-free program.

4. The synthesized control logic captured by the monitor places is incorporated into the program by instrumenting the original code.

The four steps described above are all conducted off-line. During program execution, the only on-line overhead is due to the additional lines of code pertaining to checking and updating the contents of the monitor places. In this paper, we focus on the formal DES aspects pertaining to Steps 2 and 3 (analysis part). The reader is referred to our earlier publications in computer science venues (Wang et al, 2008, 2009a) for more details on Steps 1 and 4. We start by discussing Step 2.

## 2.2 Modeling of software using Petri nets

Determining if a program is deadlock-free, for any type of deadlock, is undecidable, as it is a special instance of the halting problem for Turing machines (Hopcroft et al, 2006). We overcome this obstacle by focusing on CMW deadlocks and by making modeling assumptions. A key challenge is scalability. Real-world large-scale software contains thousands of functions and millions of lines of code. Inlining the whole program, which is required for deadlock analysis, is not an option. We first prune functions and code regions that are irrelevant to deadlock analysis. Even after pruning, inlining the whole program would result in excessively large Petri nets. We apply lock graph analysis (Engler and Ashcraft, 2003; Cano et al, 2010) to isolate the code regions that are subject to deadlock, and inline only the tail of the whole call stack that fully contains the deadlock. After pruning and lock graph analysis, we obtain a manageable model that captures all mutex interactions, and thereby all CMW deadlocks in the program. In addition to scalability, language features also pose difficulties, e.g., recursion, function pointers, and dynamic locks. When in doubt about what particular lock a given call refers to, e.g., when a dynamic lock is used, we model the lock in a conservative way; see (Wang et al,

2008) for a detailed discussion. Finally, there are Operating System, C language, and Pthread library specific features that we do not currently model, e.g., UNIX Inter-Process Communication calls that can result in other types of deadlocks, and `setjump, longjump` functions in C. Using all of the above techniques and under the above restrictions, we are able to capture all CMW deadlocks in multithreaded programs using Petri nets.

As discussed in Section 1, a wide range of sub-classes of Petri nets have been proposed in the literature, most of them motivated by applications in flexible manufacturing systems. Similarly, the class of Gadara net formally defined in this paper is motivated by the application domain of multithreaded programs, with a focus on the analysis of CMW deadlocks.

A Petri net model is obtained in Step 2 of the Gadara architecture in Fig. 1 by translating the enhanced CFG of the program. We create a place to represent each node (i.e., *basic block*) in the enhanced CFG. Moreover, a directed arc connecting two nodes in the enhanced CFG is represented by a transition and two arcs in the Petri net. For example, if there is an arc $\overrightarrow{AB}$ in the enhanced CFG that connects node $A$ to node $B$, then in the corresponding Petri net model, the two nodes $A$ and $B$ are represented by two places $p_A$ and $p_B$, respectively; further, $\overrightarrow{AB}$ is represented by three components in the Petri net: a transition $t$, an arc from $p_A$ to $t$, and another arc from $t$ to $p_B$. Lock variables are also modeled by places. The connectivity of these places to the transitions in the Petri net is determined by the actions of lock acquisitions/releases of the program. If a transition represents a lock acquisition call, we add an arc from the place modeling the lock to the transition; if a transition represents a lock release call, we add an arc from the transition to the place modeling the lock. A token in a place that represents a basic block models a thread executing in this basic block; a token in a place that represents a lock models the availability of this lock. The final Petri net model is called a Gadara net.

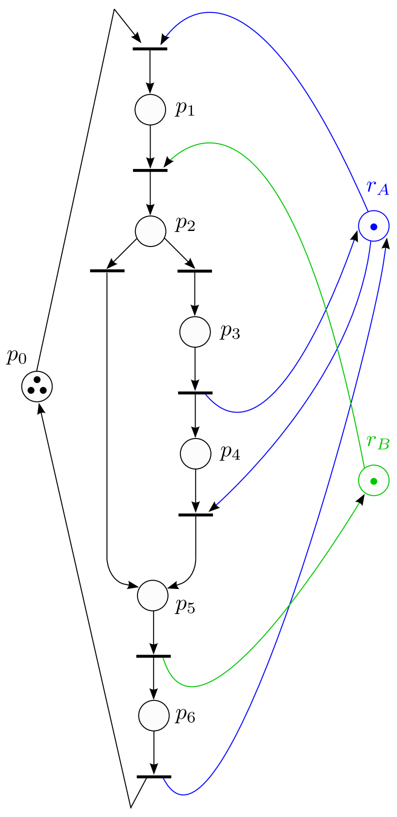
The formal definition of a Gadara net is presented in Section 3. Under the framework of Gadara nets, we are able to: (i) systematically characterize the execution of programs in terms of formal models; (ii) analyze the desired properties of programs in the context of models, and transform the goals into equivalent control specifications on Petri nets; and (iii) synthesize provably correct and optimal control logic on the model that can in turn be instrumented in the original programs.

## 2.3 Running example

A deadlock example in the BIND software is shown in Fig. 2. The acronym BIND stands for "Berkeley Internet Name Daemon," which is a popular Domain Name System (DNS) on the Internet. Figure 2(a) shows the lines of code that are related to the deadlock under consideration; the corresponding Gadara net model is shown in Fig. 2(b). The deadlock occurs if there is one token in $p_1$, which represents one thread holding lock $A$ while waiting for lock $B$, and there is one token in $p_4$, which represents another thread holding lock

```
rwlock_lock(&rbtdb->tree_lock, type);
/* LOCK(A) */
...
lock(&rbtdb->node_lock[i]);
/* LOCK(B) */
...
if(...){
   rwlock_unlock(&rbtdb->tree_lock);
   /* UNLOCK(A) */
   rwlock_lock(&rbtdb->tree_lock, READ);
   /* LOCK(A) */
}
...
unlock(&rbtdb->node_lock[i]);
/* UNLOCK(B) */
...
rwlock_unlock(&rbtdb->tree_lock);
/* UNLOCK(A) */
```

(a)



(b)

**Fig. 2** A deadlock example in BIND: (a) simplified code; (b) Gadara net model

$B$ while waiting for lock $A$. This deadlock bug occurred in the final release version 9.2.2, and was fixed in the release candidate version 9.2.3rc1. As the bug database of BIND is not open to the public due to security reasons, we confirmed the bug by the change log of 9.2.3rc1, as well as using source code comparison. The bug resided in the `rbtdb.c` file, which is a red black tree data structure that stores domain names and IP addresses. For the sake of discussion, the Gadara net model has been simplified; in particular, we model the Reader/Writer lock in this example as a mutex. We will use this Gadara net as a running example throughout the paper.

## 3 The Gadara Petri Net Model

Gadara nets, first introduced in (Wang et al, 2009b), are a special class of Petri nets that model lock allocation and release in multithreaded computer programs for the purpose of deadlock avoidance. In this section, we formally define the class of Gadara nets. When an original Gadara net is augmented with the synthesized monitor places and their associated arcs, we obtain the class of controlled Gadara nets, which are also defined. We first briefly review the Petri net preliminaries; see (Murata, 1989) for a detailed discussion.

## 3.1 Petri net preliminaries

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

The *marking* (a.k.a. *state*) of a Petri net $\mathcal{N}$ is a column vector $M$ of $n$ entries corresponding to the $n$ places. As defined above, $M_0$ is the initial marking. We use $M(p)$ to denote the (partial) marking on a place $p$, which is a scalar; we use $M(Q)$ to denote the (partial) marking on a set of places $Q$, which is a $|Q| \times 1$ column vector. The notation $\bullet p$ denotes the set of input transitions of place $p$: $\bullet p = \{t|(t, p) \in A\}$. Similarly, $p\bullet$ denotes the set of output transitions of $p$. The sets of input and output places of transition $t$ are similarly defined by $\bullet t$ and $t\bullet$. This notation is extended to sets of places or transitions in a natural way. A transition $t$ is *enabled* or *fireable* at $M$, if $\forall p \in \bullet t$, $M(p) \geq W(p, t)$. The *reachable state space* $R(\mathcal{N}, M_0)$ of $\mathcal{N}$ is the set of all markings reachable by transition firing sequences starting from $M_0$.

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. A Petri net is called *ordinary* if all the arcs in the net have unit arc weights, i.e., $W(a) = 1, \forall a \in A$; otherwise, it is called *non-ordinary.* Without any confusion, we can drop $W$ in the definition of any Petri net $\mathcal{N}$ that is ordinary.

**Definition 2** The *incidence matrix* $D$ of a Petri net is an integer matrix $D \in \mathbb{Z}^{n \times m}$, where $D_{ij} = W(t_j, p_i) - W(p_i, t_j)$ represents the net change in the number of tokens in place $p_i$ when transition $t_j$ fires.

**Definition 3** A state machine is an ordinary Petri net such that each transition $t$ has exactly one input place and exactly one output place, i.e., $\forall t \in T, | \bullet t| = |t \bullet | = 1$.

**Definition 4** Let $D$ be the incidence matrix of a Petri net $\mathcal{N}$. Any non-zero integer vector $y$ such that $D^T y = 0$ is called a *P-invariant* of $\mathcal{N}$. Further, P-invariant $y$ is called a *P-semiflow* if all the elements of $y$ are non-negative.

By definition, P-semiflow is a special case of P-invariant. A straightforward property of P-invariants is given by the following well-known result (Murata, 1989): If a vector $y$ is a P-invariant of Petri net $\mathcal{N} = (P, T, A, M_0)$, then we have $M^T y = M_0^T y$ for any reachable marking $M \in R(\mathcal{N}, M_0)$. The *support* of P-semiflow $y$, denoted as $\|y\|$, is defined to be the set of places that correspond to nonzero entries in $y$. A support $\| y \|$ is said to be *minimal* if there does not exist another nonempty support $\|y'\|$, for some other P-semiflow $y'$, such that $\|y'\| \subset \|y\|$. A P-semiflow $y$ is said to be *minimal* if there does not exist another P-semiflow $y'$ such that $y'(p) \leq y(p)$, $\forall p$. For a given minimal support of a P-semiflow, there exists a unique minimal P-semiflow, which we call the *minimal-support P-semiflow* (Murata, 1989).

## 3.2 Gadara Petri nets

As discussed in Section 2, Gadara nets are translated from the enhanced CFG of multithreaded programs. They provide a formal foundation to model the locking behavior (case of mutexes) of the program. Gadara nets share features with classes of Petri nets that arise in the modeling of manufacturing systems (Reveliotis, 2005; Li et al, 2008). More specifically, they consist of a set of *process subnets* that correspond to thread entry points in the program, and resource places that model the locks through which threads interact.

**Definition 5** Let $I_{\mathcal{N}} = \{1, 2, ..., m\}$ be a finite set of process subnet indices. A Gadara net is an ordinary, self-loop-free Petri net $\mathcal{N}_G = (P, T, A, M_0)$ where

1. $P = P_0 \cup P_S \cup P_R$ is a partition such that: a) $P_S = \bigcup_{i \in I_{\mathcal{N}}} P_{S_i}, P_{S_i} \neq \emptyset$, and $P_{S_i} \cap P_{S_j} = \emptyset$, for all $i \neq j$; b) $P_0 = \bigcup_{i \in I_{\mathcal{N}}} P_{0_i}$, where $P_{0_i} = \{p_{0_i}\}$; and c) $P_R = \{r_1, r_2, ..., r_k\}$, $k > 0$.
2. $T = \bigcup_{i \in I_{\mathcal{N}}} T_i, T_i \neq \emptyset, T_i \cap T_j = \emptyset$, for all $i \neq j$.
3. For all $i \in I_{\mathcal{N}}$, the subnet $\mathcal{N}_i$ generated by $P_{S_i} \cup \{p_{0_i}\} \cup T_i$ is a strongly connected state machine.
4. $\forall p \in P_S$, if $|p \bullet| > 1$, then $\forall t \in p\bullet, \bullet t \cap P_R = \emptyset$.
5. For each $r \in P_R$, there exists a unique minimal-support P-semiflow, $Y_r$, such that $\{r\} = \|Y_r\| \cap P_R$, $(\forall p \in \|Y_r\|)(Y_r(p) = 1)$, $P_0 \cap \|Y_r\| = \emptyset$, and $P_S \cap \|Y_r\| \neq \emptyset$.
6. $\forall r \in P_R, M_0(r) = 1$, $\forall p \in P_S, M_0(p) = 0$, and $\forall p_0 \in P_0, M_0(p_0) \geq 1$.
7. $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$.

A Gadara net $\mathcal{N}_G$ is defined to be an ordinary Petri net, because it models programs with mutex locks. **Condition 1** classifies the set of places in $\mathcal{N}_G$ into three types: (i) The *idle place* $p_{0_i} \in P_0$ is an artificial place added to facilitate the discussion of liveness and other properties. The tokens in an idle place represent the threads that "wait" for future execution. (ii) $P_S$ is the set of *operation places*. Each operation place models a *basic block* of the program. A token in an operation place represents one instance of thread that is executing in the current basic block. (iii) $P_R$ is the set of *resource places* that model mutex locks. A token in a resource place represents the availability of the mutex lock. For example, in the Gadara net shown in Fig. 2(b), place $p_0$ is an idle place, places $r_A$ and $r_B$ are resource places, and the other places in the net are operation places.

**Condition 2** defines the set of transitions in $\mathcal{N}_G$. Each subnet of $\mathcal{N}_G$ has its own set of transitions, which is not shared by any other subnet. A transition in

$\mathcal{N}_G$ models the action of lock acquisition or release by the program; a transition can also model branches in the program, such as `if/else` and `switch/case`.

$\mathcal{N}_G$ consists of a set of subnets $\mathcal{N}_i$ that define work processes, called process subnets in the literature. The process subnets are interconnected by resource places. Based on process subnet $\mathcal{N}_i$, if we further consider the resource places (and monitor places that will be introduced in the next section) associated with it, then the resulting net is called a *resource-augmented process subnet*, denoted as $\mathcal{N}_i^{aug}$. Unlike most prior work in manufacturing applications, our process subnets need not be acyclic, due to the modeling of loops in programs. We observe from Fig. 2(b) that the concurrent execution of multiple threads can even be modeled by one process subnet with multiple tokens in different operation places.

In **Condition 3**, the restriction of the process subnets $\mathcal{N}_i$ to the class of state machines implies that there is no "forking" or "joining" in these subnets. The state machine structure of a process subnet is a natural result of the translation of the enhanced CFG as described in Section 2. On the other hand, the strong connectivity of the subnets $\mathcal{N}_i$, which is also stipulated by Condition 3, ensures that in the dynamics of these subnets, a token starting from the idle place will always be able to come back to the idle place after processing. In more natural terms, this requirement for strong connectivity implies that the only reason that might prevent the completion of the considered processes is their contest for the locks that govern their access to their critical sections and not any other potential errors in the underlying program logic.

**Condition 4** models the requirement that a transition representing a branch selection should not be engaged in any resource allocation. Conditions 5 and 6 characterize a distinct and crucial property of Gadara nets. First, the semiflow requirement in **Condition 5** guarantees that a resource acquired by a process will always be returned later. A process subnet cannot "generate" or "destroy" resources. We further require all coefficients of these semiflows $Y_r$ to be equal to one. This requirement implies that the total number of tokens in $||Y_r||$, the support places of any such semiflow $Y_r$, is constant at any reachable marking $M$. Condition 6 defines the initial token content, and therefore this constant is exactly equal to one. Hence, we have the following proposition:

**Proposition 1** *For any $r \in P_R$, at any reachable marking $M$ in $\mathcal{N}_G$, there is exactly one token in the support places of P-semiflow $Y_r$.*

To illustrate the concept of P-semiflow, consider the Gadara net shown in Fig. 2(b) that has two resource places $r_A$ and $r_B$. The minimal-support P-semiflows associated with $r_A$ and $r_B$ are $||Y_{r_A}|| = \{r_A, p_1, p_2, p_3, p_5, p_6\}$ and $||Y_{r_B}|| = \{r_B, p_2, p_3, p_4, p_5\}$, respectively.

As we discussed above, if the token is in resource place $r$, the mutex lock corresponding to $r$ is available. Otherwise, it is in a place $p \in ||Y_r|| \cap P_{S_i}$ of some process subnet $\mathcal{N}_i$, which means that the thread in $p$ is holding the lock. In practice, a multithreaded program may not satisfy Condition 5 for every resource due to programming language complications and missing information in the CFG representation. For example, a thread may acquire a lock without releasing it. Gadara users may restore the P-semiflow manually, e.g., by adding annotations to program source code (Wang et al, 2008).

**Condition 6** specifies the initial markings of the three types of places. At the initial state, all the mutex locks are available; there is no thread executing in the process subnets; and, the number of threads waiting for future execution can be any positive integer.

Finally, **Condition 7** states any operation place models a basic block that requires the acquisition of at least one lock for its execution. A multithreaded program contains sections executed with at least one lock held by the executing thread, called *critical sections* in operating systems terms, and sections executed without holding any lock. Condition 7 implies that the process subnets only model the critical sections of the programs. Since the sections executed without involving any lock are irrelevant to deadlock analysis, in practice, we prune the Petri nets translated from CFGs so that our obtained Gadara nets only model the critical sections. This pruning process is automated. An illustrative example of the pruning process is shown in Figs. 3 and 4; see (Wang, 2009) for details.

## 3.3 Controlled Gadara nets

Based on the Gadara net model of the program, we want to synthesize control logic to be enforced on the net so that the controlled net corresponds to a deadlock-free program. Supervisory control based on place invariants (SBPI) is a common control technique for Petri nets (Yamalidou et al, 1996; Giua, 1992; Iordache and Antsaklis, 2006). Control specifications implemented by SBPI are represented by a set of linear inequalities on the net markings. Each linear inequality is enforced via a *monitor* place with its associated arcs that augment the original net. The added monitor place establishes a new invariant in the net dynamics and guarantees that the specified linear inequality is always satisfied in the controlled net. This invariant has a structure that is similar to that introduced by Condition 5 of Definition 5, with the monitor place playing the role of a new (generalized) resource place. When we use this control technique on the Gadara net, we obtain a controlled Gadara net, as defined below. Note that one need not associate a controlled Gadara net with any specific control policy. It is a structural definition that does not refer explicitly to the content of the linear inequalities that are enforced by SBPI.

**Definition 6** Let $\mathcal{N}_G = (P, T, A, M_0)$ be a Gadara net. A controlled Gadara net $\mathcal{N}_G^c = (P \cup P_C, T, A \cup A_C, W^c, M_0^c)$ is a self-loop-free Petri net such that, in addition to all conditions in Definition 5 for $\mathcal{N}_G$, we have

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

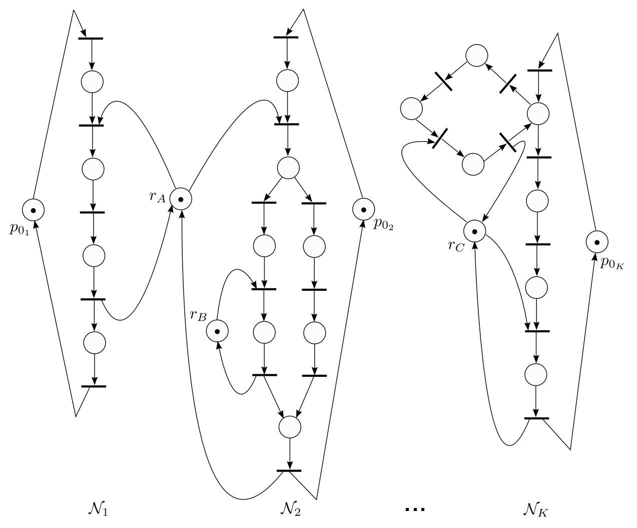


**Fig. 3** An illustrative example of Gadara nets: before pruning

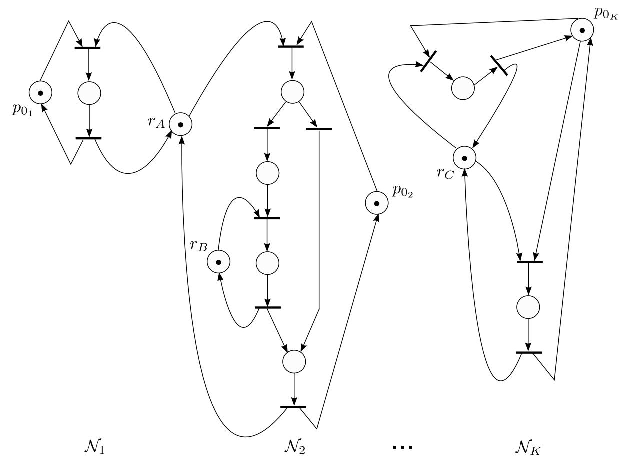


**Fig. 4** An illustrative example of Gadara nets: after pruning

Definition 6 indicates that the introduction of the monitor places into $\mathcal{N}_G^c$ preserves the net structure of $\mathcal{N}_G$ that is specified by Definition 5. **Condition 8** states that the monitor places $P_C$ share similar structural properties with the resource places $P_R$ in terms of the place invariants imposed on the net, which is inspired by the SBPI technique. But they have weaker constraints. More specifically, monitor places may have multiple initial tokens and non-unit arc weights. Therefore, $\mathcal{N}_G^c$ does not necessarily have to be an ordinary net, because of the arcs with non-unit weights that can be potentially introduced by a monitor place. **Condition 9** implies that the initial marking of a monitor place provides a number of tokens that is able to cover, in isolation, the token request posed by any stage in the support of the semiflow of that monitor place.

As a special case of $\mathcal{N}_G^c$, if all the arcs in the net have unit arc weights (or, more specifically, all the arcs associated with monitor places in the net have unit arc weights), then $\mathcal{N}_{G1}^c$, the class of controlled Gadara nets that remain ordinary, can be defined as follows.

**Definition 7** Let $\mathcal{N}_G = (P, T, A, M_0)$ be a Gadara net. A controlled Gadara net $\mathcal{N}_{G1}^c = (P \cup P_C, T, A \cup A_C, M_0^c)$ is an *ordinary*, self-loop-free Petri net such that, it satisfies Conditions 1 to 7 in Definition 5 and Conditions 8 and 9 in Definition 6.

*Remark 3* From Definitions 5, 6, and 7, we observe that $\mathcal{N}_G$ is a special subclass of both $\mathcal{N}_{G1}^c$ and $\mathcal{N}_G^c$, where $P_C = \emptyset$ and $A_C = \emptyset$. Furthermore, $\mathcal{N}_{G1}^c$ is a special subclass of $\mathcal{N}_G^c$, where $W^c(a) = 1, \forall a \in A \cup A_C$. Therefore, any property that we derive for $\mathcal{N}_G^c$ holds for both $\mathcal{N}_{G1}^c$ and $\mathcal{N}_G$ as well. In the following, for the sake of simplicity, we refer to $\mathcal{N}_G^c$ as a "Gadara net" (unless special mention is made).

## 3.4 Discussion

Petri nets have been extensively applied to the modeling and analysis of flexible manufacturing systems and other technological applications involving a resource allocation function (Li et al, 2008; Reveliotis, 2005). In this application domain, the class of $S^3PR$ nets is one of the most widely studied sub-classes of Petri nets; it consists of process-oriented nets that possess an acyclicity property (Ezpeleta et al, 1995). Many sub-classes of Petri nets have been developed to extend the formulation of $S^3PR$ in order to model special features of specific systems. Recently, a new class of Petri nets, called $STPR$, has been proposed for anomaly detection in manufacturing systems (Allen, 2010). A unique characteristic of $STPR$ nets is that the system allows resource creation and negated resources; these features are not suitable for our needs in this paper. Multithreaded software systems share some similarities with manufacturing systems, e.g., the operation of both systems require acquisition and release of resources (i.e., locks). However, loops, such as `for` and `while`, are very common in programs, and they result in internal cycles in the process subnets of their Petri net models. Thus, there is a need to relax the acyclicity constraint of $S^3PR$ nets. The resulting superclass is called $S^*PR$. Deadlock analysis is known to be difficult when the process subnets in process-oriented nets contain internal cycles (Park and Reveliotis, 2002; Jeng and Xie, 2001). In (Jeng and Xie, 2001), the authors study the class of RCN* merged nets, which arises in semiconductor manufacturing systems. The potentially degraded behaviors (e.g., reworks and failures) in this manufacturing setting necessitate the internal cycles in the model. In (Park and Reveliotis, 2002), liveness-enforcing supervision is investigated for a broad class of resource allocation systems, in the presence of uncontrollable behavior that can also lead to cyclic behavior.

(Park and Reveliotis, 2001) extends the results on liveness analysis and control of ordinary nets to the class of non-ordinary process-resource nets. There are few results on deadlock analysis in $S^*PR$ (Ezpeleta et al, 2002). Gadara nets $\mathcal{N}_G$ fall within the $S^*PR$ class, but they possess features, such as unit arc weight and 1-bounded operation places (which will be discussed in Proposition 3 in Section 5.1), which render deadlock analysis more tractable and enable the synthesis of MPLE control logic through monitor places.

## 4 Main Properties of Gadara Nets

With the class of Gadara nets formally defined, our next task is to establish the relevant properties of Gadara nets, such that the goal of deadlock-free execution of a program can be mapped to an equivalent objective in terms of its corresponding Gadara net model. This task is carried out in three steps. First, we establish in Section 4.3 that the goal of deadlock-free execution of a program is equivalent to its corresponding Gadara net model satisfying a *behavioral* property, called reversibility. Second, we prove in Section 4.4 that for a Gadara net, liveness, which is a *behavioral* property, is equivalent to the absence of certain types of siphons in the net, which is a *structural* feature. Third, we show in Section 4.5 that for a Gadara net, liveness is equivalent to reversibility. As a result of the above three steps of analysis, the *behavioral* property of deadlock-free execution of a program is mapped to an equivalent objective in terms of a *structural* property of the Gadara net. This mapping has important implications for efficient control synthesis.

There have been many results on property analysis developed for other special classes of Petri nets in the literature. However, they generally cannot be directly applied to our problem, in the context of Gadara net model of concurrent software. The set of properties that will be introduced in this section bridges the multithreaded program and its model, and lays the groundwork for efficient MPLE control synthesis (Liao et al, 2010).

We start the discussion with a series of definitions that formalize the Petri net concepts of liveness and reversibility and some additional concepts related to them.

### 4.1 Petri net liveness and reversibility

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

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

Consider the Gadara net shown in Fig. 2(b); in it, the set of places $S_{AB} = \{r_A, r_B, p_2, p_3, p_5, p_6\}$ is a siphon.

For the sake of simplicity, in the following discussion we use $R(\mathcal{N}, M)$ to denote the set of reachable markings of net $\mathcal{N}$ starting from marking $M$.

**Definition 10** A marking $M$ is *live* if $\forall t \in T$, there exists $M' \in R(\mathcal{N}, M)$, such that $t$ is enabled at $M'$. A Petri net $(\mathcal{N}, M_0)$ is *live* if $\forall M \in R(\mathcal{N}, M_0)$, $M$ is live.

**Definition 11** Petri net $\mathcal{N}$ is said to be *quasi-live* if, for all $t \in T$, there exists $M \in R(\mathcal{N}, M_0)$, such that $t$ is enabled at $M$.

**Definition 12** A Petri net is in a *total deadlock* if all the transitions in the net are disabled.

**Definition 13** Petri net $\mathcal{N}$ is said to be *reversible* if $M_0 \in R(\mathcal{N}, M)$, for all $M \in R(\mathcal{N}, M_0)$.

Clearly, Conditions 3 and 6 of Definition 5 imply that all subnets $\mathcal{N}_i$ in a Gadara net $\mathcal{N}_G$ are quasi-live. Furthermore, Conditions 5 and 6 of Definition 5 imply that quasi-liveness is preserved, when each subnet $\mathcal{N}_i$ is augmented with the corresponding resource places in $P_R$. Similarly, Conditions 8 and 9 of Definition 6 imply the preservation of quasi-liveness for the subnets $\mathcal{N}_i$ of $\mathcal{N}_G^c$ when augmented with the monitor places $p_c \in P_C$. Finally, the combination of Condition 3 of Definition 5 with the quasi-liveness of the resource and monitor-place-augmented subnets $\mathcal{N}_i$ established above, further imply the reversibility of the latter, when executing in isolation, i.e., when $M_0(p_{0_i}) = 1$.

## 4.2 Resource-induced deadly marked siphons and modified markings

The following two concepts pertain to the process-resource net structure of Gadara nets, and they play a very important role in the characterization of the liveness and reversibility of Gadara nets that is provided in the rest of this section.

**Definition 14** A siphon $S$ of a Gadara net $\mathcal{N}_G^c$ is said to be a *resource-induced deadly marked (RIDM) siphon* (Reveliotis, 2005) at marking $M$, if it satisfies the following conditions:

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

From Definition 14, we see that a RIDM siphon is defined by a siphon $S$, together with a partial marking on $S$. Thus, whenever we refer to a RIDM siphon $S$, it means the set of places that constitute $S$ as well as the partial marking on $S$.

To illustrate the notion of RIDM siphon, again, refer to the example in Fig. 2(b), and consider the reachable marking $M$, where there is one token in $p_0$, one in $p_1$, and one in $p_4$, while all other places are empty. The siphon $S_{AB} = \{r_A, r_B, p_2, p_3, p_5, p_6\}$ discussed in Section 4.1 is a RIDM siphon at marking $M$. Further, $S_{AB}$ is an *empty* siphon at marking $M$. The notion of

RIDM siphon can also be used in a non-ordinary net. In general, a RIDM siphon can be nonempty. An empty siphon is a special case of RIDM siphon. Figure 5 shows an example of a nonempty RIDM siphon in a non-ordinary net: $S = \{p_{c_1}, p_{c_2}, p_{12}, p_{13}, p_{22}, p_{23}\}$ with its associated marking illustrated in the figure.

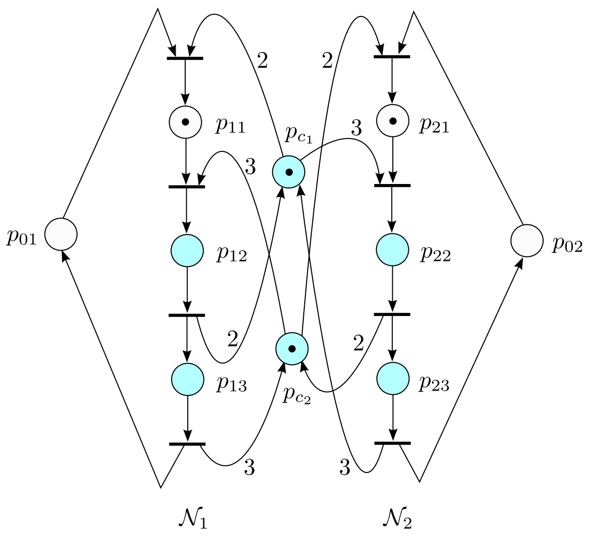


**Fig. 5** Example: $S = \{p_{c_1}, p_{c_2}, p_{12}, p_{13}, p_{22}, p_{23}\}$ is a nonempty RIDM siphon at the marking shown in the figure

**Definition 15** Given a Gadara net $\mathcal{N}_G^c$ and a marking $M \in R(\mathcal{N}_G^c, M_0^c)$, the *modified marking* $\overline{M}$ is defined by

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

Modified markings essentially "erase" the tokens in idle places. The set of modified markings induced by the set of reachable markings is defined by $\overline{R(\mathcal{N}_G^c, M_0^c)} = \{\overline{M} | M \in R(\mathcal{N}_G^c, M_0^c)\}$. Note that the number of tokens in idle places $P_0$ can always be uniquely recovered from the invariant implied by the strongly-connected state-machine structure of the subnet $\mathcal{N}_i$. Therefore, there is a one-to-one mapping between the original marking and the modified marking, i.e., $M_1 = M_2$ if and only if $\overline{M}_1 = \overline{M}_2$.

Condition 7 of Definition 5 indicates that the set of idle places do not directly interact with any resource place, and therefore they are irrelevant to the analysis of deadlocks in Gadara nets. The notion of modified markings enables us to associate the non-liveness of the net to RIDM siphons.

## 4.3 Multithreaded program and its Gadara net model

The following result provides a bridge between a program and its corresponding Gadara net model (under the assumptions discussed in Section 2.2) in terms of two relevant behavioral properties.

**Proposition 2** *A multithreaded program that can be modeled as a Gadara net $\mathcal{N}_G^c$ is deadlock-free* iff *$\mathcal{N}_G^c$ is reversible.*

*Proof:* First we show the "$\Rightarrow$" direction.

If a program is free from any CMW deadlocks[1], then for any stage the program is executing, all instances of threads in the program can always complete the rest of their executions, and terminate the processes. This corresponds to the case in the Gadara net model, where starting from any marking of the net, the tokens in the operation places can eventually return to the idle places, which leads the net back to the initial marking. Thus, the net is reversible.

Next we show the "$\Leftarrow$" direction.

(Proof by contra-positive proposition) Suppose there exist at least two threads involved in a CMW deadlock of the program; then these instances of threads are unable to complete their executions. In the corresponding Gadara net model of the program, these deadlocked threads are modeled as tokens in operation places. The fact that these threads are unable to terminate implies that the aforementioned tokens will never return to the idle places. In other words, starting from this state, the net will never return to the initial marking. Thus, the net is not reversible. □

*Remark 4* From Remark 1 and the above discussion, we know that a Gadara net model being deadlock-free does *not* guarantee that its corresponding program is free from any CMW deadlocks. For example, let us consider a Gadara net model containing $N$ process subnets. Assume that at some marking of the net: (i) there exist two process subnets, say $\mathcal{N}_1$ and $\mathcal{N}_2$, such that all the transitions in these two process subnets are disabled; and (ii) for the remaining $N-2$ process subnets, there exists at least one enabled transition in each of them. The Gadara net at this marking is deadlock-free by Definition 12. However, the underlying program has a CMW deadlock, which involves the process subnets $\mathcal{N}_1$ and $\mathcal{N}_2$. In other words, the notion of "deadlock-freeness" in Petri nets is with respect to the behavior of "total-deadlock" of the net. On the other hand, the requirement of "deadlock-freeness" in multithreaded programs cares not only about "total-deadlock", but also about "local-deadlock" as described above.

It is well known that if an ordinary Petri net cannot reach an empty siphon, then the net is deadlock-free (Reisig, 1985). But, Remark 4 implies that for the purpose of deadlock avoidance in a multithreaded program, requiring its Gadara net model to be deadlock-free is not sufficient. This motivates our

[1] Recall Remark 1: "deadlock-free" in Proposition 2 refers to the case where the program is free from any CMW deadlock.

investigation of the liveness property of Gadara nets in the next section, where we establish *necessary and sufficient* conditions for liveness (of $\mathcal{N}_G^c$, $\mathcal{N}_G$, and $\mathcal{N}_{G1}^c$) in terms of the absence of certain types of siphons.

## 4.4 Liveness of Gadara nets

Liveness and reversibility are closely related properties of Gadara nets. In fact, they are shown to be equivalent in Section 4.5. In this section, we first establish some results about the liveness of Gadara nets, which connect this *behavioral* property to a certain *structural* property in terms of siphons. Similar results exist in the literature (see Theorem 5.3 and Corollary 3 on p. 132 of (Reveliotis, 2005)) for a class of process-resource nets that are structurally similar but model processes with no internal cycles. Despite the presence of cycles and other technical differences in our process subnets, the above results in (Reveliotis, 2005) can be extended to Gadara nets.

**Theorem 1** *$\mathcal{N}_G^c$ is live* iff *there does not exist a modified marking $\overline{M} \in \overline{R(\mathcal{N}_G^c, M_0^c)}$ and a siphon $S$ such that $S$ is a RIDM siphon at $\overline{M}$.*

*Proof:* First we show the "$\Rightarrow$" direction.

(Proof by contra-positive proposition) Suppose that there exists a marking $M$ such that the corresponding modified marking $\overline{M}$ contains a RIDM siphon $S$. From the definition of the RIDM siphon, there exists a place $q \in S \cap (P_R \cup P_C)$, and a transition $t \in q\bullet$ that is disabled due to the lack of enough tokens in $q$. On the other hand, since $q \in S$, by the definition of RIDM siphons, the transitions in $\bullet q$ are all disabled. Therefore, place $q$ will never get replenished in $R(\mathcal{N}_G^c, \overline{M})$, and the disabled transition $t$ will remain non-live in $R(\mathcal{N}_G^c, \overline{M})$. Furthermore, Condition 5 of Definition 5 and Condition 8 of Definition 6 imply that $P_0 \cap \|Y_q\| = \emptyset$, and $q \notin T_I\bullet$, where $T_I = P_0\bullet$. So, when we move from the modified markings to the original markings in $\mathcal{N}_G^c$ by re-introducing the tokens in $P_0$, place $q$ will not gain new tokens, and the disabled transition $t$ will remain non-live. Therefore, the liveness of $\mathcal{N}_G^c$ implies that $\overline{R(\mathcal{N}_G^c, M_0)}$ contains no RIDM siphons.

Next we show the "$\Leftarrow$" direction.

(Proof by contra-positive proposition) Suppose that $\mathcal{N}_G^c$ is not live. We want to show that $\overline{R(\mathcal{N}_G^c, M_0)}$ contains at least one RIDM siphon. By the non-liveness assumption, we know that there exists a marking $M' \in R(\mathcal{N}_G^c, M_0)$ such that at least one transition $t' \in T$ is never enabled in $R(\mathcal{N}_G^c, M')$.

Figure 6 serves as an illustrative example to visualize the flow of the proof, but our arguments hold more generally. For the sake of simplicity, the figure presents only the critical portions of the two subnets involved in the deadlock. In the depicted example, the aforementioned marking $M'$ corresponds to the case where there is one token in $p_{11}$, one in $p_{22}$, and one in $r_C$, while all other places are empty.

In view of the structural assumptions made in defining $\mathcal{N}_G^c$, there also exists a marking $M \in R(\mathcal{N}_G^c, M')$, that satisfies the following two conditions:

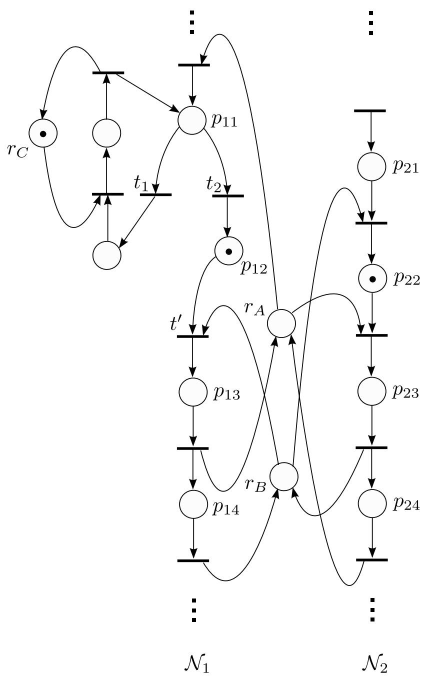


**Fig. 6** An illustrative example of the general case considered in the proof of Theorem 1

(i) There exists at least one process subnet $\mathcal{N}_i$ such that $M(p_{0_i}) < M_0(p_{0_i})$. Namely, an instantiation of the thread modeled by $\mathcal{N}_i$ is "half-way" in execution at marking $M$. Furthermore, the dead transition $t'$ must belong to one of these thread subnets.

(ii) Every transition $t \notin P_0\bullet$ is disabled at $M$. From the definition of the modified marking, this fact further implies that all the transitions are disabled at $\overline{M}$. That is, $\overline{M}$ is a total deadlock.

We claim that (i) must be satisfied, because otherwise $M_0$ is reachable from $M'$. In this case, the quasi-liveness property of $\mathcal{N}_i$, discussed in Section 4.1, implies that $t'$ is not dead at $M'$, which contradicts our assumption.

We claim that (ii) must also be satisfied. Although a process subnet of $\mathcal{N}_G^c$ may contain an internal cycle, Condition 4 of Definition 5 guarantees that a token will never be "trapped" in a cycle where it loops indefinitely. Therefore, the remaining process subnets, which are not involved in the deadlock, can eventually complete the execution of all their active thread instances and return all their tokens back to their idle places. Hence, the only enabled transitions of these subnets at marking $M$ are the output transitions of their idle places, which further implies that they deadlock at marking $\overline{M}$. In other words, a marking $M \neq M_0^c$, whose modified marking $\overline{M}$ corresponds to a total deadlock, is always reachable from $M'$.

We further illustrate the above arguments using the example of Fig. 6. Marking $M$ corresponds to the case where there is one token in $p_{12}$, one in

$p_{22}$, and one in $r_C$, while all other places are empty. Note that such a marking $M$ always exists, i.e., $M$ is always reachable from $M'$. Although at marking $M'$ subnet $\mathcal{N}_1$ can choose either $t_1$ or $t_2$ to fire, Condition 4 of Definition 5 guarantees that $t_2$ is never disabled by any resource place and will always be enabled starting from $M'$. The example of Fig. 6 demonstrates that this statement is true even in the case where the firing transition sequence from $M'$ to $M$ involves a branch selection, and one of the two branches leads to a cycle.

We are left to show that $\overline{M}$ contains a RIDM siphon. Let $S$ denote the set of disabling places at marking $\overline{M}$. Since $\overline{M}$ is a total deadlock, $S\bullet = T$, where $T$ is the set of all transitions of $\mathcal{N}_G$. Thus, we have the relationship

$$\bullet S \subseteq S\bullet = T.$$

By definition, $S$ is a siphon. Obviously, $S$ also satisfies Conditions 1 and 3 of Definition 14.

Furthermore, Condition (i) that characterizes marking $M$, when combined with the state machine structure of net $\mathcal{N}_i$ (c.f. Condition 3 of Definition 5), implies that there exists at least one transition $t \in T_i$ with $\bullet t \cap P_S = \{p\} \neq \emptyset$ and with $M(p) = \overline{M}(p) = 1$. Therefore, the deadlock at $\overline{M}$ must involve some place $q \in P_R \cup P_C$, and Condition 2 of Definition 14 is satisfied. Hence, $S$ is a RIDM siphon in $\mathcal{N}_G^c$. □

When a Gadara net is ordinary (i.e., $\mathcal{N}_G$ or $\mathcal{N}_{G1}^c$), we can characterize liveness in terms of empty siphons, which is a special case of RIDM siphons.

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

*(2) $\mathcal{N}_{G1}^c$ is live* iff *there does not exist a marking $M \in R(\mathcal{N}_{G1}^c, M_0^c)$ and a siphon $S$ such that $S$ is an empty siphon at $M$.*

The proof of this theorem is similar to the proof of Corollary 3 on p. 132 of (Reveliotis, 2005). It is presented in the appendix for the sake of completeness.

As discussed in Section 4.2, the siphon $S_{AB} = \{r_A, r_B, p_2, p_3, p_5, p_6\}$ in the Gadara net shown in Fig. 2(b) becomes an empty siphon at the reachable marking $M$, where there is one token in $p_0$, one in $p_1$, and one in $p_4$, while all other places are empty. Thus, from Theorem 2, the Gadara net depicted in Fig. 2(b) is not live. Alternatively, we can also verify that $S_{AB}$ is a RIDM siphon at $\overline{M}$; hence, from Theorem 1, we arrive at the same conclusion that the Gadara net in Fig. 2(b) is not live.

## 4.5 Reversibility of Gadara nets

In this section, we establish the equivalence between liveness and reversibility in Gadara nets. This result "links" Proposition 2 with Theorems 1 and 2, such that the goal of deadlock-free execution of the program can be mapped to the absence of certain types of siphons in the Gadara net.

**Theorem 3** *$\mathcal{N}_G^c$ is live* iff *it is reversible.*

The proof of this theorem is presented in the appendix.

### 4.6 Summary

We summarize this section with the following important results.

**Theorem 4** *(1) If a multithreaded program can be modeled as $\mathcal{N}_G^c$, then the program is deadlock-free* iff *$\mathcal{N}_G^c$ cannot reach a modified marking $\overline{M}$ such that there exists at least one RIDM siphon at $\overline{M}$.*

*(2) If a multithreaded program can be modeled as $\mathcal{N}_G$ (or $\mathcal{N}_{G1}^c$), then the program is deadlock-free* iff *$\mathcal{N}_G$ (or $\mathcal{N}_{G1}^c$) cannot reach a marking $M$ such that there exists at least one empty siphon at $M$.*

Theorem 4 implies that the problem of deadlock avoidance in a multithreaded program is *equivalent* to the problem of preventing any RIDM siphon (resp., empty siphon) from becoming reachable in the modified reachability space (resp., original reachability space) of its Gadara net model $\mathcal{N}_G^c$ (resp., $\mathcal{N}_G$ or $\mathcal{N}_{G1}^c$).

The results established in this section serve as the foundations for the development of MPLE control policies for Gadara nets based on structural analysis (Liao et al, 2010). They also provide a formal method for efficiently verifying the liveness of a Gadara net (and the deadlock-freeness of its underlying program), as we will see in the next section.

## 5 Verification of Liveness Using Mathematical Programming

From Theorems 1 and 2, we know that liveness in Gadara nets can be verified, in principle, by detecting certain types of siphons that may be reachable by the nets. The problem of siphon detection using mathematical programming has been extensively studied in the literature. In (Chu and Xie, 1997), a generic Mixed Integer Programming (MIP) formulation is presented for the detection of *maximal empty siphons* in *ordinary, structurally bounded,* Petri nets; we refer to this formulation as MIP-ES hereafter. Furthermore, MIP has also been employed to detect *maximal RIDM siphons* in general process-resource nets that are not necessarily ordinary (Reveliotis, 2005); we refer to this general MIP formulation stated on pp. 139-140 of (Reveliotis, 2005) as MIP-RS hereafter.

From the development of Theorem 1, we know that if a Gadara net is not live, then the net will eventually reach a so-called "total-deadlock modified-marking", where all the transitions in the net are disabled. This result is formally stated as Corollary 1 in Section 5.1 below. This corollary also provides us with an efficient methodology to verify the liveness of a Gadara net through mathematical programming formulations by detecting total-deadlock modified-markings. Similar in spirit to the aforementioned mathematical programming formulations, our formulations search for a total-deadlock modified marking over the broader set of markings defined by the state equation of the net. Thus, any total-deadlock modified-marking identified by these formulations might or might not be reachable in the actual net. More specifically, *if the proposed formulations do not have a solution, then the net is live*; otherwise, the net may or may not be live. A "byproduct" of these formulations is a RIDM siphon (or an empty siphon in the case of ordinary nets) that is constructed from the identified total-deadlock modified-marking through Corollary 2 in Section 5.1 below. The constructed siphon can then be used for MPLE control synthesis, as we do in (Liao et al, 2011).[2]

A more detailed description of the technical developments of this section is as follows. Exploiting the special properties of Gadara nets, we propose in Section 5.2, an efficient MIP formulation for liveness verification of $\mathcal{N}_G$. This MIP formulation is then generalized for liveness verification of $\mathcal{N}_{G1}^c$ in Section 5.3. In Section 5.4, we propose another MIP formulation for liveness verification of $\mathcal{N}_G^c$. In the following discussion, we denote the aforementioned three formulations as MIP-$\mathcal{N}_G$, MIP-$\mathcal{N}_{G1}^c$, and MIP-$\mathcal{N}_G^c$, respectively, which are self-explanatory from their names. The formulations MIP-$\mathcal{N}_G$ and MIP-$\mathcal{N}_{G1}^c$ customize the generic formulation MIP-ES; the formulation MIP-$\mathcal{N}_G^c$ customizes the generic formulation MIP-RS. The development of the customized MIP formulations was motivated by the need of efficient control synthesis for large-scale concurrent software, and it exploits the special structure of Gadara net models of multithreaded programs. These customized formulations enhance the efficiency and scalability of liveness verification of Gadara nets, which is important for deadlock analysis of large-scale software. They are also employed in the MPLE control synthesis of Gadara nets (Liao et al, 2011). In Section 5.5, we report experimental results that compare the performance of liveness verification of $\mathcal{N}_{G1}^c$ using MIP-$\mathcal{N}_{G1}^c$ with that of using MIP-ES; we also compare the performance of liveness verification of $\mathcal{N}_G^c$ using MIP-$\mathcal{N}_G^c$ with that of using MIP-RS. Although the formulations considered in our comparative study use different objective functions and produce, in general, different siphons, they all have the same implication for the purpose of liveness verification: a special property of the optimal solution or, in certain cases, the absence of such a solution itself, is a sufficient condition for the liveness of the Gadara net.

## 5.1 Key properties

We first present some properties of Gadara nets that are relevant to the development of the formulation of liveness verification.

Conditions 5, 6, and 7 of Definition 5 together lead to the following important property of Gadara nets.

[2] It should also be noticed that, in the particular case that the identified RIDM siphon is actually unreachable, the monitor places resulting from the MPLE synthesis do not compromise the maximal permissiveness of the synthesized control logic.

**Proposition 3** *Given a Gadara net $\mathcal{N}_G^c$, $\forall M \in R(\mathcal{N}_G^c, M_0^c)$, $\forall p \in P_S$, $M(p)$ is either 0 or 1. In other words, all operation places in $\mathcal{N}_G^c$ are 1-bounded.*[3]

*Proof:* Proposition 1 states that for any $r \in P_R$, there is exactly one token in the support places, $||Y_r||$, of its P-semiflow $Y_r$. This result, when considered together with Condition 7 of Definition 5, implies that for any operation place in $\mathcal{N}_G^c$, its marking is either 0 or 1. $\square$

Based on Theorem 1, we have the following results. Both Corollaries 1 and 2 follow from the "$\Leftarrow$" direction of the proof of Theorem 1.

**Corollary 1** *If $\mathcal{N}_G^c$ is not live, then $\mathcal{N}_G^c$ will reach a modified marking $\overline{M} \in \overline{R}(\mathcal{N}_G^c, M_0^c)$ and $M \neq M_0^c$, such that $\mathcal{N}_G^c$ is in a total deadlock at the modified marking $\overline{M}$.*

**Corollary 2** *In $\mathcal{N}_G^c$, given a total-deadlock modified-marking $\overline{M} \in \overline{R}(\mathcal{N}_G^c, M_0^c)$ and $M \neq M_0^c$, let $S$ be the set of disabling places at $\overline{M}$. Then, $S$ is a RIDM siphon at $\overline{M}$.*

The intuition of Corollary 1 is as follows. If $\mathcal{N}_G^c$ is not live, then $\mathcal{N}_G^c$ can reach a marking, such that there exists at least one process subnet "half-way" in execution, where no transition in it can be fired, and the process subnet (as well as $\mathcal{N}_G^c$) cannot return to its initial marking. For all other process subnets that can successfully complete their executions, their tokens in the operation places will eventually return to the idle places. After all these process subnets complete their executions, their returned tokens in the idle places are erased under the notion of modified marking (Definition 15). Now, all transitions in $\mathcal{N}_G^c$ are disabled under the modified marking. Thus, we have a total-deadlock modified-marking that is different from the modified initial marking.

Given a total-deadlock modified-marking $\overline{M} \neq \overline{M_0^c}$, we can easily construct a RIDM siphon at $\overline{M}$ using Corollary 2. Note that the modified initial marking is always a total-deadlock modified-marking. But for liveness verification, we are interested in detecting a total-deadlock modified-marking that is different from the modified initial marking. Therefore, instead of repeating the above statement, we impose this qualification on any sought total-deadlock modified-marking considered in the rest of this section.

## 5.2 Verification of liveness of $\mathcal{N}_G$

Recall from Definition 8 that a place $p$ is said to be a disabling place at marking $M$ if $p$ disables at least one of its output transitions at $M$. Further, in an ordinary net, if a place $p$ is a disabling place at marking $M$, then we have $M(p) = 0$ and $p$ disables *all* of its output transitions. By Definition 15, we know that for any place $p \in P_0$, its modified marking $\overline{M}(p) = 0$. Moreover, from Definition 5 and Proposition 3, we know that $\mathcal{N}_G$ is an ordinary net, and

[3] We use the terminology "1-bounded" instead of "safe" (Murata, 1989) because we will use the term "safe" in a different context in Section 6.

the modified marking of any place $p \in P_S \cup P_R$ is either 0 or 1. Therefore, in $\mathcal{N}_G$, the modified marking of a place $p$, $\overline{M}(p)$, can be used as a *binary indicator variable* associated with $p$, as described in the following remark.

*Remark 5* For any place $p \in P_0 \cup P_S \cup P_R$, we have: (i) $\overline{M}(p) = 0$ *iff* at $\overline{M}$, place $p$ is a disabling place and $p$ disables all of its output transitions; (ii) $\overline{M}(p) = 1$ *iff* at $\overline{M}$, place $p$ is not a disabling place and $p$ enables all of its output transitions.

According to Corollary 1, if $\mathcal{N}_G^c$ is not live, then we know *a priori* that the net will reach a total deadlock at some modified marking $\overline{M}$. Moreover, once $\overline{M}$ is reached, we know *a priori* from Corollary 2 that there exists a RIDM siphon $S$ at $\overline{M}$, which contains the set of all disabling places at $\overline{M}$. In particular, we know from Remark 5 that in the case of $\mathcal{N}_G$, this RIDM siphon $S$ is an empty siphon at $\overline{M}$.

The above discussion implies that we can verify the liveness of $\mathcal{N}_G$ very efficiently, by detecting a total-deadlock modified-marking $\overline{M}$, i.e., a modified marking $\overline{M}$ where all the net transitions are disabled. Based on the special structure of $\mathcal{N}_G$, any transition $t$ in the net can be categorized into one of the following three types:

1. Transition $t$ is an output transition of an idle place. We know that under the notion of modified marking, $t$ is always disabled.
2. Transition $t$ has only one input place, and this input place is an operation place. For $t$ to be disabled, its input place must be a disabling place.
3. Transition $t$ has more than one input places. For $t$ to be disabled, at least one of its input places must be a disabling place.

Therefore, in order to detect a total-deadlock modified-marking $\overline{M}$, we need to enforce the above three types of transitions to be disabled at $\overline{M}$, which is addressed by Constraints (4)–(6) of the MIP formulation presented below. If $\overline{M}$ is detected, then we can use Corollary 2 to construct an empty siphon, which will be used in MPLE control synthesis; otherwise, we know that the net is live. In other words, the problem of liveness verification of $\mathcal{N}_G$ can be mapped to the problem of finding a total-deadlock modified-marking in the modified reachability space of $\mathcal{N}_G$. The latter problem can be solved by the following MIP formulation, MIP-$\mathcal{N}_G$, which customizes the generic MIP-ES formulation presented in (Chu and Xie, 1997) for maximal empty siphon detection in structurally bounded ordinary nets.

$$\textbf{MIP-}\mathcal{N}_G\text{:}\quad \min \sum_{p\in P_S} \overline{M}(p) \tag{2}$$

$$s.t.\quad M = M_0 + D\sigma \tag{3}$$

$$\overline{M}(p) = M(p), \forall p \in P_S \cup P_R; \overline{M}(p) = 0, \forall p \in P_0 \tag{4}$$

$$\overline{M}(p) = 0, \forall p \in Q, \text{where,} \quad Q = \{q \in P : (\exists t \in T), (\bullet t = \{q\}) \wedge (q \in P_S)\} \tag{5}$$

$$\sum_{p\in \bullet t} \overline{M}(p) - |\bullet t| + 1 \leq 0, \forall t \text{ s.t. } |\bullet t| > 1 \tag{6}$$

$$\sum_{p\in P_S} \overline{M}(p) \geq 2 \tag{7}$$

$$\sum_{p\in P_R} \overline{M}(p) \leq |P_R| - 2 \tag{8}$$

$$M \geq 0; \sigma \in \mathbb{Z}_0^+ \tag{9}$$

We explain the MIP-$\mathcal{N}_G$ formulation presented in (2)–(9) as follows. In the objective function (2), we want 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 MPLE control synthesis (Liao et al, 2011); the details are beyond the scope of this paper. Constraint (3) is the state equation of the net, which is a necessary condition for the set of reachable markings. Constraint (4) connects an original marking with its associated modified marking based on Definition 15. From the above discussion, we want to verify liveness by finding a total-deadlock modified-marking $\overline{M}$. Constraints (4), (5), and (6) enforce that the three types of transitions, discussed above, are all disabled at $\overline{M}$. Constraint (7) follows from the fact that at least two threads must be involved in a CMW deadlock. In the context of the Gadara net model, this implies that at least two operation places are marked in a CMW deadlock. As a result, it follows from Constraint (7), and Conditions 6 and 7 of Definition 5, that at least two resource places must be empty, and hence become disabling places in a CMW deadlock; this leads to Constraint (8). Constraint (9) specifies the bounds of the variables.

The solution of MIP-$\mathcal{N}_G$, if it exists, is a total-deadlock modified-marking $\overline{M}$, based on which we can construct an empty siphon using Corollary 2. The correctness of the MIP formulation follows as a result of Proposition 3 and Corollary 1, together with the preceding discussion. The number of variables and constraints used by MIP-$\mathcal{N}_G$ is $O(|P|+|T|)$; in particular, the formulation involves $2|P|$ non-negative real variables and $|T|$ non-negative integer variables.

## 5.3 Verification of liveness of $\mathcal{N}_{G1}^c$

The class of Gadara nets $\mathcal{N}_{G1}^c$ shares all the features of $\mathcal{N}_G$. The only difference between $\mathcal{N}_{G1}^c$ and $\mathcal{N}_G$ is that $\mathcal{N}_{G1}^c$ has a set of monitor places $P_C$, whose initial markings may be greater than 1. Observing this difference, the MIP-$\mathcal{N}_G$ formulation presented in (2)–(9) in Section 5.2 can be immediately extended to liveness verification of $\mathcal{N}_{G1}^c$. Although Remark 5 remains true in $\mathcal{N}_{G1}^c$ for any $p \in P_0 \cup P_S \cup P_R$, it generally does not hold for the modified markings of monitor places. Thus, we need to introduce a new constraint on the binary indicator variables associated with the monitor places. For the sake of simplicity, with a slight abuse of notation, we also use the notation $\overline{M}(p)$ to denote the *binary indicator variable* for any $p \in P_C$ in the formulation MIP-$\mathcal{N}_{G1}^c$ presented below. That is, $\overline{M}(p)$ is not necessarily the modified marking for any $p \in P_C$ in MIP-$\mathcal{N}_{G1}^c$. $\overline{M}(p)$ is used as an indicator variable such that if $p$ is not a disabling place at $\overline{M}$, then $\overline{M}(p) = 1$; otherwise, $\overline{M}(p) = 0$.

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

The liveness of $\mathcal{N}_{G1}^c$ can be verified by detecting a total-deadlock modified-marking in the modified reachability space of $\mathcal{N}_{G1}^c$, which can be solved by the following MIP formulation:

**MIP-$\mathcal{N}_{G1}^c$**: In addition to the MIP-$\mathcal{N}_G$ formulation (2)–(9)[4], we also need Constraints (10) and (11) on $\overline{M}(p)$ for any $p \in P_C$.

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

$$\overline{M}(p) \in \{0,1\}, \forall p \in P_C \tag{11}$$

Constraint (10) characterizes the enabling/disabling feature of a monitor place $p \in P_C$ in terms of the binary indicator variable $\overline{M}(p)$. The intuition is explained as follows. Since $\mathcal{N}_{G1}^c$ is an ordinary net, if a monitor place $p \in P_C$ is a disabling place at marking $M$, then $M(p) = 0$, which, together with Constraint (10), forces the corresponding $\overline{M}(p)$ to be 0. On the other hand, if a monitor place $p \in P_C$ in $\mathcal{N}_{G1}^c$ is not a disabling place at marking $M$, then $M(p) \geq 1$, which, together with Constraints (10) and (9), forces the corresponding $\overline{M}(p)$ to be 1. Constraint (11) specifies that $\overline{M}(p)$ is a binary indicator variable associated with any $p \in P_C$.

*Remark 6* A controlled Gadara net ($\mathcal{N}_{G1}^c$ or $\mathcal{N}_G^c$) is obtained by augmenting an original Gadara net $\mathcal{N}_G$. Thus, Constraints (7) and (8) used in MIP-$\mathcal{N}_G$, which are derived based on the definition of $\mathcal{N}_G$, remain true in MIP-$\mathcal{N}_{G1}^c$, presented above, and in MIP-$\mathcal{N}_G^c$, to be presented in the next section.

Similarly to the case of $\mathcal{N}_G$, if $\mathcal{N}_{G1}^c$ is not live, then the solution of MIP-$\mathcal{N}_{G1}^c$ corresponds to a total-deadlock modified-marking, based on which we can construct an empty siphon using Corollary 2. The number of variables and constraints used by MIP-$\mathcal{N}_{G1}^c$ is $O(|P| + |T|)$; in particular, the formulation involves $2|P| - |P_C|$ non-negative real variables, $|P_C|$ binary variables, and $|T|$ non-negative integer variables.

[4] Technically, the notation $M_0$ in (3) should be substituted by $M_0^c$.

## 5.4 Verification of liveness of $\mathcal{N}_G^c$

We know from Definition 6 that $\mathcal{N}_G^c$ is not necessarily ordinary. The potential non-ordinariness makes the liveness verification formulation for $\mathcal{N}_G^c$ more complicated than those for $\mathcal{N}_G$ and $\mathcal{N}_{G1}^c$. In MIP-$\mathcal{N}_G^c$, we need to further introduce a new binary indicator variable, defined as follows.

Let $A(p,t)$ be an indicator variable associated with the directed arc from place $p$ to transition $t$ at modified marking $\overline{M}$. The dependency of $A(p,t)$ on $\overline{M}$ is suppressed in the notation for the sake of simplicity. The value of $A(p,t)$ is defined as:

$$A(p,t) = \begin{cases} 1, \text{ if place } p \text{ enables transition } t \text{ at } \overline{M}; \\ 0, \text{ if place } p \text{ disables transition } t \text{ at } \overline{M}. \end{cases} \tag{12}$$

If $A(p,t) = 1$, then the arc $(p,t)$ is said to be an *enabled arc*; otherwise, it is said to be a *disabled arc*. Note that the potential non-ordinariness in $\mathcal{N}_G^c$, which motivates the introduction of the indicator variable $A(p,t)$, can only be caused by the associated arcs of the monitor places. Therefore, we only need to introduce the indicator variable $A(p,t)$ for place-transition pairs $(p,t)$ such that $p \in P_C$ and $t \in p\bullet$.

Similar to MIP-$\mathcal{N}_{G1}^c$, we use $\overline{M}(p)$ as a binary indicator variable associated with $p \in P$ in the MIP-$\mathcal{N}_G^c$ formulation. That is, if $p$ is not a disabling place at $\overline{M}$, then $\overline{M}(p) = 1$; otherwise, $\overline{M}(p) = 0$. In the formulation, for any $p \in P_0 \cup P_S \cup P_R$, $\overline{M}(p)$ represents *both* the indicator variable associated with $p$ and the modified marking of $p$ (according to Remark 5); for any $p \in P_C$, $\overline{M}(p)$ *only* represents the indicator variable associated with $p$ (a slight abuse of notation as discussed in Section 5.3).

The liveness of $\mathcal{N}_G^c$ can also be verified by detecting a total-deadlock modified-marking in the modified reachability space of $\mathcal{N}_G^c$. This can be solved by the following MIP formulation, MIP-$\mathcal{N}_G^c$, which customizes the generic MIP-RS formulation presented in (Reveliotis, 2005) for maximal RIDM siphon detection in general process-resource nets.

$$\textbf{MIP-}\mathcal{N}_G^c\text{:}\quad \min \sum_{p\in P_S} \overline{M}(p) \tag{13}$$

$$s.t.\quad M = M_0^c + D\sigma \tag{14}$$

$$\overline{M}(p) = M(p), \forall p \in P_S \cup P_R; \overline{M}(p) = 0, \forall p \in P_0 \tag{15}$$

$$\overline{M}(p) = 0, \forall p \in Q, \text{where,} \tag{16}$$

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

$$\sum_{p\in \bullet t \cap P_C} A(p,t) + \sum_{p \in \bullet t \cap (P\setminus P_C)} \overline{M}(p) - |\bullet t| + 1 \leq 0, \tag{17}$$

$$\forall t \;\text{ s.t. } |\bullet t| > 1 \tag{18}$$

$$A(p,t) \geq \frac{M(p) - W(p,t) + 1}{SB(p)}, \tag{19}$$

$$\forall W(p,t) > 0 \text{ s.t. } p \in P_C$$

$$A(p,t) \geq \overline{M}(p), \forall W(p,t) > 0 \text{ s.t. } p \in P_C \tag{20}$$

$$\sum_{t\in p\bullet} A(p,t) - |p\bullet| + 1 \leq \overline{M}(p), \forall p \in P_C \tag{21}$$

$$\sum_{p\in P_S} \overline{M}(p) \geq 2 \tag{22}$$

$$\sum_{p\in P_R} \overline{M}(p) \leq |P_R| - 2 \tag{23}$$

$$M \geq 0; \sigma \in \mathbb{Z}_0^+; \overline{M}(p) \in \{0,1\}, \forall p \in P_C; \tag{24}$$

$$A(p,t) \in \{0,1\}, \forall p \in P_C, \forall t \in p\bullet$$

We explain the MIP-$\mathcal{N}_G^c$ formulation presented in (13)–(24) as follows. The objective function (13) and Constraints (14)–(16), (22), and (23) are the same as their counterparts in MIP-$\mathcal{N}_G$ and MIP-$\mathcal{N}_{G1}^c$. Similar to MIP-$\mathcal{N}_G$ and MIP-$\mathcal{N}_{G1}^c$, the MIP-$\mathcal{N}_G^c$ formulation aims to verify the liveness of $\mathcal{N}_G^c$ by detecting a total-deadlock modified-marking $\overline{M}$. Constraint (16) enforces that the set of transitions, which have only one input place, must be disabled. Moreover, for the set of transitions that have more than one input places, Constraint (17) enforces that at least one input place must be a disabling place. On the other hand, Constraint (19) ensures that the value of $A(p,t)$, which is associated with an enabled arc $(p,t)$ with $p \in P_C$, must be 1. Hence, all variables $A(p,t)$ that are forced to zero by Constraint (17) are indeed variables that correspond to disabled arcs. Constraint (20) recognizes any monitor place, which disables at least one of its outgoing arcs and hence is a disabling place. Further, Constraint (21) recognizes any monitor place, which enables all of its outgoing arcs and hence is not a disabling place. Constraint (24) specifies the bounds of the variables.

If $\mathcal{N}_G^c$ is not live, then the solution of MIP-$\mathcal{N}_G^c$ corresponds to a total-deadlock modified-marking, based on which we can construct a RIDM siphon using Corollary 2. Compared to MIP-$\mathcal{N}_G$ and MIP-$\mathcal{N}_{G1}^c$, the additional complexity in MIP-$\mathcal{N}_G^c$ arises from the variables and constraints associated with the direct arcs $(p, t)$, where $p \in P_C$. Therefore, the number of variables and constraints used by MIP-$\mathcal{N}_G^c$ is $O(|P|+|T|+|P_C||T|)$ in the worst case. In practice, we observe that $|P_C| \ll |P|$ in controlled Gadara net models of real-world software.

*Remark 7* The MIP formulations in Section 5 are presented in a manner that follows the theoretical results, as discussed in Section 5.1. But, more compact ones are possible by reducing the number of variables in the formulations. In the appendix, we present a simplified version of MIP-$\mathcal{N}_G$, originally introduced in Section 5.2. We also discuss the simplification of MIP-$\mathcal{N}_{G1}^c$ (introduced in Section 5.3) and MIP-$\mathcal{N}_G^c$ (introduced in Section 5.4).

## 5.5 Experimental results

In this section, we report the experimental results from a comparative analysis between the performance of the customized algorithms MIP-$\mathcal{N}_{G1}^c$ and MIP-$\mathcal{N}_G^c$ with that of the generic siphon detection algorithms MIP-ES and MIP-RS, respectively, for liveness verification of Gadara nets. The experiments were completed on a Mac OS X laptop with a 2.4 Ghz Intel Core2Duo processor and 2 GB of RAM. The mathematical programming formulations are solved using Gurobi 3.0.1.

We first compare the performance of MIP-$\mathcal{N}_{G1}^c$ with that of MIP-ES presented in (Chu and Xie, 1997). Random Gadara nets for these experiments are generated by a random-walk-style algorithm. At each step, the program randomly decides either to grab a lock or to release one already held; the number of steps is specified as an input parameter. Additional logic is applied to ensure valid behavior. The random Gadara net generator (soon to be available at `http://gadara.eecs.umich.edu/software.html`) is based on our experience in modeling real concurrent programs (Wang, 2009). Furthermore, we apply the MPLE iterative control techniques proposed in (Liao et al, 2010) to synthesize control logic for these random Gadara nets. Monitor places are added to the original Gadara nets by running a random number of control iterations for each net.[5] The resulting controlled Gadara nets, which belong to the class $\mathcal{N}_{G1}^c$, are input to the algorithms MIP-$\mathcal{N}_{G1}^c$ and MIP-ES, for the purpose of liveness verification. Their execution times on these nets are recorded as sample data.

Figure 7(a) shows the sample statistics of the execution times of the two algorithms. We group the samples according to the pair of parameters $(a, s)$ that is used in generating the random Gadara nets, where $a$ is the number of resource acquisitions per subnet, and $s$ is the number of process subnets

[5] For a given Gadara net, if the iterative control technique converges before the preselected random number of iterations are completed, we output the converged net and disregard the remaining iterations.

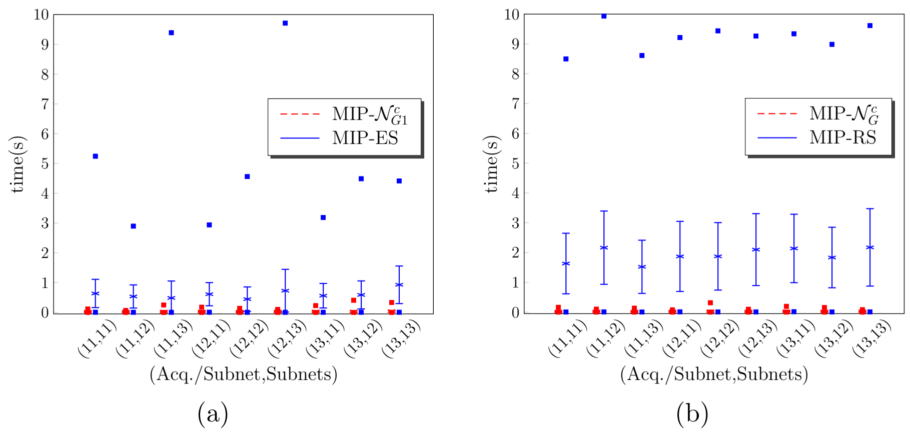


**Fig. 7** Sample statistics: (a) MIP-$\mathcal{N}_{G1}^c$ vs. MIP-ES; (b) MIP-$\mathcal{N}_G^c$ vs. MIP-RS

in the Gadara net. The $x$-axis of the figure shows the nine different groups we studied. The number of monitor places is suppressed, because it varies within a group. We report the average number of monitor places for each group in Table 1 in the appendix. In Fig. 7, the crosses represent the means, the segments represent the half-standard-deviation confidence intervals, and the solid squares represent the maxima or minima.

Next, we analyze the performance of the two algorithms using the Normalized Cumulative Frequency (NCF), which is defined as follows.

$$NCF(x) = \frac{\sum_{i=1}^{n} J_i(x)}{n} \tag{25}$$

where $n$ is the sample size of a group, and $J_i(x)$ is an indicator variable associated with the $i$-th sample and is a function of $x$ ($x \geq 0$), such that

$$J_i(x) = \begin{cases} 1, \text{ if the value of the } i\text{-th sample} \leq x; \\ 0, \text{ otherwise.} \end{cases} \tag{26}$$

The NCFs of our experiments on MIP-$\mathcal{N}_{G1}^c$ and MIP-ES are shown in Fig. 8(a), where the $x$-axis is on a log scale.

We also compare the performance of MIP-$\mathcal{N}_G^c$ with that of MIP-RS presented in (Reveliotis, 2005). In this case, we apply the Empty-Siphon-Based Control Algorithm, described in Section IV-A.1 of (Liao et al, 2010), to the Gadara nets, and choose the controlled Gadara nets that are non-ordinary and belong to the class of $\mathcal{N}_G^c$. These nets are input to the two algorithms, for the purpose of liveness verification. Similarly, the sample statistics are shown in Fig. 7(b) and the NCFs are shown in Fig. 8(b).

From the above analysis, we observe in Fig. 7 that the proposed customized algorithms are more efficient for liveness verification of Gadara nets than the generic siphon detection algorithms in all the nine groups in terms of means,

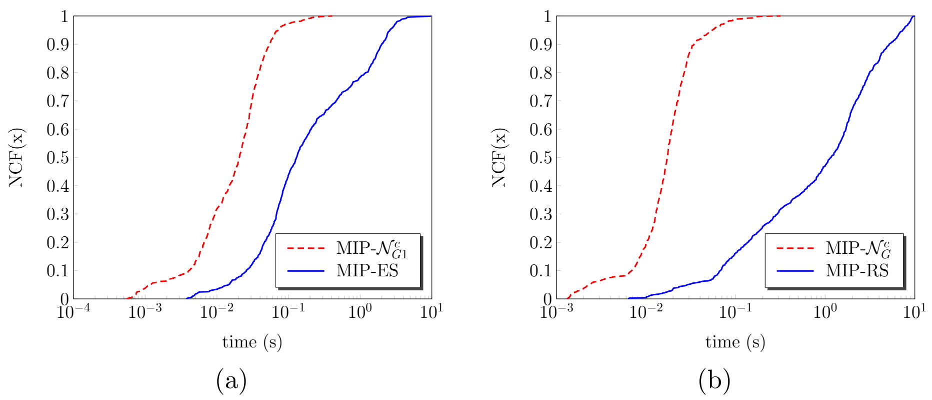


**Fig. 8** Normalized Cumulative Frequency (NCF): (a) MIP-$\mathcal{N}_{G1}^c$ vs. MIP-ES; (b) MIP-$\mathcal{N}_G^c$ vs. MIP-RS

standard deviations, and ranges. From Fig. 8, we find that for MIP-$\mathcal{N}_{G1}^c$, 98% of the samples are smaller than 0.1 second, while for MIP-ES, 40% of the samples are; further, for MIP-$\mathcal{N}_G^c$, 99% of the samples are smaller than 0.1 second, while for MIP-RS, only 15% of the samples are. More detailed statistics of the experimental results are reported in Table 1 in the appendix, from which we also see that the proposed customized algorithms seldom timed out, while the generic algorithms timed out more often. Moreover, for the nets where the proposed customized algorithms timed out, the generic algorithms also timed out. Since MIP-$\mathcal{N}_{G1}^c$ and MIP-$\mathcal{N}_G^c$ were formulated to exploit the structural properties of Gadara nets, it is not surprising that they outperform MIP-ES and MIP-RS, respectively. What is encouraging is that the results in Figs. 7 and 8 and in Table 1 demonstrate that MIP-$\mathcal{N}_{G1}^c$ and MIP-$\mathcal{N}_G^c$ are scalable to large nets, which make them attractive for analyzing deadlock freeness in large software programs.

## 6 Linear Separability of the Safe Region

In view of Theorem 1 which characterizes the liveness of Gadara nets in terms of RIDM siphons, a marking $M$ is said to be *unsafe*, (i) if there exists at least one RIDM siphon at $\overline{M}$, or (ii) if starting from $M$, the net will *unavoidably* reach another marking $M'$ such that there exists at least one RIDM siphon at $\overline{M'}$. If a marking $M$ does not satisfy Conditions (i) and (ii) above, it is said to be *safe*. If the reachable state space of a Gadara net $\mathcal{N}_G^c$ contains RIDM siphons, we want to control the net so that its reachable markings remain within the subspace of safe markings; in the following, we refer to this subspace as the *safe region* of (the state space of) the net.

Therefore, the control synthesis problem can be considered as a classification problem, where monitor places are synthesized to *separate* the safe states from the unsafe states. In particular, if the specifications associated with the monitor places are in the form of linear inequalities, the problem is reduced to a linear classification problem, which can be efficiently implemented by the SBPI technique as discussed in Section 3.3. The key question is whether linear specifications are sufficient for our purpose. In this section, we introduce the notion of linear separability, and formally establish that the safe region of a Gadara net is always linearly separable.

### 6.1 The notion of linear separability

As discussed in Section 3.3, the SBPI technique can enforce any specification in the form of a linear inequality, by adding a monitor place to the Petri net (Yamalidou et al, 1996; Giua, 1992; Iordache and Antsaklis, 2006). Thus, the control specifications of SBPI can be represented as a set of linear constraints $\{(l_k, b_k), k = 1, 2, \ldots\}$ that is enforced on the net markings, where for any $k$, $l_k$ is a weight vector, and $b_k$ is a scalar. The set of control specifications requires that after the imposition of monitor places, the linear constraint

$$l_k^T M \leq b_k \tag{27}$$

must be satisfied by the controlled net for any reachable marking $M$ and every $k$.

The safe region of a net is said to be *linearly separable*, if the set of safe states and the set of unsafe states can always be separated by a finite set of linear constraints in the form of (27). However, sometimes the safe region of a net may not be linearly separable (Giua et al, 1992). Figure 9 shows a counter-example. Define $M_1$ to be the marking where $p_{11}$, $r_B$ and $p_{0_2}$ all have two tokens while all other places are empty; define $M_2$ to be the marking where $p_{21}$, $r_A$ and $p_{0_1}$ all have two tokens while all other places are empty. Let marking $M$ be $M = 0.5M_1 + 0.5M_2$. All these three markings are reachable from the initial marking illustrated in the figure, and only $M$ is unsafe. Apparently there is no linear constraint that *separates* $M$ from $M_1$ and $M_2$. Note that the net in Fig. 9 is not a Gadara net because $P_R$ is missing. $P_R$ cannot be empty because, according to Condition 7 in Definition 5, every place in $P_S$ has to be a support place of a P-semiflow $Y_r$, $r \in P_R$.

The above notion of linear separability is closely related to the concept of *convexity* in linear algebra, except for the fact that our state space is discrete. It is well known that the solution to a set of linear inequalities is a convex region. If the safe region is not linearly separable, *maximally permissive* control is generally not feasible through monitor places. A common approach in this case is to sacrifice maximal permissiveness and synthesize control logic that constrains the controlled reachable state space to a convex safe subregion (Li et al, 2008).

In the next section, we show that the state space of a Gadara net is always linearly separable, and therefore MPLE supervision through monitor places is feasible.

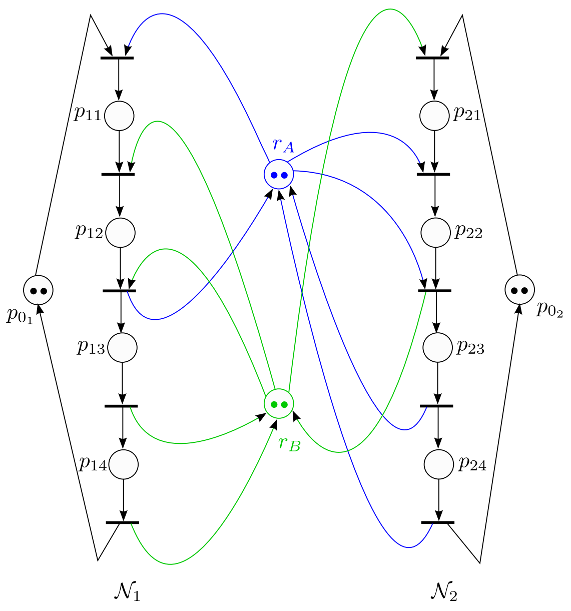


**Fig. 9** Counter example: a net with non-convex safe region

## 6.2 Linear separability of state space of Gadara nets

Similarly to the notion of modified marking used in Section 4.2, we define the notion of $P_S$-marking to facilitate the ensuing discussion.

**Definition 16** Given a Gadara net $\mathcal{N}_G^c$ and a marking $M \in R(\mathcal{N}_G^c, M_0^c)$, the *$P_S$-marking* $\overline{\overline{M}}$ is defined by

$$\overline{\overline{M}}(p) = \begin{cases} M(p), & \text{if } p \in P_S; \\ 0, & \text{if } p \notin P_S. \end{cases} \tag{28}$$

$P_S$-markings essentially "erase" the tokens in idle places, resource places, and monitor places, retaining only tokens in operation places. As in the case of modified markings, this projection does not introduce any ambiguity. There is a one-to-one mapping between the original marking and the $P_S$-marking, i.e., $M_1 = M_2$ if and only if $\overline{\overline{M}}_1 = \overline{\overline{M}}_2$. More specifically, the number of tokens in places $P_R$ and $P_C$ can be recovered from the invariants respectively established by Conditions 5 and 8 in Definitions 5 and 6; the number of tokens in places $P_0$ can be recovered in a similar manner as for modified markings. Therefore, we consider linear constraints for $P_S$-markings only, i.e., the coefficients corresponding to places $P_0$, $P_R$, and $P_C$ are all zero.

From Proposition 3, we know that $\overline{\overline{M}}$ is a binary vector, which is a key result to establish the linear separability of state space of Gadara nets. In a binary vector space, we can always construct a linear constraint to separate exactly one binary vector from all other binary vectors of the same dimension. This leads to the following theorem.

**Theorem 5** *Given a Gadara net $\mathcal{N}_G^c$ and a set of markings $V \subseteq R(\mathcal{N}_G^c, M_0^c)$, there exists a finite set of linear constraints $LC(V) = \{(l_1, b_1), (l_2, b_2), ...\}$ such that $M \in V$ iff $\forall (l_i, b_i) \in LC(V),\ l_i^T M \le b_i$.*

*Proof:* We will carry out the proof by construction. From the definition of $P_S$-marking, we know that any marking is uniquely characterized by its corresponding $P_S$-marking. Thus, for any marking $M' \notin V$, we can focus our attention on the associated $P_S$-marking $\overline{\overline{M'}}$. Moreover, we know that any $P_S$-marking is a binary vector, i.e., its component is either 0 or 1. We can construct the linear constraint associated with $M'$ based on $\overline{\overline{M'}}$ as follows.

$$l(p) := \begin{cases} -1, \text{ if } \overline{\overline{M'}}(p) = 0 \\ \ \ 1, \text{ if } \overline{\overline{M'}}(p) = 1 \\ \ \ 0, \text{ if } p \notin P_S \end{cases} ; \ b := \sum_{p \in P_S} \overline{\overline{M'}}(p) - 1 \tag{29}$$

Observe that the coefficient vector $l$ and scalar $b$ specified in (29) satisfy: $l^T M' = b + 1 > b$. If we change *any* component in $\overline{\overline{M'}}$, and obtain a different marking $M$ (i.e., $M'$ and $M$ differ in exactly one component), then the choice of $l$ and $b$ in (29) guarantees that $M$ satisfies: $l^T M = l^T M' - 1 = b$. In general, every time we change a component in $\overline{\overline{M'}}$ and obtain a resulting new marking $M$, the weighted sum $l^T M$ of the new marking will decrease by 1 as compared to the weighted sum $l^T M'$ before the change.

Any marking $M \neq M'$ can be considered as being obtained from the above thought process by changing a set of components of $\overline{\overline{M'}}$. As a result, any marking $M \neq M'$ satisfies the linear inequality $l^T M \le b$; and, $M'$ is the only marking that does not satisfy this linear inequality. In other words, if we enforce the linear inequality constraint $l^T M \le b$ on the net, then we *only* prevent one single marking $M'$ from being reachable and nothing else.

Therefore, we can construct such a linear constraint for every marking in $R(\mathcal{N}_G^c, M_0^c) \backslash V$. Since the reachable state space of $\mathcal{N}_G^c$ is finite, containing no more than $2^{|P_S|}$ states, $R(\mathcal{N}_G^c, M_0^c) \backslash V$ is finite as well, and there is a finite set of linear constraints that separates $V$ from its complement in the reachable state space of $\mathcal{N}_G^c$. □

## 6.3 Implication for control synthesis

Separating the safe region of a Gadara net using linear constraints is a special case of Theorem 5.

**Corollary 3** *The safe region of a Gadara net $\mathcal{N}_G^c$ can be separated by a finite set of constraints $LC = \{(l_1, b_1), (l_2, b_2), ...\}$ such that $M \in R(\mathcal{N}_G^c, M_0^c)$ is safe iff $\forall (l_i, b_i) \in LC,\ l_i^T M \le b_i$.*

Corollary 3 establishes the feasibility for MPLE control of Gadara nets through monitor places. That is, in principle, for every unsafe state $M_u$ we want to prevent, we can use a constraint in the form of (27) with its coefficients specified as (29), which can in turn be enforced by SBPI through a monitor place. The controlled net with this augmented monitor place guarantees that $M_u$ is not reachable. (The example of Fig. 9 also implies that the unity initial marking of the resource places is critical for this result.) Consequently, we can, in principle, employ SBPI to prevent all the unsafe markings from becoming reachable, on a marking-by-marking basis. However, this method is in general not practical in applications. In this regard, we mention that we have developed an efficient siphon-based MPLE control synthesis algorithm for Gadara nets, which exploits Corollary 3 (Liao et al, 2010, 2011); in parallel, we have also developed an alternative approach based on state space expansion and classification theory (Nazeem et al, 2010, 2011). When applied to the BIND example in Fig. 2(b), both methodologies synthesize the same control specification, given in (30). Using SBPI, we obtain the monitor place $p_c$ that enforces (30), as shown in Fig. 10.

$$p_1 + p_4 \leq 1 \tag{30}$$

## 7 Case Studies of Deadlock in Open Source Software

In addition to BIND, whose deadlock bug is used as a running example in this paper, we have used our model-based approach to perform deadlock analysis of several open-source programs so far in the Gadara project. These case studies demonstrate the benefits of a formal, model-based approach in providing an accurate and compact characterization of a deadlock scenario and in enabling systematic deadlock analysis using the techniques presented in this paper.

OpenLDAP is a popular open-source implementation of the Lightweight Directory Access Protocol (LDAP). We built the Gadara net model of version 2.2.20 of `slapd`, which is a high-performance multithreaded network server program of OpenLDAP, and has a confirmed CMW deadlock bug. The `slapd` program has 1,795 functions, of which 456 remain after the pruning process discussed in Section 3.2 (Wang et al, 2008). The discovery of the CMW deadlock bug in OpenLDAP by analyzing its Gadara net model is discussed in (Wang et al, 2010).

Apache, formally known as Apache HTTP Server, is an open-source web server software. We built the Gadara net model of Apache `httpd` version 2.2.8. Analysis of this model revealed no CMW deadlock in the software, which is consistent with the data in the Apache bug database (Wang et al, 2009a).

In the rest of this section, we discuss in detail a deadlock bug in version 2.5.62 of the Linux kernel that is captured in its Gadara net model. The deadlock example is inspired by the study conducted in (Engler and Ashcraft, 2003). Figure 11 shows this deadlock example. We annotated the lines of code that are related to lock allocations and releases. Each annotation explains the

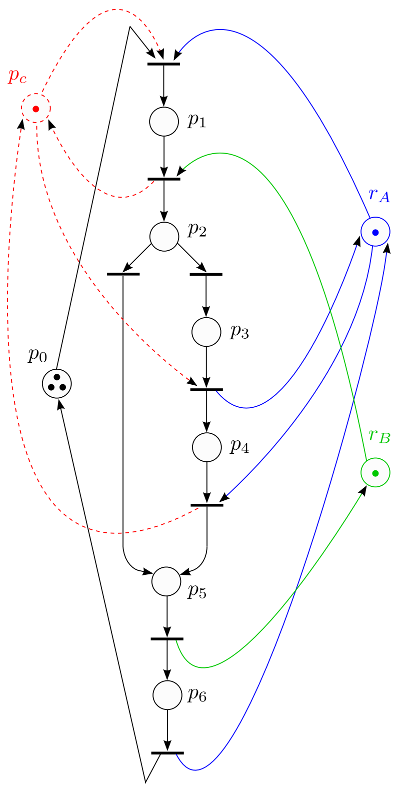


**Fig. 10** A deadlock example in BIND: controlled Gadara net model

specifics of the corresponding line of code using four components: lock/unlock action, file name, function name, and line number in the code. The deadlock involves three threads and three locks. Further, Thread 1 involves a six-level call chain, and Thread 2 calls two functions. We have inlined the chains of function calls and simplified the control flows, so that only the code that is relevant to the deadlock is presented in Fig. 11.

The Gadara net model of the considered lines of code is shown in Fig. 12. Analysis of this model using the techniques presented in this paper reveals two total-deadlock markings that are reachable from the initial marking as depicted in the figure: (i) The first total-deadlock marking is $M_1$, where there is one token in $p_{12}$, one in $p_{22}$, and one in $p_{33}$, while all other places are empty. At marking $M_1$, all three threads are involved in the deadlock. (ii) The second total-deadlock marking is $M_2$, 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 $M_2$, only Threads 1 and 2 are involved in the deadlock. As we can see, the original deadlock in the program, which involves chains of function calls and complicated branchings, is clearly captured in this Gadara net model, which lays the groundwork for formal deadlock analysis.

```
/*** Thread 1 ***/
spin_lock(&im->lock);               /* LOCK(A), igmp.c, igmp_timer_expire(), 268 */
...
if (!fl.fl4_src){
   ...
   read_lock(&inetdev_lock);        /* LOCK(B), devinet.c, inet_select_addr(), 786 */

   for (...){
      ...
      read_lock(&in_dev->lock);     /* LOCK(C), devinet.c, inet_select_addr(), 791 */
      ...
      if (...){
         read_unlock(&in_dev->lock); /* UNLOCK(C), devinet.c, inet_select_addr(), 795 */
         ...
         break;
      }
      ...
      read_unlock(&in_dev->lock);   /* UNLOCK(C), devinet.c, inet_select_addr(), 800 */
   }
   ...
   read_unlock(&inetdev_lock);      /* UNLOCK(B), devinet.c, inet_select_addr(), 803 */
   ...
}
...
spin_unlock(&im->lock);             /* UNLOCK(A), igmp.c, igmp_timer_expire(), 289 */

/*** Thread 2 ***/
read_lock(&in_dev->lock);           /* LOCK(C), igmp.c, igmp_heard_query(), 338 */
for (...){
   ...
   spin_lock_bh(&im->lock);         /* LOCK(A), igmp.c, igmp_mod_timer(), 165 */
   ...
   spin_unlock_bh(&im->lock);       /* UNLOCK(A), igmp.c, igmp_mod_timer(), 171 & 177 */
}
read_unlock(&in_dev->lock);         /* UNLOCK(C), igmp.c, igmp_heard_query(), 346 */

/*** Thread 3 ***/
read_lock(&inetdev_lock);           /* LOCK(B), devinet.c, inet_select_addr(), 759 */
...
if (!in_dev){
   read_unlock(&inetdev_lock);      /* UNLOCK(B), devinet.c, inet_select_addr(), 808 */
   return addr;
}
read_lock(&in_dev->lock);           /* LOCK(C), devinet.c, inet_select_addr(), 764 */
...
read_unlock(&in_dev->lock);         /* UNLOCK(C), devinet.c, inet_select_addr(), 775 */
read_unlock(&inetdev_lock);         /* UNLOCK(B), devinet.c, inet_select_addr(), 776 */
...
return addr;
```

**Fig. 11** A deadlock example in the Linux kernel, simplified code

## 8 Conclusion

Fear of deadlock distorts software development and diverts energy from more profitable pursuits, e.g., by intimidating programmers into adopting cautious coarse-grained locking when multicore performance demands deadlock-prone fine-grained locking. Deadlock in lock-based programs is difficult to reason about because locks are not composable: Deadlock-free lock-based software components may interact to deadlock in a larger program (Sutter and Larus, 2005). Non-composability therefore undermines the cornerstones of programmer productivity: software modularity and divide-and-conquer problem de-

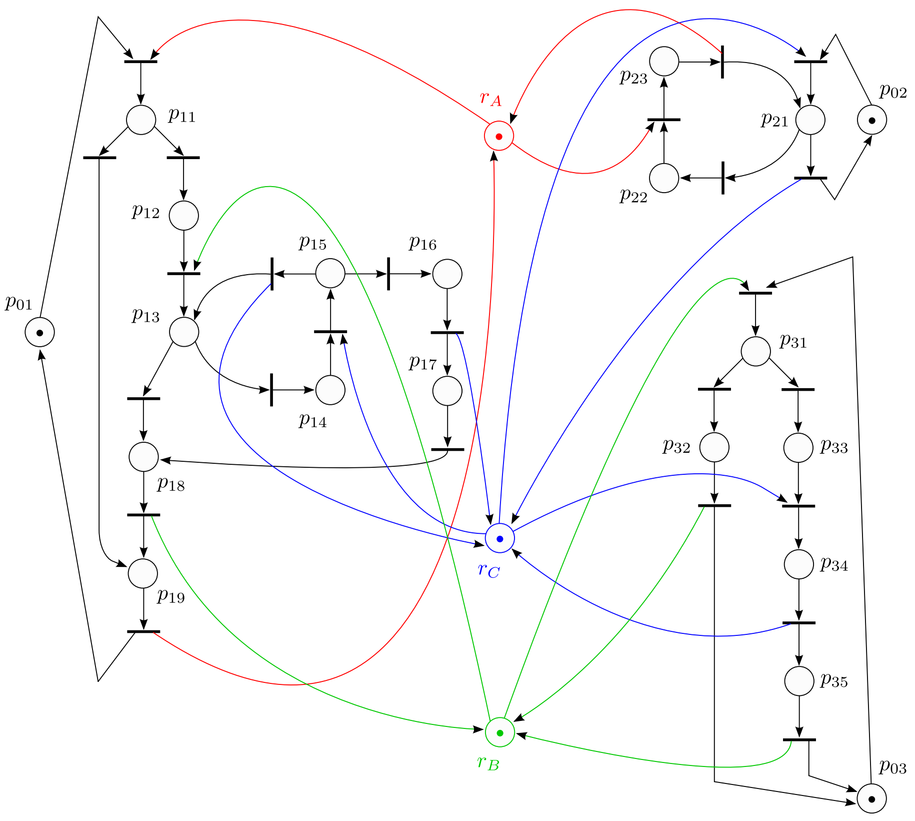


**Fig. 12** A deadlock example in the Linux kernel, Gadara net model

composition. In addition, insidious corner-case deadlocks may lurk even within single modules that are developed by individual expert programmers (Engler and Ashcraft, 2003); such bugs can be difficult to detect, and repairing them is a costly, manual, time-consuming, and error-prone chore. The above challenges have motivated the formal model-based approach that we have adopted in the Gadara project to develop a software tool that will automatically instrument a given program to provably ensure deadlock freeness at run-time.

This paper has presented our results on modeling and analysis of multithreaded programs for the purpose of deadlock analysis, which are at the basis of the Gadara software tool. Specifically, we have defined a new class of Petri nets, called Gadara nets, to systematically model lock allocation and release in this programming paradigm. We have established a set of important properties of Gadara nets. The liveness and reversibility properties provide a means to map the behavioral objective of deadlock freeness of a program to the structural requirement on its corresponding Gadara net model, which in turn lays the foundations for structure-based MPLE control synthesis of Gadara nets. The linear separability property further shows the feasibility of

MPLE control synthesis. We have proposed a set of customized algorithms for liveness verification of Gadara nets, and compared their performance with generic MIP-based siphon detection algorithms that are well-known in the literature. Our future work will report on the control synthesis framework and customized techniques that we have adopted for Gadara, on the basis of the results in this paper for the class of Gadara nets.

**Acknowledgements** We thank Ahmed Nazeem and Manjunath Kudlur for many helpful comments.

## References

Allen LV (2010) Verification and anomaly detection for event-based control of manufacturing systems. PhD thesis, University of Michigan

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

Cano EE, Rovetto CA, Colom JM (2010) An algorithm to compute the minimal siphons in $S^4PR$ nets. In: Proc. International Workshop on Discrete Event Systems, pp 18–23

Cassandras CG, Lafortune S (2008) Introduction to Discrete Event Systems, 2nd edn. Springer, Boston, MA

Chu F, Xie XL (1997) Deadlock analysis of Petri nets using siphons and mathematical programming. IEEE Transactions on Robotics and Automation 13(6):793–804

Delaval G, Marchand H, Rutten E (2010) Contracts for modular discrete controller synthesis. In: Proc. ACM Conference on Languages, Compilers and Tools for Embedded Systems

Dijkstra EW (1982) Selected Writings on Computing, Springer-Verlag, chap The Mathematics Behind the Banker's Algorithm, pp 308–312

Dragert C, Dingel J, Rudie K (2008) Generation of concurrency control code using discrete-event systems theory. In: Proc. ACM International Symposium on Foundations of Software Engineering

Engler D, Ashcraft K (2003) RacerX: Effective, static detection of race conditions and deadlocks. In: Proc. the 19th ACM Symposium on Operating Systems Principles

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

Ezpeleta J, García-Vallés F, Colom JM (2002) A banker's solution for deadlock avoidance in FMS with flexible routing and multiresource states. IEEE Transactions on Robotics and Automation 18(4):621–625

Flanagan C, Leino KRM, Lillibridge M, Nelson G, Saxe JB, Stata R (2002) Extended static checking for Java. In: Proc. the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation

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

Giua A (1992) Petri nets as discrete event models for supervisory control. PhD thesis, Rensselaer Polytechnic Institute

Giua A, DiCesare F, Silva M (1992) Generalized mutual exclusion constraints on nets with uncontrollable transitions. In: Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics, pp 974–979

Hopcroft JE, Motwani R, Ullman JD (2006) Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison Wesley

Iordache MV, Antsaklis PJ (2006) Supervisory Control of Concurrent Systems: A Petri Net Structural Approach. Birkhäuser, Boston, MA

Iordache MV, Antsaklis PJ (2009) Petri nets and programming: A survey. In: Proc. 2009 American Control Conference, pp 4994–4999

Iordache MV, Antsaklis PJ (2010) Concurrent program synthesis based on supervisory control. In: Proc. 2010 American Control Conference, pp 3378–3383

Jeng M, Xie X (2001) Modeling and analysis of semiconductor manufacturing systems with degraded behaviors using Petri nets and siphons. IEEE Transactions on Robotics and Automation 17(5):576–588

Kavi KM, Moshtaghi A, Chen D (2002) Modeling multithreaded applications using Petri nets. International Journal of Parallel Programming 35(5):353–371

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

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

Liao H, Lafortune S, Reveliotis S, Wang Y, Mahlke S (2010) Synthesis of maximally-permissive liveness-enforcing control policies for Gadara Petri nets. In: Proc. the 49th IEEE Conference and Decision and Control

Liao H, Stanley J, Wang Y, Lafortune S, Reveliotis S, Mahlke S (2011) Deadlock-avoidance control of multithreaded software: An efficient siphon-based algorithm for Gadara Petri nets. In: Submitted for conference publication

Liu C, Kondratyev A, Watanabe Y, Desel J, Sangiovanni-Vincentelli A (2006) Schedulability analysis of Petri nets based on structural properties. In: Proc. International Conference on Application of Concurrency to System Design

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

Murata T, Shenker B, Shatz SM (1989) Detection of Ada static deadlocks using Petri net invariants. IEEE Transactions on Software Engineering 15(3):314–326

Musuvathi M, Qadeer S, Ball T, Basler G, Nainar PA, Neamtiu I (2008) Finding and reproducing Heisenbugs in concurrent programs. In: Proc. the 8th USENIX Symposium on Operating Systems Design and Implementation

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

Nazeem A, Reveliotis S, Wang Y, Lafortune S (2011) Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: the linear case. to appear in IEEE Transactions on Automatic Control

Nir-Buchbinder Y, Tzoref R, Ur S (2008) Deadlocks: From exhibiting to healing. In: Proc. Workshop on Runtime Verification

Novark G, Berger ED, Zorn BG (2007) Exterminator: Automatically correcting memory errors with high probability. In: Proc. Programming Language Design and Implementation

Novark G, Berger ED, Zorn BG (2008) Exterminator: Automatically correcting memory errors with high probability. Communications of the ACM 51(12):87–95

Park J, Reveliotis SA (2001) Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. IEEE Transactions on Automatic Control 46(10):1572–1583

Park J, Reveliotis SA (2002) Liveness-enforcing supervision for resource allocation systems with uncontrollable behavior and forbidden states. IEEE Transactions on Robotics and Automation 18(2):234–240

Park S, Lu S, Zhou Y (2009) Ctrigger: Exposing atomicity violation bugs from their hiding places. In: Proc. 14th International Conference on Architecture Support for Programming Languages and Operating Systems

Qin F, Tucek J, Sundaresan J, Zhou Y (2005) Rx: treating bugs as allergies—a safe method to survive software failures. In: Proc. the 20th ACM Symposium on Operating Systems Principles, pp 235–248

Reisig W (1985) Petri Nets: An Introduction. Springer-Verlag

Reveliotis SA (2005) Real-Time Management of Resource Allocation Systems: A Discrete-Event Systems Approach. Springer, New York, NY

Silberschatz A, Galvin PB, Gagne G (2008) Operating System Concepts, 8th edn. Wiley

Sutter H, Larus J (2005) Software and the concurrency revolution. ACM Queue 3(7):54–62

Wang Y (2009) Software failure avoidance using discrete control theory. PhD thesis, University of Michigan

Wang Y, Kelly T, Kudlur M, Lafortune S, Mahlke SA (2008) Gadara: Dynamic deadlock avoidance for multithreaded programs. In: Proc. the 8th USENIX Symposium on Operating Systems Design and Implementation, pp 281–294

Wang Y, Lafortune S, Kelly T, Kudlur M, Mahlke S (2009a) The theory of deadlock avoidance via discrete control. In: Proc. the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp 252–263

Wang Y, Liao H, Reveliotis S, Kelly T, Mahlke S, Lafortune S (2009b) Gadara nets: Modeling and analyzing lock allocation for deadlock avoidance in multithreaded software. In: Proc. the 48th IEEE Conference and Decision and

Control, pp 4971–4976

Wang Y, Cho HK, Liao H, Nazeem A, Kelly TP, Lafortune S, Mahlke S, Reveliotis S (2010) Supervisory control of software execution for failure avoidance: Experience from the Gadara project. In: Proc. International Workshop on Discrete Event Systems

Yamalidou K, Moody J, Lemmon M, Antsaklis P (1996) Feedback control of Petri nets based on place invariants. Automatica 32(1):15–28

## A Appendix

### A.1 Proof of Theorem 2

*Proof:* From Theorem 1, $\mathcal{N}_G^c$ is not live if and only if there exists a marking $M \in R(\mathcal{N}_G^c, M_0)$ and $M \neq M_0$, such that its modified marking $\overline{M}$ contains a RIDM siphon $S$. From the proof of Theorem 1, we further know that $S$ is constructed by the set of disabling places at $\overline{M}$. For an ordinary net, a disabling place is essentially a place with no tokens. Since every place $p \in S$ is a disabling place, $\overline{M}(p) = 0$, $\forall p \in S$. Hence, $S$ is an empty siphon at $\overline{M}$.

Next we show that the presence of the resource-induced empty siphon $S$ at $\overline{M}$ implies the presence of an empty siphon $S'$ at the original marking $M$. Let

$$S' = \{w : w \in S \cap (P_R \cup P_C)\} \cup \{p \in P_S : M(p) = \overline{M}(p) = 0 \wedge \ \exists w \text{ s.t. } w \in S \cap (P_R \cup P_C) \wedge y_w(p) > 0\}$$

where, $y_w(p) > 0$ *iff* the operation place $p$ needs the allocation of tokens from $w$. Note that $S' \neq \emptyset$, since $S$ is a resource-induced empty siphon. Furthermore, $M(p) = 0$, $\forall p \in S$. Next we show that $S'$ is also a siphon, by considering the following two main cases:

Case 1: Consider $t \in \bullet w$ for some place $w \in S \cap (P_R \cup P_C)$. Then, by the definition of siphon, $\exists q \in S$ such that $t \in q\bullet$. If $q \in P_R \cup P_C$, then $q \in \{w : w \in S \cap (P_R \cup P_C)\} \subset S'$. On the other hand, if $q \notin P_R \cup P_C$, then $q \in P_S$. Furthermore, $y_w(q) > 0$, and $M(q) = 0$ (since $q \in S$). Therefore, $q \in \{p \in P_S : M(p) = \overline{M}(p) = 0 \wedge \exists w \text{ s.t. } w \in S \cap (P_R \cup P_C) \wedge y_w(p) > 0\} \subset S'$. So, $t \in S'\bullet$.

Case 2: Consider $t \in \bullet q$ for some $q \in P_S$ with $M(p) = \overline{M}(p) = 0 \wedge \exists w \text{ s.t. } w \in S \cap (P_R \cup P_C) \wedge y_w(p) > 0$. Let us consider to the following two subcases.

(i) If $\exists w \text{ s.t. } w \in S \cap (P_R \cup P_C) \wedge t \in w\bullet$, then, $t \in \{w : w \in S \cap (P_R \cup P_C)\}\bullet \subseteq S'\bullet$.

(ii) Otherwise, $\exists q' \in P_S \cap \bullet t$ with $\overline{M}(q') = 0$. Furthermore, since $y_w(q) > 0$, it must be that $y_w(q') > 0$ (i.e., the operation of place $q$ needs some tokens from $w$ in order to be executed, but since $w \notin \bullet(\bullet q)$, by the assumption of this subcase, there must exist an upstream operation place $q'$ which will "pass" these tokens to $q$). It needs to be pointed out that such an upstream operation place $q' \notin P_0$, because idle places do not hold any tokens from $w$, by the definition of $\mathcal{N}_G^c$. Therefore, $t \in \{p \in P_S : M(p) = \overline{M}(p) = 0 \wedge \ \exists w$ s.t. $w \in S \cap (P_R \cup P_C) \wedge y_w(p) > 0\}\bullet \subseteq S'\bullet$.

In both cases, $\forall t \in \bullet S'$, $t \in S'\bullet$. Thus, $S'$ is a siphon. □

## A.2 Proof of Theorem 3

*Proof:* First we show the "⇒" direction.

Given a marking $M \in R(\mathcal{N}_G^c, M_0)$ with $M \neq M_0$, consider a non-empty place $p \in P_S$ and its corresponding process subnet $\mathcal{N}_i$. The strong connectivity of $\mathcal{N}_i$ implies that there is a path (i.e., a sequence of feasible transitions) from $p$ to $p_{0_i}$. Let $t'$ denote the transition in that path with $t' \in \bullet p_{0_i}$.

The assumed liveness of the net implies that starting from $M$, we shall eventually be able to fire $t'$. Furthermore, the activation of the aforementioned sequence of feasible transitions does not have to involve any of the tokens in $M(p_{0_i})$. Thus, the token in $p$ at marking $M$ can eventually be collected into $p_{0_i}$.

Since the above argument holds for any non-empty place at any marking $M \in R(\mathcal{N}_G^c, M_0)$, and the total number of tokens in $P_S$ at $M$ is finite, we shall eventually be able to collect all the tokens in $P_S$ at marking $M$ into $P_0$. Denote this last marking as $M'$. Combined with Condition 5 of Definition 5, it follows that $M' = M_0$.

Next we show the "⇐" direction.

We discussed in Section 4.1 that the resource and monitor-place-augmented subnets in $\mathcal{N}_G^c$ are quasi-live. This property, together with the assumed reversibility of the net, implies that $\mathcal{N}_G^c$ is live. □

## A.3 Simplification of MIP formulations

Define $T_0 := P_0\bullet$, which is the set of output transitions of all the idle places. In MIP-$\mathcal{N}_G$ introduced in Section 5.2, our goal is to detect a total-deadlock modified-marking for the purpose of liveness verification. According to Definition 15, we know that all the transitions in $T_0$ are disabled under *any* modified marking. Therefore, in the simplified version of MIP-$\mathcal{N}_G$, denoted as MIP-$\mathcal{N}_G$-2, we can "ignore" the set of transitions $T_0$ and their input places $P_0$, because all the transitions in $T_0$ will always be disabled in the sought modified marking. Recall from Definition 5 and Proposition 3 that $M(p)$ is either 0 or 1, for any $P \in P_S \cup P_R$. We can use $M(p)$ as a binary indicator variable in MIP-$\mathcal{N}_G$-2, with $M(p)$ defined on $P_S \cup P_R$. The components of $M$ corresponding to idle places can be ignored and are irrelevant to the solution of MIP-$\mathcal{N}_G$-2.

$$\textbf{MIP-}\mathcal{N}_G\textbf{-2:} \quad \min \sum_{p\in P_S} M(p) \tag{31}$$

$$s.t. \quad M = M_0 + D\sigma \tag{32}$$

$$M(p) = 0, \forall p \in Q, \text{where,} \tag{33}$$

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

$$\sum_{p\in \bullet t} M(p) - |\bullet t| + 1 \leq 0, \tag{34}$$

$$\forall t \text{ s.t. } (|\bullet t| > 1) \wedge (t \notin T_0) \tag{35}$$

$$\sum_{p\in P_S} M(p) \geq 2 \tag{36}$$

$$\sum_{p\in P_R} M(p) \leq |P_R| - 2 \tag{37}$$

$$M \geq 0; \sigma \in \mathbb{Z}_0^+ \tag{38}$$

Let $M$ be the solution of the formulation MIP-$\mathcal{N}_G$-2, then the sought total-deadlock modified-marking can be obtained using (4). The simplified formulation MIP-$\mathcal{N}_G$-2 has fewer variables than the original formulation MIP-$\mathcal{N}_G$. More specifically, MIP-$\mathcal{N}_G$-2 involves $|P|$ non-negative real variables and $|T|$ non-negative integer variables, resulting in a reduction of $|P|$ non-negative real variables as compared to MIP-$\mathcal{N}_G$, i.e., the variables $\overline{M}(p)$, used in MIP-$\mathcal{N}_G$, disappear in MIP-$\mathcal{N}_G$-2.

The simplifications of MIP-$\mathcal{N}_{G1}^c$ and MIP-$\mathcal{N}_G^c$ can also be pursued in the same spirit. To simplify MIP-$\mathcal{N}_{G1}^c$, we can make a similar adjustment[6] as in MIP-$\mathcal{N}_G$-2, and further introduce binary variables for the monitor places as in (10) and (11). MIP-$\mathcal{N}_G^c$ can also be simplified in the same manner[7] as for MIP-$\mathcal{N}_G$ and MIP-$\mathcal{N}_{G1}^c$ discussed above.

## A.4 Experimental results of comparative analysis

Table 1 presents a summary of the experimental results of the comparative analysis, conducted in Section 5.5, on the four liveness verification algorithms: MIP-$\mathcal{N}_{G1}^c$, MIP-ES, MIP-$\mathcal{N}_G^c$, and MIP-RS. For each set of parameters (each row in the table), over 100 samples of random Gadara nets are generated.

Consider the comparison between the performance of MIP-$\mathcal{N}_{G1}^c$ and that of MIP-ES. We set a time-out threshold of 10 seconds. A net times out if its liveness cannot be determined by either MIP-$\mathcal{N}_{G1}^c$ or MIP-ES in less than 10 seconds. The number of sample nets that time out is reported in the last

[6] Constraint (34) will be rewritten, since the summation in the Left-Hand-Side now involves both $M(p)$ and $\overline{M}(p)$.

[7] Constraint (17) will be rewritten, since we only consider transition $t$ such that $|t\bullet| > 1$ and $t \notin T_0$.

column (TLE) of the table. All the other statistical data in this table are calculated over only sample nets where both MIP-$\mathcal{N}_{G1}^c$ and MIP-ES did not time out. The comparison between the performance of MIP-$\mathcal{N}_G^c$ and that of MIP-RS is carried out in a similar way.

The first column lists the four algorithms under consideration. The second (a) and third (s) columns are the number of resource acquisitions per subnet and the number of process subnets, which are input parameters to the random program generator. The fourth (P), fifth (T), and sixth (C) columns correspond to the average number of places, transitions, and monitor places in the sample Gadara nets. The seventh (SS) and eighth (US) columns describe the state space complexity, i.e., the average numbers of safe and unsafe states that are reachable in the nets. Note that the solution of the mathematical programming formulations does *not* require the construction of the state space; the numbers of safe and unsafe states were generated separately for the sake of scalability assessment. The last column (TLE) is the proportion of sample nets that did time out.

**Table 1** Experimental results of comparative analysis on liveness verification algorithms

| **Method** | **a** | **s** | **P** | **T** | **C** | **SS** | **US** | **TLE** |
|---|---|---|---|---|---|---|---|---|
| MIP-$\mathcal{N}^c_{G1}$ | 11 | 11 | 87.25 | 68.65 | 7.12 | 230581 | 91889 | 0.01 |
| MIP-ES | | | | | | | | 0.06 |
| MIP-$\mathcal{N}^c_G$ | | | 85.86 | 66.55 | 7.88 | 218741 | 85157 | 0.00 |
| MIP-RS | | | | | | | | 0.22 |
| MIP-$\mathcal{N}^c_{G1}$ | 11 | 12 | 94.84 | 76.08 | 7.15 | 496055 | 221560 | 0.01 |
| MIP-ES | | | | | | | | 0.11 |
| MIP-$\mathcal{N}^c_G$ | | | 93.66 | 73.64 | 8.46 | 444871 | 202773 | 0.00 |
| MIP-RS | | | | | | | | 0.19 |
| MIP-$\mathcal{N}^c_{G1}$ | 11 | 13 | 101.34 | 82.02 | 7.62 | 614988 | 235364 | 0.01 |
| MIP-ES | | | | | | | | 0.10 |
| MIP-$\mathcal{N}^c_G$ | | | 99.61 | 79.68 | 8.26 | 653032 | 274092 | 0.00 |
| MIP-RS | | | | | | | | 0.24 |
| MIP-$\mathcal{N}^c_{G1}$ | 12 | 11 | 89.63 | 71.52 | 6.64 | 291166 | 104067 | 0.01 |
| MIP-ES | | | | | | | | 0.06 |
| MIP-$\mathcal{N}^c_G$ | | | 87.45 | 68.48 | 7.51 | 286145 | 125343 | 0.00 |
| MIP-RS | | | | | | | | 0.16 |
| MIP-$\mathcal{N}^c_{G1}$ | 12 | 12 | 96.01 | 77.58 | 6.87 | 523258 | 203359 | 0.01 |
| MIP-ES | | | | | | | | 0.10 |
| MIP-$\mathcal{N}^c_G$ | | | 95.06 | 75.64 | 7.81 | 535029 | 241084 | 0.00 |
| MIP-RS | | | | | | | | 0.18 |
| MIP-$\mathcal{N}^c_{G1}$ | 12 | 13 | 103.89 | 84.79 | 7.49 | 862689 | 324566 | 0.01 |
| MIP-ES | | | | | | | | 0.06 |
| MIP-$\mathcal{N}^c_G$ | | | 103.14 | 83.05 | 8.41 | 745614 | 310375 | 0.00 |
| MIP-RS | | | | | | | | 0.18 |
| MIP-$\mathcal{N}^c_{G1}$ | 13 | 11 | 93.09 | 73.84 | 7.71 | 254733 | 101207 | 0.01 |
| MIP-ES | | | | | | | | 0.13 |
| MIP-$\mathcal{N}^c_G$ | | | 91.24 | 71.50 | 8.26 | 235609 | 95000 | 0.00 |
| MIP-RS | | | | | | | | 0.22 |
| MIP-$\mathcal{N}^c_{G1}$ | 13 | 12 | 98.50 | 79.62 | 7.28 | 394573 | 155436 | 0.02 |
| MIP-ES | | | | | | | | 0.08 |
| MIP-$\mathcal{N}^c_G$ | | | 97.25 | 77.62 | 8.06 | 398204 | 160820 | 0.00 |
| MIP-RS | | | | | | | | 0.18 |
| MIP-$\mathcal{N}^c_{G1}$ | 13 | 13 | 105.34 | 85.62 | 7.99 | 716595 | 314641 | 0.01 |
| MIP-ES | | | | | | | | 0.04 |
| MIP-$\mathcal{N}^c_G$ | | | 104.28 | 83.66 | 8.87 | 703018 | 298153 | 0.00 |
| MIP-RS | | | | | | | | 0.17 |