Gadara Nets: Modeling and Analyzing Lock Allocation for Deadlock Avoidance in Multithreaded Software

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

Abstract— **Deadlock avoidance in shared-memory multithreaded programs is receiving increased attention as multicore architectures and parallel programming are becoming more prevalent. In our on-going project, called Gadara, the objective is to control the execution of multithreaded programs in order to avoid deadlocks by using techniques from discrete-event control theory. In this project, Petri nets are employed to model parallel programs. This paper formally defines the class of Petri nets that emerges from modeling multithreaded programs, called Gadara nets. Gadara nets are related to, but different from, other classes of nets that have been characterized in deadlock analysis of manufacturing systems. The contributions of this paper include: (i) formal definition of Gadara nets and of controlled Gadara nets; (ii) a behavioral analysis of Gadara nets for liveness and reversibility using siphons; and (iii) identification of a convexity-type property for the set of live markings.**

I. Introduction

The era of multicore architectures in computer hardware brings an unprecedented need for parallel programming. We are interested in multithreaded programs with shared data. In this programming paradigm, programmers must use lock primitives to control access to the shared data. For example, mutual exclusion locks (mutexes) are used to protect shared data from inconsistent concurrent updates. However, improper use of mutexes can lead to the familiar "deadly embrace" problem, where a set of threads are waiting for one another and no progress can be made. These situations are called *Circular-Mutex-Wait (CMW) deadlocks*. As parallel programs are becoming more and more complex, reasoning about deadlock is becoming increasingly intricate for programmers.

In our on-going project, called Gadara[1], the objective is to control the execution of multithreaded programs in order to avoid deadlocks by using techniques from discrete-event control theory. In this project, Petri nets are employed to model parallel programs. Petri nets [8] are an efficient modeling formalism that captures the concurrency of a dynamic system while avoiding enumerating its state space. In our application domain, they provide a compact representation of the system dynamics of a multithreaded program. Moreover, the structural properties of Petri nets can greatly facilitate the analysis of programs for deadlock detection. One such property is liveness; a live Petri net will not deadlock.

In the Gadara project, the goal is to control the execution of a program so that, at run-time, CMW deadlocks never occur. Our results to date have been presented in [15], [14], [16]; they pertain to model construction, control logic synthesis and implementation, and experimental validation and evaluation. In this paper, our objective is to formally define the class of Petri nets that arises in the modeling of multithreaded programs and to characterize some of its properties that are related to deadlock detection and liveness-enforcement by control. We call these nets *Gadara nets*. Gadara nets are related to, but different from, other classes of nets that have been identified and studied in deadlock analysis of manufacturing systems, a well-researched area [6]. There have also been studies of deadlock analysis for Ada programs using Petri nets as a modeling formalism [9], [12]. However, to the best of our knowledge, no other class of Petri nets studied in the literature has the exact same specific defining features as Gadara nets have.

Prior work in the area of manufacturing systems has largely concentrated on a special class of process-oriented nets that possess an acyclicity property, the S^3PR class [1], [6]. Multithreaded programs share some similarities with manufacturing systems in terms of acquisition and release of resources (i.e., locks). However, loops are common in programs and they result in cycles in the Petri net model of the program. Deadlock analysis is known to be difficult when the subnets in process-oriented nets contain internal cycles [10], [5]. When the acyclic constraint is relaxed from the S^3PR class, the resulting superclass is called S^*PR. There are few results on deadlock analysis in S^*PR [2].

Gadara nets fall within the S^*PR class, but as we demonstrate in this paper, they possess specific features that render deadlock analysis more tractable and enable the synthesis of maximally-permissive liveness-enforcing control logic by means of *monitor* (or *control*) places [7]. First, mutexes are necessarily mutually exclusive resources; this means that there is exactly one initial token in each resource place in a Gadara net. Second, branching in a program is not associated with any resource (lock) acquisition. In manufacturing systems, branching in a process subnet typically models alternative resources a token can use to finish its processing. In multithreaded programs, branching is determined by other factors. Once a branch is selected, a token (thread) must acquire a fixed set of locks to finish the branch. Therefore, in a Gadara net, transitions that model branching are not

H. Liao, S. Mahlke and S. Lafortune are with the Department of EECS, The University of Michigan, Ann Arbor, MI 48109, USA, {hwliao, mahlke, stephane}@eecs.umich.edu. Their research is supported in part by NSF grants CCF-0819882 and ECCS-0624821, and by an HP Labs Innovation Research Award.

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, spyros@isye.gatech.edu. His research is supported in part by NSF grant CMMI 0619978.

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

[1]Gadara is the Biblical place where a miraculous cure liberated a possessed man by banishing en masse a legion of demons.

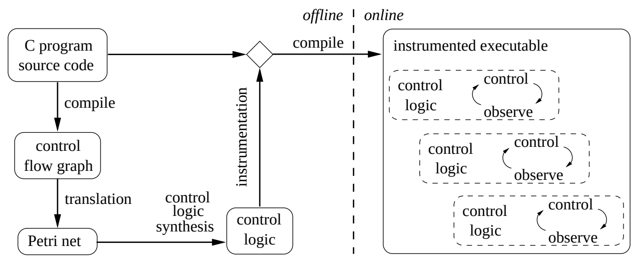

Fig. 1. Gadara architecture

connected to any resource place. The implications of these features are studied in this paper.

The main contributions of this paper are summarized as follows: (i) We formally define Gadara nets and controlled Gadara nets. (ii) We establish necessary and sufficient conditions for the liveness and reversibility of these nets that connect these two properties to the absence of certain types of siphons. These conditions extend prior conditions for S^3PR nets and other related acyclic nets that have appeared in the manufacturing systems literature [11], [6]. (iii) We show that the set of live markings of Gadara nets has a convexity-type property, which ensures the feasibility of maximally-permissive liveness-enforcing supervision through monitor places.

This paper is organized as follows: Section II gives an overview of the Gadara project. Petri net preliminaries and the formal definition of Gadara nets are presented in Section III. Section IV presents our results on liveness and reversibility of Gadara nets. Section V studies the properties of the set of live markings and liveness-enforcing supervisory control through monitor places. We discuss control synthesis for Garada nets in Section VI and conclude in Section VII. Proofs have been omitted due to space limitations; they are available from the authors.

II. Overview of the Gadara Project

The Gadara project is a multidisciplinary effort to develop a software tool, called Gadara, that automatically takes as input a deadlock-prone multithreaded C program and outputs a modified ("gadarized") version of the program that is guaranteed to run deadlock-free without affecting any of the functionalities of the program.

Figure 1 depicts the architecture of Gadara. First, the C program source code is compiled into an enhanced Control Flow Graph (CFG) by a compiler. A CFG is a high-level graphical representation of all code execution paths that might be traversed by the program. Second, the enhanced CFG is translated into a Petri net model, namely, a Gadara net. Our recent paper [16] discusses the modeling of multithreaded programs using Petri nets in detail. Places in a Gadara net represent *basic blocks* or *statements* of the program and transitions model *control jumps* among these basic blocks or statements. Threads of execution are represented by tokens flowing through these places. We model locks by *resource* places in the net. A token in a resource (lock) place represents the availability of the lock. The next step is to synthesize control logic for the Gadara net such that the controlled net corresponds to a deadlock-free program. For this purpose, the deadlocks of the original C program (a behavioral property) are mapped to the siphons in the Gadara net model (a structural property). After siphon analysis, the desired control logic is synthesized using the technique of *Supervision Based on Place Invariants* (SBPI) [4]. This phase outputs a controlled Gadara net, which is essentially an augmented version of the original Gadara net to which monitor (aka control) places and their associated arcs have been added; no new transitions are added. Finally, Gadara instruments the program to incorporate the synthesized control logic captured by the monitor places. The preceding steps are all carried out off-line, i.e., there is no run-time overhead other than the execution of the additional lines of code pertaining to checking and updating the contents of the monitor places[2]. Experimental results on the performance of gadarized code are reported in [14].

III. The Gadara Petri Net Model

A. Petri net preliminaries

We present relevant concepts and notations of Petri nets here; see [8] for a detailed discussion.

Definition 1: A Petri net $\mathcal{N} = (P, T, A, W, M_0)$ is a bipartite graph, where $P = \{p_1, p_2, ..., p_n\}$ 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, $W : A \to \{0, 1, 2, ...\}$ is the arc weight function, and for each $p \in P$, $M_0(p)$ is the initial number of tokens in place p.

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 a 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. Our Petri net models of multithreaded programs have unit arc weights, i.e., $W(a) = 1, \forall a \in A$. Such Petri nets are called *ordinary*. We drop W in the definition of any Petri net $\mathcal{N}$ that is ordinary.

Definition 2: 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., $\forall t \in T, | \bullet t| = |t \bullet | = 1$.

A pair of a place p and a transition 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, i.e., at least one of $W(p_i, t_j)$ or $W(t_j, p_i)$ is equal to zero. A self-loop-free Petri net can be defined by its incidence matrix. 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.

The marking of a Petri net $\mathcal{N}$ is a column vector M of n entries corresponding to the n places. As defined above, M_0 is the initial marking. The *reachable state space* $R(\mathcal{N}, M_0)$ of a Petri net $\mathcal{N}$ is the set of all markings reachable by transition firing sequences starting from M_0.

Definition 3: 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

[2]The strategy in Gadara corresponds to what is termed "deadlock avoidance" in computer systems; in the Petri net literature, such strategies are usually classified as "deadlock prevention."

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 [6]: A vector y is a P-invariant of Petri net $\mathcal{N} = (P, T, A, M_0)$ *iff* $M^T y = M_0^T y$ for any reachable marking M of $\mathcal{N}$.

The *support* of a 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* [8].

B. Gadara Petri nets

Our Petri nets that model multithreaded programs share similar structures as those adopted in modeling manufacturing systems [11], [6]. More specifically, they consist of a set of subnets that correspond to thread entry points in the program, and resource places that model the locks which connect these subnets together. We formally define the class of Gadara Petri nets as follows.

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

A Gadara net is defined to be an ordinary Petri net, because it models mutex locks (and not reader/writer locks for example). Conditions 1-3 are fairly common requirements for Petri nets that consist of subnets interconnected by resource places. More specifically, Conditions 1 and 2 characterize a set of subnets $\mathcal{N}_i$ that define work processes, called *process subnets* in the literature. 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. The *idle place* p_{0_i} is an artificial place added to facilitate the discussion of liveness and other properties. P_S is the set of *operation places*. P_R is the set of *resource places* that model mutex locks. The restriction of the work process subnets $\mathcal{N}_i$ into the class of state machines, by Condition 3, implies that there is no “forking” or “joining” in these subnets. On the other hand, the strong connectivity of the subnets $\mathcal{N}_i$, that 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 specification of the underlying program logic. Condition 4 models the aforestated 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.

If the token is in r, the lock is available. Otherwise it is in a place $p \in P_{S_i}$ of some subnet $\mathcal{N}_i$, which means that the thread in p is holding the lock. Finally, Condition 7 states that except for the idle places, any other place in a process subnet $\mathcal{N}_i$ models a program block that requires the acquisition of at least one lock for its execution.

Remark 1: 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 and never release it. We adjust the model based on annotations and heuristics such that the P-semiflow is restored [14].

Remark 2: A multithreaded program contains sections executed with at least one lock held by the executing thread, called *critical sections* in operating system terms, and sections executed without holding any locks. The process subnets $\mathcal{N}_i$ model only the critical sections of the underlying multithreaded programs. Each such section is modeled by a distinct process subnet. Therefore Condition 7 is true. In practice, we prune the Petri nets translated from CFGs to obtain these process subnets. An illustrative example of the pruning process is shown in Figures 2 and 3; see [13] for details.

C. Controlled Gadara nets

Supervisory control based on place invariants is a common control technique for Petri nets. This approach essentially controls the dynamics of the underlying Petri net by enforcing a set of linear inequalities on the net markings. Each of these inequalities is satisfied by the superimposition of

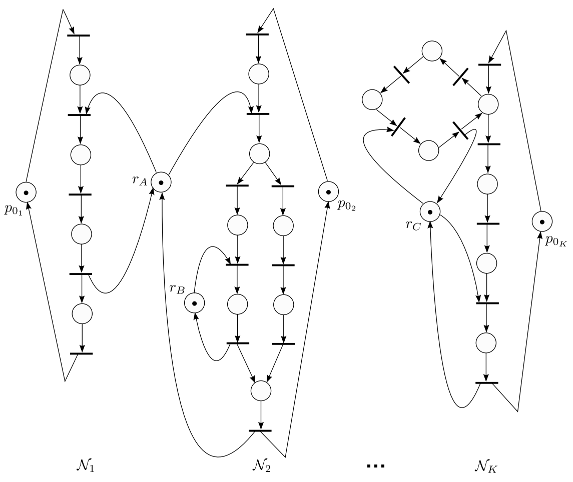

Fig. 2. An illustrative example of Gadara nets: before pruning

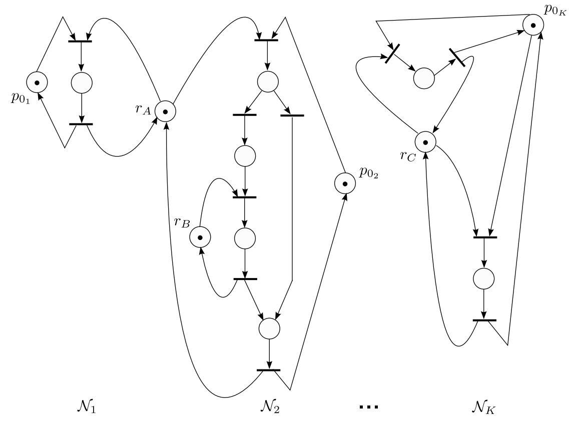

Fig. 3. An illustrative example of Gadara nets: after pruning

a *monitor* place. The monitor place connects to transitions in the Petri net and establishes a new invariant in the net dynamics that enforces the specified inequality. This invariant has a structure that is similar to that introduced by Condition 5 of Definition 4, with the monitor place playing the role of a new (fictitious) resource place. When we use this control technique on the Gadara net, we obtain an augmented net that we call a controlled Gadara net, which is formally defined as follows.

Definition 5: 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 seven conditions in Definition 4 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 $\{p_c\} = \|Y_{p_c}\| \cap P_C$, $P_0 \cap \|Y_{p_c}\| = \emptyset$, $P_S \cap \|Y_{p_c}\| \neq \emptyset$, and $Y_{p_c}(p_c) = 1$.
9) For each $p_c \in P_C$, $M_0^c(p_c) \geq \max_{p \in P_S} Y_{p_c}(p)$.

Definition 5 indicates that the introduction of the monitor places $p_c \in P_C$ preserves the net structure that is implied by Definition 4. Furthermore, it can be checked that the monitor places possess similar structural properties with the resource places in $\mathcal{N}_G$, but have weaker constraints. More specifically, monitor places may have multiple initial tokens and non-unit arc weights. We will discuss the consequences of these discrepancies for the analysis of Gadara nets in Section IV.

One need not associate $\mathcal{N}_G^c$ with any specific control policy. It is a structural definition that does not refer explicitly to the content of the inequalities that define the applied (or the underlying) control logic. More importantly, $\mathcal{N}_G$ is a special subclass of $\mathcal{N}_G^c$ where $P_C = \emptyset$. Therefore, any property that we derive for $\mathcal{N}_G^c$ holds for $\mathcal{N}_G$ as well.

IV. Main Properties of Gadara nets

A. Petri net liveness and reversibility

First, let us provide a series of definitions that formalize the Petri net concepts of liveness and reversibility and some additional concepts related to them.

Definition 6: 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 7: A nonempty set of places S is said to be a *siphon* if $\bullet S \subseteq S\bullet$.

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, and use $M[t >$ to denote that transition t is enabled at marking M.

Definition 8: A marking M is *live* if $\forall t \in T$, there exists $M' \in R(\mathcal{N}, M)$, s.t. $M'[t >$. A Petri net $(\mathcal{N}, M_0)$ is *live* if $\forall M \in R(\mathcal{N}, M_0), M$ is live.

Definition 9: Petri net $\mathcal{N}$ is said to be *quasi-live* if for all $t \in T$, there exists $M \in R(\mathcal{N}, M_0)$, s.t. $M[t >$.

Definition 10: 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 4 imply that all subnets $\mathcal{N}_i$ in a Gadara net $\mathcal{N}_G$ are quasi-live. Furthermore, Condition 5 of Definition 4 implies 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 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 4 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 $M_0(p_{0_i}) = 1$.

B. Resource-induced deadly marked siphons and modified markings of Gadara nets

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 11: A siphon S of a Gadara net $\mathcal{N}_G^c$ is said to be a *resource-induced deadly marked (RIDM) siphon* 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.

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

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

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 4 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 resource-induced deadly marked siphons.

C. Main results

In this section, we derive liveness properties for Gadara nets. Similar results exist in the literature [11] for a class of process-resource nets that are structurally similar but model processes with no internal cycles. Despite the presence of cycles in our process subnets, most proofs can be generalized to Gadara nets. Therefore we follow the same strategy as in [11] for proving liveness properties for these nets, in Theorems 2 and 3.

Theorem 2: $\mathcal{N}_G^c$ 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 resource-induced deadly marked siphon at $\overline{M}$.

Theorem 3: (1) $\mathcal{N}_G$ is live *iff* there does not exist a marking $M \in R(\mathcal{N}_G, M_0)$ and a siphon S such that S is an empty siphon at M. (2) If $\mathcal{N}_G^c$ is ordinary, then $\mathcal{N}_G^c$ is live *iff* there does not exist a marking $M \in R(\mathcal{N}_G^c, M_0^c)$ and a siphon S such that S is an empty siphon at M.

Theorem 4: $\mathcal{N}_G^c$ is live *iff* it is reversible.

V. Properties of the Live Region

If the reachable state space of a Gadara net $\mathcal{N}_G$ contains empty siphons, we want to control the net so that its marking remains within the sub-space of live markings; in the following, we shall refer to this subspace as the *live region* of (the state space of) the net. As discussed in Section III-C, the supervisory control method based on place invariants enforces liveness by adding monitor places to the Petri net [3], [7]. The control specification itself is given as a set of linear constraints $\{(l_k, b_k), k = 1, 2, \dots\}$ to be enforced on the net marking. Each l_k is a weight vector, each b_k is a constant, and it is required that

$$l_k^T M \le b_k \tag{2}$$

for any reachable marking M and every k. For example, an empty siphon can be avoided through the imposition of a linear constraint requiring that the total number of tokens in all places in the siphon is greater than or equal to one.

It is not always possible to characterize live regions by linear constraints in the form of (2) [3]. Figure 4 shows a counter example. Let marking M_1 correspond to the state where p_{11}, r_B and p_{0_2} all have two tokens while all other places are empty. Marking M_2 corresponds to the state 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 three markings are reachable from the initial marking illustrated in the figure, and only M is not live. Apparently there is no linear constraint that *separates* M from M_1 and M_2. The above argument 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 live region cannot be characterized by linear constraints, *maximally permissive* control is not feasible through monitor places. A popular approach to non-convex live regions sacrifices maximal permissiveness and further reduces the controlled reachable state space to a convex live subregion [6]. We prove next that the live region of a Gadara net is always convex, and therefore maximally permissive liveness-enforcing supervision through monitor places is feasible.[3]

Similar to the modified marking defined in Section IV-B, we define another projection here to facilitate the discussion.

$$\overline{\overline{M}}(p) = \begin{cases} M(p), & \text{if } p \in P_S; \\ 0, & \text{if } p \notin P_S. \end{cases} \tag{3}$$

Again, this projection does not introduce any confusion since we can uniquely retrieve marking M from the information provided in $\overline{\overline{M}}$, 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 is recovered from the invariants respectively established by Conditions 5 and 8 in Definitions 4 and 5. Therefore, we consider linear constraints for $\overline{\overline{M}}$ only, i.e., the coefficients corresponding to places P_R and P_0 are all zero.

Proposition 1 states that there is exactly one token in the

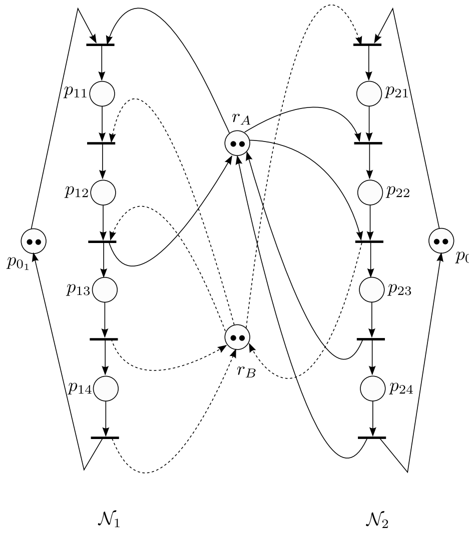

Fig. 4. Counter example: a net with non-convex live region due to multi-capacity resource places.

[3]The counter example in Figure 4 is not an $\mathcal{N}_G^c$ net because P_R is missing. P_R cannot be empty because, according to Condition 7 in Definition 4, every place in P_S has to be a support place of a P-semiflow Y_r, $r \in P_R$.

support places, $||Y_r||$, of any P-semiflow Y_r of Definition 4. This result, when considered together with Condition 7 of Definition 4, imply the following proposition, which is the key to the convexity of the live region in Gadara nets.

Proposition 5: Give a Gadara net $\mathcal{N}_G^c$, $\forall M \in R(\mathcal{N}_G^c, M_0)$, $\forall p \in P_S$, $M(p)$ is either 0 or 1, i.e., $\overline{\overline{M}}$ is a binary vector.

In a binary integer 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 6: Given a Gadara net $\mathcal{N}_G^c$ and a set of markings $V \subseteq R(\mathcal{N}_G^c, M_0)$, there exists a finite set of linear constraints $LC(V) = \{(l_1, b_1), (l_2, b_2), ...\}$ such that $M \in V$ *iff* $\forall (l_i, b_i) \in LC(V)$, $l_i^T M \leq b_i$.

Proof: (sketch) Based on Proposition 5, for any marking $M \notin V$, we construct the following linear constraint:

$$l(p) := \begin{cases} 0, & \text{if } p \notin P_S \\ 1, & \text{if } \overline{\overline{M}}(p) = 1 \\ -1, & \text{if } \overline{\overline{M}}(p) = 0 \end{cases} ; \quad b := \sum_{p \in P_S} \overline{\overline{M}}(p) - 1 \quad (4)$$

It is not difficult to verify that $l^T M > b$ and for any other marking $M' \in R(\mathcal{N}_G^c, M_0)$ (or any other binary vector of the same dimension), $l^T M' \leq b$. Therefore we can construct such a linear constraint for every marking in $R(\mathcal{N}_G^c, M_0) \backslash V$. Since the reachable state space of $\mathcal{N}_G^c$ is finite, containing no more than $2^{|P_S|}$ states, $R(\mathcal{N}_G^c, M_0) \backslash V$ 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 $\mathcal{N}_G^c$. ■

Separating the live region of a Gadara net using linear constraints is a special case of Theorem 6.

Corallary 7: The live region of a Gadara net $\mathcal{N}_G$ can be separated by a finite set of constraints $LC = \{(l_1, b_1), (l_2, b_2), ...\}$ such that $M \in R(\mathcal{N}_G, M_0)$ is live *iff* $\forall (l_i, b_i) \in LC$, $l_i^T M \leq b_i$.

Corollary 7 establishes the foundation for maximally permissive liveness-enforcing control of Gadara nets through monitor places. The example of Figure 4 also establishes that the unit initial marking of the resource places is instrumental for this result.

VI. Control Synthesis of Gadara nets

We have proved the feasibility of monitor-based maximally-permissive liveness-enforcing control for Gadara nets in the previous section. Our experiments with real-world programs (see [14], [16]) further indicate that real deadlock patterns are relatively simple because programmers typically use locks in a conservative way due to the fear of deadlocks. Our current control algorithm in Gadara first discovers all siphons in the Gadara net model of the program, and then controls a subset of them by synthesizing one monitor place for each based on a linear constraint of the type expressed in (2). Not all of the siphons need to be controlled as explained in [15]. The algorithm iterates until all empty siphons in the Petri net are controlled [16]. If the procedure does not converge, or if some of the added monitor places have non-unit arc weights, then we should control for deadly marked instead of empty siphons. Existing methods could be applied in this case. However, these methods typically either sacrifice maximal permissiveness or require state space exploration. In practice, the procedure terminates in one iteration for the real programs we experimented with.

VII. Conclusion

In our Gadara project [14], [16], the objective is to control the execution of multithreaded programs in order to avoid deadlock by using techniques from discrete-event control theory. In this project, Petri nets are used to model parallel programs. The information necessary to build the Petri net model of a program is extracted at compile time. This paper formally defined the class of Petri nets that emerges from modeling multithreaded programs, called Gadara nets. Gadara nets are related to, but different from, other classes of nets that have been characterized in deadlock analysis of manufacturing systems. Several properties of Gadara nets that are relevant to their analysis and control were presented, including: (i) characterization of liveness and reversibility in terms of siphons; and (ii) identification of a convexity-type property for the set of live markings.

References

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

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

[3] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 974–979, Chicago, IL, 1992.

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

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

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

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

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

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

[10] J. Park and S. A. Reveliotis. Liveness-enforcing supervision for resource allocation systems with uncontrollable behavior and forbidden states. *IEEE Trans. on Robotics and Automation*, 18(2):234–240, Feb. 2002.

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

[12] S. M. Shatz, S. Tu, T. Murata, and S. Duri. An application of Petri net reduction for Ada tasking deadlock analysis. *IEEE Trans. on Parallel and Distributed Systems*, 7(12):1307–1322, Dec. 1996.

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

[14] Y. Wang, T. Kelly, M. Kudlur, S. Lafortune, and S. A. Mahlke. Gadara: Dynamic deadlock avoidance for multithreaded programs. In *OSDI '08*.

[15] Y. Wang, T. Kelly, M. Kudlur, S. Mahlke, and S. Lafortune. The application of supervisory control to deadlock avoidance in concurrent software. In *WODES '08*, pages 287 – 292.

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