Maximally Permissive Deadlock Avoidance for Multithreaded Computer Programs (Extended Abstract)

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

Multicore architectures in computer hardware bring an unprecedented need for parallel programming. In the work considered in this presentation, we are especially interested in *multithreaded programs with shared data*. In this widely-used programming paradigm, "lock" primitives are employed to control access to the shared data within the program threads. For example, *mutual exclusion locks (mutexes)* are used to protect shared data from concurrent inconsistent 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 further progress can be made. These situations are called *Circular-Mutex-Wait (CMW) deadlocks* and the tale of the dining philosophers [3] is an amusing way to illustrate CMW-deadlocks. As parallel programs are becoming more and more complex, reasoning about deadlock is becoming more and more difficult for programmers. Significant effort has to be spent to detect and fix intricate deadlock bugs. Therefore, there is an emerging need for more systematic methodologies that will enable the programmers of these environments to effectively characterize, analyze, and resolve these deadlock problems.

The proposed presentation will report the progress of an ongoing project, called *Gadara*[1], that seeks to provide a systematic solution to the aforementioned deadlock handling problem. More specifically, Gadara uses discrete event control theory [1] and its specialization in the context of resource allocation [9], [15], in order to detect potential deadlocks and control the run-time execution of the underlying programs in a way that guarantees that these deadlocks never occur. In addition, the pursued solutions are required to (i) impose the minimal restrictions upon the program execution that are necessary in order to ensure deadlock-free operation, and also (ii) minimize the computational overhead that is induced by the run-time enforcement of these restrictions.

The rest of this document is organized as follows: We start with a more detailed overview of the Gadara project, outlining its particular scope and objectives, and also, the key methodology that defines the proposed solution to the considered problem of deadlock avoidance. Next we introduce the *Gadara Petri net* model, which provides a formal representation for the resource allocation dynamics that take place in the considered programs. These nets present extensive similarities to the *process-resource nets* that have been used for modeling resource allocation in other application contexts. We overview the structural and behavioral properties that are inherited by Gadara nets from their identification as process-resource nets, but also their additional attributes that enable unique, customized solutions to the deadlock avoidance problems arising in them. The discussion concludes with an overview of the particular set of such customized solutions that are currently pursued in the context of the Gadara project.

Y. Wang, H. Liao, S. Mahlke and S. Lafortune are with the Department of EECS, The University of Michigan, Ann Arbor, MI 48109, USA, {yinw,hwliao,mahlke,stephane}@eecs.umich.edu

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, spyros@isye.gatech.edu

T. Kelly is with HP Labs, Palo Alto, CA 94303, USA, 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.

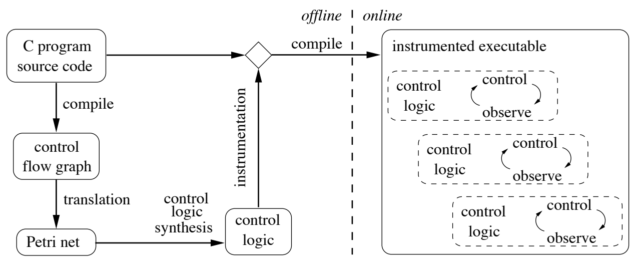

Fig. 1. The Gadara architecture

a) Overview of the Gadara project: The Gadara project can be more specifically defined as a multidisciplinary effort to develop an automated methodology, implemented in a software tool, that 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 extracted and compiled into an enhanced *Control Flow Graph (CFG)* by a compiler. A CFG is a high-level graphical representation of all the sample paths that might be traversed by the program. Second, the enhanced CFG is translated into a Petri net model, namely, a *Gadara net*. A brief description of Gadara nets is provided in the next part of this document, while a more extensive discussion on the Petri net-based modeling of multithreaded programs can be found in [13], [10]. The next step in Gadara is to synthesize control logic for the Gadara net such that the controlled net corresponds to a deadlock-free program. In this regard, the deadlocks of the original C program (a behavioral property) are mapped to the *siphons* [2] in the Gadara net model (a structural property); a more detailed account of this mapping is provided in a later part of this document. After siphon analysis, the desired control logic is synthesized using the technique of *Supervision Based on Place Invariants* (SBPI) [6]. 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. A first set of experimental results on the performance of gadarized code are reported in [12] while an extended set, based on some novel approaches that are currently pursued in this work, will be provided in the proposed presentation.

b) The Gadara Petri net model: As was already mentioned, Gadara nets possess the more general structure of process-resource nets, a Petri net sub-class that has been used extensively for modeling the dynamics of sequential resource allocation systems [9], [7]. In their basic definition, *process-resource nets* are the result of the composition of a set of *process subnets*, that model the sequential logic of the underlying processes, through an additional set of *resource places*, that model the common set of resources that are used by the aforementioned processes during the execution of their various processing stages.

In the case of Gadara nets, each process subnet models the program flow (of one or a set of threads) within one of its *critical sections* i.e., the maximal segments of the program code which require the allocation of at least one lock for their execution. With the exception of one place, all the places in each defined process subnet represent *basic blocks* or *statements* of the program, while the transitions of the subnet model *control jumps* among these basic blocks or statements. The remaining place, typically known as the (subnet) *idle place*, models the "boundary" of the critical section, i.e., those parts of the code before and after the critical section which require the allocation of no locks for their execution. Hence, tokens in the idle place of a process subnet represent threads that are waiting to start the execution of the corresponding critical section, while tokens in the remaining places of this subnet represent threads that have already initiated the execution of this section. Furthermore, in the current specification of Gadara nets, each process subnet is restricted to the topology of a *strongly connected state machine* [2]. From a more practical standpoint, the restriction of the Gadara process subnets into the class of state machines implies that the execution of a thread in a critical section can involve *"branching"* through a number

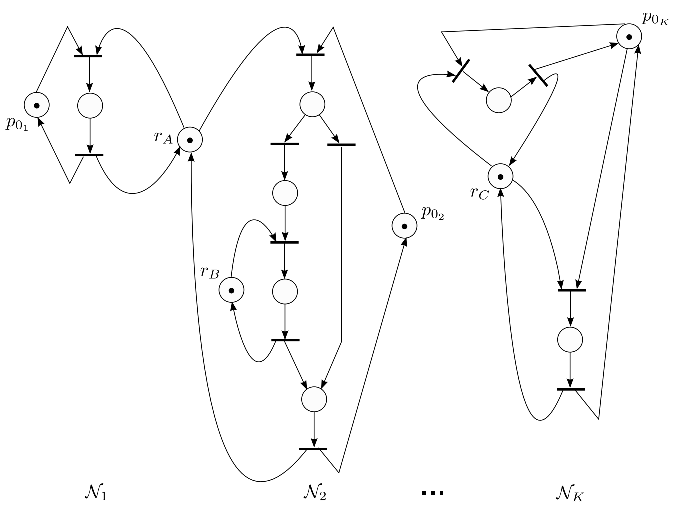

Fig. 2. Some illustrative examples of Gadara nets

of different cases, or even *"looping"* through a sequence of statements, but the thread must maintain its *atomic* nature and, except for the competition for the required locks, it will evolve independently from any other thread in the section. In particular, the current version of Gadara nets does not allow either the *coordinated / synchronized execution* of certain events among a number of running threads, or any *"merging"* and *"spawning"* activity [3]. Finally, the requirement for strong connectivity of the Gadara process subnets 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 regions and not any other potential errors in the specification of the underlying program logic.

The resource places of Gadara nets correspond to the various mutexes that control the execution of the underlying program. These places are connected to the process subnets so that the availability of a token in them indicates that the corresponding lock is currently free. Similarly, the absence of a token from any resource place indicates that the corresponding lock is currently used by some running process. Furthermore, the net connectivity guarantees (i) the atomic nature of the locks and (ii) that all locks acquired by a thread while in a critical section are properly released upon the exit of the thread from this section. This behavior is formally manifested by the fact that every resource place together with the process places that constitute potential holders for this resource, are the support of a *minimal P-semiflow* [2] in the net dynamics, with unit weights and a constant value equal to one. It also implies that Gadara nets are *ordinary* Petri nets, i.e., the firing of any transition in them transfers only one token through each of its incident arcs [2].

Finally, another element that characterizes the dynamics of a Gadara net, is the *separation* of the resource allocation and the routing functions in any given process subnet. The requirement for this separation results from the fact that in a typical program, the branching and looping decisions will not depend on the availability of the locks that are necessary for its advancement in certain directions, but on other elements inherent to the program logic. From a technical standpoint, this requirement is observed by ensuring that each transition modeling a branching or a looping decision is not connected to any of the resource places of the net.

For the purposes of deadlock avoidance, the *initial marking* of a Gadara net is naturally defined by the marking that models an execution status of the underlying program where all locks are free while a number of threads are ready to initiate their execution in the various critical sections. This situation is modeled by placing one token in each resource place, and the corresponding number of tokens in the idle place of each process subnet. Some example Gadara nets are provided in Figure 2.

c) Placing Gadara nets in the taxonomy of the process-resource nets already studied in the literature: It is clear from the above description that Gadara nets bear strong similarity to the process-resource nets studied in [9], in particular, the subclass of these nets that model *conjunctive / disjunctive resource allocation systems (C/D-RAS)*. Similar to Gadara nets, these nets of [9] allow for alternative process routing – i.e., branching in the Gadara terminology – and concurrent allocation of more than one resource types at each process stage. However, an additional element that is present in Gadara nets but is not allowed in the C/D-RAS modeling Petri nets of [9], is process *"looping"*. Technically, this effect is modeled by cycles in the process subnets of Gadata nets that do not contain the idle place, and it brings Gadara nets into the class of S^*PR nets, of [4]. It is generally true that process-resource nets with looping behavior have not received a lot of attention in the past literature, and the currently available results on them are rather limited.

Another salient element of Gadara nets, that differentiates them from the classes of process-resource nets that have been studied in the past literature, is the *"safe"* nature of all their places other than the idle places.[2] This property stems from the uniqueness of every lock that is modeled by the resource places of these nets, and from the additional fact that every process place (i.e., a place of a process subnet other than its idle place) models an operation that occurs in a critical section (and therefore, it requires at least one lock). From a more practical standpoint, the safety of the aforementioned places is instrumental for synthesizing effective and computationally efficient maximally permissive policies for Gadara nets. We shall return to this point in the next parts of this document, where we consider the structural and behavioral properties of Gadara nets and the synthesis of deadlock avoidance policies for them.

d) Behavioral and structural properties of Gadara nets: In the Petri net-based modeling framework of process-resource nets, the requirement for deadlock-free resource allocation is formally expressed by the requirement for the net *reversibility*, i.e., the ability to reach the initial marking from any marking that is reached during the evolution of the net behavior. [3] A highly celebrated result in the past studies of these systems is that for a very broad class of process-resource nets, (i) net reversibility is equivalent to net *liveness*, i.e.., the ability to enable any of the net transitions from any reachable marking, and (ii) both of these properties are equivalent to the absence from a projected version of the net dynamics that is known as the *modified marking* of the net, of a particular structural formation known as *a resource-induced deadly marked siphon* [9]. On the other hand, these same results have also established that in the case of process subnets with internal cycles, the connection of non-liveness and non-reversibility to the aforementioned siphons might not be valid any more.

However, in the presented work, we have managed to establish that the interpretation of the lack of liveness and reversibility of Gadara nets through the formation of resource-induced deadly marked siphons in the space of the modified markings of the net, remains valid, thanks to the *separation* of the resource allocation and the routing functions that is implemented by Gadara nets (cf. the relevant discussion in part (b) above). Furthermore, the *ordinary* nature of Gadara nets enables the mapping of these resource-induced deadly marked siphons to *empty* siphons that can be traced in the unprojected dynamics (i.e., the original reachable markings) of the net. A more technical exposition of these results can be found in [11], [10]. As discussed in the next part of this document, both of them have significant implications for the synthesis of effective and efficient deadlock avoidance policies for Gadara nets.

We conclude the discussion of the properties of Gadara nets by presenting an additional property that results from the safety of the process and the resource places discussed in part (c) above, and is instrumental for the development of the control policies outlined in the last part of this document. More specifically, the strongly connected state machine structure of the process subnets in Gadara nets implies that the marking of their idle places can be uniquely determined from the marking of their remaining process places. Similarly, the P-semiflows that express the reusable nature of the system resources (cf. part (b) above), imply that the marking of each resource place can also be uniquely determined by the marking of the process places that constitute potential holders of this resource. Therefore, every reachable marking of a Gadara net can be uniquely reconstructed from the information provided by its projection to the subspace that is defined by the process places. This fact further implies that any classification of the net markings, for control purposes, into *"admissible"* and *"inadmissible"* ones, can be equivalently effected by working in the projected subspace mentioned above. Finally, the aforementioned safety of the process

[2]We remind the reader that in the Petri net literature, a place is said to be *"safe"* if its marking can take only the values of 0 and 1 [2].

[3]Of course, this characterization presumes that, as in the case of Gadara nets, the net initial marking corresponds to the state of the underlying RAS where the system is idle and empty of any processes.

places in Gadara nets guarantees that all these (marking) projections will be of *binary* nature. But then, it is possible to show that any *dichotomy* of this projected set of markings can be effected through a set of *linear inequalities* [11]. The practical implications of this result are discussed in the next part of this document.

e) Monitor-based, maximally permissive, liveness-enforcing supervision of Gadara nets: We remind the reader that the primary objective of the Gadara project is the establishment of minimally restrictive (or, equivalently, *maximally permissive*) *deadlock avoidance policies (DAPs)* to govern the allocation of the various locks to the contesting threads of a multithreaded C program. As discussed in the previous parts of this document, in the Petri net-based formalism of Gadara nets, deadlock freedom is equivalent to the reversibility and the liveness of the underlying net dynamics. Hence, it is customary to also refer to this control problem as the *maximally permissive, liveness-enforcing supervision* of Gadara nets.

From a more operational standpoint, the sought supervisor must confine the evolution of the net marking within the subspace of markings that are *co-accesible* to the initial marking of the net, i.e., to those markings for which there is a feasible sequence of transitions that completes all the running threads and drives all the corresponding tokens back to their idle places. Hence, the control problem considered in this work can be reduced to the construction of a *classifier* that dichotomizes the reachable markings of the given Gadara net on the basis of their co-accessibility; the availability of such a classifier will further resolve the admissibility of any tentative transition in the net dynamics on the basis of the co-accessibility of the marking that will result from its execution.

Furthermore, according to the remarks provided in part (d) of this document, the suggested dichotomy can be effected on the projected space that is defined by the process places of the net, and the corresponding classifier can be represented by a set of linear inequalities in the net (projected) marking. This algebraic representation of the supervisory control logic subsequently enables its super-imposition on the original Gadara net through a set of *monitor* places connected to the transitions of the plant net in a way that they can be interpreted as additional *fictitious resources* to be administered to the executed processes [5], [8]. The resulting *controlled* (versions of the) Gadara nets, preserve the basic behavioral and structural properties of the (uncontrolled) Gadara nets outlined in the earlier parts of this document, but they may lose the ordinary structure of the latter.

Finally, an additional result that is also important for the effective resolution of the considered supervisory control problem, is that in the class of Gadara nets, deciding the co-accessibility of any given marking is an *NP-complete* problem [9]. The practical implication of this result is that the aforementioned classifier of the net markings might need to involve a number of linear inequalities that is a *super-polynomial* function of the size of the underlying Gadara net.[4]

Our current investigations on the problem of maximally permissive, liveness-enforcing supervision of Gadara nets seek to address the following two primary issues:

1) Given a Gadara net, how can we obtain a *minimal* set of linear inequalities, and the corresponding set of monitor places, that will effect the aforementioned classification of the net (reachable) markings into admissible and non-admissible, for the purposes of deadlock avoidance?
2) Even though we know that the classification problem addressed in Item #1 above has a *super-polynomial "worst-case" computational complexity*, what is the *"practical"* complexity of this problem for the Gadara nets that result from the CFGs encountered in contemporary programming applications?

Our work with respect to Item #1 above has evolved along two main directions. The first direction has been motivated by the ideas presented in [6], and it seeks to construct the sought classifier through the iterative identification and control, through monitor places, of any inappropriately marked siphons that arise in the dynamics of the underlying Gadara nets. A particular challenge that is encountered in this approach arises from the fact that, due to the non-ordinary structure of the controlled (aka monitor-augmented) Gadara nets, some iterations might necessitate the control of resource-induced deadly marked siphons that are not necessarily empty. On the other hand, preliminary experimentation with this approach on Gadara nets extracted from commercial programming applications, has revealed that the method has the potential to converge very fast, frequently terminating within one iteration, and without any arising need for controlling deadly marked, non-empty siphons.[5] Some results along these lines have been already reported in [12], [10].

The second line of investigation with respect to Item #1 above pursued in our current work, seeks the straightforward computation of the sought classifier through the solution of a *Mixed Integer Programming (MIP)* [14] formulation that characterizes the corresponding separation problem in the projected marking space that is defined by the process places of the Gadara net.

Finally, we notice for completeness, that an additional, more detailed problem that arises in the synthesis of the aforementioned supervisors, concerns the observance of the *uncontrollability* of the Gadara net transitions that model the routing (i.e., the branching and looping) dynamics of the underlying program. In general, this problem can be addressed through *constraint transformations* similar to those proposed in [8]. However, it is well known that these transformations of [8] do not necessarily preserve the maximal

[4]The *size* of a Gadara net is defined by the number of places and transitions in it.

[5]However, we believe that this effect is partly due to the lack of a systematic methodology for deadlock prediction and avoidance itself, since it has forced the programmers of these past applications to keep the corresponding lock allocation patterns as simple as possible.

permissiveness of the underlying control logic. Therefore, a remaining challenge in our work is the development of such transformational logic that is customized to Gadara nets and observes the requirement for maximal permissiveness to the extent possible.

The proposed presentation will provide a systematic account of our investigations with respect to the above-stated research problems.

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems (2nd ed.)*. NY, NY: Springer, 2008.

[2] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri nets*. Heidelberg, Germany: Springer Verlag, 2005.

[3] H. M. Deitel, *Operating Systems*. Reading, MA: Addison Wesley, 1990.

[4] J. Ezpeleta, F. Tricas, F. Garcia-Valles, and J. M. Colom, "A Banker's solution for deadlock avoidance in FMS with flexible routing and multi-resource states," *IEEE Trans. on R&A*, vol. 18, pp. 621–625, 2002.

[5] A. Giua, F. DiCesare, and M. Silva, "Generalized mutual exclusion constraints on nets with uncontrollable transitions," in *Proceedings of the 1992 IEEE Intl. Conference on Systems, Man and Cybernetics*. IEEE, 1992, pp. 974–979.

[6] M. Iordache and P. Antsaklis, *Supervisory Control of Concurrent Systems: A Petri net structural approach*. Boston, MA: Birkhäuser, 2006.

[7] M. Jeng, X. Xie, and M. Y. Peng, "Process nets with resources for manufacturing modeling and their analysis," *IEEE Trans. on Robotics & Automation*, vol. 18, pp. 875–889, 2002.

[8] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems using Petri nets*. Boston, MA: Kluwer Academic Pub., 1998.

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

[10] Y. Wang, "Software failure avoidance using discrete control theory," Ph.D. dissertation, University of Michigan, Department of EECS, 2009.

[11] Y. Wang, L. Hongwei, S. Reveliotis, T. Kelly, S. Mahlke, and S. Lafortune, "Modeling and analysis of a special class of Petri nets arising in multithreaded programs," EECS, Univ. of Michigan (submitted to CDC 2009), Tech. Rep., 2009.

[12] Y. Wang, T. Kelly, M. Kudlur, S. Lafortune, and S. Mahlke, "Gadara: Dynamic deadlock avoidance for multithreaded programs," in *OSDI'08*, 2008, pp. –.

[13] Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke, "The theory of deadlock avoidance via discrete control," in *POPL'09*, 2009, pp. –.

[14] W. L. Winston, *Introduction To Mathematical Programming: Applications and Algorithms, 2nd ed.* Belmont, CA: Duxbury Press, 1995.

[15] M. Zhou and M. P. Fanti (editors), *Deadlock Resolution in Computer-Integrated Systems*. Singapore: Marcel Dekker, Inc., 2004.