Eliminating Concurrency Bugs in Multithreaded Software: An Approach Based on Control of Petri Nets*

Stéphane Lafortune[1], Yin Wang[2], and Spyros Reveliotis[3]

[1] Department of EECS, University of Michigan, USA
[2] HP Laboratories, USA
[3] School of ISyE, Georgia Tech, USA

Abstract. We describe the Gadara project, a research effort whose goal is to eliminate certain classes of concurrency bugs in multithreaded software by controlling the execution of programs at run-time. The Gadara process involves three stages: modeling of the source code at compile time in the form of a Petri net, feedback control synthesis, and control logic implementation into the source code. The feedback control logic is synthesized using techniques from supervisory control of discrete event systems, where the specification captures the avoidance of certain types of concurrency bugs, such as deadlocks. We focus on the case of circular-wait deadlocks in multithreaded programs employing mutual exclusion locks for shared data. The application of the Gadara methodology to other classes of concurrency bugs is briefly discussed.

1 Introduction

The concepts and techniques of control engineering find numerous applications in computer and software engineering. For instance, classical control theory, for time-driven systems with continuous state spaces, has been applied to computer systems problems involving quantitative properties, such as throughput stabilization; see, e.g., [1]. However, many important problems in computer and software engineering involve qualitative specifications, such as deadlock avoidance, and their solution requires control-theoretic approaches for event-driven systems with discrete state spaces, i.e., Discrete Event Systems (DES). In the last few years, there has been increased interest in solving discrete-event control problems that arise in software and embedded systems; see, e.g., [2–10]. In particular, the paradigm of *controlling software execution to avoid software defects at run-time* is receiving increased attention in the control engineering, programming languages, and operating systems communities.

* Research partially supported by the U.S. National Science Foundation, under grants CCF-0819882, CMMI-0928231, and CCF-1138860 (Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering), and by HP Laboratories, under an Open Innovations Award.

J.-M. Colom and J. Desel (Eds.): PETRI NETS 2013, LNCS 7927, pp. 21–28, 2013.

We have been investigating how to control software execution to avoid certain classes of concurrency bugs under the so-called "Gadara Project" [11], a multidisciplinary effort centered at the University of Michigan and pursued in collaboration with HP Laboratories and the Georgia Institute of Technology in the U.S. In this effort, control techniques from the field of DES, such as supervisory control [12] and supervision based on place invariants [13], are employed to synthesize control logic that is instrumented into the source code and enforces the desired safety properties at run-time [14]. Since it is model-based and relies on theoretical results in DES, this approach provably guarantees the desired safety properties, subject to model accuracy. The principal safety property of interest in the work to-date is deadlock avoidance in multithreaded programs that use locking primitives to control access to shared data [15, 16]. Recent results address certain types of atomicity violations in multithreaded programs [17, 18].

This paper describes and discusses the key features of the Gadara methodology, with relevant references. It is based on, and complements, the keynote lecture of the first author at the 34th International Conference on Application and Theory of Petri Nets and Concurrency (June 2013).

2 Gadara Methodology

There is a large amount of literature in computer science on the study of deadlock using a variety of modeling and analysis techniques. Petri net models have been used for deadlock analysis in several application domains, including computer and manufacturing systems. In particular, several special classes of Petri nets have been characterized and analyzed for deadlock problems that involve a set of "processes" sharing a set of common "resources" in the context of automated manufacturing applications; see [19, 20]. Such systems are often referred to as Resource Allocation Systems, or RAS. RAS also occur in the context of software systems, where processes may be *threads* and shared resources may be *data objects*. Modeling thread creation/termination and lock/unlock operations on shared data is in fact a classical application of Petri nets [21], and Petri nets have been employed to model multithreaded synchronization primitives in the popular Pthread library for C/C++ programs [22]. Petri nets were also used to analyze deadlocks in Ada programs [23]. A review of the application of Petri nets to computer programming is presented in [24].

The methodology developed in the Gadara project for avoidance of certain classes of deadlocks in multithreaded software is also based on Petri net models. The methodology relies on the extraction of a suitable model of the program source code at compile time in the form of an enhanced Control Flow Graph (CFG) that captures the control flow and the locking behavior of all the program threads. This step generally requires the use of static analysis techniques (see, e.g., [25, 26]) to ensure a more accurate model. This model is then translated into a Petri net in a straightforward process: places in the net represent basic blocks (i.e., branch-free sets of consecutive instructions) in the CFG or locks that will be acquired by the threads; transitions in the net represent transitions

in the CFG or lock acquisition and release operations; finally, tokens represent the states of the threads and of the locks. The special class of RAS Petri nets that arises in this context is called *Gadara nets*. The reader is referred to [27] for the formal definition of Gadara nets and for a treatment of their analytical properties in the context of multithreaded programs that use mutually-exclusive locks to control access to shared data. The deadlocks caused by the use of mutually-exclusive locks are circular-mutex-wait deadlocks, where threads in a set are waiting for one another and none can proceed. Avoidance of circular-mutex-wait deadlocks in multithreaded programs is mapped to the problem of *liveness* in Gadara nets in [27]. Liveness here refers to the property that every transition is eventually firable from any reachable state of the net. Due to the structure of Gadara nets, liveness is equivalent to reversibility, i.e., the initial state must be reachable from every reachable state. Algorithms based on solving Mixed Integer Linear Programs (MILP) are presented in [27] for determining if liveness holds or not. The algorithms exploit the structural properties of Gadara nets, in terms of certain classes of siphons. In this regard, we note that many works have considered similar structural analyses for related classes of Petri nets; see, e.g., [20, 28].

The central aspect of the Gadara methodology is its focus on synthesizing a control strategy for the Petri net model so that the controlled system is provably live, with respect to the model. This control strategy, referred to as the *control logic* hereafter, must satisfy four key requirements in addition to liveness. The first two requirements, denoted by (R1) and (R2), pertain to its synthesis and the last two requirements, denoted by (R3) and (R4), pertain to its implementation. (R1): The control logic should not alter the behavior of the program; it should only act by delaying lock acquisition or release operations performed by the threads. (R2): The control logic should only intervene when absolutely necessary; this is referred to as *maximal permissiveness*. A correct strategy could be to force the threads to always execute serially; deadlock would be avoided, but no concurrency would be allowed. (R3): The control logic must be readily translatable to the original source code that is modeled by the Gadara net, thereby allowing code instrumentation as an implementation mechanism. (R4): The runtime overhead of the control logic must be minimized, so that the instrumented program runs almost as fast as the original program.

The supervisory control theory for DES initiated in [12] and widely studied since then is well suited for handling (R1) and (R2). The notion of *uncontrollable* transitions (or events) captures (R1), while maximal permissiveness is handled by the concept of the *supremal controllable sublanguage* of the legal language with respect to the uncontrolled system language. Here, the legal language is the live sublanguage of the uncontrolled system, obtained by deleting states that deadlock and those that are in a livelock, when the initial state is the only marked state. However, using the standard algorithms of this supervisory control theory, as described in [29] for instance, requires building the reachability graph of the Gadara net model of the program. Moreover, the form of the control strategy, which is now a global function over the entire reachability graph, will perform

poorly in terms of (R3) and (R4), unless it can be encoded in a different form. These two considerations have motivated the control logic synthesis research performed in the Gadara project, which is overviewed in the next section.

3 Control Logic Synthesis

Requirements (R3) and (R4) of the previous section suggest to use *control places* (also called monitor places) as the control mechanism for the Gadara net model of the program. Control places are connected to the transitions of the net, which in turn can be mapped back to specific lines of code in the program. Instrumented code can then be inserted at the appropriate location to implement the constraint imposed by the control place, which is treated as a global variable. This control mechanism only affects program execution when it reaches a point where the corresponding transition in the Gadara net is connected to a control place. The control synthesis task is therefore to determine a set of control places, their initial marking, and their connectivity to the net, such that the control logic enforced by these control places keeps the Gadara net live in a maximally-permissive manner. Moreover, the control places should never lead to the disablement of an uncontrollable transition in the net. In other words, the control logic enforced by these control places should correspond exactly to the supremal controllable sublanguage of the Gadara net subject to the live sublanguage specification mentioned earlier.

In our efforts so far, we have used the control technique called Supervision Based on Place Invariants (SBPI) to synthesize the desired control places. SBPI is a control logic synthesis framework that uses control places to enforce a set of linear inequality constraints on the reachable states of a given arbitrary Petri net [13]. Each linear inequality corresponds to a weighted sum of the number of tokens in each place of the net, and it will be *exactly* enforced by one control place, if enforceable at all; that is, the control is correct and maximally permissive with respect to the linear inequality. We have pursued two approaches that leverage the SBPI technique.

Assume that we can enumerate the set of reachable states of the Gadara net and calculate the supremal controllable sublanguage solution. This solution corresponds to a partition of the set of reachable states of the Gadara net into *legal* and *illegal* states. It is shown in [27] that this partition can be done using a set of *linear* inequalities on the states; in other words, the set of legal states is linearly separable. In [30], the problem of finding the *minimum* number of linear inequalities for effecting the desired separation of the state space is solved using concepts and techniques borrowed from classification theory. SBPI can then be invoked to obtain control logic that necessitates the minimum number of control places, which is highly correlated to the achievement of requirement (R4). This methodology is referred to as MSCL, for Marking Separation using CLassifiers, in subsequent works.

To avoid the explicit enumeration of the state space of the Gadara net that must be performed to calculate the supremal controllable sublanguage solution,

a control logic synthesis technique based on structural analysis of the Gadara net was developed. The general framework of this methodology, called ICOG for Iterative Control Of Gadara nets, is presented in [31], while its customization to the case of programs modeled by Gadara nets is presented in [32] and referred to as ICOG-O therein, since the nets involved remain ordinary throughout the iterations. This approach does not guarantee at the outset that the number of control places will be minimized. However, it leverages the structural properties associated with liveness analysis in Gadara nets from [27] in the context of an iterative scheme that eliminates illegal states by eliminating so-called *resource-induced empty siphons* [20, 27]. ICOG employs siphon analysis, coupled with SBPI, as well as a book-keeping mechanism to ensure convergence in a finite number of iterations. At convergence, a set of control places that separates the set of legal states from the set of illegal states is obtained. ICOG explicitly considers the controllability properties of transitions when synthesizing the control logic, so that no control place has an outgoing arc to an uncontrollable transition. In effect, ICOG computes the supremal controllable sublanguage solution by iterating directly on the Gadara net structure, using the notion of resource-induced empty siphon to capture illegal states.

4 Discussion

The principal focus of the Gadara project so far has been the problem of circular-wait-mutex deadlock in multithreaded software. This is an important problem due to the prevalence of multicore computer architectures. There are numerous other software problems where we believe control engineering techniques from the field of DES hold great promise. These include other types of deadlocks, such as reader-writer deadlock, condition wait/signal deadlock, inter-process deadlock, and other concurrency issues such as race, atomicity violation, and priority inversion. Results on the case of reader-writer deadlocks have recently appeared in [33], while certain types of atomicity violations have been addressed in [17, 18].

A key challenge that is posed by the consideration of reader-writer locks stems from the fact that the underlying state space is not necessarily finite; this is because one can perceive this class of RAS as one where in writing mode, the capacity of the resource is "one," while in reading mode, it is "infinite." This obstacle is addressed in [33] by taking advantage of special structure that exists in the set of inadmissible states, which enables a finite representation of this set through its minimal elements.

Detecting atomicity violations is substantially more difficult than deadlock detection. In [18], a class of atomicity violation bugs called "single-variable atomicity violations" is considered. Gadara nets are employed to capture this class of bugs by control specifications expressed as linear inequalities on the net marking; adjustments to the construction of the Gadara net at modeling time are necessary to make this possible. After adding one monitor place to enforce each linear inequality using SBPI, the ICOG methodology is then directly applied on the resulting controlled Gadara net to eliminate potential deadlocks introduced

by these control places. If one is interested in obtaining the minimum-size controller, in terms of number of added control places, then MSCL can be employed, albeit the process is more involved.

5 Conclusion

The application of the control engineering paradigm and of DES techniques to software failure avoidance opens up new avenues of research that cover the gamut from theory to implementation. While some of the above-mentioned opportunities can be solved by existing DES control theory, better customized solutions are often desirable. A crucial issue is scalability, which often necessitates the development of customized algorithms that exploit problem structure. Another crucial issue is the requirement on run-time overhead of the control logic in software applications, which is much more stringent than in other application areas such as manufacturing systems or process control, for instance. This leads to numerous opportunities to advance the state-of-the-art of DES control theory. Collaboration with domain experts is essential to construct suitable models and to understand the implementation constraints of the control logic. We wish to encourage students and researchers to consider contributing to this very promising emerging area of research.

References

1. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. Wiley (2004)
2. Wallace, C., Jensen, P., Soparkar, N.: Supervisory control of workflow scheduling. In: Proc. International Workshop on Advanced Transaction Models and Architectures (1996)
3. Phoha, V.V., Nadgar, A.U., Ray, A., Phoha, S.: Supervisory control of software systems. IEEE Transactions on Computers 53(9), 1187–1199 (2004)
4. Liu, C., Kondratyev, A., Watanabe, Y., Desel, J., Sangiovanni-Vincentelli, A.: Schedulability analysis of Petri nets based on structural properties. In: Proc. International Conference on Application of Concurrency to System Design (2006)
5. Wang, Y., Kelly, T., Lafortune, S.: Discrete control for safe execution of IT automation workflows. In: Proc. ACM EuroSys Conference (2007)
6. Dragert, C., Dingel, J., Rudie, K.: Generation of concurrency control code using discrete-event systems theory. In: Proc. ACM International Symposium on Foundations of Software Engineering (2008)
7. Auer, A., Dingel, J., Rudie, K.: Concurrency control generation for dynamic threads using discrete-event systems. In: Proc. Allerton Conference on Communication, Control and Computing (2009)
8. Gamatie, A., Yu, H., Delaval, G., Rutten, E.: A case study on controller synthesis for data-intensive embedded system. In: Proc. International Conference on Embedded Software and Systems (2009)
9. Iordache, M.V., Antsaklis, P.J.: Concurrent program synthesis based on supervisory control. In: Proc. 2010 American Control Conference, pp. 3378–3383 (2010)

10. Delaval, G., Marchand, H., Rutten, E.: Contracts for modular discrete controller synthesis. In: Proc. ACM Conference on Languages, Compilers and Tools for Embedded Systems (2010)
11. Gadara Team: Gadara project, `http://gadara.eecs.umich.edu/`
12. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. 25(1) (1987)
13. Moody, J.O., Antsaklis, P.J.: Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer Academic Publishers, Boston (1998)
14. Kelly, T., Wang, Y., Lafortune, S., Mahlke, S.: Eliminating concurrency bugs with control engineering. IEEE Computer 42(12), 52–60 (2009)
15. Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., Mahlke, S.A.: Gadara: Dynamic deadlock avoidance for multithreaded programs. In: Proc. the 8th USENIX Symposium on Operating Systems Design and Implementation, 281–294 (2008)
16. Wang, Y., Lafortune, S., Kelly, T., Kudlur, M., Mahlke, S.: The theory of deadlock avoidance via discrete control. In: Proc. the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 252–263 (2009)
17. Liu, P., Zhang, C.: Axis: Automatically fixing atomicity violations through solving control constraints. In: International Conference on Software Engineering (2012)
18. Wang, Y., Liu, P., Kelly, T., Lafortune, S., Reveliotis, S., Zhang, C.: On atomicity enforcement in concurrent software via discrete event systems theory. In: Proc. the 51st IEEE Conference and Decision and Control (2012)
19. Li, Z., Zhou, M., Wu, N.: 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 (2008)
20. Reveliotis, S.A.: Real-Time Management of Resource Allocation Systems: A Discrete-Event Systems Approach. Springer, New York (2005)
21. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
22. Kavi, K.M., Moshtaghi, A., Chen, D.: Modeling multithreaded applications using Petri nets. International Journal of Parallel Programming 35(5), 353–371 (2002)
23. Murata, T., Shenker, B., Shatz, S.M.: Detection of Ada static deadlocks using Petri net invariants. IEEE Transactions on Software Engineering 15(3), 314–326 (1989)
24. Iordache, M.V., Antsaklis, P.J.: Petri nets and programming: A survey. In: Proc, American Control Conference, 4994–4999 (2009)
25. Engler, D., Ashcraft, K.: RacerX: Effective, static detection of race conditions and deadlocks. In: Proc. 19th ACM Symposium on Operating Systems Principles (2003)
26. Cho, H.K., Wang, Y., Liao, H., Kelly, T., Lafortune, S., Mahlke, S.: Practical lock/unlock pairing for concurrent programs. In: Proc. 2013 International Symposium on Code Generation and Optimization, CGO 2013 (February 2013)
27. Liao, H., Wang, Y., Cho, H.K., Stanley, J., Kelly, T., Lafortune, S., Mahlke, S., Reveliotis, S.: Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets. Discrete Event Dynamic Systems: Theory & Applications 23 (2013) (published online May 2012)
28. Cano, E.E., Rovetto, C.A., Colom, J.M.: An algorithm to compute the minimal siphons in S^4PR nets. In: Proc. International Workshop on Discrete Event Systems, pp. 18–23 (2010)
29. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer (2008)

30. Nazeem, A., Reveliotis, S., Wang, Y., Lafortune, S.: Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: The linear case. IEEE Transactions on Automatic Control 56(8), 1818–1833 (2011)
31. Liao, H., Lafortune, S., Reveliotis, S., Wang, Y., Mahlke, S.: Optimal liveness-enforcing control of a class of petri nets arising in multithreaded software. IEEE Transactions on Automatic Control 58(5) (2013)
32. Liao, H., Wang, Y., Stanley, J., Lafortune, S., Reveliotis, S., Kelly, T., Mahlke, S.: Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control. IEEE Transactions on Control Systems Technology (2013) (published online January 2013)
33. Nazeem, A., Reveliotis, S.: Maximally permissive deadlock avoidance for resource allocation systems with r/w-locks. In: Proc. the 11th International Workshop on Discrete Event Systems (2012)