Pun, Ka I.; **Steffen, Martin**; **Stolz, Volker**

Deadlock checking by a behavioral effect system for lock handling. (English) Zbl 1246.68097

J. Log. Algebr. Program. 81, No. 3, 331-354 (2012).

Summary: Deadlocks are a common error in programs with lock-based concurrency and are hard to avoid or even to detect. One way for deadlock prevention is to statically analyze the program code to spot sources of potential deadlocks. Often static approaches try to confirm that the lock-taking adheres to a given order, or, better, to infer that such an order exists. Such an order precludes situations of cyclic waiting for each other's resources, which constitute a deadlock.

In contrast, we do not enforce or infer an explicit order on locks. Instead we use a behavioral type and effect system that, in a first stage, checks the behavior of each thread or process against the declared behavior, which captures potential interaction of the thread with the locks. In a second step on a global level, the state space of the behavior is explored to detect potential deadlocks. We define a notion of deadlock-sensitive simulation to prove the soundness of the abstraction inherent in the behavioral description. Soundness of the effect system is proven by subject reduction, formulated such that it captures deadlock-sensitive simulation.

To render the state-space finite, we show two further abstractions of the behavior sound, namely restricting the upper bound on re-entrant lock counters, and similarly by abstracting the (in general context-free) behavioral effect into a coarser, tail-recursive description. We prove our analysis sound using a simple, concurrent calculus with re-entrant locks.

MSC:

68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.)
68Q85 Models and methods for concurrent and distributed computing (process algebras, bisimulation, transition nets, etc.)

Keywords:

concurrency; deadlock prevention; static analysis; behavioral type and effect systems; simulation relation; abstraction

Software:

Cyclone; Haskell; RacerX

Full Text: DOI

References:

[1] Amtoft, T.; Nielson, H.R.; Nielson, F., Type and effect systems: behaviours for concurrency, (1999), Imperial College Press

[2] Agarwal, Rahul; Wang, Liqiang; Stoller, Scott D., Detecting potential deadlocks with state analysis and run-time monitoring, (), 191-207

[3] Blieberger, Johann; Burgstaller, Bernd; Scholz, Bernhard, Symbolic data flow analysis for detecting deadlocks in ada tasking programs, (), 225-237

[4] Bartoletti, Massimo; Degano, Pierpaolo; Ferrari, Gian Luigi; Zunino, Roberto, ν-types for effects and freshness analysis, (), 80-95 · Zbl 1250.68085

[5] Boyapati, Chandrasekhar; Lee, Robert; Rinard, Martin, Ownership types for safe programming: preventing data races and deadlocks, ()

[6] Bruni, R.; Mezzina, L.G., Types and deadlock freedom on calculus of services and sessions, (), 100-115 · Zbl 1170.68428

[7] Boyapati, Chandrasekhar; Salcianu, Alexandru; Beebee, William; Rinard, Martin, Ownership types for safe region-based memory management in real-time Java, ()

[8] Baeten, J.C.M.; van Glabbeek, R.J., Merge and termination in process algebra, (), 153-172, (also as CWI report CS-R8716) · Zbl 0636.68024

[9] Coffman, E.G.; Elphick, M.; Shoshani, A., System deadlocks, Comput. surv., 3, 2, 67-78, (1971) · Zbl 0226.68015

[10] Clarke, Edmund; Grumberg, Orna; Jha, Somesh; Lu, Yuan; Veith, Helmut, Counterexample-guided abstraction refinement, (), 154-169 · Zbl 0974.68517

[11] Clarke, Edmund M.; Grumberg, Orna; Peled, Doron, Model checking, (1999), MIT Press

[12] Christiansen, Jan; Huch, Frank, Searching for deadlocks while debugging concurrent Haskell programs, (), 28-39 · Zbl 1323.68104

[13] Corbett, J., Evaluating deadlock detection methods for concurrent software, IEEE trans. softw. eng., 22, 3, 161-180, (1996)

[14] de Boer, Frank S.; Grabe, Immo, Automated deadlock detection in synchronized reentrant multithreaded call-graphs, (), 200-211 · Zbl 1274.68084

[15] Deitel, Harvey M., An introduction to operating systems, (1984), Addison-Wesley · Zbl 0599.68003

[16] Dawson R. Engler, Ken Ashcraft, RacerX: effective, static detection of race conditions and deadlocks, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003, pp. 237-252.

[17] Flanagan, C.; Abadi, M., Object types against races, (), 288-303

[18] Flanagan, C.; Abadi, M., Types for safe locking, (), 91-108

[19] Cormac Flanagan, Stephen Freund, Type-based race detection for Java, in: Proceedings of PLDI'00, ACM SIGPLAN Conference on ACM Conference on Programming Language Design and Implementation, 2000, pp. 219-232.

[20] Manuel Fähndrich, Sriram K. Rajamani, Jakob Rehof, Static deadlock prevention in dynamically configured communication network, in: Perspectives in Concurrency, Festschrift in Honor of P.S. Thiagarajan, 2008, pp. 128-156. · Zbl 1193.68161

[21] D. Grossman, Type-safe multithreading in Cyclone, in: TLDI'03: Types in Language Design and Implementation, 2003, pp. 13-25.

[22] Richard Craig Holt, On Deadlock in Computer Systems, Ph.D. thesis, Cornell University, Ithaca, NY, USA, 1971.

[23] Igarashi, Atsushi; Kobayashi, Naoki, A generic type system for the pi-calculus, (), 128-141 · Zbl 1323.68410

[24] The Java tutorials: Concurrency. Available from: <download.oracle.com/javase/tutorial/essential/concurrency>, 2011.

[25] Kahlon, Vineet; Ivancic, Franjo; Gupta, Aarti, Reasoning about threads communicating via locks, (), 505-518 · Zbl 1081.68623

[26] Kobayashi, Naoki, A partially deadlock-free typed process calculus, ACM trans. program. lang. syst., 20, 2, 436-482, (1998), (an extended abstract previously appeared in the Proceedings of LICS '97, pp. 128-139)

[27] Kobayashi, Naoki, Type-based information flow analysis for the π-calculus, Acta inform., 42, 4-5, 291-347, (2005) · Zbl 1081.68061

[28] Kobayashi, Naoki, A new type system for deadlock-free processes, (), 233-247 · Zbl 1151.68537

[29] Kobayashi, Naoki; Saito, Shin; Sumii, Eijiro, An implicitly-typed deadlock-free process calculus, (), 489-503 · Zbl 0999.68532

[30] Kurshan, R.P., Computer-aided verification of coordinating processes, the automata theoretic approach, Princeton series in computer science, (1994), Princeton University Press

[31] Levine, Gertrude Neuman, Defining deadlock, SIGOPS oper. syst. rev., 37, 1, 54-64, (2003)

[32] Milner, Robin, An algebraic definition of simulation between programs, (), 481-489

[33] Naik, Mayur; Aiken, Alex, Conditional must not aliasing for static race detection, () · Zbl 1295.68073

[34] Nielson, Flemming; Nielson, Hanne-Riis; Hankin, Chris L., Principles of program analysis, (1999), Springer-Verlag · Zbl 0932.68013

[35] Ka I Pun, Martin Steffen, Volker Stolz, Deadlock checking by a behavioral effect system for lock handling. Technical report 404, University of Oslo, Dept. of Informatics, 2011. · Zbl 1246.68097

[36] Roscoe, Bill; Dathi, Naiem, The pursuit of deadlock freedom, Inform. comput., 75, 3, 289-327, (1987) · Zbl 0626.68019

[37] Stolz, Volker, Temporal assertions with parametrized propositions, J. logic comput., 20, 3, 743-757, (2010) · Zbl 1203.68103

[38] Tachio Terauchi, Checking race freedom via linear programming, in: Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '08, New York, NY, USA, 2008, ACM, pp. 1-10. · Zbl 1133.68387

[39] van Glabbeek, Rob; Goltz, Ursula, Refinement of actions and equivalence notions for concurrent systems, Acta inform., 37, 4/5, 229-327, (2001) · Zbl 0969.68081

[40] Williams, Amy; Thies, William; Ernst, Michael D., Static deadlock detection for Java libraries, ()