Scalable and Precise Program Analysis at NEC

Gogul Balakrishnan Malay K. Ganai Aarti Gupta Franjo Ivančić Vineet Kahlon Weihong Li
Naoto Maeda Nadia Papakonstantinou Sriram Sankaranarayanan* Nishant Sinha Chao Wang

NEC Laboratories America, NJ, USA
http://www.nec-labs.com/research/system/systems_SAV-website

*University of Colorado Boulder, CO, USA

***Abstract*—The Systems Analysis & Verification Department at NEC Labs engages in foundational as well as applied research in the areas of verification and analysis of software and embedded systems. We have developed several tools and frameworks for scalable and precise analysis of programs, some of which are now used within the company on large software projects. This extended abstract highlights their main features and provides pointers to published papers with more details.**

I. F-Soft

F-Soft is a platform for verifying source code programs [3], [4]. It can check C programs for runtime errors (such as pointer access violations, buffer overflow, memory leaks), standard library API usage, and other user-defined assertions. It can be used to check programs globally, or as an annotation checker to check user-written contracts. We have successfully analyzed large benchmark examples to find previously unknown bugs using F-Soft. An in-house product based on F-Soft, called VARVEL, is currently in use in NEC. We are now extending F-Soft to handle C++ programs.

Overall, F-Soft provides a cooperative, staged framework of various static analyses (including abstract interpretation) and model checking techniques, where the size and complexity of the model and the number of properties are successively reduced across the stages. This enables efficient use of high-precision analyses when needed, and is very effective for handling large programs with hundreds of (automatically instrumented) properties.

II. CoBe (Concurrency Bench)

CoBe is a tool for finding concurrency-related bugs in multi-threaded C programs, such as data races, deadlocks, atomicity violations, and missed notifies. It leverages a combination of static analysis, dataflow analyses, and symbolic model checking.

CoBe starts by using statically computed lockset and lock acquisition history information [7], [5], as well as happens-before constraints induced by synchronization primitives and properties [9] to generate bug warnings. Each warning is then analyzed by employing a series of static analyses in a 'telescoping' fashion, i.e., in increasing order of precision but decreasing order of scalability to decide, as cheaply as possible, whether the warning is bogus. These static analyses exploit both static constraints, i.e., interleaving constraints arising from the use of synchronization primitives like Wait/Notify (rendezvous), Wait/NotifyAll (broadcasts), etc., and semantic constraints resulting from data flow. The semantic constraints are generated by deriving sound invariants, using abstract interpretation on domains with increasing precision (range, octagonal, and polyhedral analyses). These two classes of analyses, when used in conjunction, can weed out a large fraction of the bogus warnings [8]. Finally, symbolic model checking is leveraged to generate concrete error traces for the remaining warnings [6]. Telescoping greatly enhances the efficacy of the overall process by ensuring scalability without compromising on precision. We have successfully handled many large examples, including Linux device drivers.

III. Fusion

Fusion is a platform for combining dynamic and static verification techniques, for the purpose of finding bugs in concurrent C/C++ programs. Given a test case, it executes the program to derive a *Concurrent Trace Program (CTP)* that captures that trace. The CTP can be used for exploration of alternate thread schedules, which is difficult to do in standard testing where the thread schedule is controlled by the operating system. Specifically, we have used CTPs to combine an explicit search (in the style of dynamic partial order reduction) with SMT-based analysis to improve its performance and coverage [11]. We have also used CTPs for symbolic predictive analysis, where we can detect assertion violations or atomicity violations in alternate interleavings of the same events observed in the given trace [12], [13].

For long traces, exploring all interleavings of the events can be prohibitively expensive. We are investigating additional static analysis techniques that can quickly prune away some warnings [9]. We are also studying coverage-based metrics to guide systematic testing to explore only high-risk interleavings that are likely to lead to bugs.

IV. Contessa

Testing of multi-threaded programs poses enormous challenges. To improve the coverage of testing, we present a framework named CONTESSA [10] that augments conventional testing (concrete execution) with symbolic analysis in a scalable and efficient manner to explore both thread interleaving and input data space. It works on CTPs generated by Fusion. It utilizes partial-order reduction techniques [1] to generate verification conditions with reduced size and search

space [2]. These verification conditions are checked by an SMT-solver that can generate witness traces for bugs. The tool also provides visual support for debugging the witness traces.

ACKNOWLEDGMENT

We would like to gratefully acknowledge the support and efforts of the Varvel Development team and the Software Development Environment Engineering Division team in NEC Japan.

REFERENCES

[1] Malay K. Ganai and Sudipta Kundu. Reduction of verification conditions for concurrent system using mutually atomic transactions. In *SPIN*, pages 68–87, 2009.

[2] Malay K. Ganai and Chao Wang. Interval analysis for concurrent trace programs using transaction sequence graphs. In *International Conference on Runtime Verification (RV 10)*, 2010.

[3] Franjo Ivancic, Ilya Shlyakhter, Aarti Gupta, Malay K. Ganai, Vineet Kahlon, Chao Wang, and Zijiang Yang. Model checking C programs using F-SOFT. In *ICCD*, pages 297–308, 2005.

[4] Franjo Ivancic, Zijiang Yang, Malay K. Ganai, Aarti Gupta, Ilya Shlyakhter, and Pranav Ashar. F-Soft: Software verification platform. In *CAV*, pages 301–306, 2005.

[5] Vineet Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks. In *LICS*, pages 27–36, 2009.

[6] Vineet Kahlon, Aarti Gupta, and Nishant Sinha. Symbolic model checking of concurrent programs using partial orders and on-the-fly transactions. In *CAV*, pages 286–299, 2006.

[7] Vineet Kahlon, Franjo Ivancic, and Aarti Gupta. Reasoning about threads communicating via locks. In *CAV*, pages 505–518, 2005.

[8] Vineet Kahlon, Sriram Sankaranarayanan, and Aarti Gupta. Semantic reduction of thread interleavings in concurrent programs. In *TACAS*, pages 124–138, 2009.

[9] Vineet Kahlon and Chao Wang. Universal causality graphs: A precise happens-before model for detecting bugs in concurrent programs. In *CAV*, pages 434–449, 2010.

[10] Sudipta Kundu, Malay K. Ganai, and Chao Wang. Contessa: Concurrency testing augmented with symbolic analysis. In *CAV*, pages 127–131, 2010.

[11] Chao Wang, Swarat Chaudhuri, Aarti Gupta, and Yu Yang. Symbolic pruning of concurrent program executions. In *ESEC/SIGSOFT FSE*, pages 23–32, 2009.

[12] Chao Wang, Sudipta Kundu, Malay K. Ganai, and Aarti Gupta. Symbolic predictive analysis for concurrent programs. In *FM*, pages 256–272, 2009.

[13] Chao Wang, Rhishikesh Limaye, Malay K. Ganai, and Aarti Gupta. Trace-based symbolic analysis for atomicity violations. In *TACAS*, pages 328–342, 2010.