Building Your Own Software Model Checker Using the Bogor Extensible Model Checking Framework*

Matthew B. Dwyer[1], John Hatcliff[2], Matthew Hoosier[2], and Robby[2]

[1] University of Nebraska,
Lincoln, NE 68588, USA
dwyer@cse.unl.edu
[2] Kansas State University,
Manhattan, KS 66506, USA
{hatcliff, matt, robby}@cis.ksu.edu

Abstract. Model checking has proven to be an effective technology for verification and debugging in hardware and more recently in software domains. We believe that recent trends in both the requirements for software systems and the processes by which systems are developed suggest that domain-specific model checking engines may be more effective than general purpose model checking tools. To overcome limitations of existing tools which tend to be monolithic and non-extensible, we have developed an extensible and customizable model checking framework called Bogor. In this tool paper, we summarize (a) Bogor's direct support for modeling object-oriented designs and implementations, (b) its facilities for extending and customizing its modeling language and algorithms to create domain-specific model checking engines, and (c) pedagogical materials that we have developed to describe the construction of model checking tools built on top of the Bogor infrastructure.

Motivation

Temporal logic model checking [CGP00] is a powerful framework for reasoning about the behavior of finite-state system descriptions and it has been applied, in various forms, to reasoning about a wide-variety of software artifacts. The effectiveness of these efforts has in most cases relied on detailed knowledge of the model checking framework being applied. In some cases, a *new* framework was developed targeted to the semantics of a family of artifacts [BHPV00], while in other cases it was necessary to study an *existing* model checking framework in detail in order to customize it [CAB+01]. Unfortunately, the level of knowledge and effort required to do this kind of work currently prevents many *domain* experts, who are not necessarily experts in model-checking, from successfully

* This work was supported in part by a 2004 IBM Eclipse Innovation Grant, by the U.S. Army Research Office (DAAD190110564), by DARPA/IXO's PCES program (AFRL Contract F33615-00-C-3044), and by NSF (CCR-0306607, CCF-0429149, CCF-04444167).

K. Etessami and S.K. Rajamani (Eds.): CAV 2005, LNCS 3576, pp. 148–152, 2005.

applying model checking to systems and software analysis problems. Our broad goal is to allow these experts to apply model checking without the need to build their own model-checker or to pour over the details of an existing model-checker implementation while carrying out substantial modifications.

The Bogor Extensible Software Model Checking Framework

To meet the challenges of using model checking in the context of current trends in software development, we have constructed an extensible and highly modular explicit-state model checking framework called Bogor [RDH03, SAnToS03]. Using Bogor, we seek to enable more effective incorporation of domain knowledge into verification models and associated model checking algorithms and optimizations, by focusing on the following principles.

Software-oriented Modeling Language: In contrast to most existing model checkers, Bogor's modeling language (BIR) provides constructs commonly found in modern programming languages including dynamic creation of objects and threads, garbage collection, virtual dispatch and exceptions. This rich language has enabled model checking relatively large featureful concurrent Java programs by translating them to Bogor using the next generation of the Bandera tool set.

Software-oriented State Representations and Reduction Algorithms: To support effective checking of BIR software models, we have adapted and extended well-known optimization/reduction strategies such as collapse compression [Hol97], data and thread symmetry [BDH02], and partial-order reductions to support models of object-oriented software by providing sophisticated heap representations [RDHI03], partial-order reduction strategies that leverage static and dynamic escape and locking analyses [DHRR04], and thread and heap symmetry strategies[Ios02, RDHI03].

Extensible Modeling Language: Bogor's modeling language can be extended with new primitive types, expressions, and commands associated with a particular domain (e.g, multi-agent systems, avionics, security protocols, etc.) and a particular level of abstraction (e.g., design models, source code, byte code, etc.)

Open Modular Architecture: Bogor's well-organized module facility allows new algorithms (e.g., for state-space exploration, state storage, etc) and new optimizations (e.g., heuristic search strategies, domain-specific scheduling, etc.) to be easily swapped in to replace Bogor's default model checking algorithms.

Robust Feature-rich Graphical Interface: Bogor is written in Java and comes wrapped as a plug-in for *Eclipse* – an open source and extensible universal tool platform from IBM. This user interface provides mechanisms for collecting and naming different Bogor configurations, specification property collections, and a variety of visualization and navigation facilities.

Design for Encapulation: Bogor provides an open architecture with well-defined APIs and intermediate data formats that enable it (and customized versions of it) to be easily encapsulated within larger development/verification environments for specific domains.

Courseware and Pedagogical Materials: Even with a tool like Bogor that is designed for extensibility, creating customizations requires a significant amount of knowledge about the internal Bogor architecture. To communicate this knowledge, we have developed an extensive collection of tutorial materials and examples. Moreover, we believe that Bogor is an excellent pedagogical vehicle for teaching foundations and applications of model checking because it allows students to see clean implementations of basic model checking algorithms and to easily enhance and extend these algorithms in course projects. Accordingly, we have developed a comprehensive collection of course materials [SAnToS04] that have already been used in graduate level courses on model checking at several institutions.

In short, Bogor aims to be not only a robust and feature-rich software model checking tool that handles the language constructs found in modern large-scale software system designs and implementations, it also aims to be a model checking *framework* that enables researchers and engineers to create families of domain-specific model checking engines.

Experience Using Bogor

In the past ten months, Bogor has been downloaded more than 800 times by individuals in 22 countries. We know that many of those individuals are using Bogor in interesting ways. To date, we are aware of more than 28 substantive extensions to Bogor that have been built by 18 people, only one of whom was the primary Bogor developer.

It is difficult to quantify the effort required to build a high-quality extension in Bogor. As with all software framework there is a learning curve. In the case of Bogor, which is a non-trivial system consisting more than 22 APIs, we find that reasonably experienced Java developers get up to speed in a couple of weeks. At that point extensions are generally require only a few hundred lines of code and often they can be modeled closely after already existing extensions. To give a sense of the variety of extensions built with Bogor we list a sampling of those extensions and indicate, in parentheses, the number of non-comment source statement lines of Java code used to implement the extension.

Partial-order Reduction (POR) Extensions: Multiple variations on POR techniques have been implemented in Bogor including: sleep sets (298), conditional stubborn sets (618), and ample sets (306) approaches. Multiple variations of the notion of dependence have been incorporated into these techniques that increase the size of the independence relation by exploiting : read-only data (515), patterns of locking (73), patterns of object ownership (69), and escape information (216). These latter reductions, while modest in size and complexity to implement, have resulted in more than four orders of magnitude reduction in model checking concurrent Java programs [DHRR04].

State-encoding and Search Extensions: Bogor is factored into separate modules that can be treated independently to help lower the cost of learning the framework's APIs. For example, extensions to the state-encoding and management APIs have yielded implementations of collapse compression (483), heap

and thread symmetry (317), and symmetric collection data structures (589). Extensions to Bogor's searcher APIs have enabled the POR extensions above in addition to ones supporting stateless search (14) and heuristic selective search (641).

Property Extensions: Supporting different property languages is just as important as supporting flexibility in modeling languages. Bogor's property APIs have allowed multiple checker extensions to be implemented including : regular expression/finite-state automata (1083), an automata-theoretic Linear Temporal Logic (1011) checker, and a Computation-tree Logic (1418) checker based on alternating tree automata. We have also implemented a checker extension for the Java Modeling Language [RRDH04] (3721).

Problem Domain Extensions: A main objective of Bogor was to bring sophisticated state-space analyses to a range of systems and software engineering domains. Several extensions have been built that target specific issues in reasoning about multi-threaded Java programs, for example, treating dynamic class loading (425), reasoning about event-handler behavior in program written using the Swing framework [DRTV04], and reasoning about properties of method atomicity (359) [HRD04].

Departing from the software domain somewhat, in our work on the Cadena development environment [HDD+03] for designing component-based avionics systems, we have extended Bogor's modeling language to include APIs associated with the CORBA component model and an underlying real-time CORBA event service (2593). [DDH+02, DRDH03]. For checking avionics system designs in Cadena, we have customized Bogor's scheduling strategy to reflect the scheduling strategy of the real-time CORBA event channel (439), and created a customized parallel state-space exploration algorithm that takes advantage of properties of periodic processing in avionics systems (516). These customizations for Bandera and Cadena have resulted in space and time improvements of over three orders of magnitude compared to our earlier approaches.

We are currently building extensions of Bogor for checking highly dynamic multi-agent systems. Researchers outside of our group are extending Bogor to support checking of programs constructed using AspectJ, and UML designs. Bogor is targetted as a framework for explicit state checking, and its current architecture is not necessarily amenable for incorporating symbolic techniques. We are working with researchers at Brigham Young University to refactor the framework (or develop an alternate set of APIs) to facilitate the use of symbolic techniques.

References

[BDH02] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric SPIN. *International Journal on Software Tools for Technology Transfer*, 4(1):92–106, 2002.

[BHPV00] G. Brat, K. Havelund, S. Park, and W. Visser. Java PathFinder – a second generation of a Java model-checker. In *Proceedings of the Workshop on Advances in Verification*, July 2000.

[CAB+01] W. Chan, R. J. Anderson, P. Beame, D. Notkin, D. H. Jones, and William E. Warner. Optimizing symbolic model checking for statecharts. *IEEE Transactions on Software Engineering*, 27(2):170–190, 2001.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[DDH+02] W. Deng, M. Dwyer, J. Hatcliff, G. Jung, and Robby. Model-checking middleware-based event-driven real-time embedded software. In *Proceedings of the 1st Internatiuonal Symposium on Formal Methods for Component and Objects*, pages 154–181, 2002.

[DHRR04] M. B. Dwyer, J. Hatcliff, Robby, and V. R.Prasad. Exploiting object escape and locking information in partial order reduction for concurrent object-oriented programs. *Formal Methods in System Design*, 25(2-3):199–240, 2004.

[DRDH03] M. B. Dwyer, Robby, X. Deng, and J. Hatcliff. Space reductions for model checking quasi-cyclic systems. In *Proceedings of the Third International Conference on Embedded Software*, pages 173–189, 2003.

[DRTV04] M. B. Dwyer, Robby, O. Tkachuk, and W. Visser. Analyzing interaction orderings with model checking. In *Proceedings of the 19th IEEE Conference on Automated Software Engineering*, pages 154–163, 2004.

[HDD+03] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An integrated development, analysis, and verification environment for component-based systems. In *Proceedings of the 25th International Conference on Software Engineering*, pages 160–173, 2003.

[Hol97] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–294, May 1997.

[HRD04] J. Hatcliff, Robby, and M. B. Dwyer. Verifying atomicity specifications for concurrent object-oriented software using model checking. In M. Young, editor, *Proceedings of the Fifth International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2004)*, volume 2937 of *Lecture Notes In Computer Science*, pages 175–190, Jan 2004.

[Ios02] R. Iosif. Symmetry reduction criteria for software model checking. In *Proceedings of Ninth International SPIN Workshop*, volume 2318 of *Lecture Notes in Computer Science*, pages 22–41. Springer-Verlag, April 2002.

[SAnToS03] SAnToS Laboratory. Bogor website. `http://bogor.projects.cis.ksu.edu`, 2003.

[SAnToS04] SAnToS Laboratory. Software Model Checking course materials website. `http://model-checking.courses.projects.cis.ksu.edu`, 2004.

[RDH03] Robby, M. B. Dwyer, and J. Hatcliff. Bogor: An extensible and highly-modular model checking framework. In *Proceedings of the 9th European Software Engineering Conference held jointly with the 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 267–276, 2003.

[RDHI03] Robby, M. B. Dwyer, J. Hatcliff, and R. Iosif. Space-reduction strategies for model checking dynamic software. In *Proceedings of the 2nd Workshop on Software Model Chekcing*, volume 89(3) of *Electronic Notes in Theoritical Computer Science*, 2003.

[RRDH04] Robby, E. Rodríguez, M. B. Dwyer, and J. Hatcliff. Checking strong specifications using an extensible software model checking framework. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 404–420, March 2004.