Supporting Model Checking Education using BOGOR/Eclipse*

Matthew B. Dwyer
Department of CSE
University of Nebraska-Lincoln, US
dwyer@cse.unl.edu

John Hatcliff Robby Matthew Hoosier
Department of CIS
Kansas State University, US
{hatcliff, robby, matt}@cis.ksu.edu

Abstract

This paper describes our effort on developing educational materials on model checking to help foster the adoption of software model checking. We describe the course materials that provide an in-depth theoretical background of model checking algorithms, coupled with a tool support to apply them that is built around the Eclipse platform. The educational materials presented here have been used to teach graduate-level model checking courses in a number of North American and European institutions.

1 Introduction

Temporal logic model checking [1] 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. For example, model checking frameworks have been applied to reason about software process models, (e.g., [2]), different families of software requirements models (e.g., [3]), architectural frameworks (e.g., [4]), design models (e.g., [5]), and system implementations (e.g., [6, 7]). 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 [6], while in other cases it was necessary to study an *existing* model checking framework in detail in order to customize it [3, 8].

Unfortunately, this level of knowledge and effort currently prevents many *domain* experts who are not necessarily experts in model-checking from successfully applying model checking to software analysis. Even though experts in different areas of software engineering have significant domain knowledge about the semantic primitives and properties of families of artifacts that could be brought to bear to produce cost-effective semantic reasoning via model checking, these experts should not be required to build their own model-checker or to pour over the details of an existing model-checker implementation while carrying out substantial modifications.

To enable more effective incorporation of domain knowledge into verification models and associated model checking algorithms and optimizations, we have constructed in Eclipse an extensible and highly modular explicit-state model checking framework called Bogor [9].[1] There are two dimensions to Bogor's extensibility: (1) 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.), and (2) 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.

Our own experience of customizing Bogor with domain-specific modeling primitives and optimizations has been quite positive. For analyzing models generated from Java source code in the next generation of Bandera [7], Bogor provides a rich base modeling language including features that allow for dynamic creation of objects and threads, garbage collection, virtual method calls and exception handling. For these primitives, we have extended Bogor's default algorithms to support state-of-the-art model reduction/optimization techniques that we have developed by customizing for object-oriented software existing techniques such as collapse compression [10], heap symmetry [11], thread symmetry [12], and partial-order reductions. In our work on the Cadena development environment [5] for designing component-based avionics systems, we have extended Bogor's modeling language to include APIs

*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), by NSF (CCR-0306607), by Lockheed Martin, by Rockwell-Collins, and by Intel Corporation (Grant 11462).

[1] `http://bogor.projects.cis.ksu.edu`

associated with the CORBA component model and an underlying real-time CORBA event service. [13, 14]. 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, and created a customized parallel state-space exploration algorithm that takes advantage of properties of periodic processing in avionics systems. These customizations for Bandera and Cadena have resulted in space and time improvements of over three orders of magnitude compared to our earlier approach [7, 5] which created models for Spin and dSpin. In other work, we have extended Bogor to support checking of specifications written in the Java Modeling Language (JML) [15] and GUI frameworks [16]. Researchers outside of our group have extended Bogor to support checking of programs using AspectJ.

The success and the expediency of the customization process were determined by several factors: (1) accessibility to domain knowledge, (2) intimate understanding of existing model checking algorithms, and (3) a model checking framework that allowed us to rapidly prototype ideas as concrete algorithms that we could experiment with. We believe that these factors influenced not only our specific experiences, but the experiences of others in applying Bogor as well. While issues in (1) should be addressed by the practitioners themselves, it is crucial for us to provide tutorial and reference material about Bogor's architecture and algorithms to enable others to successfully customize Bogor. Moreover, we believe Bogor itself 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 to include a variety of enhancements and optimizations.

In this paper, we give an overview of our educational materials that present foundations and applications of software model checking using Bogor, and we describe how the Eclipse framework enables a very effective presentation and implementation of Bogor and its extensibility features. Section 2 motivates (a) our pedagogical approach and teaching strategy, and (b) our choices of Bogor and Eclipse as platforms for presenting a course on model checking. Section 3 describes the educational materials that we are developing to achieve the goals described above. Section 4 describes our experiences when using the materials in classroom settings. We conclude in Section 5 with an outline of both on-going and future work to refine the materials.

2 Motivation

While there is a wide collection of model checking literature (e.g., [1]), however, the current practice of learning model checking is to use existing model checkers and to fit problems of interest to them. This is fine if one only wants to know how to use a model checker, however, as we mentioned earlier, model checking can be made significantly more effective through domain-specific customizations. Unfortunately, many existing model checkers such as SPIN [10] were not designed for extension, i.e., the implementation of the functional aspects of model checking is tangled and difficult to understand, let alone customize. In addition, there is a lack of documentation that relates how the algorithms in theoretical literature are implemented in practice. That is, many non-trivial and subtle software engineering issues that arise when implementing the algorithms efficiently are often not discussed.

Our goal is to develop project-oriented educational materials on model checking that provide the basis for a solid theoretical foundation coupled with tool support for experimentation and supporting documentation on how to implement, apply, and customize model checking algorithms. We believe this strategy enables students to better understand the algorithms described in theoretical settings and to also get hands-on experience regarding how they work.

We believe there is a strong analogy between the strategy above, and the strategy used in teaching compiler technology. Parsing evolved from a topic of purely theoretical interest, through the development of tools that embodied state-of-the-art parsing algorithms, to the situation today where there are mature tools in widespread use in practice. As this technology emerged from research into practice an accompanying shift in the pedagogy associated with compilers took place. Modern compiler course and textbooks [17] are project-oriented. Students learn the foundational concepts and those concepts are reinforced through the application of tools to solve realistic problems.

Why Bogor? In contrast to most existing model checkers, Bogor's modeling language (BIR) features high-level constructs commonly found in modern programming language such as dynamic thread and object creations, dynamic method dispatch, exception handling, etc. In addition, BIR is extensible, i.e., BIR allows the introductions of abstract data types and abstract operations as first-class constructs of the languages. This is analogous to adding a new instruction set and types in a virtual machine. This extensibility allows us to essentially build an abstract machine for each specific application domain. This customized abstract machine can be made to be more efficient because

the modeling language can move closer the abstraction level of the system description used for that particular application domain (e.g., Java bytecode and software design).

Furthermore, Bogor was designed to be modular and extensible to ease the task for understanding and customizing it. We untangled the functional aspects of model checking and made them as separate Bogor modules designed with open APIs using well-known design patterns [18] and their variations. This reduces dependency between the modules and allows local reasoning of each module. Each module of Bogor can be customized, and Bogor allows one to mix and match modules to achieve the desired abstract machine. Bogor is implemented in Java to allow a clean implementation of the modules in addition to being system independent.

We believe that the extensibility and open interfaces of Bogor make it ideal as a framework to support small to medium size projects involving the implementation of model checker components. As with compilers, this pedagogic approach will reinforce the foundational concepts learned in the course by requiring students to work with those concepts.

Why Eclipse? *Eclipse* is an open and extensible universal tool platform.[2] Eclipse provides a rich set of infrastructure for, for example, creating integrated development environment (IDEs), graphical editors, etc., that is ideal for building a user interface (UI) for a model checking tool. Eclipse provides a *plugin* facility via which one can add more features. The implementation language of Eclipse, Java, allows a tight integration of the tools. That is, we implemented Bogor as an Eclipse plugin, and in turn, Eclipse plugin facility could be used to implement replacement and extension Bogor modules. Once completed, these modules are made available to Bogor through Eclipse's own *extension point* mechanism, by which new plugins can contribute functionality to existing ones. All these features of Eclipse make it an ideal choice for developing Bogor's UI which includes a number of pedagogically-oriented features for supporting the educational materials that we are developing.

3 Educational Materials

We aim to provide a complete set of course materials packaged in a distribution that provides instructors with virtually all material necessary for running a high-quality semester-length course. The material will be packaged in an *instructor distribution* that includes solutions to exams, homeworks, etc. as well as a *student distribution* that contains all materials but exam, homework, and quiz solutions. The course is structured as a collection of modules that allow instructors to re-order, omit, or add content according to their own goals. Course modules include (1) algorithmic foundations, (2) Bogor architecture, (3) property specification and checking, (4) optimizations, abstraction and reduction, and (5) methodologies for using model checking.

Course Materials: The heart of the material is a set of course notes that present the technical material for the modules above and include exercises and suggestions for further reading. Accompanying the course notes and following a parallel structure is an on-line supplement that provides guided solutions to selected exercises as well as commentary on how technical concepts and algorithms described abstractly in the course notes are actually implemented in Bogor. Building off the course notes, the instructor distribution includes PowerPoint lecture slides, source code for lecture examples, weekly quizzes and solutions, lab exercises designed for interactive presentation, homeworks and solutions, exams and solutions. In addition, recordings of the authors presenting the lectures are available in streaming video format.

Students learn to apply Bogor to model and analyze simple concurrent systems that illustrate basic concepts of state-space exploration. Programming projects involve (re)implementing or modifying the core modules of Bogor's model checking engine, or implementing new modeling language primitives using Bogor's extensible modeling language. In addition to simply reinforcing the central concepts of model checking, the overall goal of these implementation exercises is to move students to the point where they can effectively develop model-checking tools and associated methodologies for verification of real world systems by tailoring Bogor to different application domains. The current draft of the course materials can be downloaded at our model checking course website.[3]

Methodological aspects of model checking (and Bogor, in particular) are also emphasized. This includes repeatable strategies for capturing concurrent/distributed systems as effective verification models, applying abstraction and other state-space reducing model transformations, and using a pattern-based approach to constructing temporal specifications.

Tool Support: As mentioned earlier, we leveraged the rich infrastructure already provided by Eclipse for developing Bogor UI. We developed a customized text editor for BIR that features syntax highlighting and a well-formed-ness checker (e.g., type checker) by extending Eclipse's basic text editor. If there is a syntax error or a type error, then the checker gives an informational message similar to Eclipse's Java Development Tooling

[2] `http://www.eclipse.org`

[3] `http://model-checking.courses.projects.cis.ksu.edu`

```
system SumToN {
  const PARAM { N = 5; }
  typealias byte int wrap (0, 255);

  byte x := 1;
  byte t1;
  byte t2;

  active[2] thread Worker() {
    loc loc0:
      do { t1 := x; } goto loc1;
    loc loc1:
      do { t2 := x; } goto loc2;
    loc loc2:
      do { x := t1 + t2; } goto loc0;
    }

  active thread Property() {
    loc loc0:
      do { assert (x != (byte) PARAM.N); }
      return;
    }
}
```

Figure 1: A simple BIR model

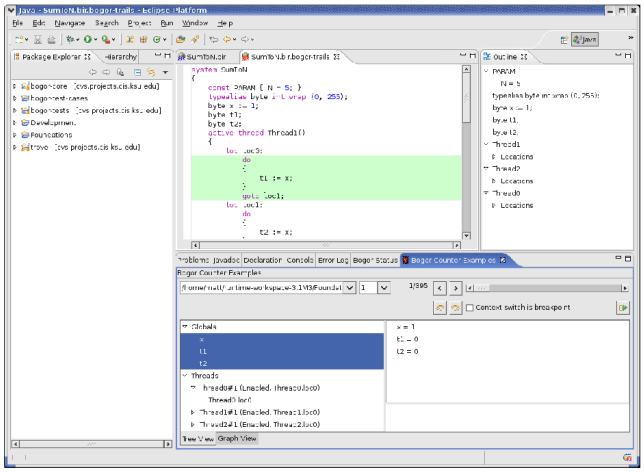

Figure 2: Counter-example view in Bogor

(JDT), i.e., by using problem markers that appear in the Eclipse problem view and also in the vertical rule of the editor that highlights the problematic lines. This helps Bogor users when modeling systems manually; for example, it helped our students when creating models for exercises, homeworks, and projects. The Bogor manual that includes BIR documentation (e.g., grammar, language description, abstract syntax tree implementation in Java, etc.) and Bogor extension tutorials is directly accessible through the Eclipse help system.

In addition to the integrated BIR editor and Bogor manual, the Bogor UI features a user-guided simulation mode and a counter-example display which mimics the "look and feel" of controls used in JDT's own debugger for Java applications. The counter-example explorer eases students' transition to verification tools by incorporating the familiar debugging idioms of breakpoints and several varieties of stepwise navigation (e.g., *step over*,*step into*, *step until return*). This viewer framework also exposes hooks which allow custom algorithm implementors to easily annotate the model display to visualize the status of computational machinery used to enforce system properties.

Bogor also provides a graphical view for BIR states as object diagrams by using the Eclipse Graphical Editing Framework (GEF).[4] These features are particularly useful for exploring the state-space of models and also to understand counter-examples. A short error trace from a simple transition system in Figure 1 is analyzed with the Bogor UI in Figure 2; the readers are encouraged to download Bogor and try it on more complex examples.

[4] `http://www.eclipse.org/gef`

4 Experiences

In this section, we compare our experiences in teaching a graduate-level model checking course at Kansas State University over the past several years using the SPIN model checker at first, and then using the educational materials described in the previous section for the last two semesters.

When we taught the course using SPIN, we taught students how to model small concurrent systems with Promela and had them use SPIN to check their systems. These illustrated the usefulness of model checking to find subtle concurrent bugs and to check safety and liveness properties. We then taught them SPIN's reduction algorithms and had them experiment and observed the effects of the algorithms to the search state space. However, we often found the students did not understand how the algorithms worked and what their effects meant. In order to address this, we believed that we had to teach the students how the algorithms were implemented and have them experiment and modified them. However, the tangled-ness of SPIN's implementation hindered us in fitting this approach in a semester-long course.

When we shifted to use the course materials described in the previous section, we could emphasize on teaching how the algorithms were actually implemented and the students could see how they work in action. We then taught them how to extend the algorithms in Bogor, and had them work on several small to medium-size projects. These projects helped reinforce their knowledge of model checking algorithms. Some of the project work that they have done were the basis of some research papers [19, 20], and some became the topics of their master theses (ongoing).

Based on the students feedbacks, there were several features in the tool support mentioned earlier

that helped their study: (1) BIR syntax highlighting and meaningful information from the well-formedness checker integrated in Eclipse helped them when modeling systems in BIR, (2) Bogor's user-simulation mode and counter-example display (both tree-based and graphical-based views) helped them to understand models (by exploring their state-space) and counter-examples easier, (3) the modularity of Bogor helped them to focus and localized their efforts when customizing it, and (4) the fact that Bogor is written in Java enabled them to use Eclipse's Java facilities to develop their Bogor extensions.

5 Conclusions and Future Work

We believe that our experiences suggest that the course materials that we are developing coupled with a tool support help the process of learning model checking and how it can be applied in software. The educational materials described in this paper have been used to teach graduate-level model checking courses at Kansas State University (USA), University of Nebraska-Lincoln (USA), Brigham Young University (USA), Queen's University (Canada), and Georg-August-Universität Göttingen-Institut für Informatik (Germany).

Ongoing work includes polishing the course materials and improving Bogor UI such as providing additional views for counter-examples (e.g., a thread sequence diagram), as well as other pedagogical views such as state-space view. Future work includes integration with other verification tools such as the next generation Bandera tool-set[5] and JMLEclipse[6].

About the Authors

Matthew B. Dwyer is a Henson Professor of Engineering at UNL-Lincoln, John Hatcliff is an Associate Professor at KSU, Robby is an Assistant Professor at KSU, and Matthew Hoosier is a research associate at KSU. They are working in the areas of model-checking, static analysis, and other forms of software verification.

References

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

[2] C. T. Karamanolis, D. Giannakopolou, J. Magee, and S. M. Wheather, "Model checking of workflow schemas," in *4th International Enterprise Distributed Object Computing Conference*, pp. 170–181, Sept. 2000.

[3] W. Chan, R. J. Anderson, P. Beame, D. Notkin, D. H. Jones, and W. E. Warner, "Optimizing symbolic model checking for statecharts," *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp. 170–190, 2001.

[4] D. Garlan, S. Khersonsky, and J. S. Kim, "Model checking publish-subscribe systems," in *Proceedings of the 10th International SPIN Workshop on Model Checking of Software*, pp. 166–180, May 2003.

[5] `http://bandera.projects.cis.ksu.edu`

[6] `http://jmleclipse.projects.cis.ksu.edu`

[5] 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*, pp. 160–173, 2003.

[6] 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.

[7] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, Robby, and H. Zheng, "Bandera : Extracting finite-state models from Java source code," in *Proceedings of the 22nd International Conference on Software Engineering*, pp. 439–448, June 2000.

[8] C. Demartini, R. Iosif, and R. Sisto, "dSPIN : A dynamic extension of SPIN," in *Theoretical and Applied Aspects of SPIN Model Checking (LNCS 1680)*, Sept. 1999.

[9] 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*, pp. 267–276, 2003.

[10] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–294, May 1997.

[11] R. Iosif, "Symmetry reduction criteria for software model checking," in *Proceedings of Ninth International SPIN Workshop*, vol. 2318 of *Lecture Notes in Computer Science*, pp. 22–41, Springer-Verlag, Apr. 2002.

[12] D. Bosnacki, D. Dams, and L. Holenderski, "Symmetric SPIN," *International Journal on Software Tools for Technology Transfer*, vol. 4, no. 1, pp. 92–106, 2002.

[13] 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*, pp. 154–181, 2002.

[14] 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*, 2003.

[15] 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*, vol. 2988 of *Lecture Notes in Computer Science*, pp. 404–420, Mar. 2004.

[16] 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*, 2004. (to appear).

[17] A. W. Appel, *Modern Compiler Implementation in Java*. Cambridge University Press, 1997.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley Pub. Co., 1995.

[19] E. Rodríguez, M. B. Dwyer, J. Hatcliff, and Robby, "A flexible framework for the estimation of coverage metrics in explicit state software model checking," in *Proceedings of the 2004 International Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart Devices*, 2004. (to appear).

[20] M. Hoosier, J. Hatcliff, Robby, and M. B. Dwyer, "A case study in domain-customized model checking for real-time component software," in *Proceedings of the 1st International Symposium on Leveraging Applications of Formal Method*, 2004. (to appear).