

# Software Model Checking using Bogor

## — a Modular and Extensible Model Checking Framework

(<http://bogor.projects.cis.ksu.edu>)  
(<http://www.cis.ksu.edu/~hatcliff/ESSCaSS04>)

John Hatcliff  
ESSCaSS 2004

### 1 Overview of Lectures

State-space exploration strategies such as model checking are emerging as a popular technology for reasoning about behavioral properties of a wide variety of software artifacts including: requirements models, architectural descriptions, designs, implementations, and process models. The computational costs of model checking are well-known, and a decade of intensive research on general techniques for reducing the complexity of model checking has made model checking tools much more efficient. Yet, past experience has shown that domain-specific information can often be leveraged to obtain state-space reductions that go beyond general purpose reductions by customizing existing model checker implementations or by building new model-checking engines dedicated to a particular domain. Unfortunately, these strategies limit the dissemination of model checking techniques across a number of domains since it is often infeasible for domain experts to build their own dedicated model checkers or to modify existing model checking engines.

To enable researchers to more easily tailor a model checking engine to a particular software-related domain, we have constructed an extensible and modular explicit-state software model checking framework called Bogor. For treating realistic designs and implementations in widely-used languages such as Java and C#, 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 built-in features, Bogor employs state-of-the-art reduction techniques such as collapse compression, heap symmetry, thread symmetry, and partial-order reductions that leverage common heap and locking structures. For tailoring to specific domains, Bogor provides (a) mechanisms for extending Bogor's modeling language with new primitive types, commands, expressions, and APIs associated with a particular application, and (b) a well-organized module facility for plugging customized domain-tailored components into the model-checking engine. Moreover, Bogor is designed to be easily encapsulated within larger domain-specific development and verification environments.

The goals of this series of lectures are to provide students with an overview of the foundations of explicit-state model checking as implemented by Bogor, and then to illustrate how Bogor can be customized to a particular domain.

#### Lecture I: Bogor Overview and Foundations of Explicit-State Model-checking

This lecture begins with an overview of the motivation for Bogor and gives a short demonstration of Bogor's input language and user interface. This is followed by an introduction to the basic concepts of explicit-state model-checking. Emphasis is placed on the primary data structures involved in the model-checker's depth-first search, as these are the data structures that one deals with when extending Bogor's input language or customizing internal modules of Bogor.

#### Lecture II: Writing Bogor Extensions—A Tutorial

A simple set example is used to illustrate how Bogor's input language can be extended with new types, new expressions, and new commands. Also illustrated are the application programming interfaces (APIs) for the primary data structures of Bogor's state-space exploration engine.

#### Lecture III: Checking Java Programs and JML Specifications

This lecture will describe how Bogor supports model-checking of Java programs and rich specifications written in the Java Modeling Language (JML). This gives insight into how the sophisticated features of a high-level language such as methods, inheritance, exceptions, etc. can be supported in Bogor, and how Bogor's reduction strategies (e.g., partial-order reductions) are tailored to features specific of the state of object-oriented software execution such as heap and locking structures. If time permits, this lecture will also briefly survey the Bandera Temporal Specification Patterns system that a number of researchers and practitioners have found useful for constructing temporal logic specifications.

#### Lecture IV: Designing Component-based Systems in Cadena and Checking Cadena Designs in Bogor

This lecture gives an overview of Cadena—an integrated development environment that we have built for the design, analysis, and implementation of systems built using the CORBA Component Model. Cadena is currently being used by research engineers at several companies including Boeing and Lockheed-Martin to demonstrate the effectiveness of model-driven component-based product-line development for avionics and command-and-control systems. Cadena uses Bogor to model-check high-level system designs, and this lecture describes how Bogor's input language and underlying state-space exploration algorithms were customized to directly support the CORBA real-time event channel middleware infrastructure used in mission- and command-control applications.

At <http://www.cis.ksu.edu/~hatcliff/ESSCaSS04> students can find materials specifically associated with these lectures including

- the distribution of Bogor which includes user manual, tutorials, Bogor source code, and API documentation, and
- electronic versions of lecture slides, supplementary lecture notes, guided exercises, and associated research papers for background reading.

These lectures report on work carried out jointly with Dr. Matthew Dwyer, Dr. Robby, and other SAnToS group members.



## 2 Overview of SAnToS Tools

Researchers at Laboratory for Static Analysis and Transformation of Software (SAnToS Laboratory) at Kansas State University have created several tools for design, analysis, and verification of software. Below we give a brief overview of these tools.

### 2.1 Bandera—A Tool Set for Verification of Java Programs

The Bandera Tool Set is an integrated collection of program analysis, transformation, and visualization components designed to facilitate experimentation with model-checking Java source code. Bandera takes as input Java source code and a software requirement formalized in Bandera's temporal specification language, and it generates a program model and specification in the input language of one of several existing model-checking tools (including Spin, dSpin, SMV, and JPF). Both program slicing and user extensible abstract interpretation components are applied to customize the program model to the property being checked. When a model-checker produces an error trail, Bandera renders the error trail at the source code level and allows the user to step through the code along the path of the trail while displaying values of variables and internal states of Java lock objects.

For the next generation of Bandera (planned for release at the end of 2004), the entire Bandera code-base has been completely redesigned and rewritten and implemented in IBM's Eclipse open-source IDE. Moreover, the back-end translations to Spin, dSpin, etc. have been replaced by a dedicated translation into Bogor. These changes have greatly increased the robustness, scalability, and usability of Bandera.

**Project URL:** <http://bandera.projects.cis.ksu.edu>

### 2.2 Bogor—An Extensible and Modular Software Model Checker

Bogor is a highly customizable and modular model checking framework aimed at easing the development of robust and efficient domain-specific model checkers for verification of dynamic and concurrent software. It provides a rich and extensible modeling language including features that allow for dynamic creation of objects and threads, garbage collection, dynamic dispatch of methods, and exception handling.

The extensible modeling language allows user-defined abstract data types and abstract operations as first class constructs. This is particularly useful when customizing Bogor to a particular family of software artifacts. Furthermore, its open modular design eases the task of customization to accommodate, for example, variations in scheduling policies, search modes for state exploration, state encodings, and checkers for specification languages.

Bogor employs state-of-the-art reduction techniques such as collapse compression, heap symmetry, thread symmetry, and partial-order reductions. Bogor has been successfully customized for efficient verification of realistic Java programs in the Bandera project and real-time avionic systems in the Cadena project.

We believe that Bogor can be especially useful to researchers interested in experimenting with new modeling languages features and new model checking algorithms.

**Project URL:** <http://bogor.projects.cis.ksu.edu>

### 2.3 Cadena—A Development Environment for Design, Analysis, of Verification of Component-based Systems

The use of component models such as Enterprise Java Beans and the CORBA Component Model (CCM) in application development is expanding rapidly. Even in real-time safety/mission-critical domains, component-based development is beginning to take hold as a mechanism for incorporating non-functional aspects such as real-time, quality-of-service, and distribution.

To form an effective basis for the development of such systems, we have built Cadena—an integrated environment for building and modeling CCM systems. Cadena provides the following capabilities:

- A collection of light-weight specification forms that can be attached to CCM's component Interface Definition Language (IDL) to specify mode variable domains, intra-component dependencies, and component state-transition semantics. These forms have a natural refinement order so that useful feedback can be obtained with little annotation effort, and increasing the precision of annotation yields more precise analysis. In addition, Cadena specifications allow developers to specify the same information in different ways, achieving a form of checkable redundancy that is useful for exposing design flaws.
- Dependency analysis capabilities allow tracing inter/intra-component event and data dependencies, as well as algorithms for synthesizing dependency-based real-time and distribution aspect information.
- A component assembly framework supporting a variety of visualization and programming tools for developing component connections.
- Integration with both C++ and Java CCM implementations including CIAO (a C++ implementation built on top of the ACE/TAO real-time CORBA implementation) and OpenCCM (a Java implementation that runs on top of a number of open source Java ORBs)
- A novel model-checking infrastructure dedicated to event-based inter-component communication via real-time middleware enables system design models (derived from CCM IDL, component-assembly descriptions and annotations) to be model-checked for global system properties.
- Java component stub and skeleton code generated using the CCM IDL compilers of OpenCCM or CIAO.

The Cadena tools are implemented as plug-ins to IBM's Eclipse IDE. This provides an end-to-end integrated development environment for CCM-based Java systems moving from editing of component definitions and connections information to editing and debugging of auto-generated code templates.

Cadena is currently being used by research engineers at several companies including Boeing and Lockheed-Martin to demonstrate the effectiveness of model-driven component-based product-line development for avionics and command-and-control systems. We are actively collaborating with researchers at the University of Vanderbilt (developers of CIAO) to more effectively support model-driven development of distributed real-time embedded systems.

**Project URL:** <http://cadena.projects.cis.ksu.edu>

### 2.4 Indus—A Toolkit to Customize and Adapt Java Programs.

Indus is a toolkit of program analyses and transformations targeted towards customization and adaptation of Java programs.

In terms of software, Indus provides a library with the following modules/features (with more coming) that the users can combine to realize customized analyses, transformations, and tools.

- A generic and primitive data flow analysis framework targeted towards inter-procedural analyses.
- An object-flow/points-to analysis that built from the above framework and that is used to construct precise call-graphs and thread-graphs.

- An assortment of program analyses.
  - monitor analysis,
  - escape analysis,
  - safe lock analysis, and
  - dependence analyses - control, sequential intra/inter-procedural data, divergence, synchronization, interference, and ready.
- A customizable program slicer for Java.
- A framework to glue various modules of the library to realize dedicated implementations.
- A collection of useful data structures and algorithms that are often required during program analysis/transformation.

In terms of tools, Indus provides a customizable implementation of the program slicer that generates executable and non-executable variants of backward, forward, and complete slices. Upcoming versions of Indus will include a partial evaluation/specialization engine for Java.

Kaveri, is a subproject of Indus that delivers the above slicer as a plugin in Eclipse. At present, the user can slice Java projects and view the slice in a Java editor in Eclipse. Future versions of the plugin will provide intuitive ways to understand the slice by chasing dependence and querying the slice.

**Project URL:** <http://indus.projects.cis.ksu.edu>

### 3 Overview of SANToS Educational Material

#### 3.1 Software Specifications

This course emphasizes tool-based formal specification methods and their role in the design and implementation of software systems. We aim to cover methods that span the development process ranging from high-level semantic modeling (Alloy), to system architecture design (USE), to coding and debugging (JML and ESC/Java).

Alloy (developed by Daniel Jackson and his students at MIT), in particular, "brings specifications to life" by allowing designers to query constructed models via its constraint analyzer and to easily assess the impact of model refinements. In addition, Alloy has a readable graphical and textual notation and it has a very small and semantically clean core language. We believe that all these factors make it an excellent pedagogical vehicle and a tool that can be applied to realistic problems with a fairly large return on the analyst's effort. (Note: our materials use an earlier version of Alloy (Alpha 1.1) rather the most recent version, because when we started this version of the course in January 2002, the documentation for the older version was more complete—in January 2005 the course will transition to the latest version of Alloy).

We cover aspects of UML due to its current popularity. The USE tool (developed by Martin Gogolla and Mark Richters from the University of Bremen) is very robust and provides some very useful capabilities for reasoning about OCL enriched UML specifications. We believe that students may well use this tool in their design/development work in the future.

We include a module on ESC/Java (developed by researchers at Compac/SRC) because we want to cover discussion of code-level specifications. The fact that ESC/Java provides some support for automatically checking these specs is in line with our preference for tool-supported methods. In Spring 2005, the course will also include checking specifications written in the Java Modeling Language (JML) (developed by Gary Leavens and colleagues) using Bogor.

The course distribution for instructors includes a variety of pedagogical materials such as PowerPoint lecture slides, streaming video for our lectures, source code for lecture examples, weekly quizzes and solutions, homeworks and solutions, exams and solutions. A separate distribution for students includes only the lecture slides and examples.

**Course URL:** <http://software-specs.courses.projects.cis.ksu.edu>

#### 3.2 Foundations and Applications of Software Model Checking

Modern computing applications increasingly require concurrent/distributed software systems that are extremely reliable. Unfortunately, current software validation techniques, such as inspections and testing, are failing to provide high levels of assurance of correctness for these systems due to system size and complexity as well as the fundamental difficulties of reasoning about state/event sequences in concurrent behavior.

Model-checking techniques (now widely used for hardware verification) hold promise for establishing crucial behavioral properties of complex software because they can automatically check to see if an abstract finite-state transition system model of the software conforms to a given state/event sequence property. If the model fails to satisfy the property, the model-checker gives a counterexample—a path through the model's transitions that violates the property. This can be used to locate and correct the corresponding software defect.

This course emphasizes a practical and project-oriented approach to learning the technical foundations of model checking and methodologies for applying model-checking tools to realistic systems. Foundational topics covered include basic explicit-state reachability algorithms, temporal specification formalisms including LTL and CTL, partial-order reductions, state-space representations (collapse compression, etc.), and alternate search strategies. In an approach similar to that used in compiler courses, these foundational and theoretical concepts are reinforced by having students implement key components of an explicit state model-checker.

The Bogor model checker developed at Kansas State University plays a central role in the course. 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.

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.

The course distribution for instructors includes a variety of pedagogical materials such as typeset lecture notes and guided exercises, PowerPoint lecture slides, streaming video for our lectures, source code for lecture examples, weekly quizzes and solutions, homeworks and solutions, exams and solutions. A separate distribution for students includes only the lecture slides and examples.

**Course URL:** <http://model-checking.courses.projects.cis.ksu.edu>