Abstract  The use of assertions to express correctness properties of programs is growing in practice. Assertions provide a form of lightweight checkable specification that can be very effective in finding defects in programs and in guiding developers to the cause of a problem. A wide variety of assertion languages and associated validation techniques have been developed, but run-time monitoring is commonly thought to be the only practical solution. In this paper, we describe how specifications written in the Java Modeling Language (JML), a general purpose behavioral specification and assertional language for Java, can be validated using a customized model checker built on top of the Bogor model checking framework. Our experience illustrates the need for customized state-space representations and reduction strategies in model checking frameworks in order to effectively check the kind of strong behavioral specifications that can be written in JML. We discuss the advantages and trade-offs of model checking relative to other specification validation techniques and present data that suggest that the cost of model checking strong specifications is practical for several real programs.

1 Introduction

The idea of interspersing specifications of the intended behavior of a program directly in the source code is nearly as old as programming itself [1]. Those foundational ideas inspired the development of more elaborate design practices and methodologies, for example, design-by-contract [2]. The use of assertional specifications has long been regarded as a means for improving software quality, but only recently have studies demonstrated support for this conjecture [3]. The increasing number of modern languages (e.g., Java, C#, PHP) and implementation frameworks (e.g., Microsoft Foundation Classes (MFC)) that include simple assertion mechanisms suggests that they will finally have the practical impact that was predicted decades ago.

To fulfill this promise, there is a need for program assertion checking mechanisms that are cost-effective, automatic, and thorough in considering both specification and program behavior. Run-time monitoring of assertions during program execution is the only mechanism that is widely used in practice today. It is both cost-effective and automatic, but can only analyze the program behaviors that are actually executed. This lack of coverage of program behavior is a significant weakness of run-time methods, especially for concurrent programs where subtle errors may depend on the order in which threads execute. To address this behavior coverage problem, a variety of static analysis approaches have been proposed to thoroughly check a program’s possible behaviors with respect to certain lightweight specifications, such as pointer non-nullness and array bounds [4], and propositional temporal properties [5]. These methods gain program coverage by sacrificing the expressiveness of their specification language.

Building on a long line of work on formal methods for manual reasoning about complete behavioral specifications of programs, several recent languages have emerged that balance the desire for completeness and the pragmatics of checkability. The Java Modeling Language (JML) is one such language [6]. With JML, one can specify properties of varying strength from lightweight assertions about
Checking JML specifications using an extensible software model checking framework

In the next section, we give an overview of JML and illustrate its main features through an example. Section 3 describes the features of the Bogor model checking framework that enable the efficient treatment of JML specifications. There are a number of different techniques for reasoning about JML specifications; we compare representatives of the main classes of techniques to our model checking approach in Sect. 6. Section 4 details our strategy for efficiently reasoning about JML specifications on-the-fly during state-space exploration of a concurrent Java program. In Sect. 5, we detail the analysis of a collection of JML annotated Java programs and report on the cost and effectiveness of checking them with Bogor and then conclude. We also refer the reader to the SpEx web site [11] for the complete JML-annotated Java code and the associated Bogor models for all of the examples considered in this paper.

2 JML: the Java modeling language

JML [6] is a Java-specific behavioral specification language [12] designed at Iowa State University by Gary Leavens and others. We illustrate JML and the treatment of JML specifications with the example in Fig. 1. This example is a concurrent linked-list-based queue from [13] with some JML specifications, written in Java comments with special tags such as //@, added to describe its behavior.

Instances of the class LinkedNode implement the nodes of the linked list representing the queue. The LinkedQueue class provides put and take methods that implement a fine-grained locking protocol, through the use of the protected methods insert and extract, to maximize concurrent access to the queue. This design leads to functional code that is nested inside synchronized statements and conditionals in those protected methods. In order to specify the behavior of that functional code we have refactored them into additional protected methods, e.g., refactoredInsert.

When a new queue is created, an object that is used to guarantee mutual exclusion of put operations is created and assigned to the putLock field and a new node is created and assigned to the head and tail instance fields (this dummy node, with an unused data field, forms the head of every list). Whenever a thread attempts to take an object from an empty queue, the thread is blocked. If the queue is not empty, then only the head is locked, and its stored value is returned. The dequeuing is done in the extract method. Whenever an object is enqueued, the tail is locked, a new node is created to store the object and one of the threads waiting to dequeue is notified.

JML specifications are phrased as invariants on instances of classes and contracts for method invocations. One important aspect of JML is that it balances support for complete behavioral specification with lightweight assertion features, such as Java assertions. Thus, it allows developers to vary the strength of their specifications across classes and within classes. The variation in the strength of specifications in the

pointer null-ness to complete functional correctness of program components; the latter we refer to as a strong property. JML is a behavioral interface specification language that allows developers to specify both the syntactic and behavioral interface of a portion of Java code. It supports the design-by-contract [2] paradigm by including notation for pre/postconditions and invariants. JML uses Java’s expression syntax and adds constructs that dramatically increase expressiveness (e.g., it is possible to quantify, universally or existentially, over objects in the heap).

In this paper, we describe how we have adapted a flexible model checking framework called Bogor [7] to check JML specifications of sequential and concurrent Java programs. Model checking adds a new and complementary approach to the existing run-time and theorem-proving technologies for reasoning about JML. While tools based on those technologies have proven effective in supporting certain kinds of Java validation and verification activities, there is currently no automatic technique for thoroughly checking a wide-range of strong JML specifications especially in the presence of concurrency. Our checking tool is automatic and exhaustive in its reasoning about general JML properties up to user defined bounds on the space consumed by a program run.

Using existing model checking techniques to verify strong specifications is problematic for several reasons. First, existing model checkers, such as Spin [8], do not provide direct support for modeling dynamically allocated objects and heap structures making it difficult to represent the program’s behavior; Bogor maintains an explicit, yet compact, representation of the dynamic program heap [9]. Second, even if one could encode the behavior in the input language of such a model checker, the underlying checking algorithms would not exploit the semantic properties of the original language to optimize the state-space search; Bogor incorporates novel partial-order reductions (POR) that exploit the semantics of a program’s heap and locking structure to achieve efficiency [10]. Finally, existing model checking frameworks support temporal properties but do not provide direct support for expressing rich data or heap-related functional properties; Bogor supports extension of the expressions sublanguage via user defined atomic expressions that can be evaluated over the full extent of a program state including the heap [7].

The contributions of this paper are as follows:

- we demonstrate that with a sufficiently feature-rich model checking framework one can check strong behavioral specifications;
- we describe how Bogor’s extension facilities can be applied to implement checking of JML specifications, including specifications that have proven difficult to check by other means such as run-time checking or theorem-proving; and
- we demonstrate that the overhead of checking JML specifications can be mitigated, and in most cases completely eliminated, through the use of sophisticated state-space reductions.