

Sireum/Kiasan for Java

*A JML Contract-based Automatic Verification and
Test Case Generation Tool-set for Java Program Units*

Sireum

|| \top \approx \times Σ \diamond \vdash \curvearrowright

a software analysis platform

A User/Developer/Researcher Manual
(DRAFT)

SAnToS Laboratory, Kansas State University
Pennsylvania State University - Harrisburg

The development of Kiasan is supported in part by the US National Science Foundation (NSF) awards [CNS-0709169](#), [CNS-0734204](#), and CAREER award [CCF-0644288](#). The development of the Sireum software analysis was supported in part by the US Air Force Office of Scientific Research, Lockheed Martin Advanced Technology Laboratories, and Rockwell Collins Advanced Technology Center.

COLLABORATORS

	<i>TITLE :</i> Sireum/Kiasan for Java		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		October 4, 2008, 12:00pm	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Introduction	1
1.1	About	1
1.2	Licensing	1
1.3	Feature Walkthrough	1
1.4	Architecture Overview	5
1.4.1	JML ^K	5
1.4.2	KiasanVM	7
1.4.3	Semanggi	7
1.5	Limitations and Known Issues	7
1.5.1	JML ^K	7
1.5.2	KiasanVM	8
1.5.3	Semanggi	8
1.6	Roadmap	8
1.6.1	JML ^K	9
1.6.2	KiasanVM	9
1.6.3	Semanggi	9
1.6.4	Other Features	9
1.6.4.1	KUnit	9
1.6.4.2	Distributed Kiasan	9
1.7	Team and Contact Information	9
1.8	Acknowledgement	10
2	Getting Started	12
2.1	System Requirements	12
2.1.1	Mac OS X	12
2.1.2	Linux	12
2.1.3	Windows	12
2.2	Downloading	12
2.3	Setting Up	12
2.3.1	Mac OS X	13

2.3.2	Linux	13
2.3.3	Windows	13
2.4	Running Kiasan	13
2.4.1	Using Command Line Interface	13
2.4.2	Using Ant	15
2.4.3	Running Kiasan Tools	16
2.4.3.1	Method Information Miner (MIM)	16
2.4.3.2	Extension Stub Generator (ESG)	18
2.4.4	Running KiasanVM Directly	18
3	Code Substitution and Modeling	19
3.1	Redirection	19
3.2	Extension	19
3.3	Reflection	19
4	Java Library Support	20
4.1	Package: <code>java.lang</code>	20
4.2	Package: <code>java.util</code>	20
5	JML Support	21
5.1	Type Specifications	21
5.2	Method Specifications	21
5.3	Primary Expressions	21
6	Case-Optimality of Kiasan's Algorithm	22
6.1	Binary Search Tree	22
6.1.1	Method <code>findMax</code> and <code>findMin</code>	22
6.1.2	Method <code>find</code> , <code>insert</code> , and <code>remove</code>	22
6.2	Red-Black Tree	22
6.2.1	Method <code>lastKey</code>	22
6.2.2	Method <code>get</code> , <code>put</code> , and <code>remove</code>	23
6.3	AVL Tree	23
6.3.1	Method <code>findMax</code> and <code>findMin</code>	23
6.3.2	Method <code>find</code> and <code>insert</code>	23
6.4	AA Tree	23
6.4.1	Method <code>findMax</code> and <code>findMin</code>	23
6.4.2	Method <code>insert</code> and <code>remove</code>	23
6.5	...	23
7	References	24

List of Figures

1.1	Container.swap	2
1.2	Kiasan's Analysis Report for Container.swap	2
1.3	Container.swap2	3
1.4	Container.swap3	3
1.5	Kiasan's Analysis Report for Container.swap3	4
1.6	Sireum/Kiasan Architecture	5
1.7	KiasanVM contract class for Container.swap3	6
2.1	A Kiasan Ant Task Example	15

List of Tables

1.1	Pre-/post-states case #0 of Container.swap3	4
2.1	Kiasan Ant Task Attributes	15
4.1	Supported classes and methods in <code>java.lang</code> package	20
4.2	Supported classes and methods in <code>java.util</code> package	20
5.1	Supported JML type specifications	21
5.2	Supported JML method specifications	21
5.3	Supported JML primary expressions	21

Chapter 1

Introduction

1.1 About

Sireum/Kiasan for Java¹ JML contract-based automatic verification and test case generation tool-set for Java program units. In contrast to regular unit testing methods, Kiasan does not need input parameters for checking units. Without assertions or contracts (e.g., pre-/post-conditions), Kiasan, by default, detects possible uncaught runtime exceptions such as `NullPointerException`, `ArithmeticException`, `ArrayIndexOutOfBoundsException`, etc. With assertions embedded in the code, Kiasan automatically determines whether the assertions are possibly violated. When [Java Modeling Language \(JML\)](#) contracts (e.g., object/class invariants, method pre/postconditions) are supplied, Kiasan can leverage the contracts to filter out input parameters (i.e., pre-states) that do not satisfy the contracts while ensuring the states after execution (i.e., post-states) satisfy the contracts. In contrast to other bug finding tools, Kiasan generates a low number of false alarms due to its formal semantic-based analysis engine, and it has a stronger and quantifiable coverage on the unit behavior that it analyzes. Furthermore, Kiasan generates helpful analysis feedback by visualizing program states (e.g., object heap graphs) and JUnit test cases useful for illustrating property violations (e.g., assertion failures) as well as for code understanding or inspection (e.g., by visualizing code effects on program states).

1.2 Licensing

Sireum/Kiasan is distributed under the [Eclipse Public License v 1.0](#).

1.3 Feature Walkthrough

Sireum/Kiasan takes as inputs a unit source code and its corresponding bytecode and analyzes whether the code might throw (uncaught) exceptions and violate assertions under certain conditions. Regardless of whether errors/exceptions are present, Kiasan generates a HTML report for the unit analysis detailing bytecode instruction and branch (MCDC) coverage, as well as object graphs illustrating possible pre-/post-states of the unit.

¹Sireum means "ants" in Javanese (i.e., the real Java language); in this context, it is a software analysis platform consisting of a collection of software analysis frameworks for building customized static analyzers for particular application domains and quality-assurance properties. Kiasan means "to reason with analogy/symbolically" in Indonesian (i.e., the Java island is one of the five biggest islands in Indonesia), which reflects the characteristics of its main algorithm.

```

1 class Container<E> {
2   E data;
3   void swap(Container<E> other) {
4     E temp = data;
5     data = other.data;
6     other.data = temp;
7   }
8   ...
9 }

```

Figure 1.1: Container.swap

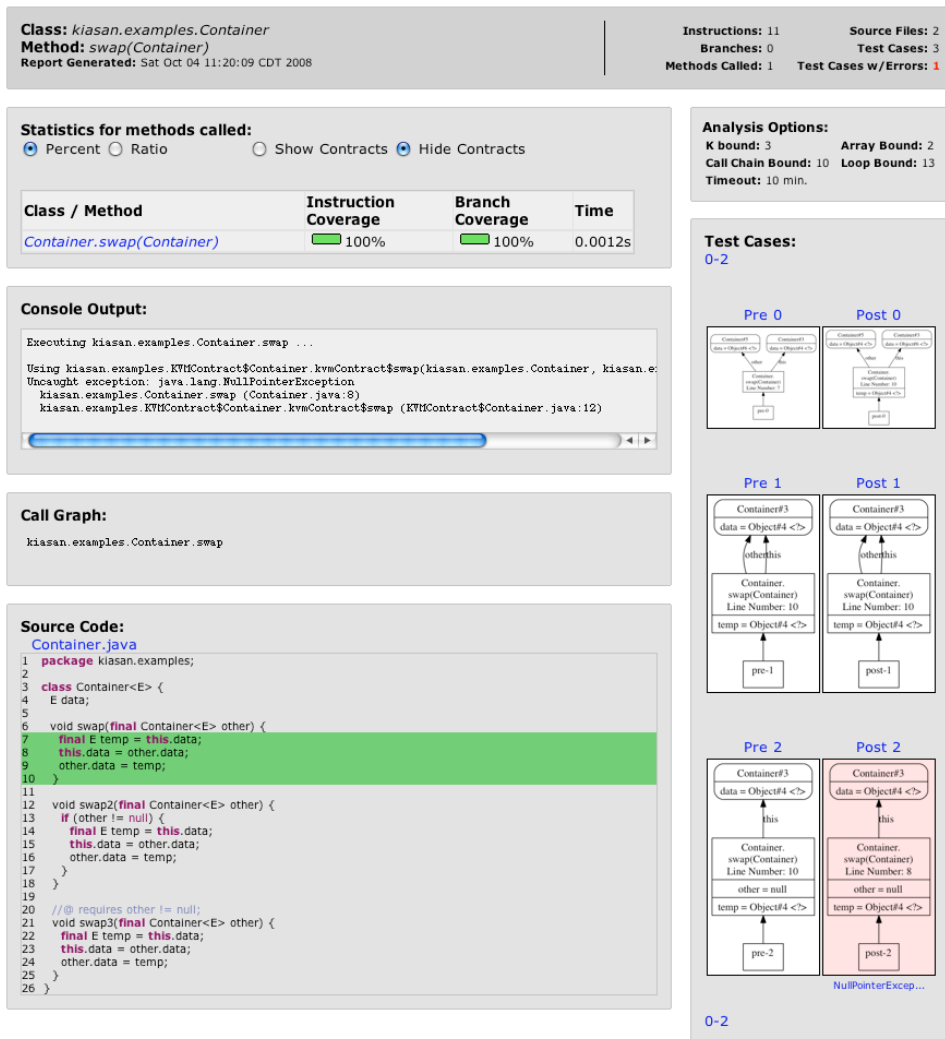


Figure 1.2: Kiasan's Analysis Report for Container.swap

For example, consider the `swap` method in Figure 1.1. After analyzing `swap`, Kiasan generates the report shown in Figure 1.2. The report page contains information about instruction and branch coverage of the unit that Kiasan analyze along with possible state/object graphs before (pre) and after (post) executing the unit. The source code of the unit is included and instruction and branch coverage information are illustrated as colored background in the code: green means fully covered, light green means partially covered, and default background color means uncovered (note: subsequent releases of Kiasan would feature a

bytecode-level view of the code). For `swap`, Kiasan determines that it is possible for `swap` to throw (uncaught) `NullPointerException` when executing the second statement at line 5 (illustrated by the pre-/post-states #2).

To guard against this, one can enclose all the statements inside a branch as illustrated in Figure 1.3.

```
class Container<E> {
  E data;
  ...
  void swap2(Container<E> other) {
    if (other != null) {
      E temp = data;
      data = other.data;
      other.data = temp;
    }
  }
  ...
}
```

Figure 1.3: `Container.swap2`

Another alternative is to require that `other` must be non-null by using the JML specification pre-condition construct as illustrated in Figure 1.4 (note: Kiasan assumes that references may be null by default; this is in contrast to JML's default semantic, but in-line with the Java Virtual Machine semantic).

```
class Container<E> {
  E data;
  ...
  //@ requires other != null;
  void swap3(Container<E> other) {
    E temp = data;
    data = other.data;
    other.data = temp;
  }
}
```

Figure 1.4: `Container.swap3`

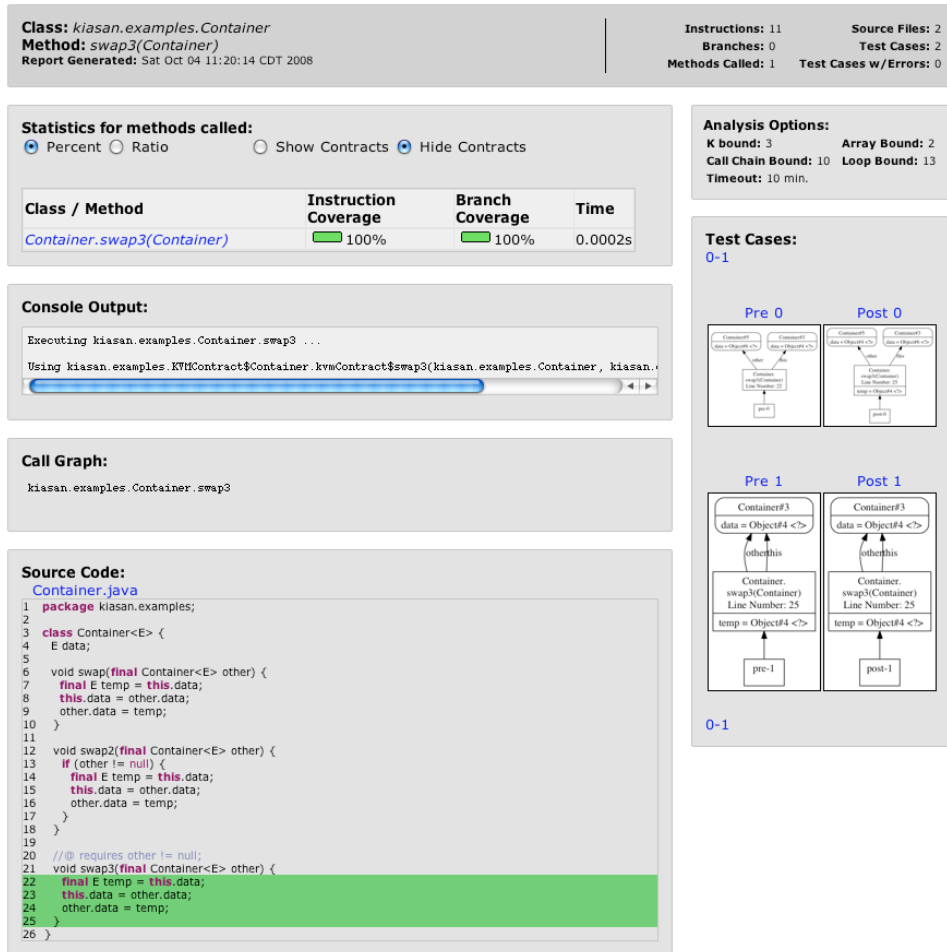


Figure 1.5: Kiasan’s Analysis Report for Container.swap3

If a unit source code is annotated with JML specifications, Kiasan will leverage them in its analysis. In this case, Kiasan assumes that `other` is non-null and only considers and generates pre-/post-states of the unit that satisfy that constraint. Under this assumption, Kiasan has verified (soundly and completely) that it is not possible for `swap3` to cause an uncaught exception/error under any possible calling context satisfying the precondition (when executed sequentially). It does so by exhaustively considering all possible calling contexts of `swap3`; that is, any calling context of `swap3` is characterized by one of the many pre-/post-states pair cases that Kiasan illustrated.

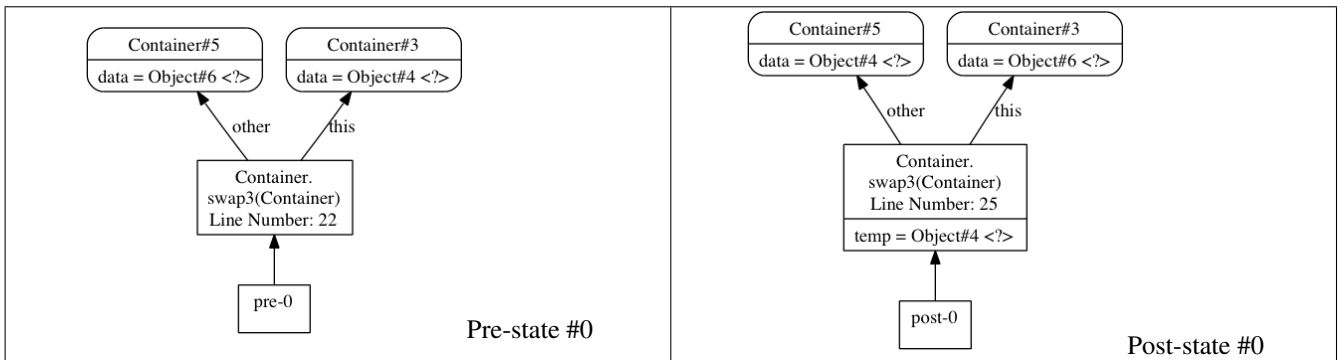


Table 1.1: Pre-/post-states case #0 of Container.swap3

Looking closely at the following pair of pre-/post-states case #0 visualized by Kiasan presented in Table 1.1. We notice that `this`

`s.data` and `other.data` points to objects of type `java.lang.Object` annotated with `<?>`. This means that `this.data` and `other.data` may refer to either any object in the heap of an actual execution that is a sub-type of `java.lang.Object`, or null (on the other hand, `<!>` annotation means non-null). Regardless of what they are, Kiasan determines that the actual values of `this.data` and `other.data` are inconsequential to the properties that it verified (in this case, the implicit property is: no uncaught exceptions).

1.4 Architecture Overview

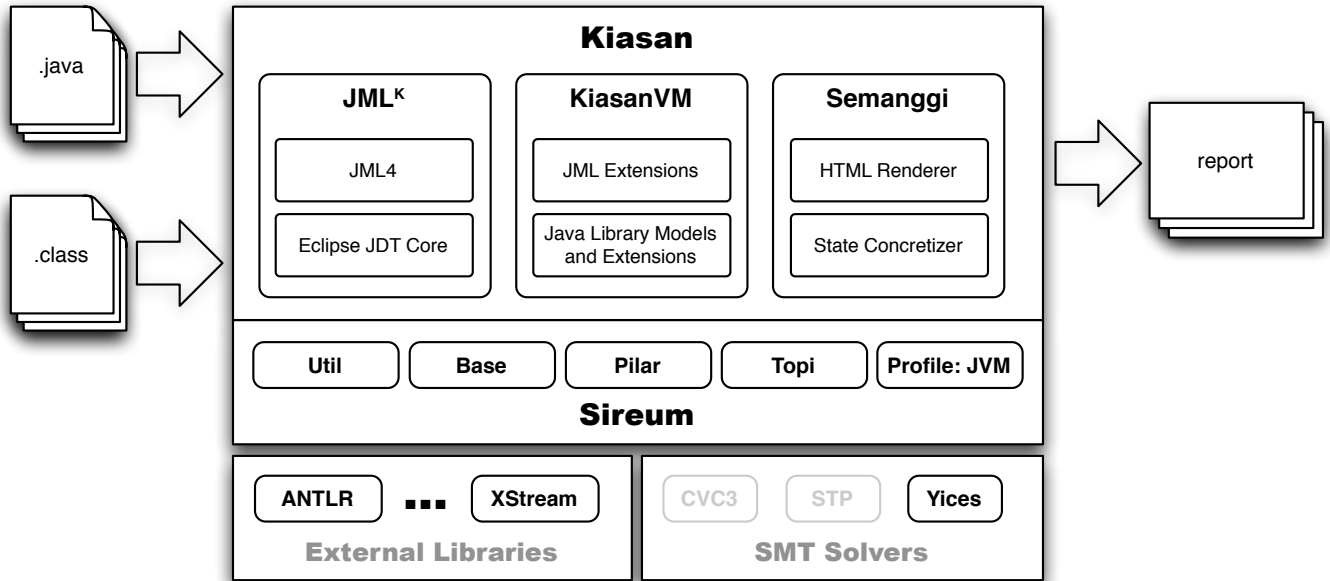


Figure 1.6: Sireum/Kiasan Architecture

Kiasan consists of three main components: (1) JML^K , (2) KiasanVM (including JML extensions and Java library models), and (3) Semanggi; the components are arranged in a pipeline architecture as illustrated in Figure 1.6. We describe each of the components in the subsequent sections.

1.4.1 JML^K

JML^K is a customized version of JML4 for Kiasan. Given a unit's Java source code and its corresponding bytecode, it translates the JML specifications in the source code to a Kiasan-amenable and executable code (Java source code) and then JML^K compiles it to Java bytecode form. Both the unit and its specification bytecode (as well as some configuration options, e.g., for contract-code substitutions) are then given to KiasanVM for analysis.

```

package kiasan.examples;

import kiasan.examples.Container;
import org.sireum.kiasan.profile.jvm.extension.Kernel;
import org.sireum.kiasan.profile.jvm.jml.contract.*;

public class KVMContract$Container {
    public static void kvmContract$swap3(final Container $receiver,
        Container other) {
        if (Kernel.getStackSize() == 1) {
            Kernel.assertTrue($receiver != null);
            final boolean $b0 = KVMContract$Container
                .kvmPre$swap3_0($receiver, other);
            Kernel.assertTrue($b0);
            KVMContract.reflect$Invoke($receiver, other);
            if ($b0) {
                Kernel.assertTrue(KVMContract$Container.kvmPost$swap3_0($receiver,
                    other));
            }
        } else {
            Kernel.assertTrue($receiver != null);
            final boolean $b0 = KVMContract$Container
                .kvmPre$swap3_0($receiver, other);
            Kernel.assertTrue($b0);
            KVMContract.reflect$Invoke($receiver, other);
            if ($b0) {
                Kernel.assertTrue(KVMContract$Container.kvmPost$swap3_0($receiver,
                    other));
            }
        }
    }

    public static boolean kvmPre$swap3_0(final Container $receiver,
        Container other) {
        return (other != null);
    }

    public static boolean kvmPost$swap3_0(final Container $receiver,
        Container other) {
        return true;
    }
}

```

Figure 1.7: KiasanVM contract class for Container.swap3

For example, JML^K translates `swap3`'s specification in Figure 1.4 into a contract class shown in Figure 1.7. Any call to `Container.swap3` will be substituted to `kvmContract$swap3`; the call to `KVMContract.reflect$Invoke` caused Kiasan to call the original `Container.swap3` code.²Therefore, in essence, Kiasan works by wrapping units by their contracts, although it does so without modifying the original code. Kiasan can do without modifying the original code because the KiasanVM symbolic virtual machine supports code substitutions; interested readers are referred to Chapter 3 for discussion on this topic.

To translate JML specifications to a KiasanVM executable form, JML^K processes the specifications by using a customized version of JML4. Once the specifications are deemed well-formed (e.g., passed type checking), JML^K computes the effective specifications (e.g., conjunct object invariants with method pre-conditions, etc.) that are then translated to Java source code. The source code is then compiled to bytecode by JML^K .³

²`KVMContract.reflect$Invoke` propagates changes to the original code's parameters up to the contract method's parameters when returning from the invocation.

³The contract classes generated by JML^K may not be compilable by a regular Java compiler.

1.4.2 KiasanVM

KiasanVM is a symbolic virtual machine for Java, and it is the main analysis engine of Sireum/Kiasan for Java. By symbolic, we mean that it can abstractly reason about unknown scalar and object values by using a sophisticated static analysis algorithm (interested readers are referred to [Deng-al:SEFM07] for a detailed discussion of the algorithm). For example, when analyzing the swap examples in Figure 1.1, Figure 1.3, and Figure 1.4, Kiasan does not require the values of `this` and `other` as inputs to its analysis; in fact, it tries to determine their possible values and determine whether they can violate safety properties in the code.

As a virtual machine for Java, KiasanVM reads in the Java bytecode of a unit and symbolically executes it. As it executes and analyzes the code, KiasanVM notifies users about *possible* safety violations and points to program locations where such violations may occur. In addition, KiasanVM measures the code instruction and branch coverage as well as other statistics such as execution times.⁴ At the end of execution, KiasanVM generates a (raw) report consisting of execution statistics as well as representative (abstract) pre-/post-states of the unit to illustrate the behaviors the analysis has covered. KiasanVM then forwards the generated report to Semanggi for processing.

1.4.3 Semanggi

Semanggi⁵ takes KiasanVM report outputs and renders an HTML report illustrating the analysis code and branch coverage and other statistics, as well as visualizing unit code effects by using object graphs as discussed earlier. Since the pre-/post-states generated by KiasanVM are abstract states, to illustrate the unit behavior in terms understandable by non-experts, Semanggi concretizes the states first before rendering the HTML report; interested readers are referred to [Deng-al:TAICPART07] for a detailed discussion of the concretization algorithm. The HTML reports generated by Semanggi are compatible with Apple Safari, Camino, Google Chrome, Mozilla Firefox, and Opera web browsers (i.e., incompatible with, e.g., Microsoft Internet Explorer).

1.5 Limitations and Known Issues

The development of Sireum/Kiasan started in late March 2008, thus, the codebase is still immature and far from its envisioned ideal form. Below we list known limitations and issues for each of Kiasan components.

Note

Although Sireum/Kiasan is open sourced, no API compatibility is guaranteed at this early stage of its development. In fact, its API will undergo a major refactoring when merging Kiasan's codebase with the second generation of the **Bogor model checking framework** built on top of Sireum. Thus, it is not recommended to build on Kiasan and customize it at this point in time.

1.5.1 JML^K

JML^K uses JML4 that supports most features of **JML Level 0**, however, JML^K includes a contract class generator which only supports a small subset of JML Level 0. Thus, JML support is still very limited in the current release. Below is a list of limitations and known issues in JML^K:

- Compositional reasoning is currently unsupported.
- Generic types are ignored when generating contract classes/methods (but, it will not affect the analysis).
- Only supports lightweight specification
- *etc.*

⁴Since KiasanVM does its analysis at the bytecode level, it computes Multiple Condition Decision Coverage (MCDC).

⁵Semanggi means clover in Indonesian, and it is also a name of a famous four-leaf clover bridge/road architecture in Jakarta, the capital of Indonesia; it is named so to pay homage to Atlassian Clover's tool as Semanggi's HTML report is heavily inspired by Clover's.

1.5.2 KiasanVM

Most of the work in KiasanVM so far has been focused on implementing the symbolic virtual machine itself. While Kiasan can interpret bytecode in the Java library, this is not always feasible or practical. For example, when a unit makes database/network connections, Kiasan does not know how to handle these methods. In practice, calls to a unit's environment are usually mocked in unit testing. KiasanVM requires a similar approach by using code substitutions (the same feature used for checking JML contracts as described above). In addition, Kiasan can scale better in some cases if abstract models are used instead of the implementations such as for the classes in the Java Collection Framework (e.g., by replacing set implementation using a red-black tree algorithm with an abstract (mathematical) model of ordered set).

In the current release, only a handful models for the Java standard library is provided (see Chapter 4) and this will be the focus for subsequent releases. In addition, as more features of JML are added in JML^K, the JML extension for KiasanVM will be enriched to handle those features. Below is a list of limitations and known issues in KiasanVM:

- Unsupported bytecode instruction: `LDC` on `Class`, and `MULTINEWARRAY`.
- Ignored instructions: `MONITORENTER`, and `MONITOREXIT`.
- Non-linear arithmetic and bit-shifting operations may cause KiasanVM to produce false alarms (also the case for Semanggi).
- KiasanVM currently uses honest-to-God, arbitrary-precision integer and real numbers (bit-imprecise), thus, it may miss bugs (unsound) and produce false alarms (incomplete) with respect to over/underflow, etc. (also the case for Semanggi). In general, the soundness and completeness of KiasanVM on scalar values depend on the properties of the underlying theorem prover being used.
- Subtyping constraints are ignored, thus, it may produce false alarms with respect to operations over object types, e.g., when using type casting, `instanceof`, object array store (also the case for Semanggi).
- *etc.*

1.5.3 Semanggi

Below is a list of limitations and known issues in Semanggi:

- Java's `assert` uses boolean static fields to determine whether assertion checking is enabled. In Kiasan, these fields' value are always true (because we always want to check assertions). Thus, Semanggi reports that branch coverage for `assert` statement is always missing the case where assertion checking is disabled.
- Semanggi HTML renderer may run slower than KiasanVM due to generation of object graphs and rendering of coverage information (and syntax highlighting).
- Semanggi coverage information includes contract coverage information. Since Kiasan assumes a unit's precondition is always true, then there are branches that are not explored at the top level contract method (i.e., when the precondition are false).
- Currently, the class-level page only displays coverage information of each of the (top-level) methods that were checked. That is, the statistics do not include coverage information from the top-level methods.
- `boolean` and `char` values are displayed as integers in object graphs (e.g., 0 for `false`, 1 for `true`).
- Semanggi uses timestamps to determine which reports should be re-rendered. In the case where KiasanVM breaks, the report output directory should be manually deleted (otherwise, Semanggi may break on subsequent runs on the same report directory).
- *etc.*

1.6 Roadmap

Below are some planned major enhancements and goals for the next releases of Kiasan (note: no specific timeline is provided for when a particular feature will be implemented).

1.6.1 JML^K

- Enhance KiasanVM contract generator and JML extensions/models to support JML Level 0.
- Add support for compositional reasoning.

1.6.2 KiasanVM

- Add models/extensions for the Java Collection Framework.
- Add models/extensions for essential classes in the `java.lang` package.
- Tighter integration with decision procedures and SMT/constraint solvers.

1.6.3 Semanggi

- Tighter integration with decision procedures and constraint solvers when concretizing states.
- Use database for storing Kiasan's analysis report.

1.6.4 Other Features

1.6.4.1 KUnit

The Sireum/Kiasan team is actively working on implementing the JUnit test case generation engine similar to Bogor/Kiasan's KUnit tool as described in [[Deng-al:TAICPART07](#)].

1.6.4.2 Distributed Kiasan

Sireum/Kiasan's algorithm can be easily parallelized/distributed over a cluster of machines or to leverage multi-core architecture. The Kiasan team is currently developing a Kiasan server implementation.

1.7 Team and Contact Information

Sireum/Kiasan for Java is developed by researchers at Kansas State University and Pennsylvania State University - Harrisburg. Below is the list of project members and their main responsibilities.

- SAnToS Laboratory, Department of Computing and Information Sciences, Kansas State University, 234 Nichols Hall, Manhattan, KS 66506, USA; 1-785-532-6350 (phone), 1-785-532-7353 (fax)
 - **Robby** (Project Leader and Main Contact Person), Faculty; Components: Sireum, KiasanVM, Topi, Semanggi's graph generator
 - **Jooyong Lee**, Postdoctoral Researcher; Components: JML^K, and KiasanVM extensions for JML
 - **Andrew King**, M.S. Student; Components: Distributed Kiasan (and the initial implementation of Semanggi's HTML report renderer)
 - **Josiah Dodds**, Undergraduate Student; Components: Semanggi's HTML report renderer
 - Other SAnToS members related to Sireum/Kiasan
 - * **John Hatcliff**, Faculty (involved in Bogor/Kiasan)
 - * **Jason Belt**, Ph.D. Student (investigating decision procedures)
- Department of Computer Science and Mathematical Sciences, Pennsylvania State University - Harrisburg, W-256 Olmsted Building, 777 Harrisburg Pike, Middletown, PA 17057, USA; 1-717-948-6081 (phone), 1-717-948-6352 (fax)
 - **Xianghua (William) Deng**, Faculty; Components: Topi, Semanggi's state concretizer, and KUnit

1.8 Acknowledgement

Sireum/Kiasan leverages the significant engineering work in JML4 -- the next generation Java Modeling Language (JML) front-end based on the Eclipse Java Development Tools (JDT), which is currently being developed by researchers at Concordia University, i.e., [Patrice Chalin](#), Perry James, and George Karabotsos; many researchers have made non-trivial contributions to the effort including [Daniel Zimmerman](#) (University of Washington at Tacoma) and Jooyong Lee.

In addition, Kiasan leverages the following open source efforts (license is specified for libraries included in the Sireum/Kiasan for Java distribution package):

- [ANT](#), "a Java-based build tool";
- [ANTLR](#), "a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages" (v2 in public domain; v3 BSD);
- [ASM](#), "an all purpose Java bytecode manipulation and analysis framework" (BSD);
- [CuTest](#), "a unit testing library for the C language";
- [dblatex](#), a DocBook to LaTeX translator;
- [Docbook](#), "a schema (available in several languages including RELAX NG, SGML and XML DTDs, and W3C XML Schema) maintained by the DocBook Technical Committee of OASIS. It is particularly well suited to books and papers about computer hardware and software (though it is by no means limited to these applications).";
- [Eclipse](#), "an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle";
- [Eclipse JDT](#), "tool plug-ins that implement a Java IDE supporting the development of any Java application, including Eclipse plug-ins" (EPL v1.0);
- [GNU GetOpt for Java](#), a Java port of GNU GetOpt command-line processor (GNU Library GPL v2);
- [GNU MP](#), "a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers" (GNU LGPL v3);
- [GNU Trove for Java](#), "high speed regular and primitive collections for Java" (GNU LGPL v2.1);
- [Graphviz](#), "an open source graph visualization software";
- [JGraphT](#), "a free Java graph library that provides mathematical graph-theory objects and algorithms" (GNU LGPL v2.1);
- [JHighlight](#), "an embeddable pure Java syntax highlighting library that supports Java, Groovy, C++, HTML, XHTML, XML and LZX languages and outputs to XHTML" (GNU LGPL v2.1);
- [JUnit](#), a unit testing framework for Java;
- [JYaml](#), "a Java library for working with the Yaml file format" (BSD);
- [LaTeX](#), "a document preparation system";
- [MinGW](#), "a collection of freely available and freely distributable Windows specific header files and import libraries, augmenting the GNU Compiler Collection, (GCC), and its associated tools, (GNU binutils)";
- [SortTable](#), "a JavaScript Library to dynamically sort HTML lists";
- [StringTemplate](#), "a java template engine (with ports for C# and Python) for generating source code, web pages, emails, or any other formatted text output" (BSD);
- [ToolTip](#), "an easy to use cross-browser Tooltip JavaScript Library that creates tooltips" (GNU LGPL v2.1);
- [XMLUnit](#), a JUnit testing framework for XML; and
- [XStream+XPP3](#), "a simple library to serialize objects to XML and back again" (BSD).

Furthermore, Kiasan benefits from the great work on SMT/constraint solvers. More specifically, Kiasan uses the following solvers as backends:

- **Cream**, a "Class Library for Constraint Programming in Java" (GNU LGPL v2.1);
- **CVC3**, "an automatic theorem prover for Satisfiability Modulo Theories (SMT) problems";
- **POOC**, "a platform for object-oriented constraint programming" (modified MIT);
- **STP**, "a constraint solver (also referred to as a decision procedure or automated prover) aimed at solving constraints generated by program analysis tools, theorem provers, automated bug finders, intelligent fuzzers and model checkers"; and
- **Yices**, "an efficient SMT solver that decides the satisfiability of arbitrary formulas containing uninterpreted function symbols with equality, linear real and integer arithmetic, scalar types, recursive datatypes, tuples, records, extensional arrays, fixed-size bit-vectors, quantifiers, and lambda expressions".

Special thanks to **John Rushby** and **Bruno Duterte** for an insightful discussion on SMT solvers and Yices.

Chapter 2

Getting Started

2.1 System Requirements

Sireum/Kiasan is implemented using the Java programming language (Java 5 or above). Ideally, it should run under all platforms supported by Java. However, Kiasan uses SMT solvers which are usually implemented using C/C++. Thus, its portability is limited by the SMT solver one chooses among all supported provers by the Sireum/Topi theorem prover and constraint solver interface.

In this version, only Yices 1.0.16 is supported. In addition, GraphViz is assumed to be installed.

Specific requirements for each computer architecture/Operating System (OS) are described below.

2.1.1 Mac OS X

Only Mac Intels with Mac OS X Leopard are supported (both 32/64-bit).

2.1.2 Linux

Only i686 hardware family is supported (both 32/64-bit). Kiasan is only tested using Ubuntu 8.04 LTS.

2.1.3 Windows

Only Windows 32-bit on x86 hardware family is supported. Kiasan is only tested using Windows XP Professional.

2.2 Downloading

Sireum/Kiasan can be downloaded at www.sireum.org.

2.3 Setting Up

The Sireum/Kiasan distribution can be installed easily by uncompressing it to a directory; let us call this directory `KIASAN_HOME`. The directory structure should contain the following directories:

- `src`, contains all the source code for Kiasan and Sireum components relevant to Kiasan
 - `bin`, contains scripts to run Kiasan and its tools
-

- `lib`, contains all the compiled code for Sireum/Kiasan and all libraries used by Kiasan
- `examples`, contains some Java examples and associated sample reports
- `licenses`, contains software licenses for Sireum and all libraries distributed with Kiasan.

Note that when moving a Kiasan installation to a different directory, one has to preserve its directory structure. Below are some specific notes for running Kiasan on different operating systems.

2.3.1 Mac OS X

Copy `KIASAN_HOME/lib/gmp/osxXX/libgmp.3.dylib` (where `XX` is either 32 or 64, depending on your architecture/OS) to `/usr/local/lib`. Similarly, copy a Yices shared library (with GMP dynamically linked) to `/usr/local/lib`.

Note

In OS X, the default Java version is Java 5 32-bit even though your system is 64-bit. If you install Yices 64-bit, then make sure you call `java` with `-d64` option or use Java 6 which only supports 64-bit; otherwise, Kiasan will throw a `java.lang.UnsatisfiedLinkError`.

2.3.2 Linux

Setup `LD_LIBRARY_PATH` to point to `KIASAN_HOME/lib/gmp/linuxXX` (where `XX` is either 32 or 64, depending on your architecture/OS) and to point to the directory containing a Yices shared library (with GMP dynamically linked).

2.3.3 Windows

The path `KIASAN_HOME` should not contain a whitespace. Setup `PATH` to point to the directory containing a Yices shared library (with GMP statically linked built using MinGW).

2.4 Running Kiasan

For analysis purposes, Sireum/Kiasan only requires Java bytecode of a unit; it works best if the code is compiled with all debugging information included (e.g., using `-g` when compiling the unit with `javac`), for example, to better illustrate error stack traces. Moreover, Semangi visualizes code instruction and branch coverage at the source code level, thus, Kiasan would need the source code for the unit for these purposes. Furthermore, JML^K currently only processes JML specifications embedded in a unit source code. Thus, the unit source code is needed for checking JML contracts. In the current release, Kiasan assumes that it can always get the unit source code along with its bytecode.

2.4.1 Using Command Line Interface

You can run Sireum/Kiasan in a command prompt (\$) as follows:

```
$ java -jar KIASAN_HOME/lib/Kiasan.jar
```

Or, you can use one of the scripts in `KIASAN_HOME/bin` appropriate for your OS (`kiasan` for Mac OS X and Linux, `kiasan.bat` for Windows). Kiasan will then output its usage information as follows:

```
Sireum/Kiasan for Java vX.Y.YYYYMMDD
http://www.sireum.org
```

```
(c) SAnToS Laboratory, Kansas State University
(c) Pennsylvania State University - Harrisburg
```

```
Parameters: [options] <class name> [<method name> [<method-descriptor>]]
```

where possible options include:

```
--classpath <paths>  sets classpath for user code
                     (default: .; alias: -cp)
--sourcepath <paths> sets sourcepath for user code
                     (default: .; alias: -sp)
--reportdir <path>   sets the output directory for report files
                     (default: ./report; alias: -rd)
--outdir <path>     sets the output directory for class files
                     (default: .; alias: -d)
--source <release>  sets source compatibility with specified release
                     (release: 1.1 .. 1.7; default: 1.5, alias: -s)
--disable-rendering disables report rendering
                     (alias: -dr)
--report-only       only performs report rendering
                     (alias: -ro)
--quiet            disables console output
```

and analysis bounding options:

```
--kbound <number>  sets the reference chain bound
                     (default: 3; alias: -kb)
--arraybound <number> sets the number of array elements bound
                     (default: 2, alias: -ab)
--loopbound <number> sets the loop iteration bound
                     (default: 13, alias: -lb)
--callbound <number> sets the method call chain bound
                     (default: 10, alias: -cb)
--timeout <minute> sets timeout to <minute>
                     (default: 10, alias: -t)
```

Most options are self-explanatory as they are similar to `javac`. Note that the `<class-name>` has to be a fully qualified name. In the current release, the fully qualified name should be the class name at the bytecode level, e.g., replace dot (.) with (\$) for inner class names; moreover, `<method-descriptor>` is a **bytecode-level descriptor** for the parameter types and the return type of the method to analyze. While cumbersome at this point, Kiasan includes a tool to mine method descriptors from class files (see Section 2.4.3.1); for convenience, the tool can generate scripts to run Kiasan both through its command-line interface and the Kiasan Ant task (see the next section). The `--disable-rendering` option is used to disable the Semanggi component, and the `--report-only` option is used to only run Semanggi. They can be used to first accumulate KiasanVM reports, and then finally use Semanggi to render the accumulated reports.

Kiasan uses a bounded analysis, the analysis bounding options can be used to control Kiasan's analysis; the larger the bounds, the more expensive the analysis cost would be, but, Kiasan can cover more behaviors of the unit. The `--kbound` option is used to control the length of (symbolic) heap object reference chains from the unit's input parameters (including the receiver object, i.e., `this`, and static fields). For example, specifying `--kbound 3` when analyzing a tree structure from a root would cause all tree structures up to height 3 to be analyzed. The `--arraybound` option is used to limit the number of distinct array indices that can be accessed on symbolic arrays (the indices themselves can be random, but still within the array index bounds). Kiasan's algorithm is quantifiable and case-optimal with respect to these bounds, thus, giving a strong behavior guarantee (relatively sound and complete) while keeping analysis cost low; interested readers are referred to Chapter 6 for more discussion on this topic.

2.4.2 Using Ant

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project">

  <typedef resource="org/sireum/kiasan/profile/jvm/tools/ant/kiasanantlib.xml" />

  <target name="check-container">
    <kiasan classpath="bin" sourcepath="src" classname="kiasan.examples.Container" />
  </target>
</project>
```

Figure 2.1: A Kiasan Ant Task Example

The Sireum/Kiasan distribution includes a customized Ant task to run Kiasan. In order to use it, make sure that Kiasan.jar is in the Java CLASSPATH when running Ant. Figure 2.1 presents an example that uses the Kiasan Ant task to analyze the swap methods. Table 2.1 describe Kiasan Ant task's attributes. The task forms an implicit FileSet and supports some attributes of <fileset> including nested <include> and <exclude> elements. The fileset is used to specify classfiles to analyze.

Attribute	Description	Required
classname	The fully qualified name of the class to analyze	No (if unspecified, a nested fileset has to be provided)
methodname	The name of the method to analyze	No (if unspecified, all methods will be analyzed)
methoddesc	The descriptor of the method to analyze (to disambiguate methods which have the same names)	No (if unspecified, all methods with the specified name will be analyzed; if specified, it requires methodname to be specified as well)
classpath	The classpath for the classes to analyze	No (defaults to the project's base directory)
sourcepath	The sourcepath for the classes to analyze	No (defaults to the project's base directory)
reportdir	The directory for generated reports	No (defaults to report under the project's base directory)
source	The Java source release compatibility: 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7	No (defaults to 1.5)
outdir	The output directory for generated contract classes	No (defaults to the project's base directory)
renderingonly	Indicates whether to only run the Semanggi report renderer	No (defaults to false)
reportDisabled	Indicates whether the Semanggi report renderer should not be run	No (defaults to false)
kbound	The bound on symbolic object reference chains from method inputs	No (defaults to 3)
arrayelementsbound	The bound on the number of distinct array indices that can be accessed on symbolic arrays	No (defaults to 2)
loopbound	The bound on the number of loop iterations	No (defaults to 13)
callchainbound	The bound on the method call chains	No (defaults to 10)
timeout	The time limit in minutes for analysis	No (defaults to 10 minutes)

Table 2.1: Kiasan Ant Task Attributes

2.4.3 Running Kiasan Tools

The Sireum/Kiasan distribution includes a couple of simple tools aimed to ease experimenting with Kiasan, which we describe in the following sections.

2.4.3.1 Method Information Miner (MIM)

The Method Information Miner (MIM) tool can be used to display information about methods in a class bytecode. You can run MIM in a command prompt as follows:

```
$ java -classpath KIASAN_HOME/lib/Kiasan.jar
  org.sireum.kiasan.profile.jvm.tools.MethodInfoMiner
Sireum/Kiasan for Java vX.Y.YYYYMMDD
Tool: Method Information Miner
http://www.sireum.org

(c) SAnToS Laboratory, Kansas State University

Parameters: <classpath> <classname> [ ant | cmd | mo | po ]
```

Or, you can use one of the `mim` scripts in `KIASAN_HOME/bin`. By default, MIM displays triples of class name, method name, and method descriptor of a class that can be used for calling Kiasan. For example, if we use the command:

```
$ mim . kiasan.examples.Container
```

under the `KIASAN_HOME/examples` directory (and assuming `KIASAN_BIN` is in your path), then MIM will display the following triples:

```
'kiasan.examples.Container' '<init>' ' ()V'
'kiasan.examples.Container' 'swap' ' (Lkiasan/examples/Container;)V'
'kiasan.examples.Container' 'swap2' ' (Lkiasan/examples/Container;)V'
'kiasan.examples.Container' 'swap3' ' (Lkiasan/examples/Container;)V'
```

The options `[ant]`, `[cmd]`, `[mo]`, and `[po]` controls the output of MIM:

- `ant`, used to output Ant scripts to call Kiasan. Below is the output (reformatted) of MIM for the `Container` example with the `ant` option specified:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="default">
  <!-- Need to set classpath to point to Kiasan.jar before running this script -->
  <typedef resource="org/sireum/kiasan/profile/jvm/tools/ant/kiasanantlib.xml" />

  <property name="kiasan.cp" value="bin" />
  <property name="kiasan.sp" value="src" />
  <property name="kiasan.rd" value="report" />
  <property name="kiasan.od" value="bin" />
  <property name="kiasan.quiet" value="false" />
  <property name="kiasan.ro" value="false" />
  <property name="kiasan.dr" value="false" />
  <property name="kiasan.kb" value="3" />
  <property name="kiasan.ab" value="2" />
  <property name="kiasan.lb" value="13" />
  <property name="kiasan.cb" value="10" />
  <property name="kiasan.time" value="10" />

  <target name="default">
    <echo message="Executing kiasan.examples.Container.<init>.( )V ..."/>
    <kiasan classname="kiasan.examples.Container"
      methodname="<init>"
```

```

        methoddesc="()V"
        classpath="${kiasan.cp}" sourcepath="${kiasan.sp}"
        reportdir="${kiasan.rd}" outdir="${kiasan.od}"
        quiet="${kiasan.quiet}"
        renderingonly="${kiasan.ro}"
        renderingdisabled="${kiasan.dr}"
        kbound="${kiasan.kb}"
        arrayelementsbound="${kiasan.ab}"
        loopbound="${kiasan.lb}"
        callchainbound="${kiasan.cb}"
        timeout="${kiasan.time}" />
    ...
</target>
</project>

```

- `cmd`, used to output batch scripts

```

export JAVA_OPTIONS="-d64"
export UNIT_CLASSPATH=
export UNIT_SOURCEPATH=
export KIASAN_HOME=
export KIASAN_OPTIONS="-cp $UNIT_CLASSPATH -sp $UNIT_SOURCEPATH"
export KIASAN_JAR="$KIASAN_HOME/lib/Kiasan.jar"
export KIASAN="java $JAVA_OPTIONS -jar $KIASAN_JAR $KIASAN_OPTIONS -dr"
export KIASAN_R="java $JAVA_OPTIONS -jar $KIASAN_JAR $KIASAN_OPTIONS -ro"

$KIASAN 'kiasan.examples.Container' '<init>' '()V'
...

$KIASAN_R

```

- `mo`, used to output XStream-serialized `org.sireum.kiasan.profile.jvm.option.MethodOption` objects that are handy for specifying code substitutions (see the next chapter):

```

<MethodOption>
  <theClassName>kiasan.examples.Container</theClassName>
  <theMethodName>&lt;init&gt;</theMethodName>
  <theMethodDescriptor>()V</theMethodDescriptor>
</MethodOption>
...

```

Note that the fully qualified class name of `MethodOption` is simplified using XStream's alias facility.

- `po`, used to output XStream-serialized `org.sireum.kiasan.profile.jvm.option.ProgramOption` objects that are handy for using KiasanVM directly (see Section 2.4.4):

```

<ProgramOption>
  <theMethodOption>
    <theClassName>kiasan.examples.Container</theClassName>
    <theMethodName>&lt;init&gt;</theMethodName>
    <theMethodDescriptor>()V</theMethodDescriptor>
  </theMethodOption>
  <theOptionalMethodExtensionClassNames>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.AssertionErrorExtension</ ↔
    string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.BooleanExtension</string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.FloatExtension</string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.ExceptionExtension</string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.IntegerExtension</string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.MathExtension</string>
    <string>org.sireum.kiasan.profile.jvm.extension.java.lang.ObjectExtension</string>
  </theOptionalMethodExtensionClassNames>

```

```
<string>org.sireum.kiasan.profile.jvm.extension.java.util. ↵  
  NoSuchElementExceptionExtension</string>  
<string>org.sireum.kiasan.profile.jvm.extension.java.lang.RuntimeExceptionExtension</ ↵  
  string>  
</theOptionalMethodExtensionClassNames>  
</ProgramOption>  
...
```

2.4.3.2 Extension Stub Generator (ESG)

PENDING

2.4.4 Running KiasanVM Directly

PENDING

Chapter 3

Code Substitution and Modeling

PENDING

3.1 Redirection

PENDING

3.2 Extension

PENDING

3.3 Reflection

PENDING

Chapter 4

Java Library Support

4.1 Package: `java.lang`

Class	Methods
<code>AssertionError</code>	<code><init></code> (constructor)
<code>Boolean</code>	<code>booleanValue</code> , <code>valueOf</code>
<code>Double</code>	<code>doubleValue</code> , <code>valueOf</code>
<code>Exception</code>	<code><init></code>
<code>Float</code>	<code>floatValue</code> , <code>valueOf</code>
<code>IllegalStateException</code>	<code><init></code>
<code>Integer</code>	<code>intValue</code> , <code>valueOf</code>
<code>Long</code>	<code>longValue</code> , <code>valueOf</code>
<code>Math</code>	<code>max</code>
<code>Object</code>	<code><init></code> , <code>equals</code>
<code>RuntimeException</code>	<code><init></code>

Table 4.1: Supported classes and methods in `java.lang` package

4.2 Package: `java.util`

Class	Methods
<code>NoSuchElementException</code>	<code><init></code>

Table 4.2: Supported classes and methods in `java.util` package

Chapter 5

JML Support

5.1 Type Specifications

Construct	Description
PENDING	PENDING

Table 5.1: Supported JML type specifications

5.2 Method Specifications

Construct	Description
assignable	PENDING
ensures	PENDING
requires	PENDING

Table 5.2: Supported JML method specifications

5.3 Primary Expressions

Construct	Description
\fresh	PENDING

Table 5.3: Supported JML primary expressions

Chapter 6

Case-Optimality of Kiasan's Algorithm

In this chapter, we demonstrate that Kiasan's algorithm is case-optimal for many complex data structures (e.g., trees). For each data structure in the following sub-sections, we mathematically analyzed the optimal numbers of cases that an analysis algorithm should consider to explore all possible data structure shapes according to the data structure invariants, in order to soundly consider all data structure operation behaviors while keeping cost low (i.e., it does not explore more behaviors than necessary). In each formula described in the following sub-sections, n relates to Kiasan's k -bound; that is, $k = n + 2$. Given a specific k such that n is positive, Kiasan generates exactly the same number of cases when computed using the mathematical formula. The readers are referred to [Deng-al:SEFM07] and [Deng:PhDThesis] for detailed discussion on how each formula is derived.

6.1 Binary Search Tree

6.1.1 Method `findMax` and `findMin`

$$a_n = \lfloor k^{2^n} \rfloor, \text{ where } k = 1.502837\dots$$

where a_n denotes the number of cases with tree height at most n .

6.1.2 Method `find`, `insert`, and `remove`

$$b_n = 2 \sum_{i=1}^n \lfloor k^{2^{i-1}} \rfloor^2 \prod_{j=i+1}^n 2 \lfloor k^{2^{j-1}} \rfloor + \lfloor k^{2^n} \rfloor, \text{ where } k = 1.502837\dots$$

where b_n denotes the number of cases with tree height at most n .

6.2 Red-Black Tree

6.2.1 Method `lastKey`

$$a(h, b) = [a(h-1, b-1) + a(h-2, b-1)^2]^2, \quad h > 1, b \geq 1 \text{ and } a(0, 0) = a(1, 0) = a(1, 1) = 1, a(h, b) = 0 \forall h < b$$

$$b_n = \sum_{b \geq 0} a(n, b), n \geq 0$$

where b_n denotes the number of cases with tree height at most n .

6.2.2 Method get, put, and remove

$$F_{h,b}(x) = [F_{h-1,b-1}(x) + F_{h-2,b-1}^2(x)]^2, h \geq 2 \wedge b \geq 1 \text{ and } F_{1,1}(x) = x^2, F_{h,0}(x) = x \forall h \geq 0, F_{h,b}(x) = 0 \forall h < b$$

$$\text{Define } G_h(x) = \sum_{i=0}^h F_{h,i}(x).$$

$$b_n = 2G'_n(1) - G_n(1)$$

where b_n denotes the number of cases with tree height at most n .

6.3 AVL Tree

6.3.1 Method findMax and findMin

$$a_n = \lfloor \theta^{2^n} \rfloor - \lfloor \theta^{2^{n-1}} \rfloor + \lfloor \theta^{2^{n-2}} \rfloor - \dots + (-1)^n \lfloor \theta^{2^0} \rfloor, \theta \approx 1.43687$$

where a_n denotes the number of cases with tree height equal to n .

6.3.2 Method find and insert

$$F_{h,b}(x) = F_{h-1,b-1}^2(x) + F_{h-2,b-1}^2(x)F_{h-1,b-1}(x), h \geq 2 \wedge b \geq 1 \text{ and } F_{1,1}(x) = x^2, F_{h,0}(x) = x \forall h \geq 0, F_{h,b}(x) = 0 \forall h < b$$

$$H_h(x) = \sum_{i=0}^h F_{h,i}(x), h \geq 0 \quad b_n = 2H'_n(1) + H_n(1)$$

b_n denotes the number of cases with tree height equal to n .

6.4 AA Tree

6.4.1 Method findMax and findMin

$$a(h,b) = a(h-1,b-1)^2 + a(h-1,b-1)a(h-2,b-1)^2, \quad h > 1, b \geq 1 \text{ and } a(0,0) = a(1,0) = a(1,1) = 1, a(h,b) = 0 \forall h < b$$

$$b_n = \sum_{b \geq 0} a(n,b), n \geq 0$$

b_n denotes the number of cases with tree height at most n .

6.4.2 Method insert and remove

$$F_{h,b}(x) = F_{h-1,b-1}^2(x) + F_{h-2,b-1}^2(x)F_{h-1,b-1}(x), h \geq 2 \wedge b \geq 1 \text{ and } F_{1,1}(x) = x^2, F_{h,0}(x) = x \forall h \geq 0, F_{h,b}(x) = 0 \forall h < b$$

$$\text{Define } G_h(x) = \sum_{i=0}^h F_{h,i}(x).$$

$$b_n = 2G'_n(1) - G_n(1)$$

where b_n denotes the number of cases with tree height at most n .

6.5 ...

PENDING

Chapter 7

References

- [Deng-al:SEFM07] Xianghua Deng, Robby, and John Hatcliff, *Towards A Case-Optimal Symbolic Execution Algorithm for Analyzing Strong Properties of Object-Oriented Programs*, IEEE Computer Society, 273-282, 0-7695-2884-8, ([extended version](#)).
- [Deng-al:TAICPART07] Xianghua Deng, Robby, and John Hatcliff, *Kiasan/KUnit: Automatic Test Case Generation and Analysis Feedback for Open Object-oriented Systems*, IEEE Computer Society, 3-12, 0-7695-2984-4, ([extended version](#)).
- [Deng:PhDThesis] Xianghua Deng, *Contract-based Verification and Test Case Generation for Open Systems*, **Ph.D. Thesis**, Department of Computing and Information Sciences, Kansas State University, 2007.
-