The Report Committee for Kilnagar S. Bhaskar
Certifies that this is the approved version of the following report:

Enhancing Usability and Applicability of Korat

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: ______________________________
Sarfraz Khurshid

____
Herb Krasner

Enhancing Usability and Applicability of Korat

by

Kilnagar S. Bhaskar, B.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2015

Dedication

To my wife Priya for her inspiration, my daughter Aditi for her distractions, and my parents and sisters for their support, without whom this project would not have been possible and would have been completed sooner.

Acknowledgements

I would like to thank Dr. Sarfraz Khurshid, Associate Professor of Electrical and Computer Engineering at The University of Texas at Austin, for his expert guidance, and Dr. Herb Krasner, Senior Lecturer in the Department of Electrical and Computer Engineering at The University of Texas at Austin, for evaluating this report from the perspective of his academic and industry experience.

A special note of thanks to my family members who assisted me with this report: my doctor wife who watched over my health, my daughter who collated the printout by page numbers, and my parents and sisters who reassured me that I could not have done any better.

Abstract

Enhancing Usability and Applicability of Korat

Kilnagar S. Bhaskar, M.S.E.

The University of Texas at Austin, 2015

Supervisor: Sarfraz Khurshid

Software testing is an integral part of the software development cycle, and involves various techniques to test software components and applications. Specification-based testing focuses on expected functionality as described in given specifications. Korat is a tool for generating structurally complex test inputs for specification-based testing of Java programs that operate on such inputs. Korat uses specifications written as Java predicates, that describe properties of expected input structures and efficiently generates all non-isomorphic valid structures within given bounds on input size.

This report describes the software requirements, application design and implementation details of our effort to improve usability and applicability of Korat. Our work involves functional enhancements to the classic Korat tool to provide support for the following elements: Graphical User Interface (GUI), Java Universal Network/Graph Framework (JUNG) output, Finite State Machine Domain (FSM), and JavaScript Object Notation (JSON) graph archival.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

Software testing is an integral part of the software development cycle. Various testing techniques are used in the industry to test software components and applications. Specification-based testing tests the functionality of code with respect to given specifications that describe expected program behaviors. Specification-based testing is typically black-box testing -- the testers are not required to have knowledge of the software architecture, design or code implementation, and they can focus on what the software does and not on how it does it.

The input domain space involved in modern day applications is typically very large, which makes manual testing costly and error-prone. An approach to deal with large input spaces is to construct their models and use tools to automate testing with respect to the models. Input space models can be written in the form of specifications in formal languages. A common approach is to use finite-state machine models to describe how the system is expected to behave.

1.1 KORAT

Korat [2] is a tool for specification-based generation of structurally complex test inputs for Java programs. Given a specification of desired inputs, Korat enumerates all inputs that meet the specification within a given bound on the input size. Korat's key strength is its ability to generate structurally complex inputs, which satisfy complex properties that relate parts of the structure. Korat requires the users to write input specifications in Java. A typical input specification is written as a predicate (i.e., boolean returning method) that inspects its given input to check whether expected properties hold and returns true or false based on the outcome of the inspection. The user also provides a

Finitization method that defines the bound on the input space. Korat outputs all non-isomorphic structures that are within the finitization bound and are determined by the predicate method to be valid.

1.2 KORAT ENHANCEMENTS

This report describes the software requirements, application design and implementation details of our work on improving the usability and applicability of Korat, specifically on the following elements:

- Graphical User Interface using Java Swing [11]
- Graph Output Format using JUNG (Java Universal Network/Graph Framework) [10]
- Finite State Machine Domain
- Graph Archival using JSON (JavaScript Object Notation) [9]

We embody these enhancements in our prototype tool named Korat-2015 [5].

Chapter 2 Requirements

The software requirements were captured and their quality scores were reviewed using Innoslate [6] requirements management tool as shown in Figure 2.1

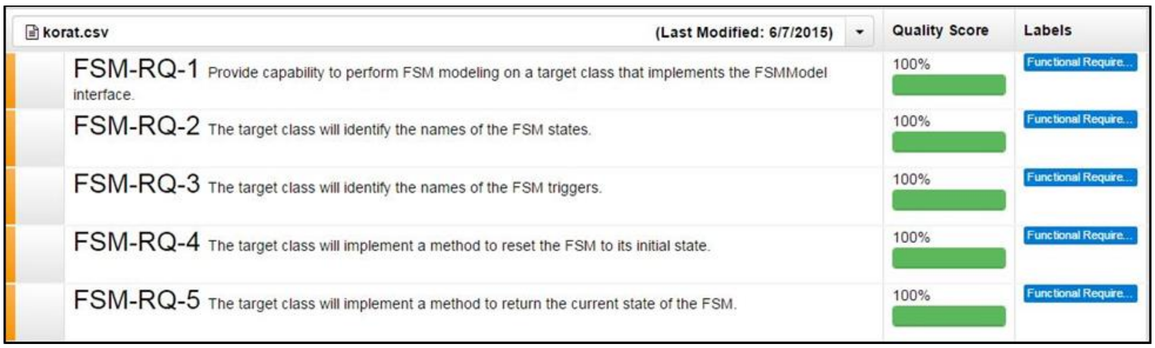


Figure 2.1: Requirements Management Tool

2.1 Functional Requirements

This section lists both functional and non-functional software requirements for Korat-2015.

2.1.1 Graphical User Interface (GUI) Support

The software requirements to add a graphical user interface to Korat-2015 are shown in Table 2.1. They include running Korat-2015 by providing specific arguments, viewing the generated text and graph outputs, viewing any console output generated, obtaining help information, saving the graph as a file, and loading a graph from a file.

ID	**Requirement Description**
GUI-RQ-1	Provide a "- -gui" command line switch to start Korat-2015 in GUI mode.
GUI-RQ-2	Invoke Korat-2015 in classical text mode if the command line does not contain the "- -gui" switch.
GUI-RQ-3	On startup, launch Korat-2015 in full screen mode and display a blank screen.
GUI-RQ-4	Display the tool name on the application tool bar.
GUI-RQ-5	A "File" menu will allow user to transform and manage the Korat-2015 generated internal models as text based files.
GUI-RQ-6	Under the "File" menu, a "Load File" submenu will allow users to navigate to the file location and choose the Korat-2015 file. This file will be used as input to display the models in a graph format.
GUI-RQ-7	The "Load File" submenu will display folders in the file chooser dialog box.
GUI-RQ-8	When a user loads a ".kjson" file, Korat-2015 will hold only the model data contained in this file and will erase any other model information that it had prior to loading the file.
GUI-RQ-9	Under the "File" menu, a "Save File" submenu will allow users to navigate to a file location to store the Korat-2015 file. This file will be used to store the Korat-2015 graph models.
GUI-RQ-10	The "Save File" submenu will display folders in the file chooser dialog box.

Table 2.1: GUI Functional Requirements

ID	Requirement Description
GUI-RQ-11	Under the "File" menu, an "Exit" submenu will allow users to quit the application.
GUI-RQ-12	A "Run" menu will allow user to specify parameters and execute Korat-2015.
GUI-RQ-13	Under the "Run" menu, a "Run" submenu will display a "Run" screen that allow users to enter the parameters required to execute Korat-2015.
GUI-RQ-14	A "Run" heading will be displayed on the "Run" screen to inform users of their current navigation point in the Korat-2015 menu.
GUI-RQ-15	The "Run" submenu will display default screen values for all the parameters.
GUI-RQ-16	Users will be able to overwrite the default screen values and enter the command parameters.
GUI-RQ-17	Korat-2015 will process only those fields that do not have the default screen values.
GUI-RQ-18	The "Run" submenu will have a "Clear" button to erase all user entered information and reset all the parameters to their default screen values.
GUI-RQ-19	The user entered information in the "Run" form will not be erased, when the user navigates to a different menu item.
GUI-RQ-20	An "Output" menu will allow users to view output generated by Korat-2015 in different formats.

Table 2.1 (continued): GUI Functional Requirements

ID	Requirement Description
GUI-RQ-21	Under the “Output” menu, a “Graph” submenu will allow users to view the Korat-2015 model output in graph format, represented as a directed graph containing vertices and edges.
GUI-RQ-22	A “Graph” heading will be displayed on the “Graph” screen to inform users of their current navigation point in the Korat-2015 menu.
GUI-RQ-23	A complete non-highlighted graph will be displayed, if one has been generated, when the user chooses the “Graph” submenu.
GUI-RQ-24	Korat-2015 will allow users to browse the generated paths on the graph, and the current displayed path will be highlighted on the graph.
GUI-RQ-25	The “Graph” screen will have a “Next” button to allow the user to browse the next path on the graph.
GUI-RQ-26	The “Graph” screen will have a “Previous” button to allow the user to browse the previous path on the graph.
GUI-RQ-27	The “Graph” screen will show the identifier of the current path displayed.
GUI-RQ-28	The “Graph” screen will contain an “Animate” button that will allow users to automatically browse by sequencing through the paths after a short delay.
GUI-RQ-29	Under the “Output” menu, a “Text” submenu will allow users to view the Korat-2015 model output in text format.

Table 2.1 (continued): GUI Functional Requirements

ID	Requirement Description
GUI-RQ-30	A “Text” heading will be displayed on the “Text” screen to inform users of their current navigation point in the Korat-2015 menu. This screen will display information sent by Korat-2015 to the standard output.
GUI-RQ-31	The “Text” screen will allow users to scroll through information on this screen.
GUI-RQ-32	The “Text” screen will have a “Clear” button to allow users to erase information on this screen.
GUI-RQ-33	The information in the “Text” screen will not be erased, when the user navigates to a different menu item.
GUI-RQ-34	Under the “Output” menu, a “Console” submenu will allow users to view the Korat-2015 system output.
GUI-RQ-35	A “Console” heading will be displayed on the “Console” screen to inform users of their current navigation point in the Korat-2015 menu. This screen will display error information sent by Korat-2015 to the standard error.
GUI-RQ-36	The “Console” screen will allow users to scroll through information on this screen.
GUI-RQ-37	The “Console” screen will have a “Clear” button to allow users to erase information on this screen.
GUI-RQ-38	The information in the “Console” screen will not be erased, when the user navigates to a different menu item.

Table 2.1 (continued): GUI Functional Requirements

ID	Requirement Description
GUI-RQ-39	A “Help” menu will provide information about the Korat-2015 tool.
GUI-RQ-40	Under the “Help” menu, a “Help” submenu will display helpful information about the Korat-2015 tool.

Table 2.1 (continued): GUI Functional Requirements

2.1.2 Finite State Machine (FSM) Support

The software requirements to run Korat-2015 on a target class that implements a finite state machine are shown in Table 2.2. They include exploring the state space, generating exploration information in text format and in an appropriate format to generate a graph, and generating coverage metrics.

ID	Requirement Description
FSM-RQ-1	Provide capability to perform FSM modeling on a target class that implements the FSMModel interface.
FSM-RQ-2	The target class will identify the names of the FSM states.
FSM-RQ-3	The target class will identify the names of the FSM triggers.
FSM-RQ-4	The target class will implement a method to reset the FSM to its initial state.
FSM-RQ-5	The target class will implement a method to return the current state of the FSM.

Table 2.2: FSM Functional Requirements

ID	Requirement Description
FSM-RQ-6	The target class will use the “@Trigger” annotation to identify the trigger methods that perform actions in the FSM.
FSM-RQ-7	The target class will define a guard method associated with a trigger method. The guard method defines when a trigger is valid and can be pulled /performed.
FSM-RQ-8	Translate the FSM state information into an appropriate format to show the states as vertices in the graph format.
FSM-RQ-9	Translate the FSM trigger information into an appropriate format to show the triggers as edges in the graph format.
FSM-RQ-10	Provide standard text output from the Korat-2015 FSM model run, in the Text Output window.
FSM-RQ-11	Provide any error information from the Korat-2015 FSM model run, in the Console Output window.
FSM-RQ-12	Provide “Trigger Coverage” metrics for modeling performed on an FSM class.
FSM-RQ-13	Provide “State Coverage” metrics for modeling performed on an FSM class.
FSM-RQ-14	Provide “Transition Coverage” metrics for modeling performed on an FSM class.
FSM-RQ-15	Provide “Transition Pair Coverage” metrics for modeling performed on an FSM class.

Table 2.2 (continued): FSM Functional Requirements

ID	Requirement Description
FSM-RQ-16	(OPTIONAL Requirement) Korat-2015 will have the capability to perform a random state reset during a modeling run on the target class.
FSM-RQ-17	(OPTIONAL Requirement) Korat-2015 will have the capability for automated test case generation based on the modeling run on the target class.

Table 2.2 (continued): FSM Functional Requirements

2.1.3 Java Universal Network/Graph Framework (JUNG) Support

The software requirements to generate a graph for an explored state space on a target class are shown in Table 2.3.

ID	Requirement Description
JUNG-RQ-1	Korat-2015 will display the graph using the information in the ".kjson" file.
JUNG-RQ-2	The paths in the graph will be displayed using vertices and edges.
JUNG-RQ-3	Korat-2015 will display the graph using least possible number of nodes and edges.
JUNG-RQ-4	The vertices in the graph will be displayed as circles.
JUNG-RQ-5	The edges in the graph will be displayed as directed edges.

Table 2.3: JUNG Functional Requirements

ID	Requirement Description
JUNG-RQ-6	The vertex information will be displayed as a label inside the vertex.
JUNG-RQ-7	The edge information will be displayed as a label alongside the edge.
JUNG-RQ-8	Highlight the vertices and the edge corresponding to the path browsed by the user, in a different color.
JUNG-RQ-9	Korat-2015 will display the graph information from any user edited ".kjson" file, provided the file contents are in the correct format.
JUNG-RQ-10	Korat-2015 will display any error that it encounters while processing the graph information.
JUNG-RQ-11	For FSM models, Korat-2015 will display the states as vertices in the graph.
JUNG-RQ-12	For FSM models, Korat-2015 will display the state transitions as directed edges in the graph.

Table 2.3 (continued): JUNG Functional Requirements

2.1.4 JavaScript Object Notation (JSON) Support

The software requirements to save a graph as a file, and load a graph from a file in the appropriate format, are shown in Table 2.4.

ID	Requirement Description
JSON-RQ-1	The Json format files will be generated in plain text format.
JSON-RQ-2	The "GFX.kjson" file name will be used to generate the Json file.

Table 2.4: JSON Functional Requirements

ID	Requirement Description
JSON-RQ-3	The current directory will be used as the default location to store the Json file.
JSON-RQ-4	After a model execution, Korat-2015 will automatically store the network paths corresponding to the generated model, in the default file at the default location.
JSON-RQ-5	Korat-2015 will allow users to save the graph information in Json format in a user specified folder.
JSON-RQ-6	Korat-2015 will allow users to load the graph information from a user specified file.
JSON-RQ-7	Korat-2015 will read the graph information from any user edited ".kjson" file, provided the file contents are in the correct format.
JSON-RQ-8	Korat-2015 will not perform modeling validation on the graph information in the ".kjson" file.
JSON-RQ-9	The user defined Json file should have a ".kjson" file extension, but can have any file name.
JSON-RQ-10	Korat-2015 will display any error that it encounters while processing the Json input file.

Table 2.4 (continued): JSON Functional Requirements

2.2 Non-Functional Requirements

The software requirements that capture the non-functional aspects are shown in Table 2.5.

ID	Requirement Description
NF-RQ-1	The GUI must launch in a reasonable time frame, not exceeding 1 minute, on a standard workstation.
NF-RQ-2	The ease of use to run FSM modeling using the GUI must be similar to that of non-FSM models.
NF-RQ-3	The graph must be capable of supporting at least 5 vertices, and at most 2 edges between any two vertices.
NF-RQ-4	The graph must display labels for the vertices and the edges in an easily readable format.
NF-RQ-5	A Json file of at least 5 lines and at least 5 data elements per line must be supported.

Table 2.5: FSM Non-Functional Requirements

Chapter 3 Design

The design principle used is to seamlessly build on top of the existing classical framework, with maximum reuse of software components to provide the enhancement functionalities. This section describes the design through sample UML diagrams.

3.1 Use Case Diagram

The use case diagram from the perspective of a Korat-2015 user is shown in Figure 3.1. It follows the possible paths that a user can navigate using the graphical user interface.

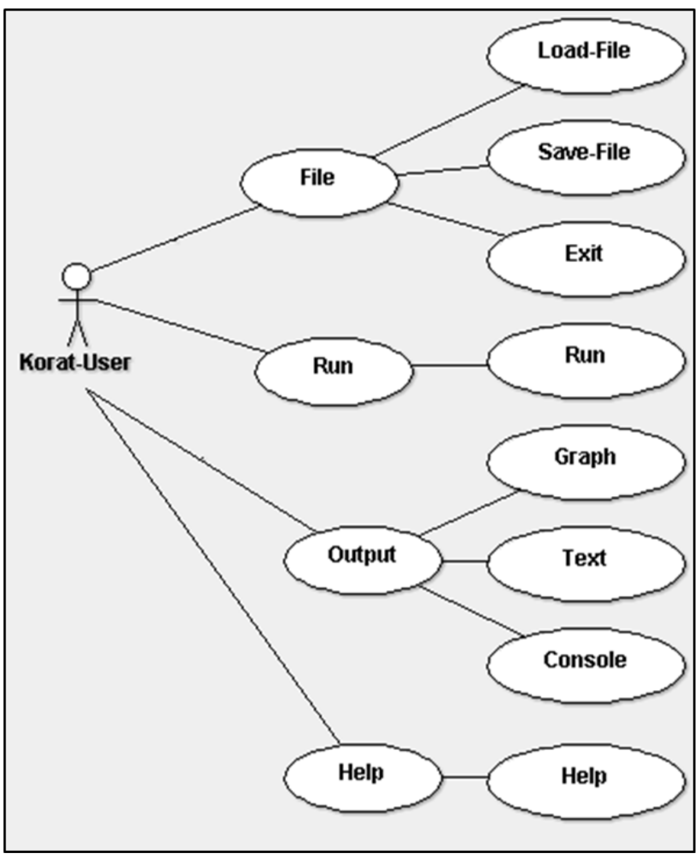


Figure 3.1: Use Case Diagram

3.2 CLASS DIAGRAM

The class diagram shown in Figure 3.2 depicts at a high level, the changes made to the existing design framework. It includes key elements that are added new or have been modified. As can be seen, the design reuses the existing software components.

The dotted lines in the diagram indicate the new dependencies that are added across the existing java packages.

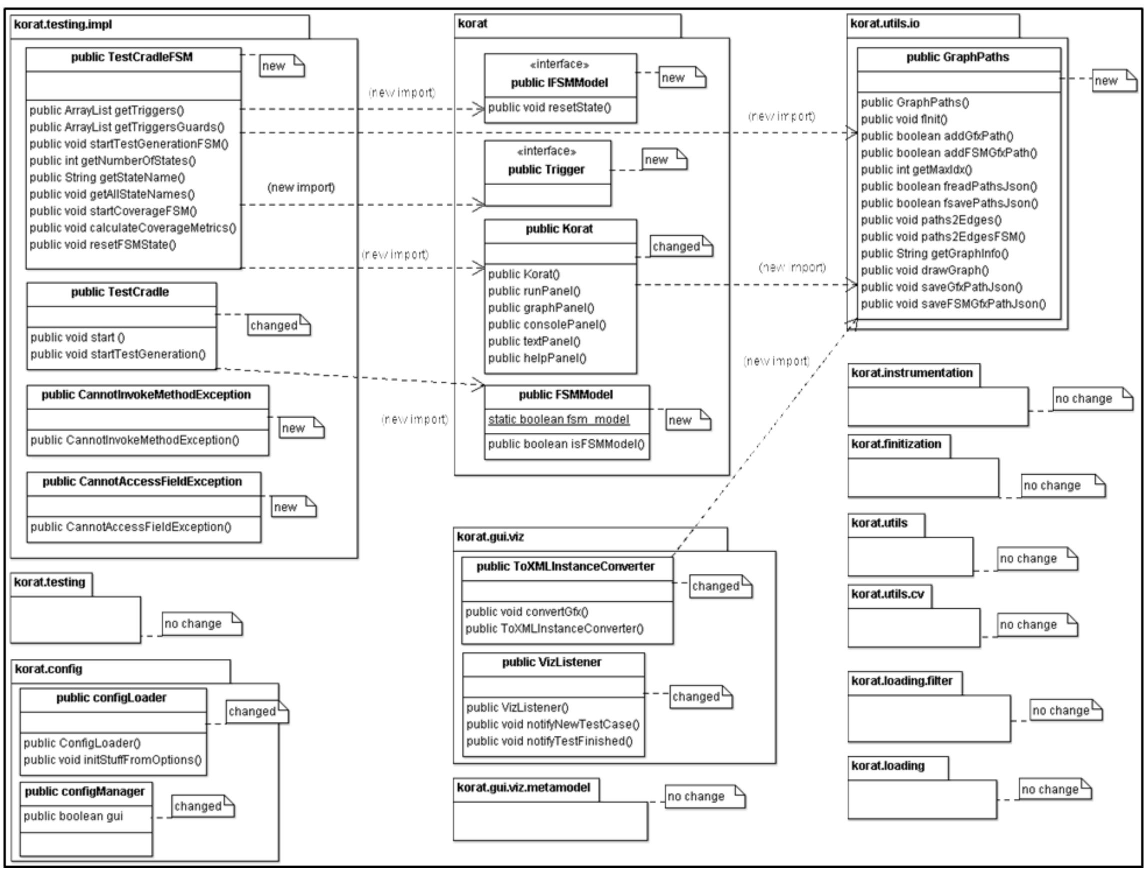


Figure 3.2: Class Diagram

3.3 SEQUENCE DIAGRAM

An overview of a sample sequence diagram is shown in Figure 3.3. It depicts the key elements and actions involved when Korat-2015 is run from the graphical user interface. Only the high level components are shown without getting into the granular details of the sequence flow.

As can be seen, every time a graph is generated, the corresponding network information is stored in a temporary file. The fInit() call clears graph information stored in the offline file.

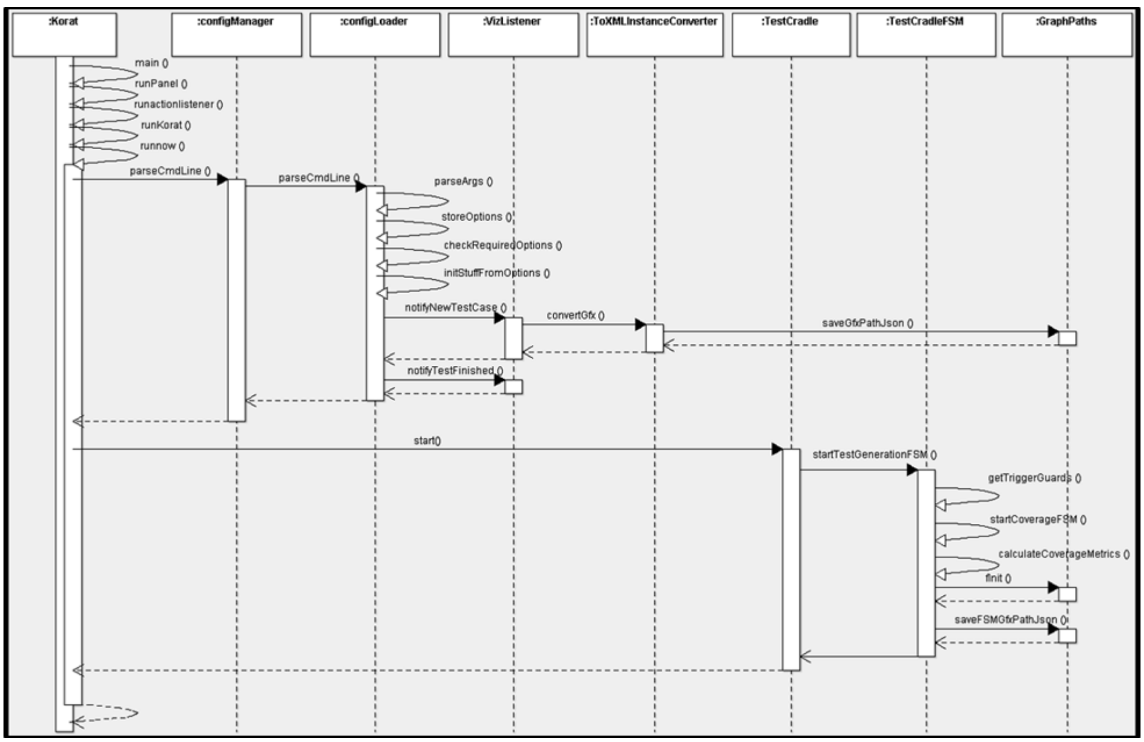


Figure 3.3: Sequence Diagram

3.4 User Interface Prototypes

This section shows the prototypes for the user interface screens for the enhanced Korat-2015 application. The graphical interface is implemented in Korat-2015 using the Java Swing library. The landing screen is launched in full screen mode with JFrame.MAXIMIZED_BOTH arguments to setExtendedState().

3.4.1 File Menu

The file menu screens are shown in Figure 3.4. The title bar displays the application name as highlighted by (1). The screen at the center is for loading the graph information from a folder using showOpenDialog(). The one on the right is for saving the graph information to a folder using showSaveDialog().

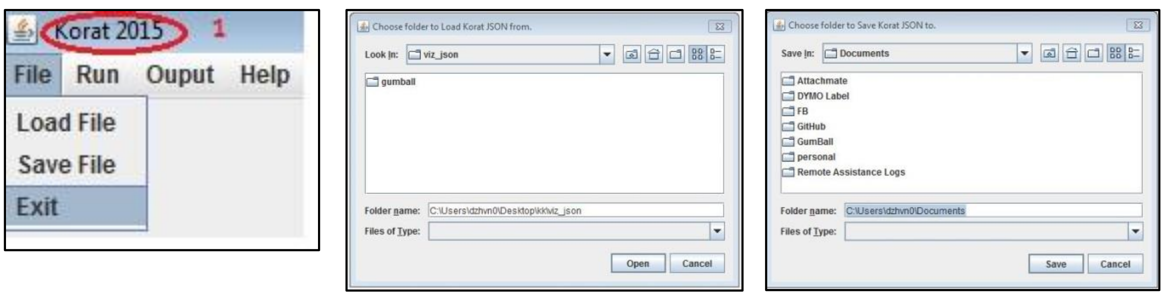


Figure 3.4: File Menu Screens

3.4.2 Run Menu

The run menu screens are shown in Figure 3.5. The screen on the right is for choosing the arguments to run Korat-2015 as highlighted by (1). The bottom of the screen highlighted by (2) contains arguments that correspond to finite state machine analysis.

A Swing ActionListener listens to event on the Clear button and resets the fields to their default values. The entered values are not cleared when the screen focus changes ensuring the values are retained across UI navigation.

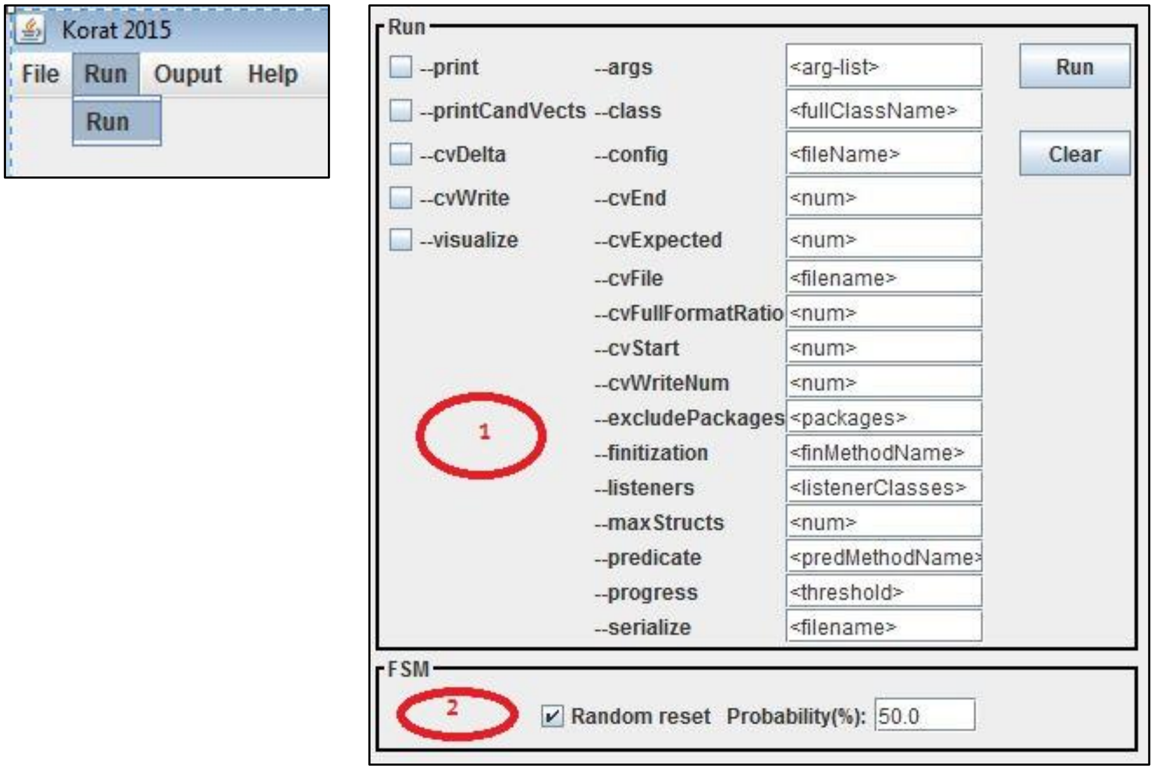


Figure 3.5: Run Menu Screens

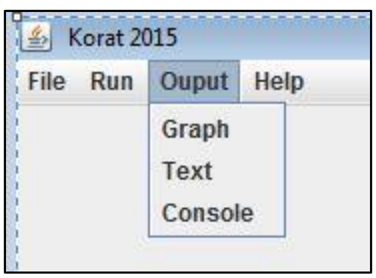
3.4.3 Output Menu

The output menu screens are shown in Figure 3.6. The screen on the top right corresponds to the graph output. On initial navigation to the Graph screen, paintGraph(PLAIN) will display a complete plain graph.

Browsing a specific graph path displays the network path information as highlighted by (1), and the network graph as highlighted by (2). The sequence number of the explored paths that is currently displayed is shown as highlighted by (3). The explored paths can be manually traversed using the buttons highlighted by (4), or traversed automatically using the check box highlighted by (5). An ActionListener listens to the events on the navigation buttons, and traverse through the GraphPath ArrayList to display the graph paths. The Animate ActionListener sequences through the graph paths with a delay between each display.

The screen on the bottom left corresponds to the text output. The current navigation point is displayed as highlighted by (1). The screen on the bottom right corresponds to the console output. The Clear button is hooked to an ActionListener to clear the screen.

The system standard output and error messages are displayed on the text and console windows, by redirecting OutputStream with System.setOut() and System.setErr().



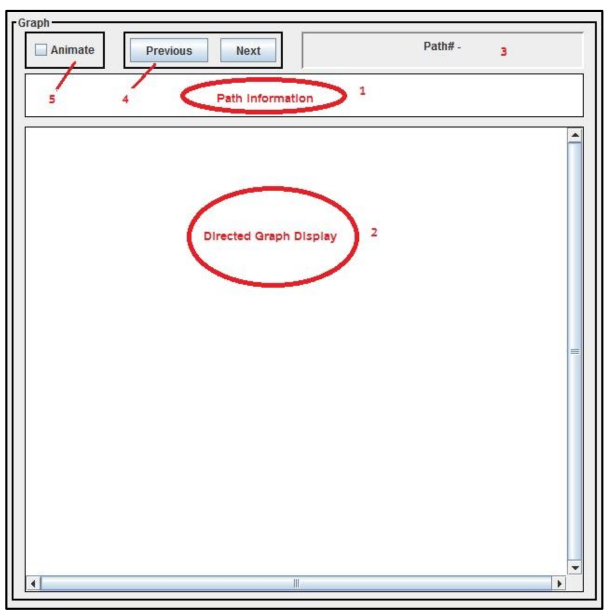


Figure 3.6: Output Menu Screens

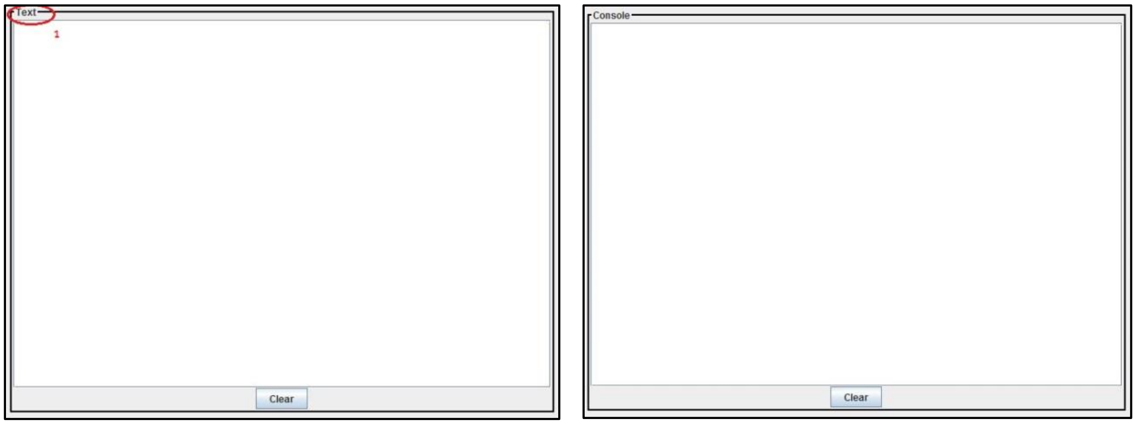


Figure 3.6 (continued): Output Menu Screens

3.4.4 Help Menu

The help menu screens are shown in Figure 3.7. The screen on the right displays the help information about the Korat-2015 application.

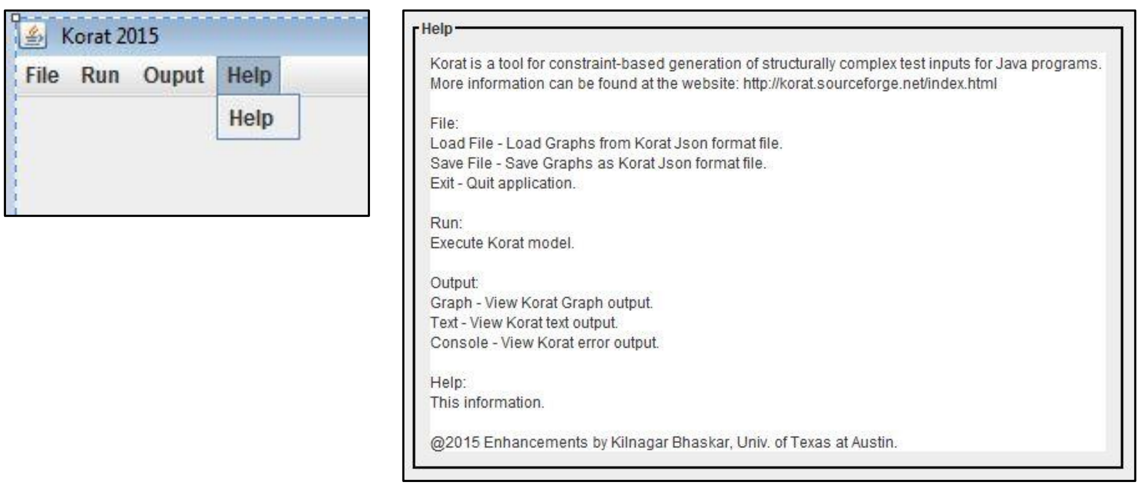


Figure 3.7: Help Menu Screens

3.5 FINITE STATE MACHINE DESIGN

This section describes the mechanics related to analyzing a FSM target class using Korat-2015.

3.5.1 Target Class

The target class is checked for the presence of @Trigger annotations to determine if the exploration and coverage analysis corresponding to a finite state machine needs to be performed.

The FSM target class implements the IFSMModel interface, defines a public String variable called State, and state names FSM_STATE_<number>. The resetState method sets the state to the initial state, and is when Korat-2015 performs a random reset during state space exploration. The trigger methods are earmarked using @Trigger annotations, and the guard methods have the same names as their corresponding trigger methods and end with "Guard".

The random reset during state space exploration can be enabled using the check box on the run screen as shown in Figure 3.8. The value of probability determines the randomness of the reset performed.

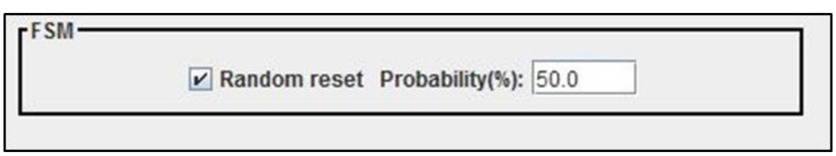


Figure 3.8: Random Reset

The skeleton of a target class is shown in Figure 3.9 with mandatory elements highlighted in bold. A finite state machine target class must have these elements.

```
public class <TargetClass> implements IFSMModel
{
        public String State;
        public String FSM_STATE_1=<Value>;
        public String FSM_STATE_2=<value>;
        ...
        public boolean <trigger>Guard () {
        }
        ...
        public @Trigger void <trigger> () {
        }
        ...
        @Override
        public void resetState() {
                State =         FSM_STATE_1;
        }
}
```

Figure 3.9: Target Class Skeleton

3.5.2 Coverage Metrics

The coverage metrics calculated by Korat-2015 on a FSM target class is shown in Figure 3.10. It includes some common metrics like state coverage, trigger coverage, transition coverage, and transition pair coverage.

```
Start of Korat Execution for <Target Class> (repOK, [<#explorations>])

FSM state reset probability: <probability between 0 and 100> %

(<from state>, <trigger>, <to state>) ****
(<from state>, <trigger>, <to state>)
…

Total explored: <# explorations done>
New & Valid found: <# unique and those validated by trigger guards>

State coverage:
<state 1>
<state 2>
…
covered: <#states covered> / <total # states> (<percent covered> %)

Trigger coverage:
<trigger 1>
<trigger 2>
…
covered: <# triggers covered> / <total # triggers> (<percent covered> %)

Transition coverage:
<transition 1>
< transition 2>
…
covered: <# transitions covered>

Transition pair coverage:
<transition 1>:<transition 2>
<transition 3>:<transition 4>
…
covered: <# transition pairs covered>

End of Korat Execution
Overall time: <execution time in seconds> s.
```

Figure 3.10: Coverage Metrics

3.6 JAVA UNIVERSAL NETWORK/GRAPH

A sample graph network displayed by Korat-2015 is shown in Figure 3.11. The states indicated by vertices are highlighted by (1) and the triggers indicated by edges are highlighted by (2). Korat-2015 displays the current path in the explored state space in a different color (green). The graph network is implemented in Korat-2015 using the JUNG library.

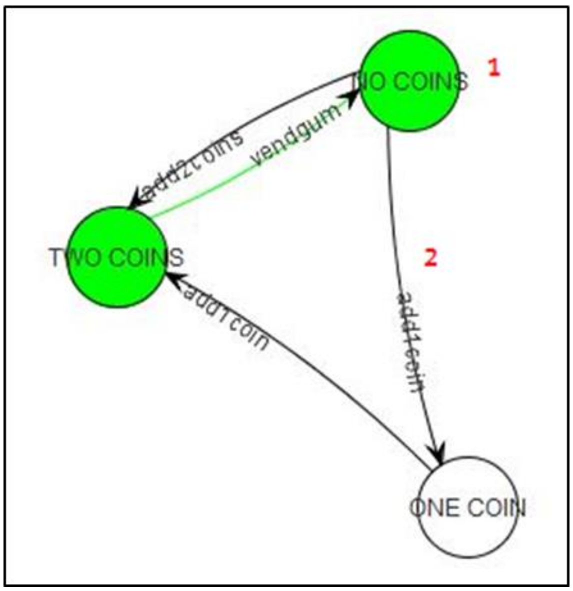


Figure 3.11: Graph Display

3.7 JAVASCRIPT OBJECT NOTATION

Korat-2015 saves the graph network information in intermediate files using the Json format. This format is viewable as a plain text and can be manually manipulated for any advanced custom analysis. The content structure of this file is shown in Figure 3.12.

Korat-2015 stores the network information in a temporary folder named “viz_json” in the current directory location. For non-FSM model analysis, the files are named GFX[n].kjson, where [n] is a running number, and each exploration graph is stored in a separate file. For FSM model analysis, the information is stored in a single file named “GFX.kjson”. Korat-2015 also uses this format when the user saves the network information as an external file or loads it from an external file. Korat-2015 used the Json file to display the graph, and does not perform model validation on the file data. The Json is implemented in Korat-2015 using the Google Gson [12] java library.

```
File Name: viz_json/GFX.kjson

File Content:
<index 1>, “from_state”, <from state>, <index 1>, “to_state”, <to state>, <index 1>,
“in_transition”, <trigger>
…

File Name: viz_json/GFX[n].kjson

File Content:
[{"fromnode":<node1>,"tonode":<node2>,"relation":<relation1>},{"fromnode":<nod
e3>,"tonode":<node4>,"relation":<relation2>}]
```

Figure 3.12: Json Structure

Chapter 4 Code Implementation

This section outlines some key aspects like algorithms, repository choices, and builds mechanisms. The implementation principle is to use and build on top of the existing code base [3].

4.1 ALGORITHM

One of the key algorithms in the support for finite state machine is the state space exploration on the target class. The UML activity diagram for this algorithm in TestCradleFSM is shown in Figure. 4.1.

The algorithm starts by scanning for variable names FSM_STATE_i to gather a list of State names. It then scans for the @Trigger annotations and gathers the list of Trigger methods. It then gathers the Guard methods based on the Trigger method names. It then iterates through invoking all Guard methods for every possible State value. The corresponding Trigger method is invoked if the Guard method returns true.

The coverage information is recorded in the iterations, and this information is used to generate the coverage metrics.

Also, the resetState () method is invoked at the probability specified when running Korat-2015.

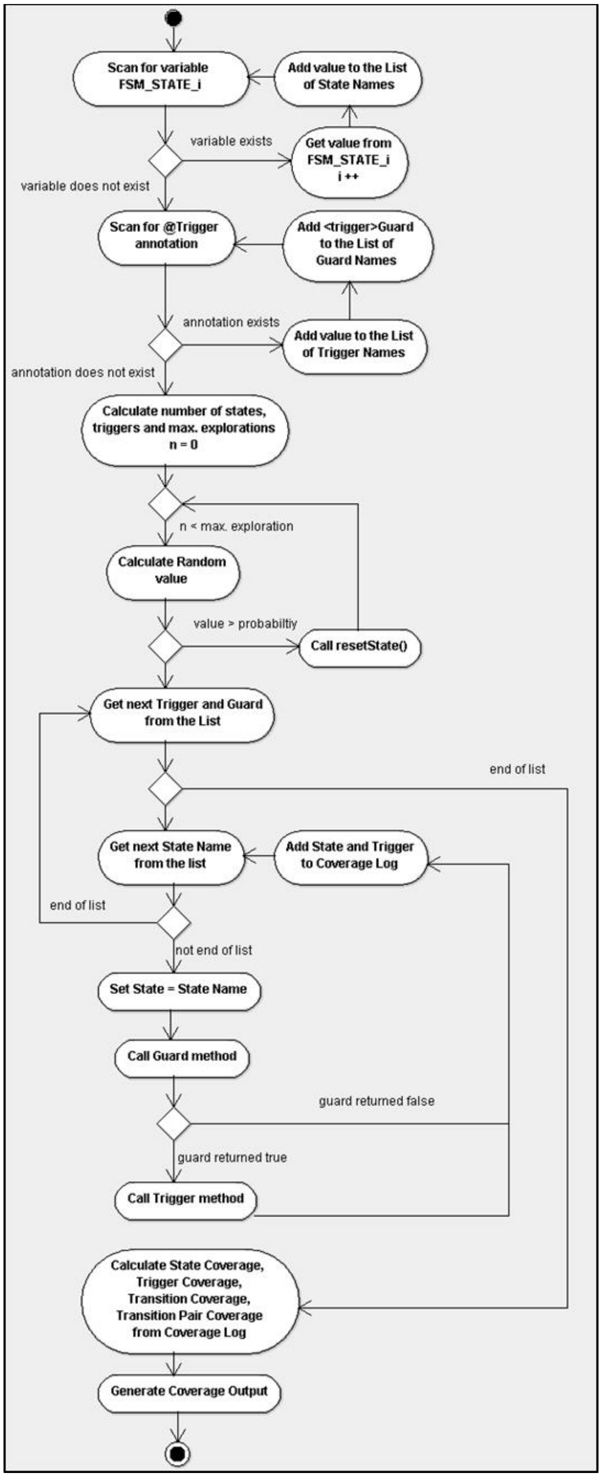


Figure 4.1: Activity Diagram

4.2 REPOSITORY

The code is maintained in the repositories shown below. The development code base is on the local machine, and also at GitHub [7] which can be integrated with Jenkins to trigger a build. The GitHub repository for the Korat-2015 code is located at https://github.com/sbhaskar17/korat2015

4.3 BUILDING KORAT-2015.JAR

The ant build.xml file has been updated to include the new java source files and the additional.jar dependencies required by the enhanced Korat-2015 application. Figure 4.2 shows a section of the build.xml file that lists the complete list of dependencies.

```
<path id="Korat.classpath">
   <pathelement location="${BUILD_DIR}" />
   <pathelement location="lib/alloy4viz.jar" />
   <pathelement location="lib/commons-cli-1.0.jar" />
   <pathelement location="lib/javassist.jar" />
   <pathelement location="lib/junit.jar" />
   <pathelement location="lib/gson-2.2.4.jar" />
   <pathelement location="lib/colt-1.2.0.jar" />
   <pathelement location="lib/concurrent-1.3.4.jar" />
   <pathelement location="lib/jung-graph-impl-2.0.1.jar" />
   <pathelement location="lib/jung-algorithms-2.0.1.jar" />
   <pathelement location="lib/jung-visualization-2.0.1.jar" />
   <pathelement location="lib/collections-generic-4.01.jar" />
   <pathelement location="lib/jung-api-2.0.1.jar" />
</path>
```

Figure 4.2: File build.xml

4.3.1 Using Command Line on Local Repository

The build can be done using the same command that was used to build the classic Korat-2015 application using the ant script. Figure 4.3 shows the log when building korat2015.jar from the command line, and the code base repository on the local computer.

```
sbhaskar17@bhaskar:~/korat$ ant createJar
Buildfile: /home/sbhaskar17/korat/build.xml

createJar:

clean:
  [delete] Deleting directory /home/sbhaskar17/korat/build
  [delete] Deleting directory /home/sbhaskar17/korat/dist

init:
   [mkdir] Created dir: /home/sbhaskar17/korat/build
    [copy] Copying 6 files to /home/sbhaskar17/korat/build

build:
    [echo] Korat: /home/sbhaskar17/korat/build.xml
   [javac] /home/sbhaskar17/korat/build.xml:58: warning: 'includeantruntime' was not
set, defaulting to build.sysclasspath=last; set to false for repeatable builds
   [javac] Compiling 184 source files to /home/sbhaskar17/korat/build
   [javac] warning: [options] bootstrap class path not set in conjunction with -source
1.5
   [javac] 1 warning
   [mkdir] Created dir: /home/sbhaskar17/korat/dist
     [jar] Building jar: /home/sbhaskar17/korat/dist/korat2015.jar

BUILD SUCCESSFUL
Total time: 9 seconds
sbhaskar17@bhaskar:~/korat$
```

Figure 4.3: Build Log

4.3.2 Using Jenkins on Local Repository

The build can be done using Jenkins, by providing appropriate build target as shown in Figure 4.4. The code base repository is on the local computer. The build log from Jenkins is shown in Figure 4.5.

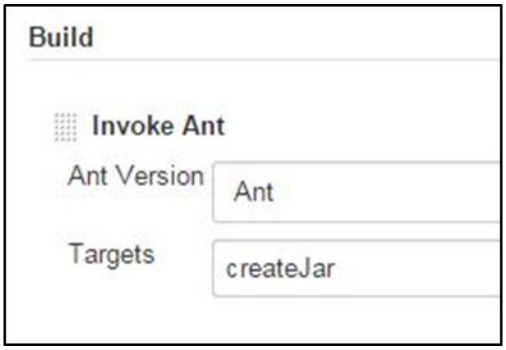


Figure 4.4: Jenkins Build

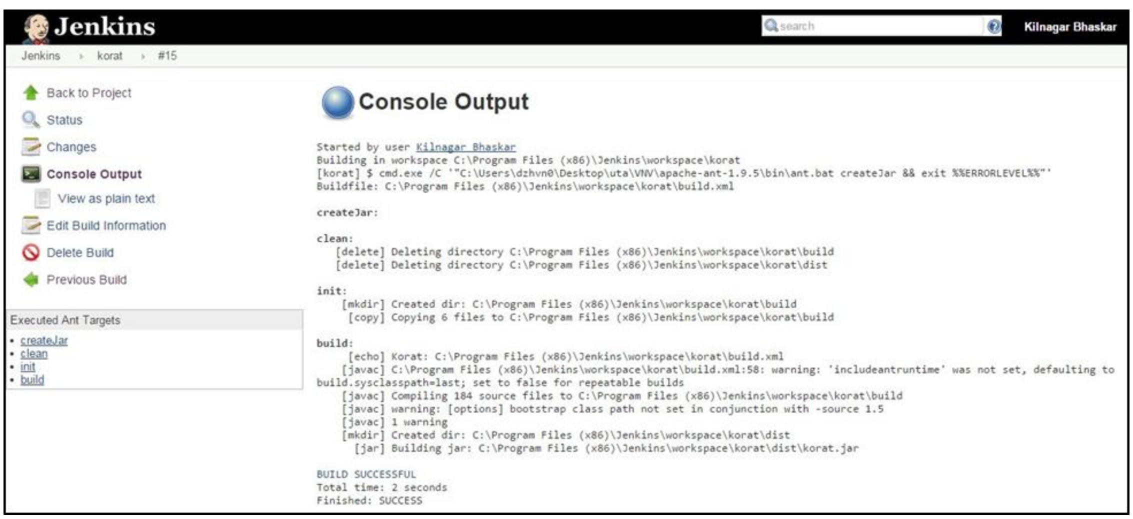


Figure 4.5: Jenkins Build Log

4.3.3 Using Jenkins on Remote Repository

The build can be done using Jenkins, on a remote code base repository like GitHub. The build log from Jenkins is shown in Figure 4.6.

Console Output

```
Started by user Kilnagar Bhaskar
Building in workspace C:\Program Files (x86)\Jenkins\workspace\korat2015
 > C:\Program Files\Git\bin\git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/sbhaskar17/korat2015.git # timeout=10
Fetching upstream changes from https://github.com/sbhaskar17/korat2015.git
 > C:\Program Files\Git\bin\git.exe --version # timeout=10
using .gitcredentials to set credentials
 > C:\Program Files\Git\bin\git.exe config --local credential.helper store --file=\"C:\Windows\TEMP\git2025742319045794647.credentials\" #
timeout=10
 > C:\Program Files\Git\bin\git.exe fetch --tags --progress https://github.com/sbhaskar17/korat2015.git +refs/heads/*:refs/remotes/origin/*
 > C:\Program Files\Git\bin\git.exe config --local --remove-section credential # timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision 59f9a01ab2a4cf74b0228eb776fdd21499e1a399 (refs/remotes/origin/master)
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -f 59f9a01ab2a4cf74b0228eb776fdd21499e1a399
 > C:\Program Files\Git\bin\git.exe rev-list 59f9a01ab2a4cf74b0228eb776fdd21499e1a399 # timeout=10
Finished: SUCCESS
```

Figure 4.6: Jenkins Build Log From GitHub

4.4 BUILDING <TARGET CLASS>.JAR

The steps to build <target class>.jar are shown in Figure 4.7.

```
sbhaskar17@bhaskar:~$ javac -cp ./korat/dist/korat2015.jar:. gumball/GumBall.java
sbhaskar17@bhaskar:~$ ls gumball
GumBall.class  GumBall.java
sbhaskar17@bhaskar:~$
```

Figure 4.7: Target Class Build

Chapter 5 Verification and Validation

This section provides the plan to test Korat-2015 application. The test principle is to use the existing test mechanisms [1, 4] and build on top of the current test framework.

5.1 Regression Tests

Our approach for regression test is to use the existing test framework as-is without any changes. The tests can be run using Jenkins with build target shown in Figure 5.1. The console output for the test is shown in Figure 5.2.

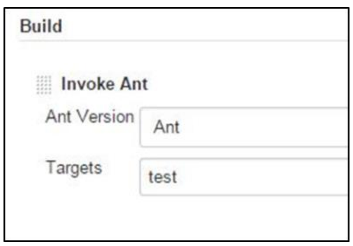


Figure 5.1: Jenkins Test Run

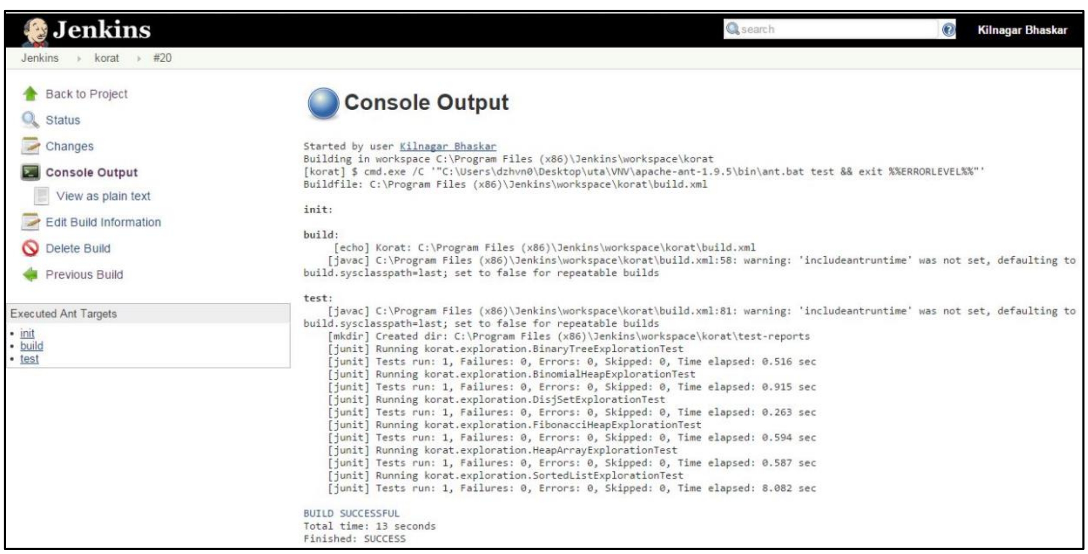


Figure 5.2: Jenkins Test Log

The regression tests can also be run from the command line or from Jenkins using the current build.xml ant and JUnit [13] test scripts. Figure 5.3 contains the log generated when testing Korat-2015 from the command line.

```
sbhaskar17@bhaskar:~/korat$ ant test
Buildfile: /home/sbhaskar17/korat/build.xml

init:

build:
     [echo] Korat: /home/sbhaskar17/korat/build.xml
    [javac] /home/sbhaskar17/korat/build.xml:58: warning: 'includeantruntime' was not
set, defaulting to build.sysclasspath=last; set to false for repeatable builds

test:
    [javac] /home/sbhaskar17/korat/build.xml:81: warning: 'includeantruntime' was not
set, defaulting to build.sysclasspath=last; set to false for repeatable builds
    [mkdir] Created dir: /home/sbhaskar17/korat/test-reports
    [junit] Running korat.exploration.BinaryTreeExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.027 sec
    [junit] Running korat.exploration.BinomialHeapExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.343 sec
    [junit] Running korat.exploration.DisjSetExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.81 sec
    [junit] Running korat.exploration.FibonacciHeapExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.248 sec
    [junit] Running korat.exploration.HeapArrayExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.747 sec
    [junit] Running korat.exploration.SortedListExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.478 sec

BUILD SUCCESSFUL
Total time: 15 seconds
sbhaskar17@bhaskar:~/korat$
```

Figure 5.3 Command Line Test Log

5.2 Enhancements Tests

There are different approaches to test, depending on how Korat-2015 is invoked by the user. One is using the command line interface and another is using the graphical user interface. The former involves adding tests to the build.xml ant script, while the latter involves manual testing. We use manual testing to test Java Swing components.

5.2.1 Automated Tests

Our approach for functions done purely from the command line interface is to use an automated test framework. These include FSM explorations and coverage, and do not include the random reset. The testing can be done using the existing ant and JUnit test framework and updating these scripts to cover the above functions.

A new test script FSMExplorationTest has been added, and the key changes to the build.xml file are shown in Figure 5.4.

File: build.xml

```
    <junit printsummary="true" fork="true" haltonfailure="false">
      <classpath refid="Korat.classpath" />
      <formatter type="plain" />
      <batchtest fork="yes" todir="${TEST_REPORTS_DIR}">
        <fileset dir="tests/">
          <include name="korat/exploration/*ExplorationTest.java" />
          <exclude name="korat/exploration/BaseExplorationTest.java" />
          <exclude name="korat/exploration/BaseFSMExplorationTest.java"
/>
          <exclude name="korat/exploration/ExplorationAllTests.java" />
        </fileset>
      </batchtest>
    </junit>
```

Figure 5.4: File build.xml Changes

The changes include new classes as shown in Figure 5.5, and test target class as shown in Figure 5.6.

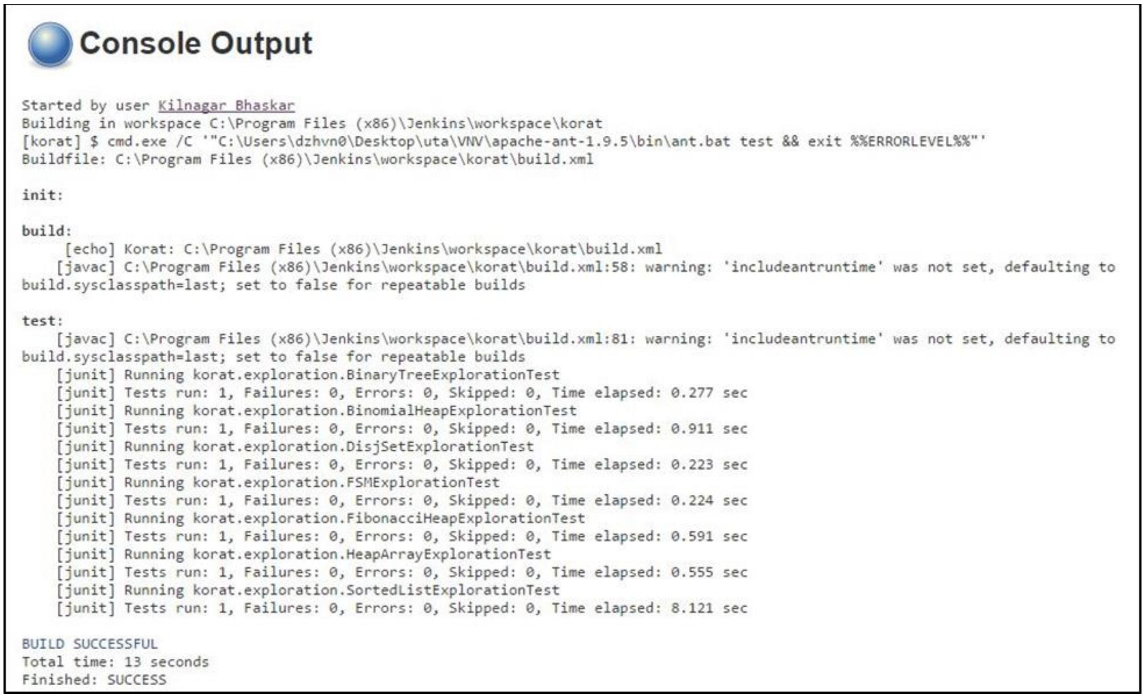
```
File: BaseFSMExplorationTest
package korat.exploration;
..
class TestConfigsFSM {
…
}

public class BaseFSMExplorationTest extends TestCase {
…
   private void doTestForAllConfigs(String[] args, int newCases, int tested) {

      TestConfigsFSM it = TestConfigsFSM.getInstance();
      it.reset();
      while (it.hasNext()) {

         it.next();
         Korat.main(args);
         assertEquals(newCases,
TestCradleFSM.getInstance().getValidCasesGenerated());
         if (tested > 0) {
            assertEquals(tested, TestCradleFSM.getInstance().getTotalExplored());
         }
      }
   }
}

File: FSMExplorationTest.java
package korat.exploration;
public class FSMExplorationTest extends BaseFSMExplorationTest {

   public void testFSM() throws Exception {
      String cmdLine = "-c korat.examples.fsm.FSM -a 3";
      doTestForAllConfigs(cmdLine, 1, 3);
   }
}
```

Figure 5.5: FSM Exploration Test Classes

```
File: FSM.java
package korat.examples.fsm;
import korat.*;

public class FSM implements IFSMModel {
	public String State;
	public String FSM_STATE_1="NO COINS";
	public String FSM_STATE_2="ONE COIN";
	public String FSM_STATE_3="TWO COINS";

	public boolean add1coinGuard() {
		return (State.equals(FSM_STATE_1) || State.equals(FSM_STATE_2));
	}

	public @Trigger void add1coin() {
		State = State.equals(FSM_STATE_1) ? FSM_STATE_2 :
FSM_STATE_3;
	}

	public boolean add2coinsGuard() {
		return (State.equals(FSM_STATE_1));
	}

	public @Trigger void add2coins() {
		State = FSM_STATE_3;
	}

	public boolean vendgumGuard() {
		return (State.equals(FSM_STATE_3));
	}

	public @Trigger void vendgum() {
		State = FSM_STATE_1;
	}

	@Override
	public void resetState() {
		State = FSM_STATE_1;
	}
}
```

Figure 5.6: Test Script Target Class

Figure 5.7 shows the Jenkins console output from running the above JUnit script for the finite state machine test. Figure 5.8 shows the test report generated.




```
Started by user Kilnagar Bhaskar
Building in workspace C:\Program Files (x86)\Jenkins\workspace\korat
[korat] $ cmd.exe /C '"C:\Users\dzhvn0\Desktop\uta\VNV\apache-ant-1.9.5\bin\ant.bat test && exit %%ERRORLEVEL%%"'
Buildfile: C:\Program Files (x86)\Jenkins\workspace\korat\build.xml

init:

build:
     [echo] Korat: C:\Program Files (x86)\Jenkins\workspace\korat\build.xml
    [javac] C:\Program Files (x86)\Jenkins\workspace\korat\build.xml:58: warning: 'includeantruntime' was not set, defaulting to
build.sysclasspath=last; set to false for repeatable builds

test:
    [javac] C:\Program Files (x86)\Jenkins\workspace\korat\build.xml:81: warning: 'includeantruntime' was not set, defaulting to
build.sysclasspath=last; set to false for repeatable builds
    [junit] Running korat.exploration.BinaryTreeExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.277 sec
    [junit] Running korat.exploration.BinomialHeapExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.911 sec
    [junit] Running korat.exploration.DisjSetExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.223 sec
    [junit] Running korat.exploration.FSMExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.224 sec
    [junit] Running korat.exploration.FibonacciHeapExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.591 sec
    [junit] Running korat.exploration.HeapArrayExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.555 sec
    [junit] Running korat.exploration.SortedListExplorationTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 8.121 sec

BUILD SUCCESSFUL
Total time: 13 seconds
Finished: SUCCESS
```

Figure 5.7: Jenkins Output From JUnit Run

```
File:  TEST-korat.exploration.FSMExplorationTest.txt

Testsuite: korat.exploration.FSMExplorationTest
Tests run: 1, Failures:  0, Errors: 0, Skipped:  0, Time elapsed: 0.229 sec
------------- Standard Output ---------------

Start of Korat Execution  for korat.examples.fsm.FSM  (repOK, [3])

FSM state reset probability:  0.0 %

(NO COINS, add2coins,  TWO COINS) ****
(ONE COIN, add2coins,  ONE COIN)
(TWO COINS, add2coins,  TWO COINS)
Total explored:  3
New & Valid  found:  1

State coverage:
TWO COINS
covered: 1 / 3 (33 %)

Trigger coverage:
add2coins
covered: 1 / 3 (33 %)

Transition  coverage:
NO COINS:add2coins:TWO  COINS
covered: 1

Transition  pair coverage:
covered: 0

End of Korat Execution
Overall time: 0.172 s.
------------- ---------------- ---------------

Testcase: testFSM took  0.226 sec
```

Figure 5.8: JUnit Test Report

5.2.2 Manual Tests

Our approach for the functions done purely from the graphical user interface is to test manually. These include FSM random reset, Swing UI, JUNG graph network, and Json functions. There is scope to automate most of these which can be considered for future work.

5.2.3 Comparison Tests

A sample run of original Korat and the enhanced Korat-2015 on a binary tree exploration is shown in Figure 5.9. The results are comparable, with Korat-2015 taking a slightly longer execution time for the sample run.

```
C:\Users\dzhvn0\Desktop\kk>"C:\Program Files (x86)\Java\jdk1.7.0_75"\bin\java -cp
korat-1.0.jar;.\lib\*;. korat.Korat --class korat.examples.binarytree.BinaryTree --print -
-args 3,3,3

Start of Korat Execution for korat.examples.binarytree.BinaryTree (repOK, [3, 3, 3])

korat.examples.binarytree.BinaryTree@1b7a553
korat.examples.binarytree.BinaryTree@1b7a553
korat.examples.binarytree.BinaryTree@1b7a553
korat.examples.binarytree.BinaryTree@1b7a553
korat.examples.binarytree.BinaryTree@1b7a553
Total explored:63
New found:5

End of Korat Execution
Overall time: 0.133 s.

C:\Users\dzhvn0\Desktop\kk>
```

Figure 5.9: Korat and Korat-2015 Comparison Run

```
C:\Users\dzhvn0\Desktop\kk>"C:\Program Files (x86)\Java\jdk1.7.0_75"\bin\java -cp
korat2015.jar;.\lib\*;. korat.Korat --class korat.examples.binarytree.BinaryTree --print
--args 3,3,3

Start of Korat Execution for korat.examples.binarytree.BinaryTree (repOK, [3, 3, 3])

korat.examples.binarytree.BinaryTree@10e2558
korat.examples.binarytree.BinaryTree@10e2558
korat.examples.binarytree.BinaryTree@10e2558
korat.examples.binarytree.BinaryTree@10e2558
korat.examples.binarytree.BinaryTree@10e2558
Total explored:63
New found:5

End of Korat Execution
Overall time: 0.135 s.

C:\Users\dzhvn0\Desktop\kk>
```

Figure 5.9 (continued): Korat and Korat-2015 Comparison Run

The graph outputs from the runs on a FSM and non-FSM target classes are shown in Figure 5.10.

```
Non-FSM (left)
Command:
java -cp korat2015.jar;.\lib\*;. korat.Korat –gui
Arguments:
 --class = korat.examples.binarytree.BinaryTree
--args = 3,3,3

FSM (right)
Command:
java -noverify -cp korat2015.jar;.\gumball;.\lib\*;. korat.Korat --gui
Arguments:
--class = gumball.GumBall
--args = 10
```

Figure 5.10: FSM and non-FSM Graphs

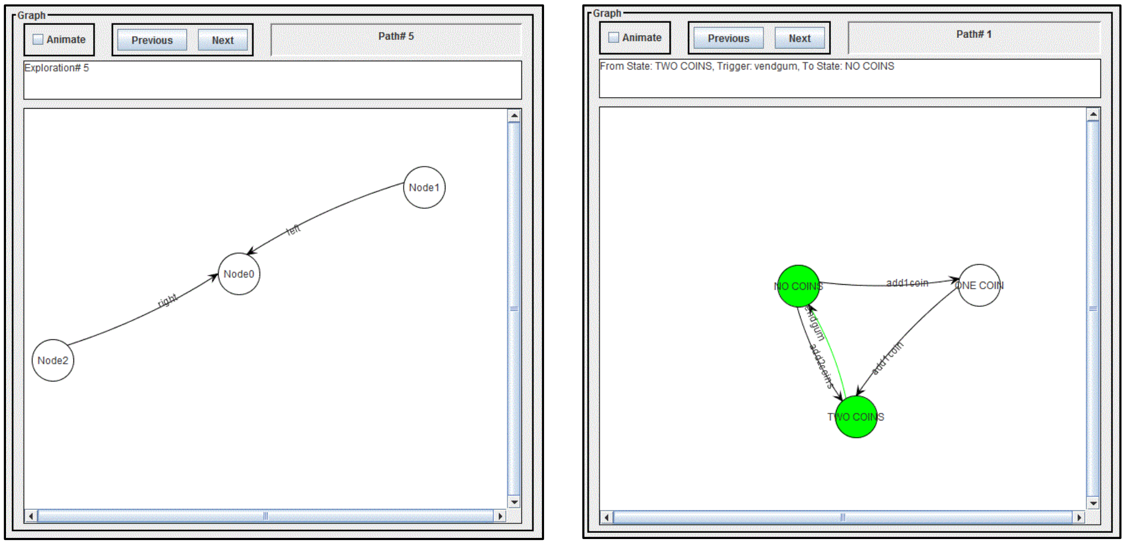


Figure 5.10 (continued): FSM and non-FSM Graphs

Chapter 6 Downloading, Installing and Using Korat-2015

This section provides the instructions to build, install and use Korat-2015. The information is also available at SourceForge [8] as described later in this section.

6.1 DOWNLOADING

There are multiple ways to download the Korat-2015 application:

- Download Korat-2015 bundle. This bundle contains the libraries including those for generating graphs. If you instead need to use GraphViz for graphs, it needs to be downloaded and installed separately.
- Download Korat-2015 lite. This is just the core Korat, and does not contain the libraries. The following libraries need to be downloaded separately: alloy4viz.jar, collections-generic-4.01.jar, colt-1.2.0.jar, commons-cli-1.0.jar, concurrent-1.3.4.jar, gson-2.2.4.jar, javassist.jar, jung-algorithms-2.0.1.jar, jung-api-2.0.1.jar, jung-graph-impl-2.0.1.jar, jung-visualization-2.0.1.jar, and junit.jar.
- Download source files and build. The deployed code can be used to build the korat2015.jar file, or a custom korat2015.jar file.

The code, deployment jar files and support information are hosted on SourceForge at http://korat2015.sourceforge.net

6.2 INSTALLING

The current directory/ folder that Korat-2015 runs, contains the korat2015.jar file. The korat2015.jar file can be obtained as described in the above section. The current directory contains two sub-directories, <target> and lib. The <target> directory contains the <target>.jar file. The lib sub-directory contains the library files.

Figure 6.1 explains the directory structure on Windows. The structure on Linux would be the same.

Figure 6.1: Directory Structure

```
C:\Users\dzhvn0\Desktop\kk>dir gumball
 Volume in drive C is OSVol
 Volume Serial Number is 666B-D925
 Directory of C:\Users\dzhvn0\Desktop\kk\gumball
05/28/2015  10:41  AM   <DIR>         .
05/28/2015  10:41  AM   <DIR>         ..
06/15/2015  04:44  PM          1,141  GumBall.class
05/29/2015  08:00  PM            838 GumBall.java
05/17/2015  09:27  AM   <DIR>         viz_json
            2 File(s)          1,979  bytes
            3 Dir(s)  382,914,117,632  bytes free

C:\Users\dzhvn0\Desktop\kk>
```

Figure 6.1 (continued): Directory Structure

```
C:\Users\dzhvn0\Desktop\kk>dir  korat2015.jar
 Volume in drive C is OSVol
 Volume Serial Number is 666B-D925

 Directory of C:\Users\dzhvn0\Desktop\kk
06/15/2015  01:02 PM          364,791 korat2015.jar
               1 File(s)        364,791 bytes
               0 Dir(s)  382,914,117,632 bytes free

C:\Users\dzhvn0\Desktop\kk>dir  lib
 Volume in drive C is OSVol
 Volume Serial Number is 666B-D925

 Directory of C:\Users\dzhvn0\Desktop\kk\lib
06/15/2015  07:07 PM    <DIR>          .
06/15/2015  07:07 PM    <DIR>          ..
06/15/2015  08:41 AM           716,757 alloy4viz.jar
06/15/2015  08:41 AM           531,557 collections-generic-4.01.jar
06/15/2015  08:41 AM           581,945 colt-1.2.0.jar
06/15/2015  08:41 AM            30,117 commons-cli-1.0.jar
06/15/2015  08:41 AM           189,284 concurrent-1.3.4.jar
06/15/2015  08:41 AM           190,418 gson-2.2.4.jar
06/15/2015  08:41 AM           471,005 javassist.jar
06/15/2015  08:41 AM           233,113 jung-algorithms-2.0.1.jar
06/15/2015  08:41 AM            40,975 jung-api-2.0.1.jar
06/15/2015  08:41 AM            62,329 jung-graph-impl-2.0.1.jar
06/15/2015  08:41 AM           324,398 jung-visualization-2.0.1.jar
06/15/2015  08:41 AM           118,808 junit.jar
              12 File(s)      3,490,706 bytes
               2 Dir(s)  382,914,117,632 bytes free

C:\Users\dzhvn0\Desktop\kk>
```

Figure 6.1 (continued): Directory Structure

6.3 Creating Target Class

The target class can be created by downloading Gumball.java, and compiling the target gumball.GumBall class as shown in Figure. 6.2.

```
Compiling target class:

Windows
javac -Xlint:deprecation -cp .;.\korat2015.jar .\gumball\GumBall.java

Linux
javac -Xlint:deprecation -cp .:./korat2015.jar ./gumball/GumBall.java
```

Figure 6.2: Target Class Compilation

6.4 USING KORAT-2015

This section describes how to use Korat-2015 on a couple of popular platforms. One method to run Korat-2015 is using command line interface, and providing the application arguments with the command. The other method is to use the new graphical user interface. In this case, the application argument "--gui" is the only application argument provided on the command line. The other application arguments are entered on the Run screen.

6.4.1 Running on Linux

Figure 6.3 shows the command to run Korat-2015 on Linux.

```
Using Graphical user interface:

sbhaskar17@bhaskar:~$ java -noverify -cp ./korat2015.jar:./lib/*:./gumball:.
korat.Korat --gui
```

Figure 6.3: Running on Linux

```
Using Command line interface:

sbhaskar17@bhaskar:~$ java -noverify -cp ./korat2015.jar:./lib/*:./gumball:.
korat.Korat -c gumball.GumBall -a 3

Start of Korat Execution for gumball.GumBall (repOK, [3])

FSM state reset probability: 0.0 %

(NO COINS, add1coin, ONE COIN) ****
(ONE COIN, add1coin, TWO COINS) ****
(TWO COINS, add1coin, TWO COINS)
Total explored: 3
New & Valid found: 2

State coverage:
TWO COINS
ONE COIN
covered: 2 / 3 (66 %)

Trigger coverage:
add1coin
covered: 1 / 3 (33 %)

Transition coverage:
NO COINS:add1coin:ONE COIN
ONE COIN:add1coin:TWO COINS
covered: 2

Transition pair coverage:
NO COINS:TWO COINS
covered: 1

End of Korat Execution
Overall time: 0.568 s.
sbhaskar17@bhaskar:~$
```

Figure 6.3 (continued): Running on Linux

6.4.2 Running on Windows

Figure 6.4 shows the command to run Korat-2015 on Windows.

```
Using Command line interface:

C:\Users\dzhvn0\Desktop\kk> java -noverify -cp korat2015.jar;.\gumball;.\lib\*;.
korat.Korat -c gumball.GumBall -a 3

Start of Korat Execution for gumball.GumBall (repOK, [3])

FSM state reset probability: 0.0 %

(NO COINS, add1coin, ONE COIN) ****
(ONE COIN, add1coin, TWO COINS) ****
(TWO COINS, add1coin, TWO COINS)
Total explored: 3
New & Valid found: 2

State coverage:
TWO COINS
ONE COIN
covered: 2 / 3 (66 %)

Trigger coverage:
add1coin
covered: 1 / 3 (33 %)

Transition coverage:
NO COINS:add1coin:ONE COIN
ONE COIN:add1coin:TWO COINS
covered: 2

Transition pair coverage:
NO COINS:TWO COINS
covered: 1

End of Korat Execution
Overall time: 0.213 s.
```

Figure 6.4: Running on Windows

Using Graphical user interface:

```
C:\Users\dzhvn0\Desktop\kk> java -noverify -cp korat2015.jar;.\gumball;.\lib\*;.
korat.Korat --gui
```

Figure 6.4 (continued): Running on Windows

Chapter 7 Traceability Matrix

The below Table 7.1 maps requirements, design and verification & validation sections in this document.

Requirement	Design	V&V	Requirement	Design	V&V
GUI-RQ-1	6.3	5.2.2	GUI-RQ-18	3.4.2	5.2.2
GUI-RQ-2	6.3	5.2.1	GUI-RQ-19	3.4.2	5.2.2
GUI-RQ-3	3.4.1	5.2.2	GUI-RQ-20	3.4.3	5.2.2
GUI-RQ-4	3.4.1	5.2.2	GUI-RQ-21	3.6	5.2.2
GUI-RQ-5	3.4.1	5.2.2	GUI-RQ-22	3.4.3	5.2.2
GUI-RQ-6	3.4.1	5.2.2	GUI-RQ-23	3.4.3	5.2.2
GUI-RQ-7	3.4.1	5.2.2	GUI-RQ-24	3.4.3	5.2.2
GUI-RQ-8	3.3	5.2.2	GUI-RQ-25	3.4.3	5.2.2
GUI-RQ-9	3.4.1	5.2.2	GUI-RQ-26	3.4.3	5.2.2
GUI-RQ-10	3.4.1	5.2.2	GUI-RQ-27	3.4.3	5.2.2
GUI-RQ-11	3.4.1	5.2.2	GUI-RQ-28	3.4.3	5.2.2
GUI-RQ-12	3.4.2	5.2.2	GUI-RQ-29	3.4.3	5.2.2
GUI-RQ-13	3.4.2	5.2.2	GUI-RQ-30	3.4.3	5.2.2
GUI-RQ-14	3.4.2	5.2.2	GUI-RQ-31	3.4.3	5.2.2
GUI-RQ-15	3.4.2	5.2.2	GUI-RQ-32	3.4.3	5.2.2
GUI-RQ-16	3.4.2	5.2.2	GUI-RQ-33	3.4.3	5.2.2
GUI-RQ-17	3.4.2	5.2.2	GUI-RQ-34	3.4.3	5.2.2

Table 7.1: Traceability Matrix

Requirement	Design	V&V
GUI-RQ-35	3.4.3	5.2.2
GUI-RQ-36	3.4.3	5.2.2
GUI-RQ-37	3.4.3	5.2.2
GUI-RQ-38	3.4.3	5.2.2
GUI-RQ-39	3.4.4	5.2.2
GUI-RQ-40	3.4.4	5.2.2
FSM-RQ-1	3.5.1	5.2.1
FSM-RQ-2	3.5.1	5.2.1
FSM-RQ-3	3.5.1	5.2.1
FSM-RQ-4	3.5.1	5.2.1
FSM-RQ-5	3.5.1	5.2.1
FSM-RQ-6	3.5.1	5.2.1
FSM-RQ-7	3.5.1	5.2.1
FSM-RQ-8	3.6	5.2.1
FSM-RQ-9	3.6	5.2.1
FSM-RQ-10	3.4.3	5.2.1
FSM-RQ-11	3.4.3	5.2.1
FSM-RQ-12	3.5.2	5.2.1
FSM-RQ-13	3.5.2	5.2.1
FSM-RQ-14	3.5.2	5.2.1
FSM-RQ-15	3.5.2	5.2.1

Requirement	Design	V&V
FSM-RQ-16	3.5.1	5.2.2
FSM-RQ-17	N/A	5.2.1
JUNG-RQ-1	3.7	5.2.2
JUNG-RQ-2	3.6	5.2.2
JUNG-RQ-3	3.6	5.2.2
JUNG-RQ-4	3.6	5.2.2
JUNG-RQ-5	3.6	5.2.2
JUNG-RQ-6	3.6	5.2.2
JUNG-RQ-7	3.6	5.2.2
JUNG-RQ-8	3.6	5.2.2
JUNG-RQ-9	3.7	5.2.2
JUNG-RQ-10	3.4.3	5.2.2
JUNG-RQ-11	3.6	5.2.2
JUNG-RQ-12	3.6	5.2.2
JSON-RQ-1	3.7	5.2.2
JSON-RQ-2	3.7	5.2.2
JSON-RQ-3	3.7	5.2.2
JSON-RQ-4	3.7	5.2.2
JSON-RQ-5	3.4.1	5.2.2
JSON-RQ-6	3.4.1	5.2.2
JSON-RQ-7	3.7	5.2.2

Table 7.1 (continued): Traceability Matrix

Requirement	Design	V&V
JSON-RQ-8	3.7	5.2.2
JSON-RQ-9	3.4.1	5.2.2
JSON-RQ-10	3.4.3	5.2.2
NF-RQ-1	ALL	5.2.2
NF-RQ-2	ALL	5.2.2
NF-RQ-3	ALL	5.2.2
NF-RQ-4	ALL	5.2.2
NF-RQ-5	ALL	5.2.1

Table 7.1 (continued): Traceability Matrix

Chapter 8 Conclusion

This report presented our work on enhancing the usability and applicability of Korat, which is a tool for automated specification-based testing of Java programs. Our enhancements address the following elements: Graphical User Interface (GUI), Java Universal Network/Graph Framework (JUNG) output, Finite State Machine Domain (FSM), and JavaScript Object Notation (JSON) graph archival. We hope our work provides a foundation that enables more developers and testers to benefit from automated test input generation offered by Korat.

References

1. A. Milicevic, S. Misailovic, D. Marinov, and S. Khurshid. Korat: A Tool for Generating Structurally Complex Test Inputs. Formal Research Demo at the 29th International Conference on Software Engineering (ICSE Demo 2007), Minneapolis, MN, May 2007.

2. C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on Java predicates. International Symposium on Software Testing and Analysis (ISSTA 2002), pages 123-133, Rome, Italy, July 2002.

3. Aleksandar Milicevic, Sasa Misailovic, Darko Marinov, Sarfraz Khurshid. Java implementation of korat http://korat.sourceforge.net

4. S. Misailovic, A. Milicevic, N. Petrovic, S. Khurshid, and D. Marinov. Parallel test generation and execution with Korat. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007), Dubrovnik, Croatia, Sept. 2007.

5. Kilnagar S. Bhaskar. Korat-2015 http://korat2015.sourceforge.net

6. Innoslate https://www.innoslate.com/

7. GitHub https://github.com/

8. SourceForge http://sourceforge.net/

9. JSON http://json.org/

10. JUNG http://sourceforge.net/projects/jung/

11. Java Swing http://docs.oracle.com/javase/tutorial/uiswing/start/

12. Google GSON https://sites.google.com/site/gson/

13. JUnit http://junit.org