5-3-2008

Using Rule-based Structure to Evaluate Rule-based System Testing Completeness: A Case Study of Loci and Quick Test

Stephen Charles Medders

USING RULE-BASED STRUCTURE TO EVALUATE RULE-BASED SYSTEM TESTING COMPLETENESS: A CASE STUDY OF LOCI AND QUICK TEST

By

Stephen Charles Medders

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2008

USING RULE-BASED STRUCTURE TO EVALUATE RULE-BASED SYSTEM TESTING COMPLETENESS: A CASE STUDY OF LOCI AND QUICK TEST

By

Stephen Charles Medders

Approved:

Edward B. Allen Associate Professor of Computer Science and Engineering (Major Professor and Graduate Coordinator)	Edward A. Luke Associate Professor of Computer Science and Engineering (Committee Member)
Jeffrey C. Carver Assistant Professor of Computer Science and Engineering (Committee Member)	Roger King Associate Dean for Research and Graduate Studies of the Bagley College of Engineering

Name: Stephen Charles Medders

Date of Degree: May 02, 2008

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Edward Allen

Title of Study: USING RULE-BASED STRUCTURE TO EVALUATE RULE-BASED SYSTEM TESTING COMPLETENESS: A CASE STUDY OF LOCI AND QUICK TEST

Pages in Study: 62

Candidate for Degree of Master of Science

Rule-based systems are tested by developing a set of inputs which will produce already known outputs. The problem with this form of testing is that the system code is not considered when generating test cases. This makes software testing completeness difficult to measure. This is important because all the computational models are constructed within the code. Therefore, to show the models of the system are tested, it must be shown that the code is tested.

Chem uses the Loci rule-based application framework to build computational fluid dynamics models. These models are tested using the Quick Test suite. The data flow structure built by Loci, along with Quick Test, provided a case study for the research. The test suite was compared against three levels of coverage. The measures indicated that

the lowest level of coverage was not achieved. This shows us that structural coverage measures can be utilized to measure rule-based system testing completeness.

Key words: rule-based, structural, testing, Loci, scientific computing, software engineering

DEDICATION

I dedicate this thesis to my wife, Lauren, for her constant encouragement and help throughout my graduate career.

ACKNOWLEDGMENTS

This work was supported in part by grant 0132673 from the National Science Foundation. The findings and opinions in this thesis belong solely to the author, and are not necessarily those of the sponsor.

I thank my committee for their comments on this thesis, Dr. Edward Allen for hiring me and for his advice, and the Empirical Software Engineering group at the Computer Science and Engineering Department, MSU for their helpful discussion. Dr. Luke provided data from Quick Test and Loci for the case study as well as useful analysis of case study results.

I would also like to thank my wife Lauren for being my providing help in reading through the work and correcting what I write.

All graphs were drawn using yEd Java Graph Editor [20].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Rule-based systems build computing environments by specifying sets of rules which are executed, or fired, upon receiving data that matches a particular template. These types of systems are commonly found in artificial intelligence, but are also used in scientific computing to define physics models [13]. One issue that rule-based systems have is that rarely are their programs tested in regards to the program's code structure [5]. Some more recent testing methods use rule-based structures, but not for the purpose of identifying coverage. They instead are used for design [1], requirements verification [10], and identifying rule inconsistencies such as contradictions and infinite loops [19].

1.1 Problem Specification

Rule-based systems define their computations and logic through interactions of rules and data. These rules work together resulting in models representing complex computations and logical decisions. These systems are commonly tested by using a set of test cases with known outcomes [2]. The problem is that this testing method ignores the code and code structure of the rule-based system. Models represent the numerical computations and decisional logic which rule-based systems are intended to perform. Since these models are built in the code, the models themselves need to be tested directly. Therefore,

testing simple functional requirements based on predefined scenarios is inadequate to provide assurance in the testing of a rule-based system. The code and structures produced by relations of elements found within the code must be considered in test case generation in order to provide assurance that the models of the rule-based system have been tested.

1.2 Hypothesis

Rule-based systems produce structures in the relations between the rules. These relations, defined by data and control flow, can become the basis for structural testing. They can be used in structural testing to support rule-based systems by indicating, quantitatively, the coverage over the rule-based system code that is achieved during testing. Therefore, the hypothesis of this research can be stated as follows:

> Rule-based systems can benefit from structural testing techniques. This is because structural testing uses system code and structures formulated within the code to generate test cases and to quantitatively show coverage of the system by a test suite, ultimately providing assurance that the models of the rule-based system are tested. The relations between the rules defined by the flow of data from one rule to another can be used to define the structure to use in structural testing.

In other words, structural testing can provide increased assurance in the testing of rule-based systems. This benefit results from the fact that structural testing generates test cases and measures completeness of a test suite based on the code of the system. With models for rule-based systems constructed from code, structural testing provides the means to explicitly show how the these models are tested, rather than just functional requirements.

1.3 Case Study

Loci, developed at Mississippi State University, uses Datalog-like syntax to build computational fluid dynamics models [13]. Chem is a computational fluid dynamics software package which uses Loci as its back end [11]. Currently, Chem is black box tested through a regression test suite known as Quick Test.

Loci generates a rule-based structure by mapping the output of rules to the input of other rules. This structure is currently used for scheduling concurrent execution of these rules in order to perform complex calculations within a reasonable time. Quick Test tests the models against a set of predefined inputs, or test cases. A test case is considered successful when the output is within a set range of acceptable outputs. Whenever a change is made to the C++ back-end code for Loci, or when the models for Chem are changed, Quick Test is run to ensure that these changes did not break the system.

The Loci structure can be used to evaluate the completeness of Quick Test by examining the execution of the test suite with respect to coverage of the rule-based structure. Measurements indicating the level of coverage reached can provide criteria for defining the quality of the test suite and for proposing new test cases in order to improve coverage. This case study will attempt to measure the quantity of coverage Quick Test achieves over the Loci structures representing the physics models of the Chem software package.

1.4 Research

The research leading to this thesis was composed of four parts. A literature review was done in order to understand prior work done in this field. Then, a graph analysis tool was

built in order to facilitate completing the case study. Next, Quick Test was analyzed by the graph analysis tool as a case study. Finally, the measurements from the graph analysis tool were reviewed to produce a conclusion. These steps worked together toward the goal of answering the following questions:

1. Can relations between rules in a rule-based system provide a structure to use in structural testing techniques for rule-based systems?
2. Can structural coverage measurements be used to evaluate the effectiveness of a rule-based system test suite?
3. What level of coverage is achieved by test cases in Quick Test on the rule-based system structure generated by Loci for Chem?
4. What portions of the rule-base structure for the models of Chem, if any, are currently untested by Quick Test?

The first step in the research was an analysis of different rule-based structures already proposed. This analysis was done via a systematic literature review (see Appendix A). The literature review revealed two main types of structures. Antecedent-consequence rules relate the variables and facts in a rule-based system to the rules [2, 7, 10, 13, 17, 19]. Rule-to-rule structures relate rules to each other, abstracting away the data [5, 9]. Our case study was on Chem, which uses the Loci application framework. Loci uses an antecedent-consequence rule structure in its processing. The Loci structures gives us a rule-based structure to evaluate the completeness of the test suite for Chem.

A software tool was needed to take the graphs from the case study, along with the test cases, and simulate program flow to calculate coverage. This tool was built using the Gravisto library. The tool measures three metrics based on those presented by Barr [2]. The Classes measurement showed the ratio of class nodes in the structure which are covered by testing. Second, the Finding-Class Pairs metric indicated the amount of coverage

for paths between finding nodes and class nodes. Third, the Edges metric provided a more detailed measurement of path coverage by showing the percentage of edges covered by the test suite.

Finally, the test cases in Quick Test — the test suite for Chem — were analyzed against the structures of the rule-based models found in Chem. The structures built by the Loci framework for Chem were input to the tool along with the test case inputs. These measurements were then analyzed to determine whether or not the structural analysis provides support for the Quick Test suite.

CHAPTER 2

RELATED WORK

In order to identify prior work on rule-based systems (RBS) and structural testing, a systematic literature review was performed. This review was based on a template provided by Mian et al. [15]. This type of review is intended to provide repeatability of the literature review, in order to help future researchers find the same sources used and identify sources that are new since the review was completed. This chapter presents the results of the review. For the review methods and statistics, please see appendix A.

2.1 Rule-Based Structures

The first question asked in this research is whether or not relations from rule to rule can be used to build a structure for structural testing. This section presents the results of a review identifying prior work specifying relations within a rule-based system which build structures. All the structures reviewed are divided into two types. One type of structure specifies antecedents and consequences to rules. The other structure defines abstracted relations between rules.

2.1.1 Antecedent-Consequence Structures

These structures explicitly define facts or variables which act as inputs — the antecedents — of rules. They also define the facts or variables which act as the output, or decisions, of these rules. The output is known as the consequence. These structures were the most commonly found and present the most detail of the system when compared to the rule-based structures discussed later in this section. The relations in this type of structure connect data in the system to rules in the system. Relations between rules can be identified in this structure by showing output decisions or data of one rule acting as input of another.

The first structure of this type to be discussed is the Petri-net [7, 10, 17, 19]. Petri-nets have two entities. Places, which represent facts and variables, can be inputs or outputs of transitions, which represent the rules. Edges show flow through the rule base and can be marked to imply negation of the antecedent or consequence. Figure 2.1 shows a simple rule $a, b \rightarrow c$. Petri-Nets represent a rule-based system starting from inputs and ending in the computed values or decisions inferred by the rule base. Structural testing techniques could start by marking variables in the system provided by inputs, then traversing the graph by identifying which rules are capable of being fired and marking these rules as fired and their outputs and edges. This structure would require knowledge of the logic within a rule to generate test cases as well as analyze completeness.

Lee et al. [10] proposes a different type of Petri-net used for task verification. This structure defines a transition as a task which can be composed of multiple sub-tasks, each composed of multiple rules. These rules are also represented by transitions, which result in multiple Petri-nets of varying levels of abstraction. This structure at its lowest level

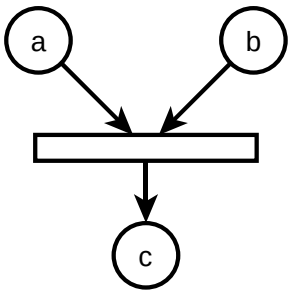

Figure 2.1

Petri-Net for Rule-Based Systems

of abstraction is like other Petri-Nets and has the same benefits and down falls as other Petri-Nets.

Wu and Lee [18] propose a token flow system for verifying rule-based systems. This system is similar to Petri-nets, but instead of places and transitions, it uses P-Nodes and R-Nodes, which represent the same thing. The added entity that makes this structure useful is the sink node that represents goals in the system. The token flow structure represents a rule-based system much the same way that a Petri-Net does. Token flows do not use marks on edges to indicate value negation, but it does use a sink node to indicate the end of the structure. This structure supports coverage analysis by providing a single end from which to analyze path coverage with.

Argwal and Tanniru [1] propose the parameter dependency network. Similar to the structure proposed by Lee et al. [10], this structure supports several types of abstraction which can actually convert the graph into a rule-based graph. This structure would provide

benefit in hierarchically building a structural testing suite. Paths to end rules can first be found, building up to identifying different sets of inputs covering multiple data flow paths.

Barr [2] discusses the TRUBAC structure, which is the most detailed type of structure. This structure does not just abstract a rule to one entity, it contains a separate entity for each boolean relationship between facts. This structure gives significant detail about a rule-based structure and could even be used to give the greatest amount of path analysis by showing logical paths within the rules themselves.

Ramaswamy et al. [16] present a hypergraph where nodes represent facts, compound hypernodes represent groups of nodes required for a rule, and edges represent rules. This type of structure presents a method of performing path analysis through the structure without having to know the intricacies of the logic within the rules. This would lead to the ability to have testers which do not necessarily have to understand the detailed logic of each rule in order to traverse the graph from test case inputs to final outputs.

Zhang and Luke [21] present a structure used by the Loci high-performance computing package for scheduling parallel processes. Because this system is the case study for this research, this structure was the most interesting. Figure 2.2 shows an example from the same rule as Figure 2.1. The difference between the Loci structure and regular Petri-nets lies in the semantics and restrictions of the structure. This structure is a directed graph with alternating entities. The first entity, represented by the circles, are the facts and variables. The second entity, represented by the rectangles, are the rules. The edges show both data and control flow.

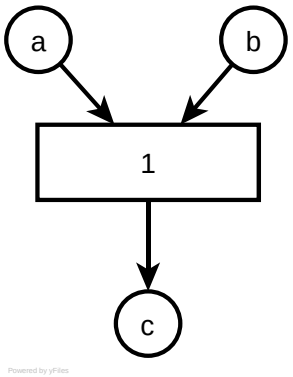

Figure 2.2

DAG for Loci

2.1.2 Rule-Based

Rule-based structures represent rule-based systems by graphically showing the relations between rules while abstracting away the data of the system. Each structure is built on a set of relations. These relations represent how one rule can be fired as a result of another rule being fired. These types of structures can benefit rule-based system structural testing by representing control-flow paths through the system. By identifying which rules are fired, testers can identify paths which have been exercised by the test suite. This would result in a quantifiable means of determining completeness of a rule-based system test suite.

Grossner et al. [5] presented a structure via a set of set-theory based relations between rules. In this structure, only one type of entity exists, and this entity represents rules. The only one-to-one relation is the depends-upon relation that states if rule B depends on rule A, then the right hand side of rule A asserts a set of facts that match the template in the left hand side of rule B. The next two relations are one-to-many. Rule A is "reachable"

from a rule set W if A depends-upon each rule in set W. This is called the "reachability" relation. Also, rule set V "enables" rule A if all the rules in set V are required to assert the facts in the left hand side of rule A. This relation is called "enablement." Using these relations, a developer can identify control paths from rule to rule. These paths can then be used to generate test cases to maximize coverage.

Kiper [9] proposes the logical path graph which represents the logical paths that can be traversed from one rule to another. Entities for this graph — nodes — represent rules. The edges represent a depends-upon relation that is the same as proposed by Grossner et al. [5]. When incoming edges to a node are connected by an arc, then all the source nodes for these edges are required for flow to the target node. This relation is the same as the "enablement" relation [5] described above. Figure 2.3 shows an example taken from Kiper's work. This graphical representation of the rule-to-rule relations gives testers the ability visually trace paths and generate test cases intended to maximize coverage.

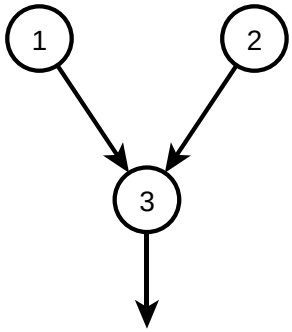

Figure 2.3

A Logical Path Graph

2.2 Coverage Metrics

Coverage metrics are needed to quantify the amount of a rule-based system which is exercised by a test suite. During the review, two types of coverage metrics studies were reviewed. Traditional coverage metrics are used to measure software built using imperative programming languages. These metrics are used to analyze coverage of a program based on control flow through the sequential list of instructions. Rule-based coverage metrics seek to quantify coverage over the set of rules defined in the code.

2.2.1 Traditional Coverage Metrics

Jorgensen [8] presents several aspects on software testing, including traditional coverage based on DD-Paths. A DD-Path — *decision-to-decision path* — is a structure of nodes and edges where each node is a decision statement in a program and each edge is a path from one decision to the next. These measurements are pass-fail measurements indicating that the testing has either achieved the particular level of coverage or it has not. They are considered hierarchical in nature in that if one level of coverage is reached, previous levels are also considered to have been reached. These different levels are presented in Table 2.1 from Jorgensen [8].

2.2.2 Rule-Based Coverage

Only one study was found discussing coverage metrics for rule-based systems [2]. Like traditional coverage analysis, the levels of coverage discussed by Barr [2] represent hierarchical levels of coverage, making a sequential set of goals to attain through testing.

Table 2.1

Structural Testing Coverage Metrics

Metric	Description of Coverage
C_0	Each statement in source executed
C_1	Every DD-Path (predicate outcome)
C_{1p}	Every predicate to each outcome
C_2	C_1 coverage + loop coverage
C_d	C_1 coverage + Every Dependent pair of DD-Paths
C_{MCC}	Multiple condition coverage
C_{ik}	Every program path that contains up to k repetitions for a loop (usually k = 1)
C_{stat}	"Statistically Significant" fraction of paths
C_∞	All possible execution paths

These levels of coverage start by focusing on classes, which are goals to be achieved by running the rule-based system. The following levels of coverage seek to cover more paths leading to these classes. The levels of coverage are enumerated in the following list starting with the lowest level of coverage and leading to the highest [2].

1. *Each-Class:* A class is a sink node in the graph and represents a goal, which is a calculation or decision requested by the user. This measure states that there is at least one path to each class in the rule-base that is traversed by the test suite.

2. *Each-hypoth:* The test suite covers paths that traverse each sub-class leading to each class.

3. *Each-class-every-sub:* For each sub-class/class combination, the test suite covers at least one path containing the two.

4. *Each-class-every-finding:* A finding is a fact that is established during the execution of the rule-based toward a specific class. This measure states that for each finding/class pair which has an execution path between, the test suite covers at least one path connecting the two.

5. *All-edges:* This is the highest level of feasible coverage. All edges in the graph are covered by the test suite.

6. *All-paths:* This measure states that every possible path in the structure or graph is traversed by the test suite. Because there are too many paths in a real-world system to test in a reasonable amount of time, reaching this level is infeasible.

2.3 Recommendations

First, this review shows that few studies have been done toward structural testing of rule-based systems (see section A.5). While several structures were found, these structures were developed with varying intents including proving completeness, finding conflicting rules, and initial design. Only one of these structures was designed with the intent of structural testing [2]. All together, these results indicate that rule-based systems do form a structure which should be considered in developing test cases.

Second, of the studies found, few were done on sizeable case studies. Instead, the most common approach was to build a small "toy" project and experiment with it. While these toy projects can show that the concept has some merit, it does not show its validity in real world situations. Case studies would do more to show how useful these rule-based structures are for real world systems.

Finally, to truly evaluate the completeness of a test suite, coverage metrics need to be used. While one study was found to propose a set of metrics, no coverage metrics were found over the code itself. Code coverage includes goals such as covering all executable lines of code and covering iterations through loops. Structural coverage metrics abstract the code into entities of a structure and seek to maximize the paths covered through this structure. While structural coverage is important, code coverage can be meaningful as well. However, the structural metrics proposed are useful for our intended research, and

can be used on the structures already in use by the Loci system. Therefore those metrics will be included in the evaluation of the test suite currently in use for the Loci system.

2.4 Other Literature Reviewed

Other literature was reviewed toward the understanding of software coverage metrics. These were not included in the review because they did not include structures or metrics for rule-based systems, or the software quality aspects were not focused on internal structure.

Numerical software has its own set of quality issues. These studies were not included in the systematic review because the studies did not discuss structural analysis or testing of numerical software. However, these studies did present quality issues of numerical software systems.

Hatton [6] presents the results of what are referred to as the T-experiments, which reveal a number of errors in numerical software. The first experiment, T1, presented a static analysis, being an analysis of the code without running the program. The second experiment, T2, performed a dynamic, or runtime, analysis. Hatton concluded that all commercial numerical systems will have statically detectable errors and that current methods of designing numerical systems are insufficient.

Gropp [4] presents several issues causing problems with numerical software. He lists issues such as lack of modularity and orienting the numerical software according to the algorithm used rather than the problem to be solved. These reasons cited are also presented as the basis for building PETSc [4], a numerical software library, from the ground up rather

than using existing libraries. Gropp concludes with a set of design decisions to optimize PETSc.

CHAPTER 3

CASE STUDY

Loci is a rule-based application framework developed at Mississippi State University by Dr. Ed Luke [12]. Loci builds a graph structure from the rules defined for its framework to generate and schedule parallel execution of computations. Chem is the first application written around the Loci framework [11]. Used for computational fluid dynamics, Chem builds a large variety of physics models with the rule-based syntax of Loci. Quick Test is used to test these models with a set of inputs with known results. The structures Loci builds for the Chem rule base and the Quick Test inputs together makes the case study for this research.

3.1 The Loci System

Loci is a framework designed to reduce complexity in developing software for finite-element applications [12]. The intent of Loci is to reduce errors made in typical numerical applications caused by mistakes in control structures in code, and is accomplished by generating control structures from the rule-based syntax.

Loci uses graphs to represent databases of facts and rules which, when combined, produce transitions calculating new facts. One type of entity in this graph is a node representing values and variables in the system. Another type of entity is a node representing

rules in the system [21]. Edges are entities in the graph representing data flow through the system built with the Loci framework.

Values are mapped to the places in the structure representing facts [12]. These values are then input to the rules to calculate new facts. This flow continues from the rules into the next set of variable nodes. For each rule, all variables flowing into it must have values mapped to them or the rule will not be executed by the system. This provides a means for scheduling and parallelizing execution.

The Loci framework produces one of these graphs each time input is given to its rule-based models. The graph is first produced starting at the top (where the inputs given are in the rule database) and then pruned from the bottom up to produce a graph which defines the execution structure for that specific set of inputs. For this case study, the pre-pruned graphs were used. However, since these graphs are produced at run-time based on inputs, they never really give an overall picture of the entire system. In order to do this a master graph was produced, showing all the rules, yet ignoring iterations on looping rules.

3.2 Chem and Quick Test

Chem is a modeling program for chemical reaction flow models, or computational combustion models [11]. This program, using the Loci framework, builds several physics models. Quick Test is the test suite used for initial verification of the models built in Chem. This means that Quick Test is used to show that the equations programmed into Chem are the equations intended at that time.

Quick Test is the testing suite that was analyzed for this case study. The inputs for Quick Test were compared against the graph structure produced by the Loci framework for the Chem models. The tool created for this study took the inputs and graph structures and measured the coverage amount of each graph based on those inputs (see Chapter B).

3.3 Graph Analysis Tool

A software tool was required to analyze structure graphs for the Quick Test case study. The requirements of the tool were to accept the graphs and test inputs in files. The tool then needed to measure coverage of the graphs that resulted from traversal starting with input nodes down to class nodes.

3.3.1 Metrics

For each metric used in this case study, definitions applying to the Loci framework were needed. Then, using these definitions, the metrics were selected based on what was considered useful, both for this study, and to the developers of Loci. The three metrics selected were Each-Class, Each-class-every-finding, and All-edges. Each metric was returned by the tool as a ratio. That particular level of coverage was considered achieved if the returned value was 100%.

The use of ratios provided a comparison of individual test case measurements against measuring the coverage over the master graph for the Chem models. As will be seen in chapter 4, the master graph actually shows different coverage than the individual test cases. While neither the master graph or any of the individual test cases show any level

of coverage completed, the master graph had a higher ratio of classes covered. Also, Dr. Luke stated that many of the classes are achieved through use of experimental features. This leads to an acceptance of lower coverage due to the fact that the test cases do not need to cover experimental features.

The metrics were output by the Graph Analysis Tool in a four-column spreadsheet. The first column represented the name of each test case, while each column afterward presented the measures from the Graph Analysis Tool in order: Classes, Finding-Class Pairs, and Edges. This spreadsheet was put in a Comma Separated Variables file.

3.3.1.1 Classes Metric

In order to use the Each-Class metric, the term class had to be defined in terms of Loci. Barr [2] defined a class as a value at the end of the set of computations or the final value produced by the sequence of rules fired. In this case, a class is defined as the last set of variable nodes in the flow of execution in the graph or variable nodes that have edges flowing into them and no edges flowing out of them.

This measure was based on the Each-Class metric [2]. In the Graph Analysis Tool, a variable node is differentiated from a rule node by a true value for the `isVar` attribute for the node. The Classes metric was measured as a ratio of class nodes with `isFired` set to true, to the total number of class nodes.

$$\frac{\text{classesCalculated}}{\text{totalClasses}} \tag{3.1}$$

3.3.1.2 Findings-Class Pairs

The Findings-Class Pairs metric is derived from the Each-class-every-finding metric proposed by Barr [2]. In section 2.2, a finding is described as "a fact that is established during the execution of the rule-based system toward a specific class." In terms of the Loci graph, a finding is a variable node in the path between the initial input variable nodes and the class variable nodes having both incoming and outgoing edges. In order to measure the finding-class pair metric, each pair had to be identified. Then, a ratio was defined as the number of finding-class pairs which had a path between the two nodes of the pair traversed during program execution, divided by the total number of finding-class pairs. This level of coverage is considered complete with a measure of 1.0 from the Graph Analysis Tool.

$$\frac{\text{findingClassPairsIncluded}}{\text{totalPair}} \tag{3.2}$$

3.3.1.3 All-Edges

The final metric did not require any special definitions. The All-edges metric was modified to a ratio, as the other metrics used were modified. This ratio was the number of edges traversed by program flow divided by the total number edges in the graph.

Edges in the Graph Analysis Tool hold a boolean attribute named `isTraversed`. This attribute is set whenever a rule is fired, indicating flow down the structure graph. The All-Edges metric is defined by Barr [2] as traversal of all edges in a graph. In our tool, this metric would be defined as all edges in the graph holding a true value in its `isTraversed` attribute. This tool returns a ratio of the number of edges meeting this

criteria to the total number of edges. Again, 100% for this ratio indicates successful completion of this level of coverage.

$$\frac{\text{edgesTraversed}}{\text{totalEdges}} \tag{3.3}$$

3.3.2 Rule Lists

The rule lists were original used to debug the Graph Analysis Tool. They were kept in the tool because it aids the Loci developers by specifically showing untested rules, adding support for path analysis and test case generation. There were two types of rule lists. One list was output for each test case in Quick Test. This contained each rule not fired during testing, followed by an indented list of variables required for firing the rule but not present during testing. The second list was generated due to overlap in the test case structures. The structures for individual test cases had overlap in the rule nodes. However, a rule may be omitted by one test case but included in another. The second rule list contains all the rules which were never fired in the whole test suite, explicitly showing untested portions of the structure.

CHAPTER 4

RESULTS

There were two main results from the case study. The metrics taken (see section 3.3.1) numerically show the amount of coverage accomplished by testing in respects to the rule-based structure. The rule list augments these metrics by identifying the specific portions of the structures which remain untested. Outputs were given for each test case. This is because for Loci, a graph structure is produced when the inputs are given.

4.1 Measurements

Each test case has its own set of outputs. The metrics were output to a spreadsheet with each test case being a separate row. The rule lists for each test case were put in different text files.

4.1.1 Metrics

The metrics produced by analysis of Quick Test were put into a four column table (see chapter B). The first column in the table contains the names of each test case. The next three columns contained the metrics for each test case in order as follows: Classes Metric, Findings-Class Pairs, and Edges Metric. The metrics, each indicating an increasing order

of coverage over the previous, did not achieve 100% on any of the test cases or in the analysis of the master graph. The table with the metrics can be found in appendix C.

Table 4.1 presents a small sampling of the results. The first set has the lowest level of Classes coverage of the test cases. The second row presents about a middle level for the test cases. However, very few metrics reached this, most staying around $.47$ (47%). The last row represents the coverage over the master graph. At 77% this coverage is the highest of them all, with the 60% coverage being the closest to full coverage. This coverage is interesting because the master graph is supposed to represent the system as a whole rather than the individual test cases. The exception to this is that looping structures found in the individual test cases are removed from the master graph. The master graph could be used as a overall barometer of the test suite while the individual test cases help to guide test case generation for higher coverage.

Table 4.1

Metrics Sampling

Graph file Classes Metric	Findings-Class Pairs	Edges Metric
Inviscid_lowSpeed_TEST_shock_tubegraph.dat 0.44444445	0.35183823	0.6638935
Viscous_lowSpeed_TEST_wallLawTwallgraph.dat 0.6	0.43229812	0.69109666
masterGraph 0.7777778	0.1840504	0.4032941

The Classes metric indicates that there are classes in the structure that are not exercised by the test cases. Further analysis of the test cases can reveal which classes are not being computed by the Quick Test suite. By identifying paths in the structure which lead to each class not included by the testing, further test cases can be developed to include these classes. This will increase the test coverage to 100% in respects to the Classes Metric.

The Finding-Class Pairs metric indicates that there are finding-class pairs (defined in section 3.3.1.2) which have no path between the two invoked by the test suite. While this information is easily deduced from the fact that there are classes alone which are not covered by the test suite, what is not seen is that some classes may be covered but without some findings paired with the class. After the Classes metric reaches 100%, each finding-class pair not covered can be identified. Then, a path between the pair can be identified, allowing the developer to create new test cases to cover the previously omitted pairs.

Finally, the Edges metric indicates that there are edges in the graph structure left uncovered by the test suite. Again, this lack of coverage is easy to see when it is recognized that there are paths uncovered leading to classes and between finding-class pairs. However, with full coverage on these previous measurements, there can still be multiple paths leading to classes and between a class node and its preceding findings. After Finding-Class Pairs coverage is complete, missing edges can be identified, allowing for additional test cases which will incorporate these edges into the Quick Test suite.

4.1.2 Rule Lists

The rule list enumerates the rules which were not fired during testing. There are two types of rule lists for this case study. The first lists all the rules for each test case that did not fire. Each rule is followed by a list of all the data not present but required to fire the rule. These lists were stored in text files, one for each test case, and another for the master graph (see chapter B). Since the graphs produced were not pruned for the inputs, there is overlap in structure from graph to graph, causing some rules to fire in one graph,but not in others. For this reason, a second rule list was also created consisting of rules that were never fired in all of the test cases combined. These rule lists can be found in Medders' technical report on the graph analysis tool [14].

The metrics indicate a clear need for additional test cases. The rule lists show where the need can be found. Paths uncovered can be extracted from rules dependent on one another present in the list. As each level of coverage in the metrics is sought, the rule lists provide a map to this goal. What needs to be remembered, however, is that the two types of rule lists should be used in conjunction with one another. If a rule is exercised in one test case but not another, then it may be adequately covered.

4.1.3 Master Graph

The master graph is a compiled graph consisting of all the rules in the system. This graph is "flattened" by omitting time-based iteration data that is found in the individual test case graphs. All the inputs of each individual test case compiled produce the input for the master graph. These metrics and rule lists provide an extra set of measurements

which could be used to compare against the individual graphs and test cases. It may be considered adequate to achieve completeness on the master graph. However, this graph does not include timing and iteration data found in the details of the individual test cases. If these details are considered important, then overlap of the test cases must be considered in determining full coverage.

One of the rule lists output by the graph analysis tool (see section 3.3.2) represents the list of rules not fired in all the test cases combined. A difference between this list and the list of rules not fired during traversal of the master graph indicate that the difference in the graph affects analysis. If more rules are covered in the master graph than in the domain of the combined test cases, then an analysis of the rules not covered in the test cases would be necessary to determine why they were not exercised. Such a scenario could reveal execution paths which exist in the system but are not tested by the suite. New test cases could be generated in consideration of these overlapping portions between test cases structures. The master graph could also prove to have fewer rules covered than the combined test cases. Such a finding would show that the master graph does not represent enough of the structure for the models to provide an accurate picture of test coverage, and therefore should be not used.

CHAPTER 5

CONCLUSIONS

5.1 The Research

In the beginning of the research, a literature review presented several possible structures built on rule-based systems. Several structures were found, as seen in chapter 2. Section 2.1.1 shows structures based on the flow of data, showing values as inputs and outputs for rules. We also see control flow structures in section 2.1.2 illustrating rule-to-rule relations. Loci, the application framework for the case study, utilizes a structure of the former type in its methods of scheduling and executing the rule base [21]. This graph provides a structure for the case study.

The case study was the test suite used for the Chem application developed around the Loci framework. Known as Quick Test, this suite provides a set of inputs to the rule-based models in Chem, each producing a set of computations with known outcomes. These inputs, along with the rule-based structures from the Loci framework, offered a case study which helped to understand how structural analysis can benefit rule-based systems.

In order to perform the case study, a software tool was needed to trace the graph starting with the inputs provided. Marking the graph as it is traversed, the tool measures the amount of coverage achieved by the test cases. Three metrics are used for this. The Classes metric

gives the ratio of the number of class nodes in the graph reached by traversal to the total number of classes. The Findings-Class Pairs metric gives the percentage of finding-class pairs in the graph that have one or more paths between then exercised by the testing. Finally, the Edges metric measures the number of edges traversed divided by the total number of edges in the structure. Finally, after these metrics are calculated, all the rules not covered by testing are listed, along with the missing variables needed to fire the rule. These lists provide a way for the developers of Chem to identify parts of the model which Quick Test omits. This extra input from the developers added to the analysis of the structural testing for rule-based systems.

The final results of the research were produced from the metrics and the rule lists produced by the graph analysis tool. Each of the metrics produced failed to reach 100%. These metrics show that, in fact, the Quick Test suite does not test the entire structure for Chem, and therefore does not adequately exercise the Chem models. Also, the list of unfired rules, when analyzed by the developers, enabled them to identify the exact portions of the models which remained untested.

5.2 Developer's View

These results were presented to the primary developer of Loci, Dr. Ed Luke of Mississippi State University. He first stated that the metrics did not have a clear application to Quick Test. However, the rule lists did provide a means to identify specific portions of Chem models left untested by Quick Test. Dr. Luke pointed out that the rule lists showed that some of the untested portions consisted of experimental features not intended for end

users. Therefore, the lack of coverage on these portions of the models is acceptable. Other uncovered portions identified by the rule lists were not experimental. These specific uncovered portions did in fact show deficiencies in the test suite of which he was unaware. Dr. Luke's final assessment was that the structural analysis did provide beneficial information regarding the completeness of Quick Test and indicated what portions of the Chem model needed more focus by the test suite.

One additional comment from Dr. Luke was that the automated analysis provided by the Graph Analysis Tool is useful to his work. The structures produced for the Chem models were very large and complex. Manual analysis of these structures would have been very long and tedious, as well as error prone if done by hand. The automated process produced quick reliable results which revealed portions of the Chem models currently untested.

5.3 Conclusions

Four questions guided this research based on the hypothesis presented in section 1.2. These questions, listed in section 1.4 are as follows:

1. Can relations between rules in a rule-based system provide a structure to use in structural testing techniques for rule-based systems?
2. Can structural coverage measurements be used to evaluate the effectiveness of a rule-based system test suite?
3. What level of coverage over the rule-based system structure generated by Loci is achieved by test case in Quick Test?
4. What portions of the rule-base structure in Quick Test, if any, are currently untested?

Question 1 asks if there is a structure, relating rules to one another, which can be used in structural testing for rule-based systems. During the literature review, several possible structures for rule-based systems were found. They were originally proposed for several different purposes. Barr, however, presented using one structure, TRUBAC [2], for the purpose of structural testing.

Dr. Luke produced a rule-based high performance computing system known as Loci [12]. The Loci framework produced, as part of its internal functions, a structure representative of the variables and rules within the models. In our case study, we used the structure from Chem, built on the Loci framework, to measure coverage of Quick Test and identify untested portions of the models constructed by Chem. Analysis of these results by the Chem and Loci developer Dr. Luke indicates that this type of structural analysis does provide insight into finding untested portions of the system. This shows that rule-based systems can be represented by a structure which is useful in structural testing techniques.

The next question builds on the first by asking if metrics taken on a rule-based system structure can adequately quantify testing completeness. The literature review showed us that Barr [2] had proposed a set of sequential goals to be achieved through testing. These goals start with class values, or final computations, of the rule-based system. They increment in complexity as they seek to increase the number of possible paths to these class values which are covered by testing.

During the case study, three measurements were taken, all based on those proposed by Barr. These measures are as follows: Classes covered, Finding-Class Pairs covered, and Edges covered. These measurements all indicated that portions of the Chem rule-based

structures were in fact not tested. Analysis of the results by the Loci and Chem developer, Dr. Luke, allowed him to use the rule lists to find the specific classes and the specific rules which were lacking coverage. Therefore, it is reasonable to state that structural coverage metrics can support testing assurance in a rule-based system by showing the need, or lack thereof, for more testing for a system.

An important aspect of these results is that Dr. Luke identified some of the uncovered portions to be experimental. These sections of the models did not need testing and therefore incomplete coverage is acceptable in this scenario. This leads to the conclusion that the use of ratios for coverage metrics, rather than pure pass-fail sequential goals can be beneficial to Dr. Luke because he can determine the amount of the system consisting of experimental features and use this to establish a goal in terms of coverage ratios to achieve in order to consider that level of coverage adequate for his system.

The final two questions were intended to direct the case study. The metrics chosen (see sections 3.3.1.1 - 3.3.1.3) indicated that Quick Test had not achieved even the lowest of the proposed levels of coverage. Along with this finding, the list of rules not fired during tested indicated the specific areas not tested by Quick Test. Some of these sections were considered experimental. The developers consider it acceptable to leave these experimental features untested. Even so, some sections which need to be tested were omitted by Quick Test. The rule lists enabled Dr. Luke to identify these untested portions of the system.

One unexpected benefit from the research came from identifying some of the untested portions of the structure which consist of experimental features. Dr. Luke stated that by

identifying these features in the structure, he was able to begin to consider which parts of the models utilized these experimental features. Occasionally, in order to obtain results, these features were inserted into the model. This structural analysis helped Dr. Luke to consider which portions of the model are still using experimental features. Dr. Luke could then identify which computations are still considered experimental and not intended for the end-user.

The hypothesis for this work stated that structural testing can provide evidence that the models constructed within the rule-based system have been adequately tested. The case study revealed the ability to use structural coverage metrics to identify the need for further testing, and structural path analysis to identify untested portions of the system. Structural path analysis can also produce new test cases to maximize coverage. Therefore, this research shows that structural testing provides additional assurance to rule-based system testing by showing completeness of the test suite against the models the rule-based system is intended to compute.

5.4 Contributions

Rule-based systems, like those built on the Loci framework, define computations through the interaction of data with logical rules. Sometimes rules will interact with each other by having decisions and computations of one set of rules act as input to another. Together, these interactions create models intended to represent either physical conditions, like that found in Chem's computational fluid dynamics models, or logical conditions though processes. While structures have been proposed, most of these structures are intended for

other purposes like design [1], specifically finding error in logic such as infinite loops or self-contradictions [19], or requirements based testing [10]. These types of testing, known as black box testing, do not consider code or code structure.

While black-box testing can be useful in supporting assurance, it does not show the correctness of the code itself. Since the computational and logical models are found in the code of a rule-based system, it is important to test the code, rather than just the functionality. This testing can provide assurance in the correctness of the implementation of the functionality. Structural testing techniques provide a method for developers to generate test cases which are centered around the code and the code structure of the rule-based system. This testing technique provides a means to show how much of the code has been tested, and therefore provide improved assurance in the implementation of that system.

This research provides support to rule-based system testing by showing that structural analysis can support the development of rule-based systems by improving testing techniques. Structural analysis of Quick Test provided a means for the Loci and Chem developers to find untested computations of the models. While Dr. Luke expected some portions to be untested, he was not aware of what was untested and stated that he had no way before this research to identify where deficiencies in Quick Test lay.

5.5 For Further Research

There remains much more research that can be performed from this point. The analysis by the Graph Analysis Tool worked from the fact that each rule in the code requires all the inputs to fire. This is different than the average rule-based system which can use complex

logical expressions to determine whether a rule fires or not. Further research involving case studies that use analysis on such a rule-based structure would help to provide better understanding of how structures in rule-based systems lend themselves to providing white box testing benefits.

Interestingly, extended research to measure more complex rule-based structures can still use Chem or any other Loci-based application. After the research was complete, it was learned that, while syntactically the rules do require all inputs to fire, the "priority" rules behave in a way that creates a complex relation semantically which behaves like a typical rule-based system. If the Graph Analysis Tool was designed to dynamically generate its graph traversal engine based on results of parsing "priority" rules, this would result in a study on a rule-based structure which uses a more complex mechanism for scheduling rules to fire.

Even without branching into more complex rule structures, detailed path analysis could be very beneficial to the field of rule-based system testing. As was shown in the case study, after the metrics were analyzed, the list of non-fired rules enabled Dr. Luke to specifically identify portions of the Chem models that have not been tested. The rule lists could in fact then be used to support path analysis that would lead to new test cases. There could even be a new tool generated which could identify uncovered paths which if tested would lead to higher levels of coverage.

Finally, more metrics could be researched. Dr. Luke stated that the metrics used in this research did not appear to have clear meaning to his system. While that does not mean that they would not have clear meaning to any rule-based systems, it does show that identifying

which metrics provide greater analysis support to rule-based system developers is needed. Also, as more complex structures are used in future rule-based system testing research, more metrics could be included. This could help determine which metrics provide meaningful results to the testers and the developers. Barr presented several more metrics which could be utilized in future research [2].

5.6 Potential Applications

The first of new applications this research can be used for is the development of new testing techniques for rule-based systems. Testers can identify structural components in their system which contribute to the different levels of coverage. For example, a tester could first identify each class and make a test case to complete at least one path to each of these classes. Then, paths between findings for these classes could be identified, adding one more level of coverage. As further research improves the understanding of rule-based systems, these techniques could result in more efficient test cases, reducing the time spent in testing for a system.

Software tools, like the one built for the case study, can also be built to benefit testers. These tools could be made to identify components of the rule-based structure which are not yet tested, and possibly even highlight paths which would add necessary coverage to the test suite. This coverage could lead to automated test case generations which maximizes test coverage while reducing time spent in testing. Such a tool could increase quality assurance of products designed around logical programming while possibly decreasing cost of development.

REFERENCES

[1] R. Agarwal and M. Tanniru, "Structured Tools for Rule-Based Systems," *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, vol. 3, Jan 1991, pp. 8 – 11.

[2] V. Barr, "Rule-Based System Testing with Control and Data Flow Techniques," *Annals of Software Engineering*, vol. 4, 1997.

[3] Gravisto, "Gravisto::Graph Visualization Toolkit," http://gravisto.fim.uni-passau.de/ (current Nov. 19, 2007).

[4] W. Gropp, "Why we couldn't use numerical libraries for PETSc," *Quality of Numerical Software: Assessment and Enhancement*, R. F. Boisvert, ed., New York, July 1996, NIST, pp. 249 – 252, Chapman and Hall.

[5] C. Grossner, A. Preece, G. Chander, T. Radhakrishnan, and C. Suen, "Exploring the Structure of Rule Based Systems," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., July 1993, pp. 704 – 709.

[6] L. Hatton, "The T-Experiments: Errors in Scientific Software," *Quality of Numerical Software: Assessment and Enhancement*, R. F. Boisvert, ed., New York, July 1996, NIST, pp. 13 – 30, Chapman and Hall.

[7] H. Inazumi and N. Omoto, "A New Scheme for Verifying Rule-Based Systems Using Petri-Nets," *Systems, Man, and Cybernetics: Conference Proceedings*. 1999, vol. 1, pp. 860 – 865, IEEE.

[8] P. Jorgensen, *Software Testing: A Craftsman's Approach*, second edition, CRC Press LLC, New York, 2002.

[9] J. Kiper, "Structural Testing of Rule-Based Expert Systems," *ACM Transactions on Software Engineering Methodology*, vol. 1, no. 2, April 1992, pp. 168 – 187.

[10] J. Lee and L. Lai, "A High-Level Petri Nets-Based Approach to Verifying Task Structures," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 2, March 2002, pp. 316 – 335.

[11] E. Luke, "CHEM: A Full Featured Navier-Stokes Solver for Chemically Reacting Flows," http://www.cse.msstate.edu/ luke/chem/index.html (current Jan. 4, 2008).

[12] E. Luke, "Loci: A Deductive Framework for Graph-Based Algorithms," *Lecture Notes in Computer Science*, , no. 1732, December 1999, pp. 142–153.

[13] E. Luke and T. George, "Loci: A Rule-Based Framework for Parallel Multidisciplinary Simulation Synthesis," *Journal of Functional Programming*, vol. 15, no. 13, 2005, pp. 477 – 502.

[14] S. Medders, *Graph Analysis Tool for Quick Test Loci Structures*, Tech. Rep. MSU-080125, Department of Computer Science and Engineering, Mississippi State University, MSU, MS, 2007.

[15] P. Mian, T. Conte, A. Natali, J. Biolchini, and G. Travassos, "A Systematic Review Process for Software Engineering," *3rd ESELAW - Experimental Software Engineering Latin American Workshop*, Uberlandia, 2005, vol. 1.

[16] M. Ramaswamy and S. Sarkar, "Using Directed Hypergraphs to Verify Rule-Based Expert Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 2, March 1997, pp. 221 – 237.

[17] C. Wu, "Enhanced High-Level Petri Nets with Multiple Colors for Knowledge Verification/Validation of Rule-Based Expert Systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 27, no. 5, Oct 1997, pp. 760 – 773.

[18] C. Wu and S. Lee, "A Token-Flow Paradigm for Verification of Rule-Based Expert Systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 30, no. 4, Aug 2000, pp. 616 – 624.

[19] S. Yang, A. Lee, W. Chu, and H. Yang, "Rule Based Verification Using Petri Nets," *Proceedings of the Twenty Second Annual International Computer Software and Applications Conference*, Aug 1998, pp. 19 – 21.

[20] YWorks, "yEd," http://www.yworks.com/en/products_yed_about.htm.

[21] Y. Zhang and E. A. Luke, "Dynamic Memory Management in the Loci Framework," *Parallel and Distributed Computing Practices*, vol. 7, no. 3, 2006.

APPENDIX A

METHODS FOR THE SYSTEMATIC LITERATURE REVIEW

A systematic literature review was performed in order to identify previous work on structural testing for rule-based systems. This review was based on the outline from Mian et al. [15]. A systematic literature review is a formal review process which documents the search, selection criteria, and the selection. This assured repeatability so future researchers can use similar methods to determine what new work has developed since this review was performed. The results of this review can be found in chapter 2. This appendix presents the structure and methods used for the systematic review.

A.1 Review Needs and Background

The first step the systematic literature review is to establish the needs of the review as well as the current knowledge of the reviewer. This section presents the problem which inspired the literature review. From there, questions which we hope to answer with the review are discussed. In order to establish a starting point, the researcher's current knowledge of structural testing and rule-based systems are presented. Finally, this section shows what is hopefully to be gained from the work.

A.1.1 Problem

This literature review is intended to support research in identifying the usefulness of structural coverage testing techniques for rule-based systems. Coverage testing for any system requires two main aspects. These are structures defined over the system and metrics defined over these structures. Therefore, this review needs to identify previous work done

in order to define a structure contained within the knowledge base of a rule-based system, and metrics defined which measure the coverage across these structures.

A.1.2 Questions

The following questions are the focus of the literature review:

- What structures can be extracted from a rule-based system?
- What metrics can be used to identify test coverage on a rule-based system?

A.1.3 Prior Knowledge

The following list contains the knowledge held by the reviewer prior to the review:

- logical programming — a basic understanding of building and using rule-based systems
- rule-based dependency — rules can effectively "call" one another by making assertions in the right hand side which match the left hand side of another rule
- coverage testing — the process of testing a system in a manner to assure the maximum coverage of the system's code and structure
- control flow and data flow — measurements and graphs depicting how control of the system and data being processed is transferred from one portion of the program to another

A.1.4 Keywords and Synonyms

In order to effectively find prior studies, the terms used in the rule-based systems and structural testing literature, including various synonyms, needed to be identified. Since often times authors use different words to describe the same thing, it is important to know and understand different varieties. The following list presents the terms found to be used commonly in literature:

- Rule-based Systems:
  - rule based, rule-based
  - expert system
  - production system
  - knowledge base
- Structural Testing:
  - structure
  - metrics
  - coverage
  - testing
  - verification
  - validation

A.1.5 Intended Observations

We intended to find existing literature on defining structures on rule-based systems. These structures were to show data flow, control flow, relations between rules, and logical paths leading to intended calculations, decisions, or goals. We also intended to find existing literature on coverage testing and coverage metrics defined on rule-based systems.

A.1.6 Expected Results

We expected to find literature about rule-based system structure, including structures which define dependency, and structures defining data flow. We also expected to find measurements across these structures defining path coverage and rule coverage.

A.1.7 Population

We were searching the literature available in academic databases, the Internet in general, and in the MSU Mitchell Memorial Library, as well as any other libraries searchable via the online search at the MSU library website.

A.1.8 Application

We expect the results of this work to be useful to anyone developing software using logical programming techniques.

A.2 Selection of Sources

The following sections list the criteria used to select sources and databases to be searched.

A.2.1 Definition of Criteria

For a source to be included in the search, it must meet the following requirements:

- Language: The studies need to be documented in English.
- Accessible: Studies must be able to be found in one of the following:
  - MSU Mitchell Memorial Library
  - Public Libraries participating in Inter-Library Loan
  - Online databases where full text is accessible to students of Mississippi State University
- Searchable: The source needs to support a search option. This is because the topic may be covered in multiple areas and an exhaustive search through various sources without a computer-aided search method is infeasible.
- Subject Search: The source needs to support narrowing searches to computer science studies.

A.2.2 Identification of Sources

Sources were selected before the search began, and this selection evolved as the review revealed other useful sources as well as showing that previously selected sources were not helpful.

- Sources Search Methods:
  - Expert Interview: librarians and professors
  - MSU Library Database: database of academic sources with brief descriptions
  - Google Scholar: Internet search engine indexing multiple academic sources
- Search String: Several search strings were used. While not all databases support the complex boolean searches as they are, the search strings can be altered accordingly. The searches are split up into searches for structures and searches for metrics
  - Structure:
    - (“rule based” OR “rule-based”) AND structure
    - “expert system” AND structure
    - “production system” AND structure
    - “knowledge base” AND structure
    - (“rule based” OR “rule-based”) AND (verification OR validation)
    - “expert system” AND (verification OR validation)
    - “production system” AND (verification OR validation)
    - “knowledge base” AND (verification OR validation)
  - Metrics:
    - (“rule based” OR “rule-based”) AND “coverage metrics”
    - (“rule based” OR “rule-based”) AND “coverage testing”
    - “expert system” AND “coverage metrics”
    - “expert system” AND “coverage testing”
    - “knowledge base” AND “coverage metrics”
    - “knowledge base” AND “coverage testing”
    - “production system” AND “coverage metrics”
    - “production system” AND “coverage testing”
    - (“rule based” OR “rule-based”) AND metrics AND (NOT complexity)
    - “expert systems” AND metrics AND (NOT complexity)
    - “production systems” AND metrics AND (NOT complexity)

    - “knowledge base” AND metrics AND (NOT complexity)
    - (“rule based” OR “rule-based”) AND testing AND coverage
    - “knowledge base” AND testing AND coverage
    - “expert systems” AND testing AND coverage

- Sources List: These sources were identified initially for the search.
    - MSU Library Indexes and Databases
    - Science Citation Index
    - EBSCOHost
    - ACM Digital Library
    - ACM Guide
    - IEEEXplore
    - Google Scholar

A.2.3 Source Selection after Evaluation

The following list of sources contained included studies:

- ACM Digital Library
- IEEEXplore
- Google Scholar

A.3 Selection of Studies

Primary studies for this review were selected with the criteria presented in this section.

A.3.1 Study Selection Definition

Here, the methods for selecting studies are defined.

- Inclusion Criteria:
    - Discusses structures built on rule-based systems, production systems, expert systems, or knowledge bases

  - Discusses measurements for rule-based system structure for evaluating test coverage for rule-based systems, production systems, expert systems, or knowledge bases

- Exclusion Criteria:

  - Discusses building rule-based systems to identify structures of other software systems
  - Discusses algorithms or other processes that use structures without defining said structures
  - Repeat studies by the same authors on previously found structures, except studies that extend the prior study, for example, from a small "toy" project to a real world case study

- Study Types: The following types of studies were included:

  - Journal articles
  - Conference presentations
  - Theoretical definitions
  - Case studies

- Procedure: The selection was done in the following phases:

  1. Title filtering: excluding studies whose titles clearly indicate the criteria is not met
  2. Abstract filtering: excluding studies whose abstracts contain exclusion criteria or do not contain inclusion criteria
  3. Introduction reading and study skimming:
     (a) Read study introduction
     (b) Skim study material
     (c) Exclude studies with no promising definitions or figures
  4. Data extraction:
     (a) Read article
     (b) Extract data to data extraction form (see section A.4.2)
     (c) Exclude if no data can be extracted (see section A.4.1)

A.3.2 Selection Execution

A small number of relevant studies were found. While a formal review including parallel reviews and critiques was not performed, a brief evaluation of their content was done.

- Included Studies: The following is the list of studies selected for this review.
  - "Structural Testing of Rule-Based Expert Systems" [9]
  - "Exploring the Structure of Rule-Based Systems" [5]
  - "Rule-Based System Testing with Control and Data Flow Techniques" [2]
  - "Dynamic Memory Management in the Loci Framework" [21]
  - "Structured Tools for Rule-Based Systems" [1]
  - "Rule-Base Verification Using Petri Nets" [19]
  - "Enhanced High-Level Petri Nets with Multiple Colors for Knowledge Verification/Validation of Rule-Based Expert Systems" [17]
  - "A New Scheme for Verifying Rule-Based Systems Using Petri Nets" [7]
  - "Using Directed Hypergraphs to Verify Rule-Based Expert Systems" [16]
  - "A Token-Flow Paradigm for Verification of Rule-Based Expert Systems" [18]
  - "A High-Level Petri Nets-Based Approach to Verifying Task Structures" [10]
- Study Quality: Quality was recorded in data extraction (see A.4.2) but was not used for inclusion or exclusion criteria.

A.4 Information Extraction

Information was selected and extracted from the selected studies by the following means.

A.4.1 Inclusion and Exclusion Criteria

After studies were selected, the information extraction criteria was divided into information for structures and graphs representing rule-based systems and coverage metrics for rule-based system structure.

A.4.1.1 Graphs and Structures

Several graphs and structures were found in literature, but not all were found to be useful. The following inclusion and exclusion criterion were used during the review process.

- Inclusion:
  - Well defined structures: at a minimum, verbal descriptions of entities and relations, formal definitions preferred
  - Relations between rules can be shown:
    - output of one rule acts as input to another
    - entities are rules and relations between entities define relations between rules
  - Unique structures: redefinitions excluded
- Exclusion:
  - Structures defined over non-rule-based systems
  - Structures with no clear mapping to rule-based system attributes

A.4.1.2 Metrics

Metrics defined for the purpose of testing turned out to be fairly rare. Other metrics were found however, and the following criterion were used to determine which studies were pertinent the needs of this study.

- Inclusion:
  - Metrics defined for test coverage
  - Metrics defined over structures for rule-based systems
- Exclusion:
  - Metrics defined for complexity
  - Metrics defined over code
  - Metrics defined over problem-space or knowledge space
  - Metrics defined specifically for non rule-based system structures

A.4.2 Data Extraction

Table A.1 was used for extraction of structure data for rule-based systems. Table A.2 was used for extraction of coverage metrics for rule-based system structure.

Table A.1

Structures for Rule-Based systems

Data Item	Description
Bibliographic Data	Full Citation
Database or Search engine	Database that study was found in
Query used	Query used to find study
Type of Article	Journal, Conference presentation, workshop, etc
	Theoretical, Toy Project, Case Study, Empirical
Structure	Name of structure defined in study
Entities	Entities in study, these usually are antecedents or rules
Relations	Relations between entities in structure
Comments	Miscellaneous comments about paper and structure

A.5 Distribution of Studies

The studies selected were found across three search engines, and varying in publication types, study types, and specific topics. In this section, the distribution across these aspects will be presented. Numbers presented are simply the count of the studies found out of the ten included (see Section A.3.2).

What can be seen in Table A.3 is that the highest concentration of studies is with IEEEXplore. The two studies found using Google Scholar were indexed on personal web sites. The study listed as “Personal Web Site” was found on the developer’s Web site for

Table A.2

Coverage Metrics for Rule-Based System Structures

Data Item	Description
Bibliographic Data	Full Citation
Database or Search engine	Database that study was found in
Query used	Query used to find study
Type of Article	Journal, Conference presentation, workshop, etc
	Theoretical, Toy Project, Case Study, Empirical
Metrics	Definition of metrics
Comments	Miscellaneous comments about paper and metrics

the Loci high-performance computing package. Because Google Scholar indexes several databases, it was searched last. When no more studies were found to be indexed on other databases, then we concluded that the search was complete.

Table A.4 shows that the included studies were approximately balanced between journal articles and conference presentations. No workshop presentations or technical reports on this subject were found. Table A.5 shows that there were twice as many toy projects studied than case studies, and no controlled or real-world experiments. This indicates a need for more full-scale studies.

A.6 Publication Bias

The problem of publication bias refers to the tendency of positive results being more likely to be published than negative results. The question proposed for this review was to find what structures and metrics have been proposed. This does not include what have been shown to be useful. Therefore, this problem does not present a threat to this review.

Table A.3

Database/Search Engine Distribution

Database/Search Engine	Number of Papers found
IEEEXplore	6
Google Scholar	2
ACM Digital Library	1
Personal Web Site	1

Table A.4

Article Type Distribution

Article Type	Number of Studies
Journal Article	6
Conference Presentation	4
Workshop Presentation	0
Technical Report	0

Table A.5

Study Type Distribution

Study Type	Number of Studies
Theoretical only	1
Toy Project	6
Case Study	3
Controlled Experiment	0
Real-World Experiment	0

APPENDIX B

GRAPH ANALYSIS TOOL

A software tool was required to analyze structure graphs for the Quick Test case study. The requirements of the tool were to accept the graphs and test inputs in files. The tool then needed to measure coverage of the graphs that resulted from traversal starting with input nodes down to class nodes.

The graph analysis tool took several weeks to develop. The tool was used to analyze graphs representing the structure of the Quick Test rule-base. The tool accepts the folder containing the graph files. These graphs files are parsed, then measured. This chapter describes the tool and how it supported this research.

B.1 Input Files

During execution, Loci generates a graph structure representing the relations of variables and rules to each other. The purpose of this structure is to determine which calculations can be performed in parallel and which must be completed sequentially. These structures were saved to a file for the purpose of this research. This section discusses the format of the files, and how they were parsed for the tool.

The file was an ASCII text file with a `.dat` file extension. The file amounted to a list of inputs for that graph, a list of nodes, and a list of connections. Each node had an identifier that was a non-zero integer. The difference between variable nodes and rule nodes were that variable nodes had positive identifiers and rule nodes had negative identifiers.

The first section of the format is an optional section specifying a list of inputs for the test case. After the list of inputs, the graph is given as a list of nodes and a list of connections. The list of nodes is a set of colon separated integer-string pairs. The integer is

a node identifier. A negative number indicates the node represents a rule in the rule-based system. A positive number indicates the node is a variable. If the node is a rule node, the string is the rule as it appears in the original source code. Should the node represent a variable, the string represents the name of the variable in the rule-based system. Each connection is represented in the connection list by one integer followed by a set of integers. Each integer is a node identifier appearing in the node list earlier. The list represents edges going from the first node to all the following node in the list for that set of connections.

B.2 Development

A graphing library was needed to represent the Quick Test graphs in memory and provide a method for analyzing these graphs. The library needed to be able to iterate through the nodes, number and list incoming and outgoing edges, and save the graph to a portable file format. GraphML is a file format which uses XML syntax to represent the graph's nodes and edges. Meta-data can be represented by attributes to the nodes and edges.

Two options in developing the tool were building a graphing API or using an off-the-shelf API. The needs of the API would require data structures to represent the graph, the nodes, and the edges. Also, the API would need to be able to store meta-data in the graph and save the graph to a GraphML file. Gravisto provided all of these needs. Prior experience with Gravisto gave this option no learning curve and no needed development time. For this reason, the Gravisto library was chosen over developing a customized graphing library.

B.2.1 Gravisto API

Gravisto provides classes to represent the graph, the nodes, and the edges. From a graph object, a program can iterate through all the nodes or all the edges. A graph object contains node objects and edge objects. This object can store attributes about the graph of all major data types: Integer, Float, Double, Boolean, String, and generic Object. These attributes are referred to by names, known as paths. Graphs can be directed or undirected. Directed paths differentiate the direction of edges as incoming or outgoing from a particular node. Undirected graph semantics define edge flow as bidirectional.

The node interface provides methods for many node-based functions. Using this interface, a program can obtain a collection of all incoming or outgoing edges. The interface also provides methods for acquiring adjacent nodes in the graph. The methods can differentiate between adjacent nodes attached to incoming edges from those attached to outgoing edges. Finally, nodes also provide attribute support. These attributes can provide a method for identifying variable nodes in a rule-based structure from rule nodes. A boolean attribute can also be used to identify rules which have been fired and variables which have been input or calculated.

Edge interfaces also provide many useful graph functions. Methods in the interface allow a user to acquire the source node and target node objects for the edge. Attributes for the edge, such as the node interface and the graph class, can be stored in most Java data types. These attributes provide a means of identifying edges which have been traversed within the rule-based graph.

B.2.2 Traversing the Graph

Each node in the graph is given a boolean attribute labeled `isFired`. For variable nodes, a true value for this boolean means the variable has either been provided as input or calculated in the firing of a rule. In a rule node, this attribute indicates whether or not the rule has been fired.

For all Quick Test graphs except the master graph, the file will have a set of inputs listed. Each node with a name attribute matching one of the names in the input list has the `isFired` attribute set to true. This sets the graph in its initial stage with input data present. The master graph is a special graph which represents the system as a whole. This graph removes time-based iterations but includes all rules within the system. The tool aggregated the inputs of all the test cases together in order to initiate traversal of the master graph.

In Loci, rules can be fired if all data is provided to the system as input [13]. Therefore, for traversal, when a rule node is reached, all incoming node neighbors are iterated through. If all incoming node `isFired` attributes are set to true, the rule node's `isFired` attribute can be fired, indicating the rule would be fired in the Loci system.

The first step in traversing the graph involves collecting a list of all rule nodes in the graph. This list is iterated through multiple times. For each rule node, if the rule can be fired, the `isFired` attribute is set, and all incoming and outgoing edges have the `isTraversed` boolean attribute set to true. Also, the `isFired` attribute for all outgoing neighbor nodes is set, and the rule node is removed from the list of rule nodes.

This iteration is repeated until the list is empty, or until one full iteration is completed without a single rule node being fired.

B.3 Output

The Graph Analysis Tool presented three outputs. The metrics spreadsheet showed coverage measurements of Quick Test over the Loci structures. The rule lists presented rules that were not fired in test cases along with variables not calculated. Finally, graphML files were output representing uncovered portions of the Loci structures.

B.3.1 Metrics Spreadsheet

A comma separated variables file was used to represent a spreadsheet of metrics taken for each test case in Quick Test by the Graph Analysis Tool. This spreadsheet consisted of four columns. The first column listed each test case name according to the name of the graph file provided for that test case. The following three columns represented the levels of coverage achieved by each test case: Classes, Finding-Class Pairs, and Edges. These metrics are discussed in further detail in section 3.3.1.

B.3.2 Rule Lists

While debugging the program, some rules in the graph were not being traversed, even though it was known that these rules were being computed in the test cases. In order to understand why, a text file listing each rule node which was not traversed was output. After each rule, an indented list of required input variable nodes for that rule followed. One such file was written per graph.

After debugging was complete, this feature was left in the tool, in order to allow the Loci developers to analyze the results better. The list of untested rules will allow the developers to identify new test cases. Such an analysis could also provide extended benefits to this research by showing the usefulness of structural testing to the developers.

B.3.3 GraphML File

After processing the graph from the file, all nodes and edges covered were removed, and the file was rewritten to disk with the changes. To do so, all edges with a true value `isTraversed` were removed from the graph. Then, all nodes with no incoming or outgoing edges were also removed. This process left a graph with only uncovered portions remaining. As with the list of uncovered rules, this graph can be used by developers to determine new test cases to cover the previously uncovered portions.

This graph is stored in the graphML format. This format is supported in the Gravisto library [3]. As was discussed before, the graphML format is an XML-based format. The Gravisto library automatically generates the XML and saves the graphML file. This file can be viewed in any visualization toolkit which supports graphML. The graphML files were not used in analysis of the case study, because they proved to be large and complex. No visualization toolkit was found capable of rendering an organized presentation of the graph along with its meta-data. This would lead to too much information presented in an format which could not be organized and analyzed in the amount of time given for this research.

APPENDIX C

METRICS TABLE FROM THE GRAPH ANALYSIS TOOL

The following tables present the metrics produced through use of the graph analysis tool described in chapter B.

Table C.1

Metrics Results

Graph file Classes Metric	Findings-Class Pairs	Edges Metric
Viscous_highSpeed_TEST_wallLawTwallgraph.dat 0.6	0.43193159	0.6918446
Viscous_lowSpeed_TEST_diffusiongraph.dat 0.47368422	0.4218492	0.6764627
Viscous_lowSpeed_TEST_lidgraph.dat 0.47368422	0.4166172	0.6679629
Viscous_lowSpeed_TEST_wallLawAdiabaticgraph.dat 0.6	0.43063426	0.6894229
Viscous_lowSpeed_TEST_wallLawTwallgraph.dat 0.6	0.43229812	0.69109666
masterGraph 0.7777778	0.1840504	0.4032941

Table C.1

Metrics Results cont'd

Graph file Classes Metric	Findings-Class Pairs	Edges Metric
Inviscid_lowSpeed_TEST_inflowgraph.dat 0.47368422	0.39823008	0.70780456
Inviscid_lowSpeed_TEST_isentropicgraph.dat 0.47368422	0.399287	0.6550662
Inviscid_lowSpeed_TEST_shock_tubegraph.dat 0.44444445	0.35183823	0.6638935
Inviscid_lowSpeed_TEST_superSonicgraph.dat 0.47368422	0.3969278	0.69996923
Viscous_highSpeed_TEST_adiabaticgraph.dat 0.6	0.47372863	0.71039355
Viscous_highSpeed_TEST_Twallgraph.dat 0.6	0.47332913	0.71170986
Viscous_highSpeed_TEST_wallLawAdiabaticgraph.dat 0.6	0.43050477	0.68981904

Table C.1

Metrics Results cont'd

Graph file Classes Metric	Findings-Class Pairs	Edges Metric
Inviscid_highSpeed_TEST_cfitgraph.dat 0.47368422	0.39757943	0.7013294
Inviscid_highSpeed_TEST_cylindergraph.dat 0.47368422	0.3975939	0.6566139
Inviscid_highSpeed_TEST_fixedMassgraph.dat 0.47368422	0.40121844	0.65475154
Inviscid_highSpeed_TEST_inflowgraph.dat 0.47368422	0.39823008	0.70780456
Inviscid_highSpeed_TEST_isentropicgraph.dat 0.47368422	0.399287	0.6550662
Inviscid_highSpeed_TEST_nozzlegraph.dat 0.47368422	0.39848942	0.7094409
Inviscid_highSpeed_TEST_shock_tubegraph.dat 0.44444445	0.35183823	0.6638935
Inviscid_highSpeed_TEST_superSonicgraph.dat 0.47368422	0.39785275	0.7083731
Inviscid_highSpeed_TEST_wedgegraph.dat 0.52380955	0.4710093	0.6894494
Inviscid_lowSpeed_TEST_cylindergraph.dat 0.47368422	0.3975939	0.6566139
Inviscid_lowSpeed_TEST_fixedMassgraph.dat 0.47368422	0.40121844	0.65475154