MINISTRY OF EDUCATION & TRAINING
THE UNIVERSITY OF DANANG

TRINH CONG DUY

AUTOMATIC REGRESSION TESTING FOR LUSTRE/SCADE APPLICATIONS

Major: COMPUTER SCIENCE
Code of Major : 62 48 01 01

DOCTORAL THESIS SUMMARY

Da Nang, 11/2018

The doctoral dissertation has been finished at:

THE UNIVERSITY OF DANANG

Advisors:

1) Assoc Prof Dr. Nguyen Thanh Binh

2) Prof Dr. Ioannis Parissis .

Reviewer 1:……

Reviewer 2:……

Reviewer 3:……

The dissertation is defended before The Assessment Committee at The University of Danang

Time:….....h.....

Date:...../……/…….

The dissertation is available at:

- National Library of Vietnam
- Learning & Information Resources Center, The University of Danang

INTRODUCTION

1. Context and Motivation

Lustre [4, 5] is a formal declarative and synchronous dataflow programming language that can be also used as a temporal logic of the past. Variables and expressions are represented by data flows, that is, infinite sequences of values whose evaluation is governed by a discrete global clock. SCADE [6] is a graphical environment commercialized. It is based on the synchronous language Lustre. Therefore, it is often referred to as Lustre/SCADE. Lustre/SCADE is usually used to build the applications of reactive systems.

Software maintenance is the last stage of software life cycle aiming at, correcting errors, making modifications to functionality and deleting existing features. These changes may cause the system to work inaccurately. Thus, there is a need for regression testing. Regression testing purpose is to make sure that the changes and modifications to the software did not introduced new bugs. Regression testing can be applied for testing a system efficiently by methodically selecting the proper minimum test sets needed to cover a specific modification adequately. Our objective is to determine a method for detecting the most possible errors with the minimum of test data.

This motivates us to choose the topic **"Automatic regression testing for Lustre/SCADE applications"** for the doctoral dissertation. This thesis aims at studying regression testing process issues with a focus on automating test data generation, in the framework of reactive systems developped in the Lustre/SCADE.

2. The Goals, Objectives and Scope of the Research

The objectives of the thesis are to propose automatic regression

testing techniques in the Lustre/SCADE environment. To do so, we proceed in the following steps:

– Firstly, we present a state of the art on regression testing and regression testing techniques.

– Secondly, we analyze the features of reactive systems, the synchronous approach, the Lustre language and SCADE environment; we study the structural coverage of Lustre programs and the activation conditions of paths on operator.

– Thirdly, we focus on using model checking in software testing. The thesis proposes the approach of using model checking to generate the test data based on the activation conditions in operator network of Lustre/SCADE programs.

– Finally, we propose an approach to generate test data in regression testing for Lustre programs. In this approach, a Lustre program is modeled by an operator network. Then we determine its set of paths and compute symbolically the path activation conditions for each version. Test cases for regression are generated by comparing paths between versions. To validate this solution, we have developed a tool named LUSREGTES.

3. The contributions of this thesis

– We have suggested the solution which uses activation conditions on the operator network from Lustre programs, combined with the use of a model checker on main source of Lustre program to create test data for Lustre/SCADE programs. This solution helps removign the manual inputs definition in the model checking, as well as saving time and effort since this solution could be fully automated.

– The thesis proposes the solution to generate test cases for

regression testing. We proposed and experimented three approaches: GSRS - Generation of test cases in regression test using software requirements in natural language; GSCR - Generation of test cases in regression test using SCR; GOPN - Generation of test cases in regression test using operator network. Most importantly, we can propose the best and best suited solution based on special characteristics of Lustre programs. With GOPN approach, we identify the correlation among activation conditions of paths on the operator network and test data. When the changes appear upon Lustre/SCADE program, we identify which data have just been removed, re-used from the old version or need to be created.

– Suggest algorithms to complete automatic solutions. In particular, the two algorithms AGTC and AGTR are the most important. The GOPN approach will be developed based on these two algorithms.

– We have developped LUSREGTES tool, it implementing the GOPN approach. The tool automatically creates test data for regression testing of Lustre/SCADE programs. LUSREGTES tool helps removing the unnecessary test data; therefore it saves considerable time and effort for regression testing process.

4. Thesis structure

In addition to introduction, conclusion and future work sections, the structure of thesis contains the following chapters.

Chapter 1. Lustre/SCADE and Regression testing: basic concepts. This chapter presents the fundamental concepts of regression testing, the techniques and application of regression testing as well as a state of the art on this testing technique. Beside that, the chapter

introduces the overview of Lustre/SCADE environment, the features and basic components of Lustre programs and SCADE environment, we focus on introducing the contents related to the Lustre language as: Operator network, paths and activation conditions.

Chapter 2. Using model checker for testing Lustre/SCADE programs. Model checkers are formal verification tools, which are of providing counter-examples violating properties. In this chapter, we have proposed an approach to use a model checker to generate the test cases for Lustre/SCADE programs.

Chapter 3. Regression testing approach for Lustre/SCADE programs. This chapter presents the solution to generate test cases in regression testing of Lustre programs. We studied and proposed three approaches: GSRS, GSCR and GOPN.

Chapter 4. LUSREGTES: a regression testing tool for Lustre/SCADE programs. Chapter 4 presents the LUSREGTES tool for regression testing Lustre/SCADE programs implementing and illustrating the solution that has been proposed in Chapter 3.

Chapter 1. LUSTRE/SCADE AND REGRESSION TESTING: BASIC CONCEPTS

1.1 Testing techniques

This section present an overview of software testing, which includes two techniques black box testing and white box testing.

1.2 Regression testing

1.2.1 Introduction

Regression testing is the process of re-testing of software to ensure its quality and check whether changed parts work as intended

and unchanged ones are not influenced by the modifications. Regression Testing Techniques.

There are various regression testing techniques [20]: (1) Retest all; (2) Regression Test Selection; (3) Test Case Prioritization; (4) Hybrid Approach.

1.2.2 Regression test techniques

1.2.3 Regression test selection techniques

In this section, we study the regression test selection techniques as well as give a try on finding relevant techniques or concepts that can be applied appropriately in our context.

1.2.4 Regression Testing Tools

1.3 Introduction to Lustre/SCADE

1.3.1 Reactive system

Reactive system [29] is a system that responds to its external environment. This system can be autonomous or controlled orientation to respond to external influences. Continuous response system reacts to the environmental impact, while the environment cannot be synchronized in a reasonable manner with the system. That is, the environment cannot wait and strict real-time constraints need punctual response must be taken into account by the system.

1.3.2 Synchronous programs

1.3.3 Lustre language

Lustre is a data-flow synchronous language, which is designed for programming reactive systems [4, 33]. Advantages of this language are the merging of both synchronous and dataflow paradigms and its simple graphical syntax along with the notion of discrete time.

1.3.4 Specification of a software in Lustre

1.3.5 Flows and Clocks in Lustre

1.3.6 SCADE environment

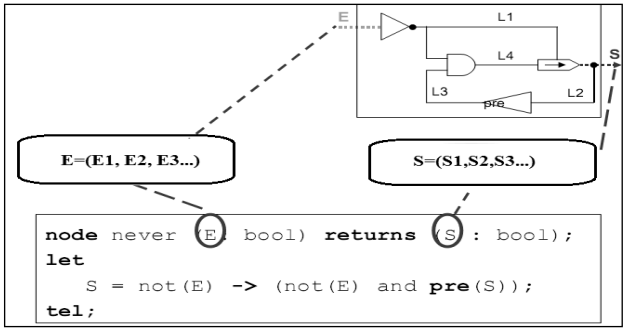

Figure 1.6. Illustrating a SCADE model and Lustre program.

SCADE [6] is a graphical environment commercialized by Esterel Technologies. It is based on the synchronous language Lustre. Therefore, it is often referred to as Lustre/SCADE. Figure 1.6 illustrates a SCADE model and the corresponding Lustre source for ***never*** program.

1.4 Structural model in Lustre programs

Lustre is a data-flow language: the input flows of a program are transformed into output flows through a set of dependent or independent operators. As in SCADE descriptions, the most usual representation for Lustre programs is directed graph, called operator network.

1.4.1 Operator network

Lustre programs are usually represented as operator networks [60, 61, 62, 59]. An operator network is a labeled graph connecting operators by means of directed edges. An **operator** (logical or

numerical) specifies data-flow transfers from inputs to outputs. An **edge** specifies the data flow between two operators.

1.4.2 Paths in an operator

Paths in an operator network represent the possible directions of flows from the input through the output. Table 1.3 provides the examples of paths in the ***never*** program.

Table 1.3. The examples of paths.

#	**Path**	**Type**	**Length**
1	*(E, L0, S)*	*acyclic*	*3*
2	*(E, L1, L3, S)*	*acyclic*	*4*
3	*(E, L0, S, L2, L3, S)*	*cyclic*	*6*
4	*(E, L1, L3, S, L2, L3, S)*	*cyclic*	*7*

1.4.3 Operator Predicate

1.4.4 Activation conditions

The condition upon which a data flow is transferred from the input edge to the output edge of an operator is called activation condition [61]. An activation condition is associated with each path.

Table 1.4 summarizes the formal expressions of the activation conditions for most of the Lustre operators [63].

Table 1.4. Activation conditions for all Lustre operators.

Operator	**Activation condition**
s = NOT (e)	*AC (e,s) = true*
s = AND(a,b)	*AC (a,s) = not(a) orb* *AC (b,s) = not(b) ora*

$s = OR(a,b)$	$AC\ (a,s) = aornot(b)$ $AC\ (b,s) = bornot(a)$
$s = ITE\ (c,a,b)$	$AC\ (c,s) = true$ $AC\ (a,s) = c$ $AC\ (b,s) = not(c)$
relational operator	$AC\ (e,s) = true$
$s = FBY\ (a,b)$	$AC\ (a,s) = true \rightarrow false\ AC\ (b,s) = false \rightarrow true$
$s = PRE\ (e)$	$AC\ (e,s) = false$ -> $pre(true)$

For example, with the path p_2 = (*E*, *L1*, *L3*, *S*) in the corresponding operator network for the node *never* (Table 2.2), the boolean expression for the activation condition of the selected path is:

$$AC\ (p_2) = false \rightarrow E\ or\ pre(S).$$

In practice, in order for the path output to be dependent on the input, either the input has to be true at the current execution cycle or the output at the previous cycle has to be true; at the first cycle of the execution, the path is not activated.

1.5 Testing Lustre/SCADE programs

1.5.1 Test data generation

According to the information available about the system, test data generation techniques are generally divided into two classes. In this section, we mainly focus on Lustre programs and as stated before, the techniques of the first class are based on black-box testing and merely require functional specification to generate test data, while the second class has the techniques based on white-box testing which needs the Lustre code of the program under test. And, in this section, we have mentioned some testing tools for Lustre applications.

1.5.2 Coverage criteria for Lustre programs

1.6 Conclusion

In chapter 1, we have discussed the basic of regression testing, the technologies that improve effectiveness of regression testing. With the fast development of software industry and shorter software upgrade cycle, regression testing is used more and more widely as well as developed quickly both in technical aspect and scope of application. Also in this chapter, we have presented an overview of reactive systems and characteristics of these systems. The thesis focused on researching the special characteristics of Lustre language and SCADE environment, the concepts of operator network, paths and activation conditions, which are important elements used as a basis for solutions proposed in this research. Additionally, we have mentioned some testing tools for Lustre applications such as Lutess, GATeL, etc.

Chapter 2. USING MODEL CHECKER FOR TESTING LUSTRE/SCADE PROGRAMS

2.1 Model checking technique

2.1.1 Introduction to model checking

A model checker is a tool used for formal verification [43]. It takes as input an automaton based model of a system and a temporal logic property, and then effectively explores the entire state space of the system in order to determine whether the model violates the property or not, Figure 3.1 is a typical model checking work-flow. If a property violation is encountered, then a counterexample is returned to illustrate the violation to the analyzer.

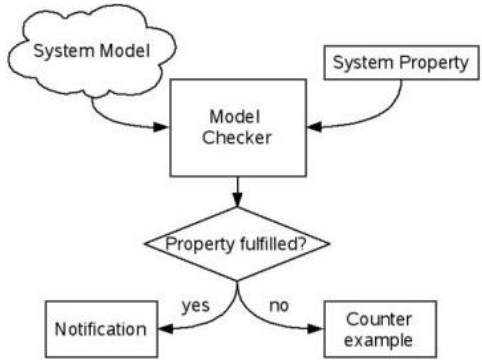

Figure 2.2. A typical model checking work-flow

2.1.2 Kripke structure

2.1.3 Temporal logics

2.2 Testing with model checker

The idea of testing with model checkers is to interpret counterexamples as test cases. In many cases, testing with model checkers is applied to reactive systems, which read input values from sensors and set output values accordingly. The system reacts to inputs by setting output values, so that a logical step in a counterexample can be mapped to an execution cycle of the system under test. In the reactive system scenario, counterexamples can directly be interpreted as test cases [43].

2.3 Lesar: Tool for model checking Lustre/SCADE programs

2.4 The solution to generate test data for Lustre/SCADE programs

2.1.4 Using model checker in generation test cases for Lustre/SCADE programs

In this study, we propose the solution to generate test data for Lustre/SCADE program. In details, we use activation conditions on the operator network of Lustre program and model checker to automatically generate test cases, according to those conditions. Lesar

model checker will be applied to test model for a specific Lustre program, then, to generate test cases based on the results from this test.

With a given Lustre programs, test cases are needed to be generated automatically, according to activation conditions of paths on operator network of the equivalent Lustre program.

2.1.5 The AGTC Algorithm

The algorithm in Figure 2.11 implements the solution, it is called as AGTC (AGTC - Algorithm for Generating Test Cases).

Input: - *SourceFile*: The source code of Lustre/SCADE program

Output: - *T*: The list of test cases for testing

- *DatabaseSuite*: The database of relation between Paths, Activation Conditions, Trap properties and Test case

Algorithm:

1. **Begin**
2. *SourceCode*=**readSourceCode***(SourceFile)*
3. *T*= **initTestCaseList()**
4. *DatabaseSuite* =**initDatabaseSuite()**
5. *P*= **getAllPathInSourceCode**(*SourceCode*) // Define list of paths from source code
6. *i=1*
7. **while** (not **isEndOfListPath(***P*))
8. *AC[i]*=**getAc**(*P[i]*) //get Activation condition of path
9. *Trap [i]*=**defineTrapPro**(*AC[i]*) //create Trap properties with AC
10. *ModelFileList[i]*=**CreateFileForLesar**(*SourceCode*, *Trap[i]*)
11. **UpdateDatabase**(*DatabaseSuite*, *P[i]*, *AC[i]*, *Trap[i]*, *ModelFileList[i]*)
12. *i++*
13. **end while**

14. **foreach** (*LesarInputFile* **in** *ModelFileList*)
15. *Counter_Example*=**modelChecking**(*LesarInputFile*) // Run Lesar
16. *TestCase*=**defineTestData**(*Counter_Example*)
17. **T=addTestCaseList***(T, TestCase);*
18. **UpdateDatabase**(*DatabaseSuite*, *T*)
19. **end foreach**
20. **Optimize**(*T*); //Optimize the list of test cases
21. **Optimize**(*DatabaseSuite*); //Optimize the database of relation
22. **End**

Figure 2.1. The AGTC algorithm

The AGTC algorithm takes a Lustre/SCADE program as the input and produces the output including the list of test cases and the database of relationships between Paths, Activation Conditions, Trap properties and Test cases.

Let's note *n* as the number of paths of the program. The first loop of the algorithm repeats *n* times, so its complexity is O(*n*). The second loop repeats also *n* times. Hence, the complexity of the algorithm is O(*n*) + O(*n*), so it is O(*n*).

2.1.6 Case study

In this section, we present the case study to generate test data for Lustre/SCADE program, using the AGTC algorithm presented above.

2.5 Conclusion

Chapter 2 has shown how model checkers may be used to generate test data for reactive systems developed in Lustre. The basic idea of this solution is to apply trap properties for the process of model checking; the final goal is to create counter-examples from which test cases are created to be used in the testing process.

The ability to construct counter-examples by a model checker has been proposed as a way of deriving test cases. We have suggested to use activation conditions on the operator network from Lustre programs to build model and trap properties for model checking process. This approach could be fully automated so it can increase the efficiency of testing process for Lustre/SCADE programs.

Chapter 3. REGRESSION TESTING APPROACH FOR LUSTRE/SCADE PROGRAMS

3.1 Motivation

Regression testing is applied when previously tested code is changed, in order to ensure that no new errors are introduced. A straightforward approach to regression testing is retest all. Here, all available test cases are executed, which might be very time consuming and expensive. Therefore, selective retesting tries to select only a subset of the available test cases, which are sufficient to detect faults introduced with the changes. Traditionally, only changes in the source code are considered. Changes in the specification, however, also require regression testing.

Currently, there are some studies related to the test cases selection techniques in regression testing, but are not proposed for Lustre/SCADE.

Besides, the Lustre/SCADE applications usually require very high quality and rigorous testing activities before being deployed. During the development process, the Lustre program is often updated, regression test should be performed to detect bugs. However, the regression testing process takes excessive time and resources if we re-run all test cases in the old version and generate the new test cases.

Thus, we need to optimize the number of test cases for regression testing. As a result, we need to define a technique that selects a subset of the existing test suite.

3.2 Research scope in regression testing for Lustre/SCADE programs

To support cost-effective regression testing for Lustre/SCADE programs, we propose a novel approach of combining regression testing via model checking with the following research goals.

Suppose that we have a Lustre program L_1 and that L_2 is a new version of program L_1. Comparing two versions:

- Remove some functions;
- Add some new functions;
- Change in some functions;
- And leave some functions unchanged.

In the first version, we don't use regression testing but model checking to generate test cases. In this version, the selection of test cases has not been applied, all test cases are considered. In the second version, regression testing will be applied. We need to apply a technique of selection of test cases in regression testing and eliminate unnecessary test cases to reduce costs. We call this technique TSTG (Technique of selection and automatic generation of test cases). The details of this technique will be presented in next sections.

3.3 Proposed approaches for regression testing

In regression testing, the main issue is to create a set of test cases to test the new version. Here, we have two versions, the current version L_2 and the previous version L_1.

The basic idea in the construction of the set of test cases is as follows: We compare version L_2 and version L_1 of the system to determine differences between them:

– The new parts added into L_2;

– The parts of L_1 changed in L_2;

– The parts of L_1 not affected in L_2;

– The parts of L_1 removed from L_2.

In the first stages of the software testing process, the development of test plans, test cases and other content for testing is often based on system descriptive documents such as solid documentation: the requirements of the software in natural language, or the format documents of design specifications of the system or the source code of the program. Depending on the context in which they are used, the testing process will apply different solutions with different inputs.

In this part of the thesis, we propose the following three approaches in regression testing for Lustre programs:

– GSRS - Generation of test cases in regression test using Software Requirements Specification (SRS) in natural language;

– GSCR - Generation of test cases in regression test using Software Cost Reduction (SCR);

– GOPN - Generation of test cases in regression test using Operator Network of Lustre program.

3.4.1 The GSRS approach

The basic idea in the construction of the set of test cases is as follows: We compare the current version (L_2) and the previous version (L_1) of the system to determine: The new requirements, the

requirements that need change, the requirements that are not affected and the requirements that are removed in the new version (in L_2).

The GSRS approach can also be used to generate test cases for regression testing, but based on the requirements is not simple, and automating this process is very difficult.

3.4.2 The GSCR approach

For this, we are using the SCR (Software Cost Reduction) [62] technique. In this section, we present the solutions of automatic generation of test cases in regression testing from program requirement specifications using SCR requirements. First, we define SCR specification from requirement of two programs. Then SCR 1 and SCR 2 are fed into a comparator. However, there are some issues when using this approach:

– Translating a Lustre program into SCR may be costly;

– Comparison A and B must be done manually: Comparing two SCR versions may be difficult;

– To use a model checker, GSCR approach requires to define models and properties, based on SCR. Therefore, SCR must be translated to Promela or SMV and LTL which is very costly. Developping a model checker for SCR specification is also costly. The automation of this process is impossible at this phase.

3.4.3 The GOPN approach

Both approaches GSRS and GSCR are often used in black box testing techniques. They are commonly used in manual testing with input as natural language or SCR method. If the input is a Lustre program with source code, both approachs cannot be used because

Lustre/SCADE has its own characteristics. In this part, we propose a new approach based on characteristics of Lustre/SCADE programs.

As mentioned earlier, a Lustre program is represented by a network of operators with corresponding paths and activation conditions. Figure 3.9 illustrates the proposed approach.

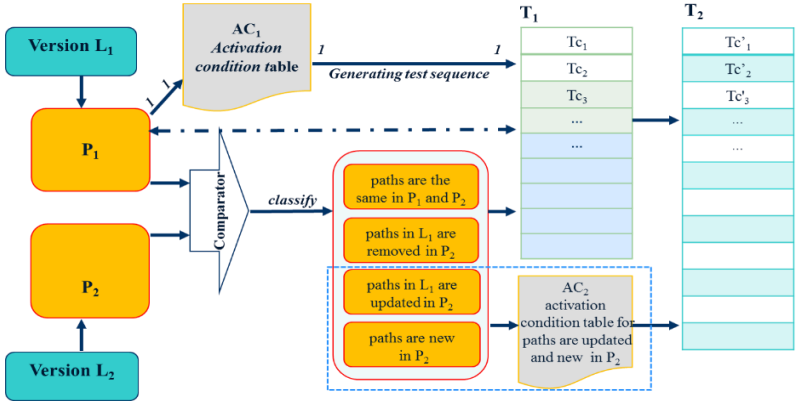

Figure 3.9. The overall approach of regression testing for Lustre programs.

The GOPN approach consists of two following phases:

– Generating test cases for version L_1

– Selecting and generating test cases for version L_2

Phase 1: Generating test cases for version L_1

We first determine the operator network, path table and activation condition table for version L_1. Then we generate test cases (test sequence) covering the activation condition table. This solution have been presented in Chapter 3. The result of thiss phase is a set of test sequences T_1 and the relationship between path table P_1 and T_1.

Phase 2: Selecting and generating test cases for version L_2

In this phase, we reuse a subset of test cases generated in phase 1 for testing version L_2 and we create new test cases covering the new paths in L_2. This phase consists of three steps as follows:

2.1) Determining paths table P_2 from operator network of L_2.

2.2) Comparing two tables P_1, P_2 and classifying paths, we obtain three subset P_A,P_B and P_C .

2.3) Building the set of test cases for version 2 based on P_A, P_B and P_C as follows.

We design AGTR algorithm (AGTR - Algorithm for Generating Test cases in Regression) to implement the GOPN approach, it is presented in Figure 3.1.3. In this algorithm, we reuse the AGTC algorithm proposed in Chapter 2, and some processes to handle comparisons, execute model checking and update data.

Input: - *SourceFile2*: The source code of Lustre/SCADE in version 2
-*DatabaseSuite1*: The database of relation between Paths, Activation Conditions, Trap properties and Test case of version 1.

Output: - *T2*: The list of test cases for regression testing
- *T2all*: The list of all test cases for version 2
- *T1remove*: The list of test cases removed
- *T1reuse*: The list of test cases reused
- *DatabaseSuite2*: The database of relation

Algorithm:

1. **Begin**
2. *SourceCode2*=**readSourceCode***(SourceFile2)*
3. *T2=T2all=T1remove=T1reuse=* **initTestCaseList()**=null
4. *DatabaseSuite2* =**initDatabaseSuite()**=null
5. *P1=* **getAllPathInDatabase***(DatabaseSuite1)*

```
6.  P2= getAllPathInSourceCode(SourceCode2)
7.  Pa=P1
8.  Pb=null
9.  Pc=P2
    m=getCount(P2) // get the number of paths P2
    n=getCount(P1) // get the number of paths P1
10. for j=1 to m
11. for i=1 to n
12. if (compare(P2[j], P1[i])==0)
13. Pb=addPath(Pb, P2[j])        //Pb= Pb - P2[j]
14. Pa=removePath(Pa, P1[i])   //Pa= Pa - P1i]
15. Pc= removePath(Pc, P2[j])  //Pc= Pc - P2[j]
16. end if
17. end for i
18. end for j
19. T1remove= getTestCases(DatabaseSuite1, Pa)
20. T1reuse= getTestCases(DatabaseSuite1, Pb)
21. i=1
22. T2=AGTC(SourceFile2, Pc) //the AGTC algorithm proposed
23. Optimize(T2)  //Optimize the TestCaseList set
24. T2all=T1reuse+T2
25. UpdateDatabase(DatabaseSuite2, T2, T1reuse, T1remove)
26. Optimize(DatabaseSuite) //Optimize the database
27. End
```

Figure 3.1. The AGTR algorithm

Basing on the algorithm, two nested loops take O(*nm*). The AGTC algorithm takes O(*m*). So the complexity of the AGTR is O(*nm*).

Table 3.6. Comparing 3 approaches GSRS, GSCR and GOPN.

Solution / Criteria	GSRS	GSCR	GOPN
Input	Specification requirement in natural language	Specification requirement in SCR	Source code of Lustre program
Context of use	Black-box testing	Black-box testing	White-box testing
Deployment costs	High	Medium	Low
Automation capabilities	Low	Medium	Very High
Quality evaluation	Satisfying requirement based coverage	Satisfying: Requirement based coverage and control flow based coverage	Satisfying: Control flow based coverage and data flow based coverage
Meet the coverage criteria	No criteria	No criteria	100% for BC 100% for ECC 70% for MCC

Table 3.6 shows the comparison of GSRS, GSCR and GOPN. Depending on the context, we can use one out of the three approaches, the testing process can apply different approaches with different inputs. However, in three approaches, GSRS and GSCR have many difficulties in automation. So we decide to use GOPN approach for

automation regression testing of Lustre/SCADE programs. We have chosen to automate and develop the tool LUSREGTES.

3.4 Conclusion

Chapter 3 has discussed the solutions to generate test cases for regression testing of Lustre/SCADE programs. We studied and proposed 3 approaches: GSRS, GSCR and GOPN. We have compared the three approaches, identifying the appropriate and inappropriate characteristics of each approach to our problem. Most importantly, we propose to automate the process of generating test data for regression test based on special characteristics of Lustre programs which are operator network, paths on the operator network and activation conditions of provocation equivalent paths on the operator network.

Chapter 4. LUSREGTES: A REGRESSION TESTING TOOL FOR LUSTRE/SCADE PROGRAMS

4.1 Introduction

With the solution proposed in Chapter 3, we have developed a tool named LUSREGTES. This tool automatically tests data generation in regression testing for Lustre/SCADE programs.

4.2 Test Execution Environment

LUSREGTES mainly consists of a test data generator. It has two modules:

- Module generating test data for the first version;
- Module selecting and generating test data for regression testing for the second version.

Module 1 of LUSREGTES requires these inputs: The first version of Lustre program to analyze, length of path and the maximum number of loops in a path (that is, the number of times that a pre

operator will be repeated in a path). This module automatically runs Lesar tool with activation conditions.

Module 2 uses the result in module 1 and the new version of Lustre program as the inputs. It selects and generates test data for regression testing in the second version. The process consists of three automated steps as follows.

First, from the source code of program new version, the tool defines path collection. Then, the tool will automatically compare this collection with the paths collection of previous version.

Second, the tool will automatically perform model checker using Lesar. The result returned is the counter-examples.

The last step, the tool will analyze the counter-examples and identify test data for the regression testing process.

4.3 The LUSREGTES tool

4.4 Case study

In this section of this thesis, we will demonstrate its functions to test two systems: Heater Controller System and U-turn Section Management System.

4.5 Result evaluation

This section presented the comparison for three tools: LUSREGTES, Lutess and Lutess, based on the basic criteria of a testing tool.

4.6 Conclusion

In this chapter, we have presented the LUSREGTES tool developped based on the previously proposed techniques. The tool automatically creates test data for regression testing of Lustre/SCADE programs, identifies test data which should be removed and re-used. .

In addition, we also present two specific test cases to test and illustrate the correctness of the solution.

CONCLUSIONS AND FUTURE WORKS

Compared to the objectives, the thesis has achieved the results as follows:

– Studying the basic issues in regression testing and regression testing techniques, applications of regression testing, the state of the art in regression testing.

– Studying and analysing basic features of reactive systems, synchronous approach, the Lustre language and SCADE environment; identifing the structural coverage of Lustre programs; defining the activation conditions of paths on operator network of the equivalent Lustre programs.

– Figuring out the overview of model checker, using model checking in software testing. We have suggested the solution which uses activation conditions on the operator network from Lustre programs combined with model checker on main source of Lustre program to create test data for Lustre/SCADE programs. This solution helps to remove the manual input definition from the model checking.

– We proposed and experimented three approaches: GSRS - Generation of test cases in regression test using software requirements in natural language; GSCR - Generation of test cases in regression test using SCR; GOPN - Generation of test cases in regression test using operator network. Most importantly, we can propose the best suited solution to automate the process of generating test data for regression test based on special characteristics of Lustre programs which are

operator network, paths on the operator network and activation conditions.

– A regression testing tool is developped implementing the GOPN approach, called LUSREGTES. The tool automatically creates test data for regression testing of Lustre/SCADE programs, identifies test data which should be removed and re-used. This tool helps removing the unnecessary test data; therefore it saves considerable time and effort for regression testing process.

In addition to the obtained results in the thesis, some issues may be studied in the future:

– We intend to support the construction of reports for test management with test cost, test coverage, and defects found.

– The results obtained by the experimental evaluation showed that in many cases, one test suite satisfies several activation conditions. It would be interesting to investigate more thoroughly the coverage ratio that the criteria yield with regard to the number of the required test cases and hence explore ways of finding a minimal test cases set that satisfies all the activation conditions for a given criterion.

– To demonstrate scalability of the method and applicability to more realistic, complicated software systems, we plan to use larger specifications taken from real life examples.

– Currently, there are not many similar studies in automated regression testing for the Lustre/Scade programs, so evaluating our solutions and other solutions is difficult.

– The LUSREGTES tool developed in this work can be further extended in the future. We will study to develop more additional new functionality and integrate the existing tool.

PUBLICATIONS

[1] Trinh Cong Duy, Nguyen Thanh Bình, Ioannis Parissis, "Automatic Generation of Test Cases in Regression Testing for Lustre/SCADE Programs," *Journal of Software Engineering and Applications (ISSN: 1945-3124)*, P: 27-35, Vol.6, No.10A, 2013.

[2] Trịnh Công Duy, Nguyễn Thanh Bình, Ioannis Parissis, "Sinh ca kiểm thử tự động trong kiểm thử hồi quy cho các hệ thống phản ứng," *Kỷ yếu Hội nghị Quốc gia lần thứ VI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR)*, P: 484-494, 2013.

[3] Trinh Cong Duy, Nguyen Thanh Bình, "Applying model checking for regression testing based on requirements specifica," *13th International Symposium on Advanced Technology - ISAT13 - Engineering innovation for sustainable future*, P: 61-67, 2014.

[4] Trịnh Công Duy, Nguyễn Thanh Bình, "Sinh ca kiểm thử cho các hệ thống phản ứng sử dụng công cụ kiểm chứng mô hình NUSMV," *Chuyên san Kỹ thuật và Công nghệ, Tạp chí khoa học Đại Học Huế*, P: 55-66, Số 7, 2015.

[5] Trịnh Công Duy, Nguyễn Thanh Bình, "Xây dựng công cụ sinh dữ liệu thử cho chương trình Lustre/SCADE dựa trên kiểm chứng mô hình," *Tạp chí khoa học và công nghệ Đại học Đà Nẵng*, Trang 84-90, Số 9(94), 2015.

[6] Trịnh Công Duy, Nguyễn Thanh Bình, Ioannis Parissis, "Sinh dữ liệu thử cho ứng dụng Lustre/SCADE sử dụng điều kiện kích hoạt," *Kỷ yếu Hội nghị Quốc gia lần thứ VIII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR)*, P: 628-639, 2015.

[7] Trinh Cong Duy, Nguyen Thanh Binh, Ioannis Parissis, "A regression testing approach for Lustre/SCADE programs," *SoICT 2015 The Sixth International Symposium on Information and Communication Technology Hue City*, ACM Publisher, New York, USA, 2015.

[8] Nguyen Thanh Binh, Trinh Cong Duy, Ioannis Parissis, "LusRegTes: A Regression Testing Tool for Lustre Programs," *International Journal of Electrical and Computer Engineering (ISSN: 2088-8708, a Q2 SCOPUS indexed Journal);* Vol 7, No 5, 2017.