Automatic Test Generation for LUSTRE/SCADE programs

Virginia Papailiopoulou - Supervisor: Ioannis Parissis
University of Grenoble 1 - LIG
E-mail: Virginia.Papailiopoulou@imag.fr

1. Introduction

LUSTRE [2] is a declarative, data-flow language, which is devoted to the specification of synchronous and real-time applications. It ensures efficient code generation and provides formal specification and verification facilities. It is based on the synchronous approach which demands that the software reacts to its inputs instantaneously. In practice, that means that the software reaction is sufficiently fast so that every change in the external environment is taken into account. These characteristics make it possible to efficiently design and model synchronous systems.

A graphical tool dedicated to the development of critical embedded systems and often used by industries and professionals is SCADE (Safety Critical Application Development Environment). SCADE is a graphical environment based on the LUSTRE language and it allows the hierarchical definition of the system components and the automatic code generation. From the SCADE functional specifications, C code is automatically generated, though this transformation (SCADE to C) is not standardized. This graphical modeling environment is used mainly in the aerospace field (Airbus, DO-178B); however it is also used in the domains of transportation, automotive and energy.

Due to their criticality, testing of synchronous real-world applications is a significant part of the development process. Extensive studies have been carried out on testing focusing on the automatic test data generation. Several tools have been proposed, mainly concerning the formal specification processing to produce test input sequences. Some tools for programs specified in LUSTRE are Gatel, Lurette and Lutess. This research work is partially concerned with Lutess, a testing environment which automatically transforms formal specifications into test data generators.

In major industrial applications, it is also important to assess how thoroughly the produced test data has tested the corresponding specification. As far as programs written in sequential languages are concerned, several adequacy criteria have been presented in the past, such as path/branch coverage criteria, LCSAJs and MC/DC. Due to the fact that these criteria are not in line with the synchronous paradigm and cannot be applied on LUSTRE programs, especially designed structural coverage criteria have been proposed [4]. The tool LUSTRUCTU [3] automates the coverage assessment process for these programs.

Our research targets the amelioration of the testing process and the automation of the test data generation with a view to improving and simplifying the testing of real industrial applications, especially in the aerospace domain. Within this context, there are two main areas of interest. First, we study the structural coverage assessment with regard to LUSTRE programs. The existing criteria defined for the LUSTRE language should be extended to fully support the SCADE language. This extension is divided in three main aspects. First, it consists in taking into account the use of multiple clocks in a LUSTRE program. Second, integration testing as opposed to unit testing should be further studied and new criteria have to be defined allowing the coverage measurement of large-sized systems. Third, the simultaneous use of data-flow diagrams and finite state automata is also to be studied, since the new version of SCADE supports both. Moreover, we are interested in the automated test data generation aiming at the facilitation of the testing process. We focus on synchronous software testing using Lutess. In particular, we examine how the definition of an environment model, based on formal specifications, could guide the test data generation through different testing techniques according to a test objective.

The long-term goal is to obtain a concrete testing methodology, especially suitable for testing the control parts of avionic systems.

2. Background

2.1. The LUSTRE language

Contrary to imperative languages which describe the control flow of a program, LUSTRE describes the way that the inputs are turned into the outputs. Any variable and expression denotes a flow, i.e. each infinite sequence of values is defined on a clock, which represents a sequence of time. Thus, a flow is the pair of a sequence of values and a clock.

The clock serves to indicate when a value is assigned to the flow. That means that a flow takes the *n*-th value of its sequence of values at the *n*-th time of its clock. Any program has a cyclic behavior and that cycle defines a sequence of times, i.e. a clock, which is the *basic clock* of a program. A flow on the basic clock takes its *n*-th value at the *n*-th execution cycle of the program.

A LUSTRE program is structured into nodes; a node is a set of equations which define its outputs as a function of its inputs. Each variable can be defined only once within a node and the order of equations is of no matter. Specifically, when an expression *E* is assigned to a variable *X*, *X*=*E*, that indicates that the respective sequences of values are identical throughout the program execution; at any cycle, *X* and *E* have the same value. Once a node is defined, it can be used inside other nodes like any other operator.

The operators supported by LUSTRE are the common arithmetic and logical operators (`+`, `-`, `*`, `/`, `and`, `or`, `not`) as well as two specific temporal operators: the *precedence* (`pre`) and the *initialization* (`->`). The `pre` operator introduces to the flow a delay of one time unit and it is used to refer to the previous execution cycle. The `->` operator -also called *followed by* (`fby`)- allows the flow initialization and it is used to refer to the first execution cycle. The operators that affect the clock of a flow are `when` and `current`. The `when` is used to **sample** an expression on a slower clock. Let *E* be an expression and *B* a boolean expression with the same clock. Then *X*=*E*`when`*B* is an expression whose clock is defined by *B* and its values are the same as those of *E*'s only when *B* is *true*. The `current` operates on expressions with different clocks and is used to **project** an expression on the immediately faster clock. The expression *Y*=`current`*(X)* has the same clock as *B* and its value is the value of *X* at the last time that *B* was *true*. Note that until *B* is *true* for the first time, the value of *Y* will be *nil*.

A simple LUSTRE program is given in Figure 1. This program receives a signal `set` and returns a boolean `level` that must be true during `delay` seconds after each reception of `set`. The seconds are provided by a boolean input `second`, that is true whenever a second elapses. The program reuses the node `STABLE`, calling it at a suitable clock, by filtering its input parameters.

```
node TIME_STABLE(set, second:  bool; delay:  int)
returns (level:  bool);
var ck:  bool;
let
  level = current(STABLE((set, delay) when ck));
  ck = true -> set or second;
tel;

node STABLE(set:  bool; delay:  int)
returns (level:  bool);
var count:  int;
let
  level = (count>0);
  count = if set then delay
        else if false->pre(level) then pre(count)-1
        else 0;

tel;
```

Figure 1. Example of a LUSTRE program.

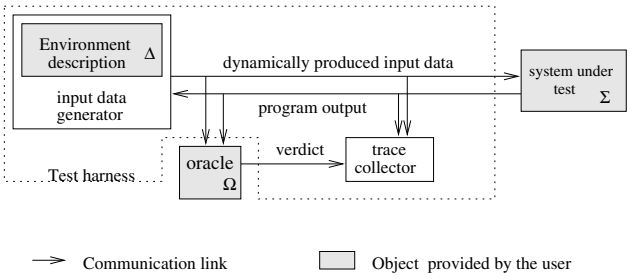

Figure 2. The Lutess testing environment.

2.2. The Lutess testing environment

Lutess is a "black box" specification-based tool designed to automate the test process in synchronous reactive software development.

To perform the test operation, Lutess [6] requires three components (see Figure 2): the software environment description (Δ), the executable code of the system under test (Σ) and a test oracle describing the system requirements (Ω). The system under test and the oracle are both synchronous executable programs. The environment description is composed of a set of properties, stated as invariants, that the system environment is assumed to satisfy. These invariants are LUSTRE expressions and constrain the possible environment behaviors.

Lutess builds a random generator from the environment description as well as a test harness which links the generator, the system under test and the oracle. Lutess coordinates their execution and records the input and output sequences as well as the associated oracle verdicts.

The test is operated on a single action-reaction cycle driven by the generator. The generator randomly selects and sends an input vector to the system under test; the latter reacts with an output vector and feeds back the generator with it. The generator proceeds by producing a new input vector and the cycle is repeated. The oracle observes the program inputs and outputs and determines whether the software specification is violated. The testing process is stopped when the user-defined length of the test sequence is reached.

2.3. Coverage criteria for LUSTRE programs

The Basic Criterion (BC), the Elementary Conditions Criterion (ECC) and the Multiple Conditions Criterion (MCC) are the three coverage criteria that have been especially designed for LUSTRE/SCADE programs [4]. Al-

though these criteria are comparable to the existing data-flow based criteria, they are not the same. They are of increasing strength and aim at defining intermediate coverage objectives and estimating the required test effort towards the final one. They are defined on the graphical representation of a LUSTRE program, the *operator network*, which is a directed graph that depicts how the input flows are transformed into output flows through a set of operators. The criteria are based on the notion of the path *activation condition*. Informally, this is strongly related to the propagation of the effect of the input edge through the output edge. Equally, a path is activated if every change in the input value causes a consequent change in the output value. Therefore, the selection of a test set satisfying the activation conditions for all the paths in an operator network leads to the program coverage.

The above criteria have been implemented in LUSTRUCTU [3], a tool that automatically measures the structural coverage of LUSTRE/SCADE programs. It requires three inputs: the LUSTRE program to analyze, the path length and the maximum number of loops in a path and finally the criterion to satisfy. The tool analyzes the program and extracts the conditions that a test input sequence must satisfy in order to meet the given criterion. Then it computes the coverage ratio achieved after the execution of a test sequence.

3. Motivations and Objectives

Even though the conducted research on testing and the current testing techniques have progressed well, there are still certain issues to be farther studied. Not only improving the validation and the testing of critical real-time embedded systems but also reducing its cost is of vital concern. Providing the means to fully automate the test data generation and consequently evaluate its quality consists in the primary interests of our research work.

Briefly, our main purpose is to enhance the theoretical and empirical results of the existing research work on the testing of LUSTRE/SCADE programs and adapt it to the real and current industrial aspects, with a view to providing tools and methods directly applicable on avionics applications. More precisely, structural coverage criteria for LUSTRE programs, integration and regression testing as well as the use of multiple clocks are some of the concepts yet not thoroughly studied that should also be taken into account. Moreover, specifying a complete and precise testing methodology based on the Lutess tool and tailored to industrial needs and real applications is a necessity.

3.1. Coverage assessment

One of the basic difficulties in the testing process is the lack of a formal relation between the test objective and the system formal specification. Indeed, the tester must ensure that the generated test cases cover the system functional requirements and the test objective that the latter define. Hence, coverage criteria specially adjusted to the synchronous paradigm (BC, ECC, MCC) provide an adaptable framework for evaluating the test data thoroughness of LUSTRE programs.

The existing LUSTRE coverage criteria cannot be performed on the complete operator set of the language; they can be applied only on specifications that are defined under a unique global clock. The global clock is a boolean flow that always values *true* and defines the frequency of the program execution cycles. Other, slower, clocks can be defined through boolean-valued flows. They are mainly used to prevent useless operations of the program and to save computational resources by forcing some program expressions to be evaluated strictly on specific execution cycles. Thus, nested clocks may be used to restrict the operation of certain flows when this is necessary, without affecting at the same time the rest of the program variables. Real systems are more complex and usually operate on multiple clocks. This fact would introduce modifications on the criteria definition. The object of our research is to extend these structural test coverage criteria to deal with LUSTRE flows defined on different clocks. That means that the activation conditions must be defined for the two temporal operators `when` and `current` which handle multiple clocks. These criteria must be also tailored to the latest version of SCADE that supports both data-flow and control flow graphs.

Aside from that, integration testing must be considered in addition to unit testing. That means that in order to compute the system coverage, the internal nodes that are used as single operators (like in the example of Figure 1) should not be unfolded. If already measured, the coverage of such nodes should be reused or otherwise approximately estimated. As a result, new criteria must be defined including a notion of abstraction to make it possible to measure the program coverage also in the case of large-sized programs without unfolding the internal nodes and, thus, leading to a huge number of paths, difficult to execute.

Finally, the criteria extension should also support the combined use of both data-flow diagrams and control flow graphs, which is the case in the new SCADE version.

Empirical evaluation will be carried out involving lab LUSTRE components as well as case studies extracted from the industrial domain. We are interested in complexity issues in terms of the cost of computing the activation conditions, the relative difficulty to meet the criteria and the fault-detection power of the latter.

This part of our research work is conducted within the framework of the SIESTA project (www.siesta-project.com), funded by ANR, the French National Research Foundation.

3.2. Automatic test data generation

In [5], a new version of Lutess is presented based on constraint logic programming. The testing process is filled in with various techniques depending on the way test sequences are generated. The main idea is, based on the problem specification, to enhance the system external environment by transforming the conditions that the latter must satisfy into formal expressions. Basically, the environment description represents an environment model consisted of all the constraints that describe the correct operation of the system under test. The goal is to obtain valid and suitable test suites in line with the system specification and the test objective. Apart from the random testing, Lutess supports the use of operational profiles to guide the test data generation. Moreover, aiming at the improvement of the ability of the generated test cases to detect faults or erroneous situations, testing can be guided by the safety properties defined in the oracle. This strategy can be further extended to use hypotheses on the system under test defined by the user that provoke property violations.

Within the scope of our research, we look into the composition of a testing methodology using Lutess, taking advantage of this progressively increasing approach of test data generation according to the environment description. This approach would not only provide the means to adequately test a system but also guide the system towards different modes of operation as well as the detection of faults. In particular, we are interested in establishing the basic steps towards the construction of the environment model in order to automate the testing process as much as possible without requiring further intervention by the tester. This implies that the conditions on the system inputs which should be invarianltly satisfied must be formally expressed. These conditions are also hypotheses under which the software is designed. With the intention of applying this approach on real-word avionic systems, we are interested in validating this methodology on case studies from this field and providing the guidelines in the procedure of testing a system in the Lutess environment.

4. Contributions & ongoing work

We are currently working on the coverage of LUSTRE/SCADE programs using one or more clocks. We have defined the activation conditions for the `when` and `current` operators and have applied these definitions to simple programs. The theoretical results of the coverage measurement are promising. The next step is to extend the tool for the automatic coverage assessment of LUSTRE programs developed in our lab to verify our approach in practice.

In addition, we have defined a testing methodology dealing with the new features of Lutess. We have performed the step-by-step environment model on a realistic and well known case study, the steam boiler control system [1]. The objective is to validate and assess the effectiveness of the tool as well as to evaluate its scalability in testing real-world applications, since it was so far evaluated on a simple reactive program example, an air-conditioner controller. So far, we have guided the test data generation using operational profiles. A general observation to obtain is that the effort needed for testing is not easy to assess as it depends on the desired thoroughness of the test sequences which may lead the tester to write several operational profiles corresponding to different situations. However, it is interesting to guide the testing process towards the violation of safety properties and the detection of faults, especially making some hypotheses on the boiler system.

References

[1] Jean-Raymond Abrial. Steam-boiler control specification problem. In *Formal Methods for Industrial Applications*, pages 500–509, 1995.

[2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[3] A. Lakehal and I. Parissis. Lustructu: A tool for the automatic coverage assessment of lustre programs. *In proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005)*, pages 301–310, Chicago, USA, November 2005.

[4] A. Lakehal and I. Parissis. Structural test coverage criteria for lustre programs. *In the 10th International Workshop on Formal Methods for Industrial Critical Systems (FMICS), a joint event of ESEC/FSE'05*, pages 35–43, Lisbon, Portugal, September 2005.

[5] Besnik Seljimi and Ioannis Parissis. Using clp to automatically generate test sequences for synchronous programs with numeric inputs and outputs. In *ISSRE*, pages 105–116, 2006.

[6] Besnik Seljimi and Ioannis Parissis. Automatic generation of test data generators for synchronous programs: Lutess v2. In *DOSTA '07: Workshop on Domain specific approaches to software test automation*, pages 8–12, New York, NY, USA, 2007. ACM.