

Model-Based Verification of Software Interface

Dinh Thang Pham

Faculty of Information Technology, Huflit University, Ho Chi Minh city, Viet Nam

Abstract: In Software Engineering, There is much technique to testing. They are Unit testing, Design testing, Code testing ... It depends on Development model of a project, the testing maybe before or after. In Viet Nam, the testing for program verification is almost by manually. This thing conducts many shortcoming or overload with big project. Although, there is many try to improve. Such as: present the requirement by XML or looking for many different approaches. But there is no an approach to test become engineering by automatically.

From that requirement, it needs a program verification of software interface. There is a tool for testing on the computer by automatically. With this thing, they can save much money, time, and effort.

Project is created with two concrete works. One of many importance things is choose a tool that is powerful enough to solve the counter examples. In this project, I choose the Altarica language as that tool. And so, it must be created the input data from XML that Altarica can readable. This operation must be implemented by a parser.

To this time, there are two functions Deadlock and Live that are implemented by successfully. And they are used for test run. In the future, there is more function that will develop for maximum support to design.

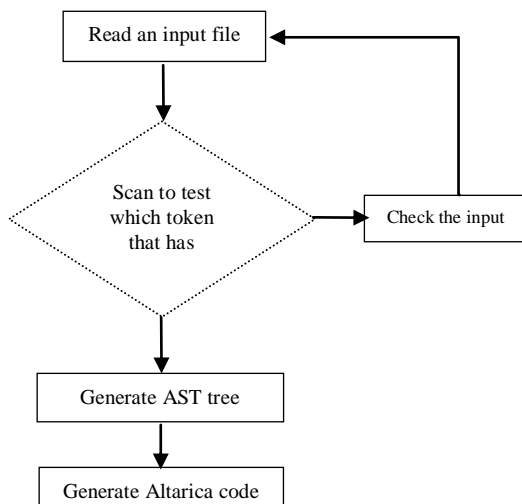
Keywords: Model-Based Verification, Model Checking, Altarica for Model Checking, Testing, Formal Design

I. INTRODUCTION

In this project, I concentrate on program verification. Concrete is investigated to test interface. The project is as diagram below:



From a specification, the program must transfer to kinds of data that Altarica can understand. This thing is implemented via a parser from XML to Altarica (XML2Altarica)



The result of this phase can create two importance define.

- Define of node.
- Define transactions between nodes.
-

The next uses the logic theory to test for counter examples. Such as: Live, Deadlock.... If it passes, this design will be accepted and after that is coding. And if it does not pass, this design will be examine again.

II. MODEL-BASED VERIFICATION

A. Model Checking

Model Checking is a part in a cycle testing. It is a logic testing and work with formal design. Model checking has been proven extremely useful to verify the correctness/completeness of a well-defined system. Especially, the model checking technique is able to generate counter-example when encountering a potential error/flow. Model Checking provide logic test on counter examples. This property makes model checking highly potential for automatic program verification. Model Checking can show all of case in a problem, it make we handle each of case. From there, we can save time, expenditure, effort.

B. Technique in program verification

In program verification, the general problem is to verify that a given program satisfies a given behavioral property or does not violate a given property (for example, mutual exclusion, absence of deadlock, or absence of livelock).

The combination of the automaton with the program transition graph is efficiently done on the fly. The program transition graph is in practice an abstract model of a program in which only key variables are retained (typically variables implementing synchronization and communication among processes). It is automatically obtained from an abstract program modelling language (e.g., Promela in SPIN) or a fully-fledged programming language (e.g., Java Path Finder that translates Java programs into equivalent Promela programs before verifying them. There is an others program that is powerful to resolves this part. That is AltaRica. This tool is not only feature fully but also vivacious for any case that user can interfere.

So, some tools are AltaRica, SPIN, Tina, PIPN, SMV....

C. Altarica Language

The AltaRica project is born from the wish of industrial partners and academic researchers to build gates between:

- The domain of Formal Methods and the one of Reliability and Risk Assessment.
- The quantitative analysis of dysfunctions and the qualitative analysis of functional behaviours.
- The tools and methods used for the modelling of systems.

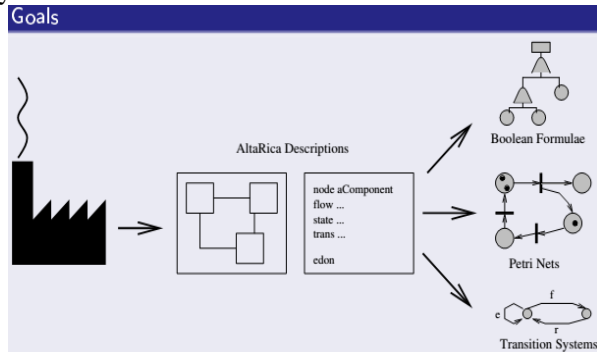


Fig.1 AltaRica Description

The aim of the project is to provide to designers an integrated workbench for the qualitative/quantitative analysis of complex and/or critical systems.

By many researches on various tools that serve for program verification. In this project, the model checking I use to check is AltaRica to solve the problem. The reason is I was learned this knowledge. Altarica is a tool that uses in many big projects such as: Airbus (Rosas, A350), Dassault System (Catia System), ClearSy (Atelier B)... in other side, this tool is developed by Labri and this is a promise for expanding and perfect tool in the future.

D. Dicky's Logics

All rule in testing is based on Dicky's logic. The logical property of an automaton can be seen as the set of all entities that satisfy the formula. This property can be checked by putting some mark during a depth-first-search algorithm on the reachability graph.

There are two kinds of properties for a graph $G(E, V)$. All state properties ($S \subseteq E$) and transition properties ($T \subseteq E \times V \times E$)

III. EXPERIMENT

There are two testing case in this experiment is deadlock and live

A. Deadlock

When you go from a node, through many transactions and go to another node that has no way to go. Now, there is deadlock. And the node that has no way to continuous is called dead node.

In fact, there is much design that exposed irrational when they meet this case. Almost the people must restart or turn off the application that is running

The following small example can describe deadlock and using Altarica to testing.

In this test case, there is a situation for a web project with some contents:

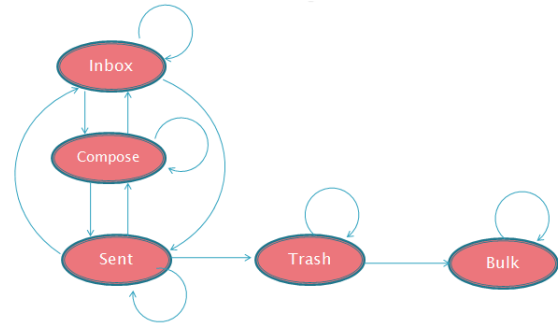


Fig.2 The symmetric with initial node: inbox (deadlock case)

1) Specification in XML:

Example for node **Bulk**

```
<role name="bulk">
  <status name="enable">
    <screen name="trash">
      <status name="view"></status>
    </screen>
    <screen name="bulk">
      <status name="view"></status>
    </screen>
  </status>
  <status name="disable">
    <screen name="inbox">
      <status name="view"></status>
    </screen>
    <screen name="compose">
      <status name="view"></status>
    </screen>
  </status>
</role>
```

2) Specification in Altarica:

```
node Button
state selected:bool:public;
event click:parent;
  off;
trans
  selected=false|-click->selected:=true;
  true|-off->selected:=false;
init
  selected:=false;
node Content
state loaded:bool:private;
event open,load:public;
trans
  true|-open->loaded:=false;
  true|-load->loaded:=true;
init
  loaded:=true;
edon
```

3) Testing for Deadlock case:

```
Define testing by logical
deadlock := any_s - src(any_t - self_epsilon);
```

This result points out there are 16 cases from initial node (inbox) that exist deadlock. And node Bulk is dead node

```

smile@t7: ~/Altairca_Tools/bin
File Edit View Terminal Tabs Help
arc>load test.spe
Loading file 'test.spe'.
/*
 * Properties for node : system
 * # state properties : 5
 *
 * initial = 1
 * any s = 108
 * state_bulk = 16
 * all_trans_to_bulk = 20
 * deadlock = 16
 *
 * # trans properties : 8
 *
 * any t = 276
 * epsilon = 108
 * self_epsilon = 108
 * not_deterministic = 0
 * livelock = 249
 * SCC = 192
 * self = 144
 * deadlock_at_bulk = 0
 */
TEST(deadlock=0) [FAILED] actual size = 16
TEST(livelock=0) [FAILED] actual size = 249
TEST(SCC=0) [FAILED] actual size = 192
TEST(deadlock_at_bulk=0) [PASSED]
TEST(state_bulk=0) [FAILED] actual size = 16
TEST(all_trans_to_bulk=0) [FAILED] actual size = 20
arc>

```

Fig.3 result of Deadlock

B. Live

Live is the case that there is as least a way to go, but never return initial node. This case make people cannot call the homepage, return main menu or main function when they go to child page or child function.

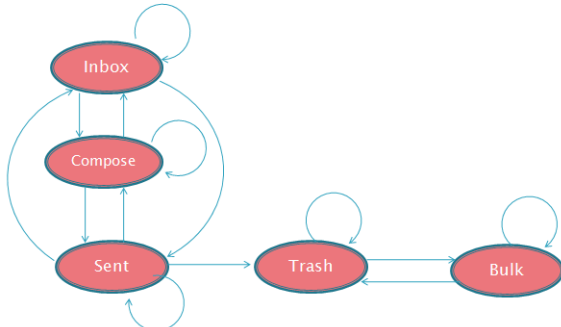


Fig.4 The symmetric with initial node: inbox (Live case)

1) Specification in XML:

Example for node **Inbox**

```

<root>
<role name="inbox">
  <status name="enable">
    <screen name="inbox">
      <status name="view"></status>
    </screen>
    <screen name="compose">
      <status name="view"></status>
    </screen>
    <screen name="sent">
      <status name="view"></status>
    </screen>
  </status>
  <status name="disable">
    <screen name="trash">
      <status name="view"></status>
    </screen>
  </status>
</role>
<screen name="bulk">

```

```

<status name="view"></status>
</screen>
</status>
</role>
.....

```

2) Testing for Live case:

Define testing by logical

`livelock := any_t-loop(rsrc(initial),any_t);`

The result shows that there is not deadlock. But there is live lock with 265 cases.

```

smile@t7: ~/Altairca_Tools/bin
File Edit View Terminal Tabs Help
arc>load speweb7.spe
Loading file 'speweb7.spe'.
/*
 * Properties for node : system
 * # state properties : 7
 *
 * state_trash = 38
 * any s = 105
 * initial = 1
 * state_bulk = 8
 * all_trans_to_bulk = 27
 * deadlock = 0
 * all_trans_to_trash = 20
 *
 * # trans properties : 9
 *
 * self = 140
 * deadlock_at_trash = 3
 * any t = 292
 * self_epsilon = 105
 * livelock = 265
 * SCC = 265
 * epsilon = 105
 * not_deterministic = 0
 * deadlock_at_bulk = 1
 */
TEST(deadlock=0) [PASSED]
TEST(livelock=0) [FAILED] actual size = 265
TEST(SCC=0) [FAILED] actual size = 265
TEST(deadlock_at_bulk=0) [FAILED] actual size = 1
TEST(state_bulk=0) [FAILED] actual size = 8
TEST(all_trans_to_bulk=0) [FAILED] actual size = 27
TEST(deadlock_at_trash=0) [FAILED] actual size = 3
TEST(state_trash=0) [FAILED] actual size = 38
TEST(all_trans_to_trash=0) [FAILED] actual size = 20
arc>

```

Fig .5 Result of Live

There is a transaction between them

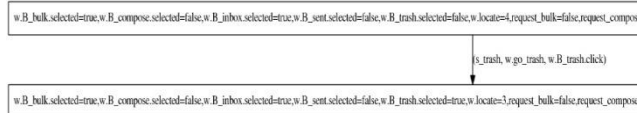


Fig .6 Bulk and trash

C. Applying Model-Based in program Verification

LARION Computing is a software outsourcing services and business Solutions Company. LARION is founded in 2003. The company has more 70 employees and recruiting.

Address:Room 10, Hall 5, Quang Trung Software City, Tan Chanh Hiep Ward, Dist 12, HCMC, Vietnam
Phone: (+84.8) 37155742

Website: <http://www.elarion.com>

At Larion Company, I received a request from a design. It is presented by a XML file. I same like:

```

<root>
<user_roles>
<role name="quan_tri_cao_cap">
<stateuses>
<status name="enable">
<screen name="Quan_Ly_Cau_Hinh" status="view"/>
<screen name="Quan_Ly_Nguoi_Dung" status="view"/>
<screen name="Quan_Ly_Nhan_Vien" status="view"/>

```

```

<screen name="Quan_Ly_Bao_Cao" status="View BCDVNgay"/>
</status>
<status name="disable"></status>
</stateuses>
</role>
<role name="quan_tri">
<stateuses>
<status name="enable">
<screen name="Quan_Ly_Nguoi_Dung" status="view"/>
<screen name="Quan_Ly_Nhan_Vien" status="view"/>
<screen name="Quan_Ly_Bao_Cao" status="View BCDVNgay"/>
</status>
<status name="disable"/>
</stateuses>
</role>
</user_roles>
<screens>
<screen name="Quan_Ly_Cau_Hinh">
<statescreens>
<status name="view">
<screen name="View_QLCH"/>
</status>
<status name="edit">
<screen name="Add_QLCH"/>
<screen name="Edit_QLCH"/>
<screen name="Delete_QLCH"/>
</status>
</statescreens>
</screen>
<screen name="Quan_Ly_Cau_Hinh_1">
<statescreens>
<status name="view">
<screen name="View_QLCH"/>
</status>
<status name="edit">
<screen name="Add_QLCH"/>
<screen name="Edit_QLCH"/>
<screen name="Delete_QLCH"/>
</status>
</statescreens>
</screen>
.....
</screen>
</screens>
</root>

```

Specification spe:

```

with system do
//test deadlock
Deadlock := any_s - src(any_t - self_epsilon);

//test livelock
livelock := any_t-loop(rsrc(initial),any_t);
//test SCC
SCC:=loop(any_t,any_t);

show(all);

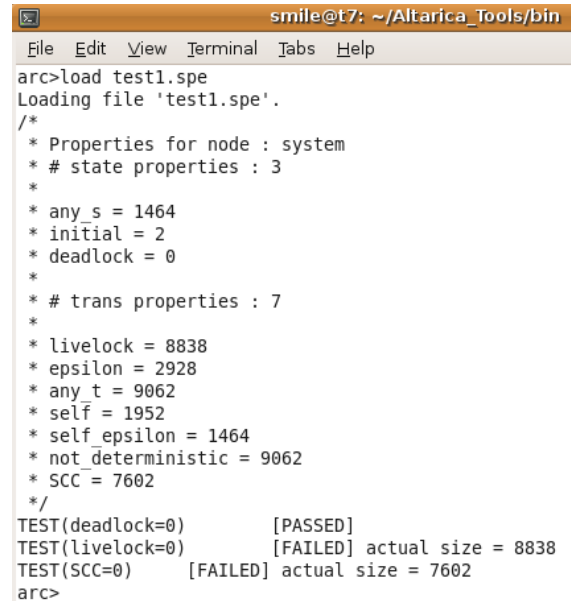
```

```

test(deadlock,0);
test(livelock,0);
test(SCC,0);
done

```

After test by this program the result of testing for livelock and deadlock is:



```

smile@t7: ~/Altarica_Tools/bin
File Edit View Terminal Tabs Help
arc>load test1.spe
Loading file 'test1.spe'.
/*
 * Properties for node : system
 * # state properties : 3
 *
 * any_s = 1464
 * initial = 2
 * deadlock = 0
 *
 * # trans properties : 7
 *
 * livelock = 8838
 * epsilon = 2928
 * any_t = 9062
 * self = 1952
 * self_epsilon = 1464
 * not_deterministic = 9062
 * SCC = 7602
 */
TEST(deadlock=0) [PASSED]
TEST(livelock=0) [FAILED] actual size = 8838
TEST(SCC=0) [FAILED] actual size = 7602
arc>

```

Fig.7 Result for fact case

As the result, this design has not deadlock. But there is livelock with 8838 cases.

Larion Company estimated this tool is good and it can be applying to fact.

IV. CONCLUSION

From the result of the work, we can apply to software engineering in Viet Nam. However, there are many works to develop such as input data checking, accepting multi input data from many kinds, or more functions, point out the trace that has Deadlock, Live....

By Dicky's logics, we can make many define for test. That is copious and satisfies with requirements. According to my think, this project will go far in developing and applying to fact.

ACKNOWLEDGMENT

I would like to thanks all of partner in this project. The knowledge from **Professor Alain Griffault** and **Professor Ann Dicky** has provided the foundation for this experiment. Besides, with guidance and advice from **Professor Thanh Tho Quan** and **Professor Huu Thang Bui** keep the job all right and can implement in fact. There is many development to applying this application. So, I need the coloration of all. Thanks again.

REFERENCES

- [1] Industrial Use of Formal Methods: Formal Verification, Jean-Louis Boulanger, 2013, chapter 3.
- [2] Altarica language. <http://altarica.labri.fr/>
- [3] Formal Design, Alain Griffault, 2007
- [4] Dicky's Logics, <http://www.dept-info.labri.fr/~dicky/>
- [5] Principles of Model Checking, Christel Baier, Joost-Pieter Katoen, MIT Press, 2008.

BIOGRAPHY



MSc. Dinh Thang Pham
Graduated from Bordeaux 1 –France
Model Checking, Logics, Network Security,
Reverse Engineering