Ada Runtime Error Generator

Functional Specification

Date:27/11/2020

Student: Derry Brennan

Student number: C00231080

Supervisor: Chris Meudec

DECLARATION

I hereby declare that this research project titled "Ada runtime error generator" has been written by me under the supervision of Dr. Christophe Meudec.

The work has not been presented in any previous research for the award of bachelor degree to the best of my knowledge.

The work is entirely mine and I accept the sole responsibility for any errors that might be found in the work, while the references to published materials have been duly acknowledged.

I have provided a complete table of reference of all works and sources used in the preparation of this document.

I understand that failure to conform with the Institute's regulations governing plagiarism constitutes a serious offence.

Signature: Derry Brennan   Date: 29/04/2021

Derry Brennan (Student)

C00231080 (Student Number)

The above declaration is confirmed by:

Signature: Chris Meudec   Date: 29/04/2021

Dr. Christophe Meudec (Project Supervisor)

Table of Contents

Table of Figures 3

1. Introduction 4

2. Project Description 4

3. Users 5

4. Use Cases 6

4.1 Ada Runtime Error Generator Use Case 6

4.1.1 Use case 1. 6

4.1.1.1 Brief Use Case 6

4.1.1.2 Detailed Use Case 6

4.2 Mika Visual Studio Code Extension Use Case 8

4.2.1 Use case 1. 8

4.2.1.1 Brief Use Case 8

4.2.1.2 Detailed Use Case 8

4.2.2 Use case 2. 9

4.2.2.1 Brief Use Case 9

4.2.2.2 Detailed Use Case 10

6. FURPS+ 11

6.1 Functionality 11

6.1.1 Core Functions 11

6.1.2 Secondary Functions 11

6.2 Usability 11

6.3 Reliability 11

6.4 Performance 11

6.5 Supportability 12

7. Metrics 12

8. Testing 12

Table of Figures

Figure 1 Ada runtime error generator Use case diagram 6
Figure 2 Mika Visual Studio Code Extension Use Case 8

1. Introduction

As technology expands throughout the world, taking control of complex tasks such as avionics and missile control, the need to make sure such software is free from as many errors as possible is of paramount importance. Programming languages have many forms of error checking in place already, with the integrated development environments having both semantic and syntactic errors being detected as the code is being written. But runtime errors are more difficult to find and not as much research and development has gone into the finding of such errors before.

Runtime errors such as division by zero, integer overflow and index out of bounds errors can cause a program to output unexpected results or to cease functioning entirely, neither of which is a good result, especially where lives are at stake.

Ada is a programming language that has its focus on safety and was seen as an ideal candidate to provide the testing ground for such a tool.

The Ada runtime error generator will be a tool that can be used by an Ada programmer who wishes to test their code for the presence of possible runtime errors.

2. Project Description

It will take an Ada program as its input and then the Mika tool, a test data generator will provide test data to achieve one hundred percent branch coverage of the code. The code will then be run with the test data as its inputs and the presence of runtime errors will be determined.

The tool will be constrained to a very limited subset of Ada that excludes external library calls.

The tool aims to detect division by zero errors, integer overflows and the possibility of detecting index out of bounds errors.

The tool aims to provide a detailed response to the user about what type of runtime error was encountered and where it happened in the code, along with the variable values that lead to the error. And the possibility of having an exportable version of the report in LaTeX or a format of their choosing.

With this information the developer would be provided with suitable information to implement a fix to prevent such errors from occurring in the future.

The possibility of adding annotations to the code where the error was found is also under consideration.

The test data generation will have to be steered towards values most likely to cause runtime errors and through methods such as symbolic execution the project aims to see if those values are possible within the constraints of the given program.

An additional feature of the project is the development of a Visual Studio Code extension to complement the Mika program that would allow a user to select a line of code and generate test input for the code that would lead a supplied boolean condition supplied by the user to be true.

This extension was selected to be used in Visual Studio Code as it is a very popular text editor and has a broad ecosystem of extensions.

3. Users

Power user: Ada programmer.

This user would be comfortable with both the GUI and the command line interface for interacting with the tool. They would understand terms related to programming and runtime errors and do not need too much extra explanations about terms.

4. Use Cases

4.1 Ada Runtime Error Generator Use Case

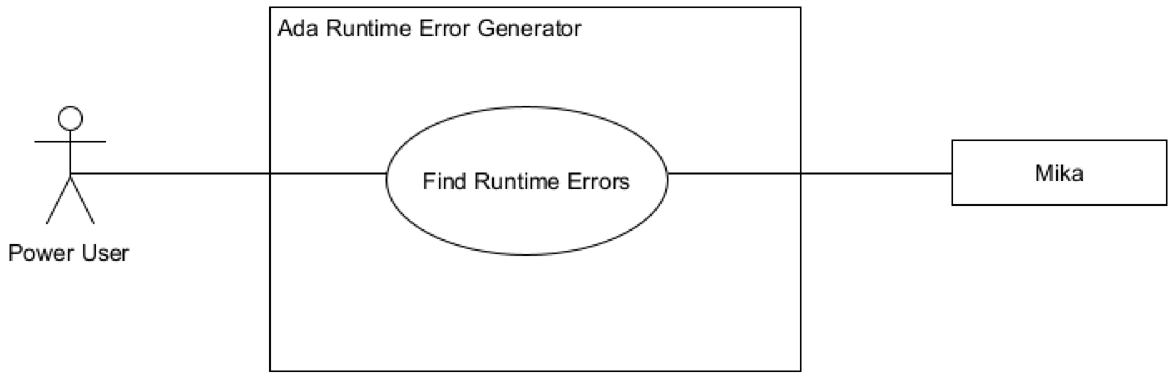

Figure 1 Ada runtime error generator Use case diagram

4.1.1 Use case 1.

Find Runtime Errors

4.1.1.1 Brief Use Case

Actor(s): Power User

Description: This use case begins with a power user supplying Ada code to the Mika software. Beginning the parsing of this code which generates a prolog file of the source code. Then selecting the exception flag as a condition for running. Mika will perform symbolic execution on the supplied code looking for variable conditions that would lead to a runtime error and supply back a report to the user detailing the error found if any.

4.1.1.2 Detailed Use Case

Use Case : Find Runtime Errors

Scope : Supply code and generate test inputs that would cause runtime errors if present

Level : Power User goal

Primary Actor : Power User

Stakeholders and Interests:

Power User: Wants to detect the possibility of runtime errors in the supplied code.

Preconditions: Developer has windows 10 environment with Sicstus Prolog and GNAT 2010 installed

Success Guarantee : Detection of runtime error if any present, lack of finding runtime error still does not guarantee code safety just gives an indication that no errors were found.

Main Success Scenario:

1.1 The power user selects the file that they want to carry out tests for runtime errors on.

1.2 The power user then parses the file to generate the prolog code.

1.3 The Power user then selects the exception flag as a condition for the generation of test inputs.

1.4 The power user conducts the test code generation.

1.5 Mika supplies back test input generation information with the variable values that cause the runtime error to occur.

Alternatives:

1.5 No runtime errors found.

1.5.a Mika supplies a back report with no test input as no conditions could be met to generate a runtime exception.

4.2 Mika Visual Studio Code Extension Use Case

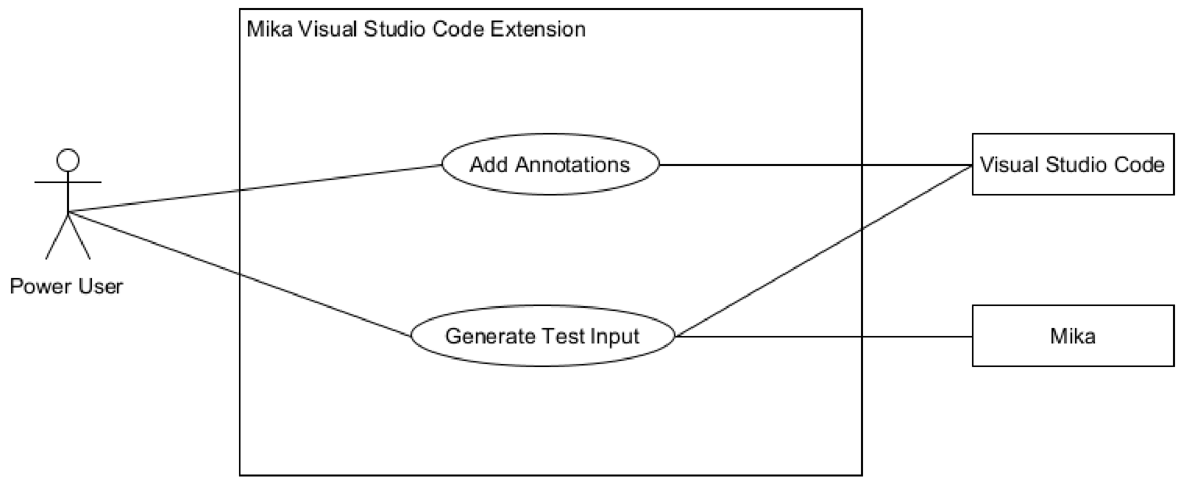

Figure 2 Mika Visual Studio Code Extension Use Case

4.2.1 Use case 1.

Add Annotations

4.2.1.1 Brief Use Case

Actor(s): Power User

Description: This scenario begins when a Power User wants to add a special comment to an Ada file that will allow them to generate a test input report for the variables mentioned within the comment.

4.2.1.2 Detailed Use Case

Use Case	: Add Annotations
Scope	: Add special comment into the specific line of code
Level	: Power user Goal
Primary Actor	: Power User

Stakeholders and Interests:

Power User: Power user wants to add a comment into their code that will allow the extension to generate test input.

Preconditions: The code supplied is Ada code and open in the main tab of visual studio code.

Success Guarantee: If the special comment is inserted into the selected line of code.

Main Success Scenario:

1.1 The Power User opens an Ada file in Visual Studio Code.

1.2 The Power user moves the cursor to the desired line to insert a Mika comment.

1.3 The Power user then selects the command menu in Visual Studio Code and chooses “Mika Ada annotations”.

1.4 The comment is inserted at the cursor's position in the code.

1.5 The Power User then alters the default area of the comment to their desired boolean condition.

4.2.2 Use case 2.

Generate Test Input

4.2.2.1 Brief Use Case

Actor(s): Power User

Description: This use case begins after adding a Mika comment is inserted into the code and the condition is specified, the Power User then runs the “Mika generate test inputs” command from the command menu in Visual Studio Code. A report is then generated and displayed within Visual Studio Code if Mika finds conditions that make the supplied comment be true.

4.2.2.2 Detailed Use Case

Use Case : Generate Test Input

Scope : Generate Test input for variables at a specific line of code to match supplied condition

Level : Power user Goal

Primary Actor : Power User

Stakeholders and Interests:

Power User: Wants to generate test input that will make a condition on a certain line of code true for the supplied variables.

Preconditions: A Mika comment has been inserted within the code and that the syntax of the boolean condition provided is valid Ada

Success Guarantee: A Report is displayed in an adjacent tab to the source code with values for the variables that would make the supplied condition true.

Main Success Scenario:

2.1 Power User opens the command Menu.

2.2 Power User selects the "Mika generate test inputs" command.

2.3 A report from Mika is displayed in a tab Adjacent to the source code.

2.4 The Power User can then save this document in a location of their choosing.

Alternatives:

2.3 The provided or default Mika or GNAT paths within the extension are invalid.

2.3.a An Error Message is displayed to the Power User stating this fact.

6. FURPS+

6.1 Functionality

The functionality of the Ada Runtime Error Generator is mainly in the detection of errors, but there are some other secondary functions to be considered too.

6.1.1 Core Functions

- Detection of division by zero
- Detailed report on found error
- Detection of integer overflow
- Detection of index out of bounds

6.1.2 Secondary Functions

- Visual Studio Code extension for Mika

6.2 Usability

Ada runtime error generator requires Windows 10, the Mika test code generation software, Sicstus Prolog and GNAT GPL 2010.

Other platforms are not supported yet, and the target language is Ada.

6.3 Reliability

The tool should be able to recover from failures 95% of the time and the tool should run 99% of the time.

6.4 Performance

Addition of the runtime error checks should keep in track with the performance of the Mika tool.

6.5 Supportability

The code must be written with maintainability in mind with a detailed research document detailing the steps taken to reach the desired outcome.

7. Metrics

To determine the success of the project to correct detection of runtime errors within supplied test code should be found, the number of achievable runtime errors that checks can be implemented for has yet to be determined.

- It can detect errors within test code.
- It can run its tests within 10 seconds 95% of the time, highly dependent on complexity.

8. Testing

Testing will be carried out using sample Ada code used to target the runtime errors being targeted and an attempt to find suitable Ada code on a resource such as GitHub to carry out tests on will be undertaken to test the boundaries of the tool.

9. Conclusion

The project is mainly a research project that requires writing additional functionality into the already existing Mika tool to support the handling of runtime errors. The viability of whether this is even possible has yet to be determined and hopefully the above functionalities will be met if it does indeed prove to be possible.