# Ada Runtime Error Generator

Project Report

Date:01/04/2021

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

# 1. Introduction

This project was aimed at the detection of runtime errors within Ada code and commenced in October 2020 and ended in April 2021. This Project was conducted as a fourth year module from the Institute of Technology Carlow in preparation for a software development degree.

The final report breaks down the project into sections. The project description first details the overall aim of the project and what was intended to be produced. It will then examine the conformance to the specifications that were detailed in the functional specification document and if there were any features that were not achieved or additional features added. Finally, it will detail the accumulation of the knowledge gained from this project in the learning outcomes section and the review and conclusions of the project in the Project review section.

# 2. Description of Project

The main goal of the project was to find the presence of runtime errors within Ada code and supply the inputs that would cause these errors to occur back to the developer, along with a description of the error found and on what line of code it was encountered. This would use a test input generating software called Mika which already had the functionality of supplying test inputs for coverage strategies of branch, decisions and modified condition / decision coverage (MC/DC). The aim of the project was to add a new coverage condition of “exception coverage” to the tool.

Mika uses symbolic execution and prolog to generate the test inputs for its desired coverage option supplied. The prolog file is generated from the source code using a custom Ada language file, ada.y, a Yet Another Compiler Compiler (yacc) file where the decisions and branches and exception stages of the code are defined as prolog relationships. Prolog’s intrinsic backtracking ability allows it to follow a path down until it finds an answer to the supplied query or if it turns out to be false, to go back up a step and take another path.

The initial starting point of the project was to write additions into the ada.y file that would build prolog relationship statements for runtime exceptions where they were possible to be found within the Ada language and to have the symbolic execution determine if there were variable values which would make this condition true.

An additional idea for a Visual Studio Code extension was also incorporated into the project. This would allow a developer to open an Ada file within Visual Studio Code and insert a special comment onto a specified line that would have a boolean condition in it, such as X = 5.

Once the comment was in place they could run a command that would call Mika to provide test inputs for the code that would make this condition be true at the specified line the comment was entered.

## 3. Conformance to specification

The project mostly conformed to the initial specifications in that implementations of the division by zero and the array index out of bounds exception checks work within Mika. The division by zero check was a success and had hopes high for moving on to the more complex exceptions, but when the array index out of bounds was being implemented it quickly became clear that the parser was not the ideal place to be implementing these checks, as it has no typing information and is quite simplistic.

Furtherance of the project would require a change in direction of implementing the checks within the symbolic execution where all typing information is available.

An additional area of the project was introduced where an extension for Visual Studio Code was to be implemented that would allow for dynamic code querying. Two functional commands are provided by the extension to accomplish this. One simple command: “Mika Ada Annotations” inserts a boilerplate comment on the highlighted line of code, allowing the developer to insert a boolean condition they would like to generate variable values to fulfil.

The next command: “Mika generate test inputs” makes a copy of the source code the comment was inserted in and programmatically inserts a new procedure into which the provided boolean condition is entered as an argument. Next the extension calls Mika on this file with the -Tquery flag and the test inputs if any are present are returned in a json file. The extension then parses this json file and displays the results in a tab beside the source code.

The overall conformance to the specification was not full as I was unable to provide support for all the runtime errors that were hoped to be met due to the unforeseen typing problem encountered while implementing the array out of bounds exception checks.

However, the overall value of the project was then supplemented with the Visual Studio Code Extension. This is a valuable addition, being able to provide real values to the developer to meet the conditions they want to test at any stage of their code from within Visual Studio Code.

# 4. Learning Outcomes

Throughout the project I was exposed to numerous learning outcomes, both technological and personal. Over this section I will detail what my personal experience was and how it helped me to grow  as a developer over the course of the year.

## 4.1 Ada

Before commencement on this project I had not even heard of Ada as a language, so it was necessary to familiarise myself with it. It is a somewhat niche language, but it is very good at what it does right, being safety and validation and verification of the code produced.

Ada’s natural language approach to the language was pleasant, I prefer the usage of English words where possible over their symbolic counterparts e.g. ‘and’ vs ‘&&’ couples with the absence of curly brackets made the produced code both look and read well. This is something that over the years has stood out to me; you may only write a program once, but you come back to it numerous times to either enhance it or just to reread it and having the code be more easily readable and understandable is priceless.

One of the most notable features for me was also the ability to create custom types. If you want an integer that goes from 0-100 and nothing else is allowed you can create your type, call it what you want and set the range. Now anytime you use this type Ada will enforce your type rules on it as it would any standard type. This can be both a blessing and a curse. With everything in Ada being a type it was a bit frustrating at the start to get used to, but once I realized its full potential of steering the developer towards a more stable program overall I was sold on this feature.

Working with Ada has helped me to learn how its techniques and features are applied to solve problems faced by aviation and military companies. By enforcing strict rules from the inception of a project you can prevent or drastically reduce the possibility of drastic errors. This will help me immensely in progressing through my career.

## 4.2 Symbolic execution

Symbolic execution was also unknown to me prior to this project. Its practicality in application has only recently become more prominent with the different solvers used in conjunction with it having achieved sufficient power to make the application of this technique more practical.

Reducing the variables down to a symbolic term and then producing boolean equations that match the source codes branches and conditions, the solver can then, given the constraints of those variables, provide an answer as to whether this boolean condition could ever be true and provide the values that it found to make it true. Each branch or condition can be executed independently and once they reach an end point a solver can then compute a concrete value for the variables, if one is possible.  This can be used to navigate very complex code following certain paths down to the desired point allowing for much faster traversal through code than coming up with inputs that would get you to the same area manually.

It is not a one size fits all approach. As the size of the code under test increases so do the possible paths that it will need to track, this ‘path explosion’ is made even greater once loops are taken into account. But even with its limitations symbolic execution and its ability to find values that will progress through code to certain sections is very valuable in the validation and verification of code. It takes a great deal of time to manually calculate values to do the same for a developer and knowing that this is a technique that can be put into practise is a big takeaway from the project.

## 4.3 Flex

My understanding of the lexical analysis step was part of my research and although I had some knowledge of this prior to the project, getting to look inside and interact with the ada.l file was fascinating. Seeing how the language lexemes are broken down into the tokens was very instructive. The number of tokens for such a complex language as Ada was also fewer than I would have thought with the whole ada.l file coming in just over 850 lines of code.

With the greater understanding of lexical analysis, I now feel able to write up the language patterns for something customised. The ability to write your own language is both empowering and practical.

## 4.4 Yacc

The Yet Another Compiler Compiler (Yacc), or in the case of this project the updated bison version, was where most of the practical work of the project was proposed to take place. This file holds the information on the language's grammar, uses the tokens produced by the lexical analyser, and also contains C code for the actions to be to be taken at certain stages. In the project, this was the automated writing of a prolog file of the source code provided.

The ada.y file in the project was incredibly intimidating, coming in at over 5000 lines of code and a long portion of my time in the project was aimed at reading through it and gaining a comprehension of what was going on in order to better understand where to apply the checks for the runtime errors that were being looked for. In hindsight, the length and complexity of this file should have not been the mental hurdle for me that initially it was due to the fact that the file structure does follow a logical order and regardless of its length is just another file. Being able to make additions in this file and see their outputs start to produce results was a good feeling.

Working with both Flex and Yacc helped me to learn how to manage myself when confronted with large daunting files that seem initially to be very complex, this will help me in future when working within other large scale projects that also contain such files.

## 4.5 Prolog

As part of the research for the project, before I knew exactly what area I would be working on I spent some time looking into prolog. I knew that Mika used prolog in conjunction with the symbolic execution to produce the test input values and I figured it would be a good Idea to familiarise myself with its workings.

Prolog's logical programming was intriguing. The idea of relationships and facts making up the language and thus allowing you to explain what you would like to accomplish and it telling you how to achieve that comes counter to regular programming where we provide the steps to achieve a desired goal.

Prolog also provides a unique backtracking ability that makes its queries extremely useful. If it follows a branch of execution to a point where the query cannot find an answer, but all query branches have not been taken, the variables binding since the last branch taken are discarded and it returns back to that choice and follows a different branch. This can be seen as somewhat similar to a depth first search.

My limited exposure to this language has left me craving more and hopefully this is where the ultimate answer to the project's proposed goal of finding runtime errors in code can be accomplished. Prolog's innate backtracking ability allows it to quickly do tasks that would otherwise be very complex to do, this feature coupled with its logical programming is something that when it can be applied is going to be on my mind for other large projects.

## 4.6 Visual Studio Code

A proposed addition to the project was to add an extension to Visual Studio Code for Mika that would allow code to be queried in the text editor, such as 'find me test inputs that make this condition true at this line of code'.

Visual Studio Code is my preferred text editor already and as a user of a number of useful and different extensions already I was very happy at the idea of also creating our own.

This brought up the difference in experiences between being a user of a product and being a developer within said product. On starting the research into how one would commence development of the extension, I learned quite a lot more about Visual Studio Code than I had previously known.

Most interestingly of which, for me personally was that Visual Studio Code is made with electron [1] which is an application framework made using Javascript, HTML and CSS. Maybe naively, I had thought it was made using C# being a Microsoft product, but the flexibility and customization options on this framework are amazing.

Both working on and with Visual studio code and managing the project from within it has made the management of a large scale project like this much easier. Coupled with Git for version control made keeping track of the files and changes being carried out over the course of the project much easier and every experience with both is a crucial learning experience for me as I know from my work experience how important these are in the industry.

## 4.7 Javascript

After the decision to add an extension to Visual Studio Code was made, I looked into how they were made. Here was a choice between Typescript or Javascript for the language to write in. Since I had some experience with

Javascript for my internship I decided this would be a good choice. In this project I worked with node.js, the package manager and I ended up using a few packages from within this.

This filesystem package fs was great for making and removing directories and copying files between them. The vscode package has all the commands for interacting directly with Visual Studio Code such as inserting text into the current open file.

The glob package was used for pattern matching. It uses patterns to match the desired file. I used it to get the folder name of a folder with a generated name based on date/time that would have been very difficult to determine otherwise.

And lastly I used the child process package, without which I couldn't imagine how to make sure Mika had finished running its reports before looking for the output from Mika.

Having all these packages at my disposal during the project was fantastic. These packages give you a way of accomplishing difficult tasks without having to reinvent the wheel. If a tool is available it is better to use that tool than to spend much more time constructing your own tool for the same purpose. I have to say that javascript has gotten a bit of a bad name for itself among programmers, but I found the process very enjoyable. I can see why typescript was needed too though; as was mentioned earlier in the Ada section having types is very helpful if a bit frustrating at the start.

Having a shell to also debug in was very helpful and it is something that I will miss if going back to Ada to program. In future projects where a shell is available for a language I will be conducting my prototyping of algorithms here. This gives me a greater understanding of how all the function calls work and being able to see them in action before applying them to the code base of a project. Being able to quickly test out an idea in the shell and make small adjustments to it before putting it into the source code is really something all languages that can do this should do.

## 4.8 Python

During the project a step that was needed was to try to find real world Ada code to test the runtime error generator on. There were a few limitations on Mika, one being that it cannot handle code with external libraries. Trying to browse GitHub for code that matched our requirements proved fruitless and so the decision was made to write a web scraper to do the searching for me. I chose Python here because I had heard it had a very powerful library called ‘beautiful soup’ [2]. Between Python and beautiful soup the process was much easier than I imagined. I scraped the webpage ethically too leaving a 3 second gap between actions so as not to over tax the website.

Unfortunately the web scraper did not produce any usable code to test upon, but the process of making it was very beneficial giving me greater knowledge of HTML and Ada also. Every step even if unsuccessful is still a step forward. Knowing how to perform such a step as web scraping enables me to reach into my quiver of knowledge and draw upon this in the future whenever it is applicable.

My experience here with python being an extremely popular language and particularly its application in data science will be very beneficial to me as it is an area of further study that appeals to me.

## 4.9 C

Within the Yacc file Ada.y file C is used to provide instructions once a certain token is found. I had experience with C++ in third year but never dealt with C until now. As the C sections in the Ada.y file were dealing with constructing strings to then write into the foo.pl file to generate prolog from the source code, the allocation of memory for each string was needed. Using malloc you needed to basically count the characters that you would be using in the string and assign the correct value to the allocated. This low level management of a program was foreign to me, but it certainly does give the developer far greater control over every step of the process.

## 4.10 Mika

Mika, the Ada test input generator developed by Dr. Chris Meudec, was at the heart of the project and to fully understand and progress with the project I had to thoroughly familiarise myself with this software. There is a graphical user interface (GUI) for Mika and also a command line interface (CLI); both provide the same features but for development the CLI was mostly used.

Through the documentation available, a user guide and a developer's guide, I was able to get everything up and running locally. The process was not without a few hiccups along the way. For example, building the parser within Visual Studio proved to be quite complicated. But having followed the documents and working with the software I feel that I gained a good understanding of the inner workings of this software. It provides great insight into code and is a huge time saver for developers.

## 4.11 Time Management

Time management has never been a strong suit of mine. Being on time for appointments or meets is fine, but dividing my time up into productive sections has always been very difficult for me. Especially with studying from home during coronavirus, staying productive has only gotten harder.

Over the year I did start to implement timetabled sections of time for myself to tackle the project exclusively and this proved to be very focusing. If I had had this ability from the start of the year, I feel it would have reduced a lot of the stress felt as the deadlines were approaching. I allotted certain blocks of time to just project work outside of the allotted hours during college. With 3 hours on each Sunday, a further 2 hours on each Tuesday and 3 more hours on Fridays to make sure I maximise my time well.

Now knowing that in advance the importance of spreading the workload out over the full length of a project I feel will make me generally a more productive student/worker. This is especially true of a research project where there are endless possibilities of reading and expanding your knowledge. Taking my learning from this year into future endeavours.

## 4.12 Presentation Skills

I always enjoy doing presentations, even if the nerves do get to me a little.  I feel they are a great way of demonstrating what you have been doing and an opportunity for you to show off your work in the best light, through your own words.

The two presentations this year on the project were very difficult, but rewarding. Trying to condense down such a project into just five minutes and have it make sense and still be engaging and not lose people in technicalities was the goal and I feel I accomplished this well.

Getting direct questions from the group also gives an opportunity to further demonstrate your knowledge on topics not necessarily prepared for. I feel that with each presentation I am getting better at public speaking and being able to get across complex topics in an interesting and engaging manner. Getting across complex topics in a clear and concise manner is not always easy as people's attention may drift if not engaged quickly. Being able to pick out the key ideas you want to deliver and engage people with those to get them on board and invested in your presentation is key. As I feel the topic of this project is quite complex to explain to people not fully familiar with the topic, presenting this topic this year has been an invaluable experience. It prepared me well to make future presentations on equally complex topics and bring my experience for this year to them also.

### 4.13 Research Skills

With this project being primarily a research project, it was imperative that research was the focus from commencement. I read quite a lot to familiarise myself with the topic and task at hand, but I do feel that my research skills at the start of the project were lacking. With everything that I was reading and learning, I was not necessarily properly documenting everything that I was doing. Of course, the point of the research project is not to just familiarise myself with the topic but to produce documents that will do the same for others in the future.

The project put the focus on the experience of trying to write good literature, reading others works in the area to understand the topics and procedures conducted. This enabled me to not retread old ground and spend time doing what someone else has already done.

Knowing this now I feel that if I was to engage in future research, I would be better equipped to produce better results overall. Additionally the differences between a research project and a regular project where the goals are clearly laid out from the start, something that was difficult for me to understand over the course of the project.

This understanding came late to me within this project after a colleague of mine reminded me about the complexity of the task at hand compared to the construction of an app. They are both beneficial but are hardly comparable. If I had gotten this understanding cemented in my mind from the beginning it would have been a much less stressful year. In future I will spend more time working at the task at hand and less time worrying about what has to be done. This is a useful skill to bring forward in life and not just in software development.

## 4.14 Technical writing

From an academic perspective, technical writing may be  my weakest area. Many times writer's block has struck, and the direction of some documents have felt weak and not well structured. I have used technical templates to help add initial structure to my documents and provide a logical flow to the writing. I have read some writing articles to try and improve myself in this area and also asked for help from people who are more experienced in this area to read over my documents and provide me feedback on improving.

I do feel that the experience has improved my skills, but there is still a large amount of room for improvement in this area. I’m sure all that it will take is more practice and as a result I am considering a summer course in technical writing.

## 4.15 Collaboration

Collaborating with my tutor on this project was very enjoyable; seeing how certain aspects were approached gave me ideas to take onboard myself. One such example was a work diary keeping track of whatever you are working on each day. I found it is a very good idea to remind yourself about important discoveries and as a form of documentation that you can later use while writing the more formal documents.

Doing the documents online in Google Drive, while not having the same presentation of a latex document certainly did allow for immediate and useful feedback between my tutor and myself on topics, where side discussions in the comments gave greater understanding of what was being written about.

Also collaborating with my fellow colleagues was very beneficial in talking out a tricky area of research or tackling a complex area of coding. A problem shared is a problem halved. After being stuck on an area of the extension where terminal commands needed to be run sequentially before the program progressed, I got in a call with a colleague and between us talking it through and exploring options together the problem was solved very quickly. Alternatively I have also helped provide support to my fellow colleagues on areas that caused them to get stuck. Oftentimes you just need fresh eyes on a problem or even to simply explain the problem out loud and the answer will become apparent.

I do feel I work well with others. Having worked on a number of group projects over the course of this degree and also being part of a production team while on work experience. I do feel that I am most productive when working closely with others and enjoy having the company too. Although I think in this project I was slightly embarrassed to share and ask for help from my tutor on areas I didn’t fully comprehend. It may have been that talking to fellow students who are in a similar situation of learning to me seemed more manageable than admitting my shortcomings to an expert felt difficult to broach. My communication could have been better overall, with more back and forth communications about all stages of the project rather than just waiting until the weekly meeting to discuss what had been done. I will take this as a major learning outcome from this project. In the future I will not be as self conscious about my own lack of knowledge in an area and bring up my problems with those with the skills to direct me in the proper direction. Learning is difficult enough without letting a fragile ego hinder it further.

# 5. Future Developments

Over the course of the project I have developed an understanding of the techniques involved applying this to find real world solutions to the problem of finding runtime errors in code.

With that in mind I feel that given a few more months to work within the symbolic executor and apply the necessary checks there instead of within the parser as was the initial goal that more substantial progress could be made than the division by zero and the array index out of bounds.

Also there is the possibility of applying this same technique to other more widespread languages such as C, this would require a building from the ground up of the necessary systems and files necessary, but the underlying principle would be the same.

Being able to branch out into a more widespread language would also provide an abundance of existing code bases that could be used for testing purposes, which unfortunately were very hard to find for Ada over the course of the project.

There is definitely an application of this research and it will be very interesting to see it progressed to a fully satisfying conclusion.

## 6. Conclusion

I found this project to be very challenging. The subject matter is complex and there were not a lot of areas for me to find direction in my research online to provide me with a concrete answer as to how to progress or what the best option was as to how to tackle the project. Even similar projects or products that are aimed at doing something similar for any language are few and far between.

Also comparing my project with those of my peers who were producing an application or a website, tangible things, I sometimes felt disheartened with how little coding I was producing for my project. As the year progressed this only grew, but in talking with numerous people on the topic this seems to be the nature of a research project. I do feel I would like to have more to stand over at the end of the year to call my own, but I have still gained a lot of knowledge from this project that will stand me well in the future.

It has been an overall  positive experience. It did come with its share of stress and anxiety, but this is the nature of extended projects with unknown outcomes. If I were to start the project again, I would approach it differently from a technical perspective,  knowing the limitations of the approach taken this time around. I would also approach it differently from a mental standpoint and not get as down about my output and just focus on the task at hand.

Throughout the project I tried to approach it in an Agile approach as much as possible, there was some up front documentation necessary that didn't quite fit with how I had learned about Agile in college and my own experience in the workplace. Enhancing the collaborative aspects of the project and adding value to it where necessary with a shift in direction to the extension. Having the flexibility to work with your findings from each iteration was invaluable and the benefits of Agile are readily apparent to me after my experiences with it.

Working to add value to a large scale project such as the Mika test input generation software was also a very important step. Being able to familiarise myself with code written by someone else and add meaningful additions to it that provide real value is very rewarding. Coming from my own work place experience it is also going to account for a majority of work conducted in the workplace, there is far more building upon existing code bases than the creation of entirely new ones. Being able to read and understand the flow of code and what it is doing is a great skill to bring forward to future employment or further studies.

Also the ability to apply coding techniques to solve real world problems, such as the application of the dynamic code querying and the search for runtime errors in code are skills that are very empowering to have. Knowing that no matter the complexity of the problem, with the correct application of what I have learned through my course, almost any problem has a reachable solution. This is something that not everyone experiences and I am thankful that I have this knowledge with me as I move forward to whatever the next step of my journey.

## 7. Acknowledgements

A big thank you to all those that helped throughout this project and provided great ideas. And of course to  my supervisor, Dr. Chris Meudec, who is also the author of the Mika software.

I would also like to thank all of the Institute of Technology staff and faculty for providing a very enjoyable and enlightening course in software development. Coming back as a mature student was never an issue and I have learned a great deal in my four years here.

# 8. Bibliography

[1] Foundation, O., 2021. *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.*. [online] Electronjs.org. Available at: <https://www.electronjs.org/> [Accessed 15 April 2021].

[2] Richardson, L., 2020. *Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation*. [online] Crummy.com. Available at: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> [Accessed 18 April 2021].

[3] Software BV, T., 2021. *index | TIOBE - The Software Quality Company*. [online] Tiobe.com. Available at: <https://www.tiobe.com/tiobe-index/> [Accessed 25 April 2021].