Ada Runtime Error Generator

Derry Brennan C00231080, Institute of Technology Carlow, April 2021

Abstract

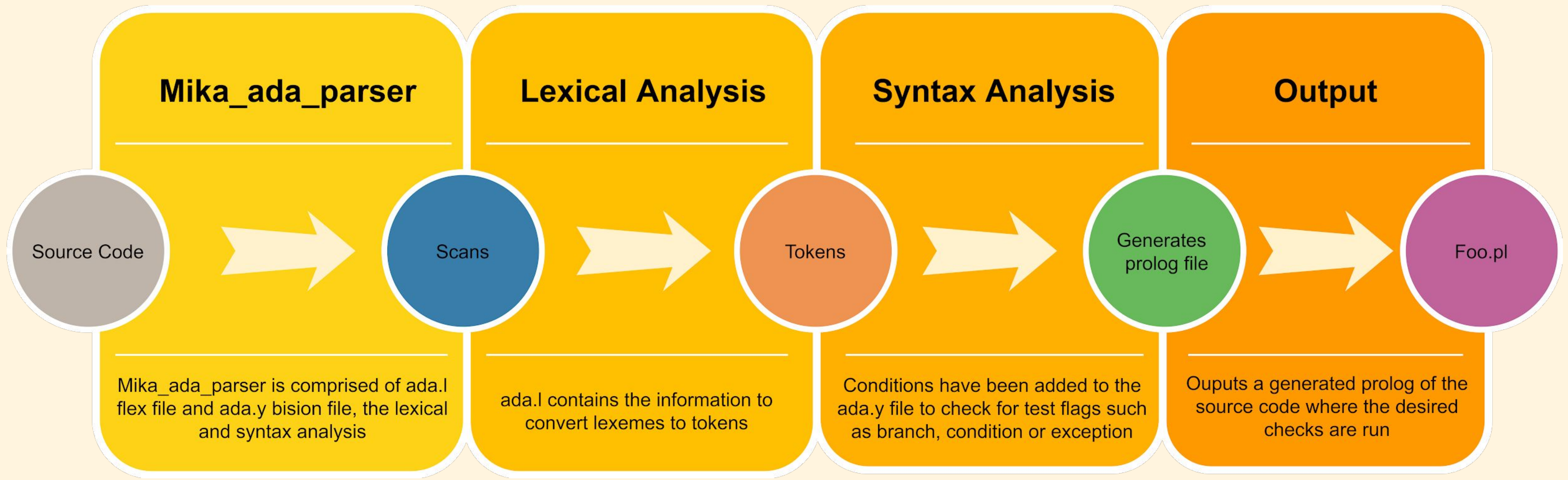
"Ada is a state-of-the-art programming language that development teams worldwide are using for critical software: from microkernels and small-footprint, real-time embedded systems to large-scale enterprise applications, and everything in between." [1] It is particularly used by the military, avionics and many other fields where safety is of critical importance.

With the reliance of safety in Ada it is pertinent to investigate the ability to have run-time error free programs. An error that happens in the field could cause the loss of life or the destruction of property. My goal is to produce a prototype proof-of-concept tool which will be able to take Ada code and be able to tell the programmer if there is any possibility of runtime errors in their code and where, with the test inputs provided that would cause these errors to occur. This poster will be focused on the research into the different areas required for this project such as Ada itself, parsers and the different types of runtime errors.

AdaCore
www.adacore.com

Methods used

Mika has both a lexical (Flex [3]) and a syntax analysis (Bison [4]) file for Ada that it uses to parse the source code. To achieve the checks for possible exceptions within the supplied source code addition were made to the parsing file of Mika, ada.y. Firstly an addition for the division by 0 was made, followed by another check for array index out of bounds. The division by 0 looked at the inserted a check after any division symbols found within the source code and determined if there was any possible conditions within the program that would make this value equate to be 0. Another addition in the parsing file was for when any element was being indexed, in Ada syntax "Array(I)".


Mika_ada_parser
Lexical Analysis
Syntax Analysis
Output
Source Code
Scans
Tokens
Generates prolog file
Foo.pl
Mika_ada_parser is comprised of ada.l flex file and ada.y bision file, the lexical and syntax analysis
ada.l contains the information to convert lexemes to tokens
Conditions have been added to the ada.y file to check for test flags such as branch, condition or exception
Ouputs a generated prolog of the source code where the desired checks are run

Objectives

The goal of the project has been to investigate the possibility of using a test input generation software Mika [2] to also perform checks for possible runtime errors in the supplied Ada code. Runtime errors in critical systems such as avionics and rocketry, as well as any other system where human lives or other extremely valuable assets are depending on the smooth operation of the system. These systems need to be tested and for all outcomes and errors and if there was a tool that could do these things it would be of great benefit to developers.

Some Examples of runtime errors include:

•Division by 0

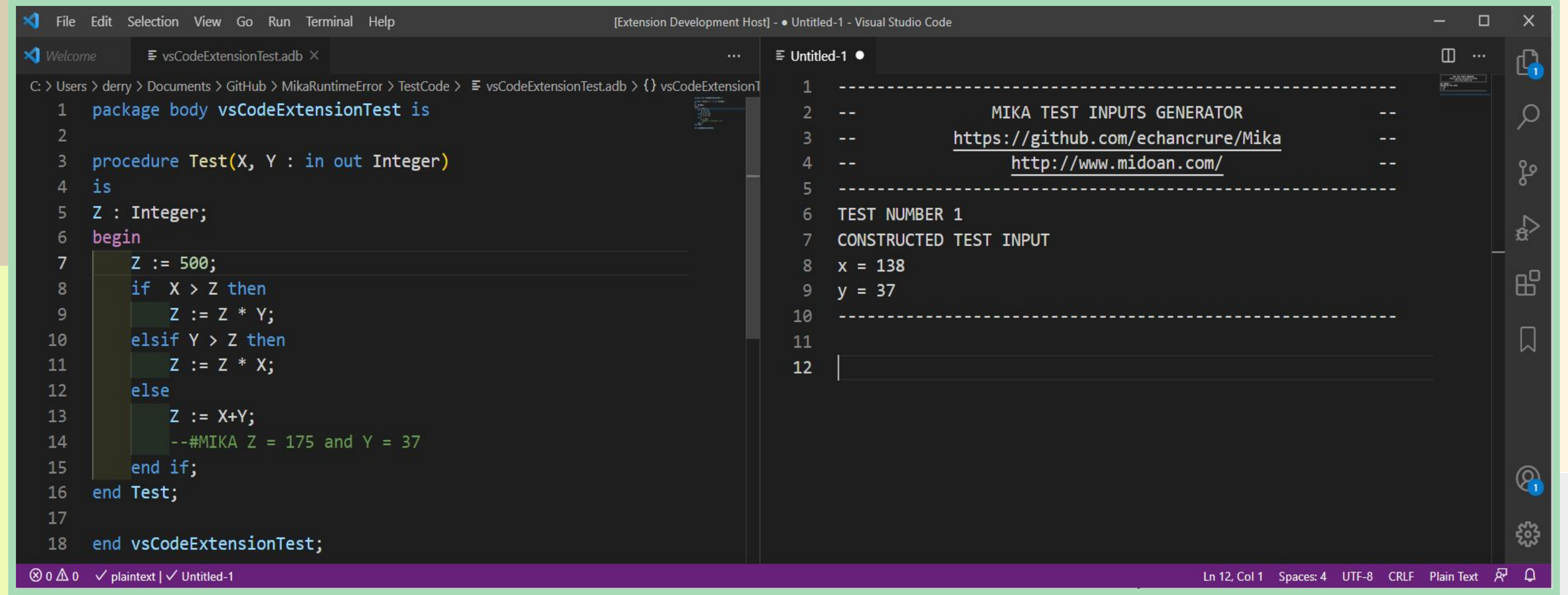
•Array out of bounds indexing

•Integer overflow

Dynamic Code Querying

A Visual Studio Code extension was created to allow a developer to generate tests inputs that satisfy any Boolean condition they wish anywhere in the code.

The developer can use Visual Studio Code (VS Code) command menu to insert a special Mika comment or to run Mika on the code open in VS Code which will supply back the answer in a tab next to the source code. This can be very useful to fully understand how code flows and is something that at present is not readily available within a text editor like VS Code elsewhere.


File Edit Selection View Go Run Terminal Help
[Extension Development Host] - ● Untitled-1 - Visual Studio Code
Welcome
vsCodeExtensionTest.adb
C: > Users > derry > Documents > GitHub > MikaRuntimeError > TestCode > vsCodeExtensionTest.adb > {} vsCodeExtensionT
Untitled-1 ●
⊗ 0 ⚠ 0 ✓ plaintext | ✓ Untitled-1
Ln 12, Col 1 Spaces: 4 UTF-8 CRLF Plain Text

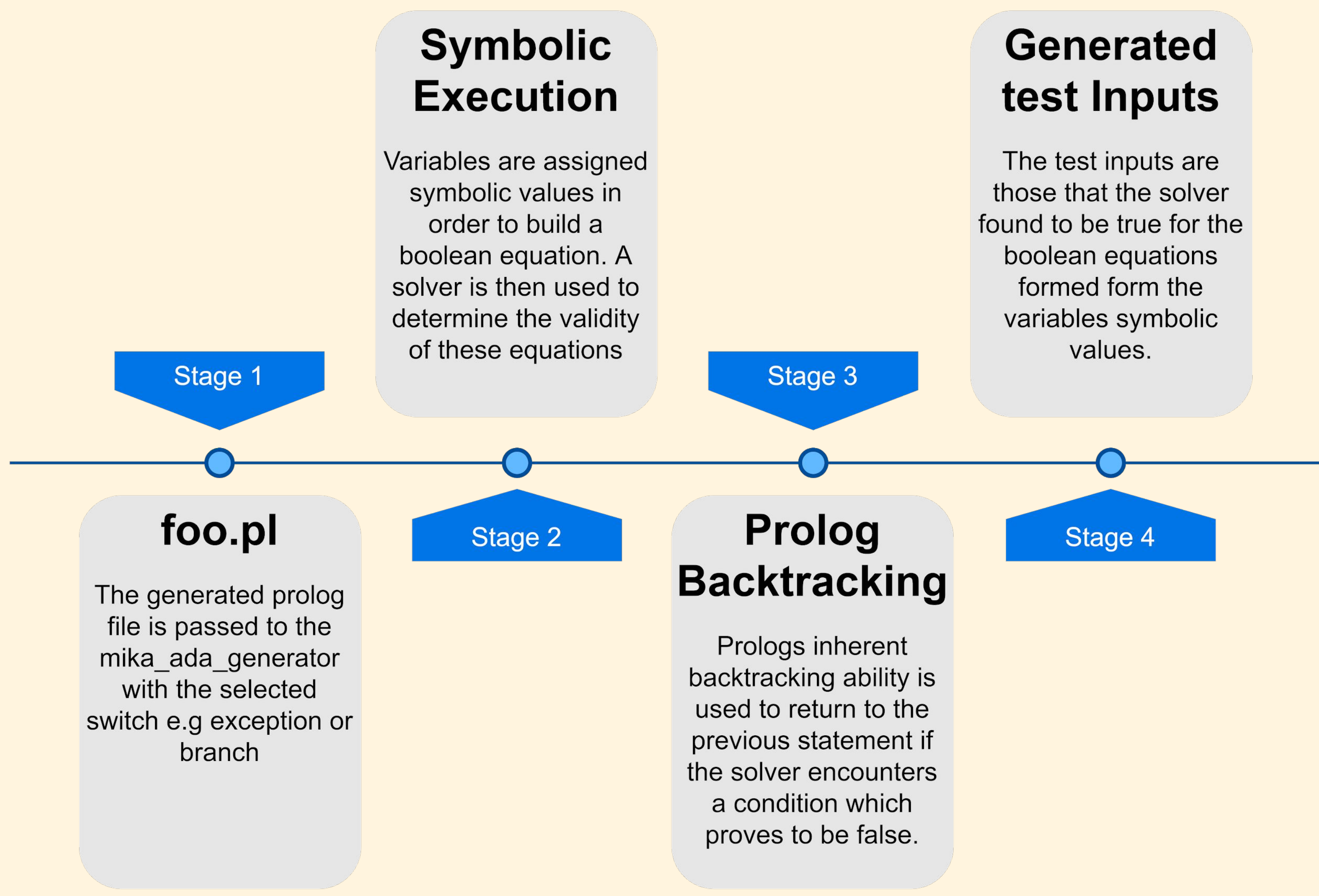
```
package body vsCodeExtensionTest is

procedure Test(X, Y : in out Integer)
is
Z : Integer;
begin
    Z := 500;
    if  X > Z then
        Z := Z * Y;
    elsif Y > Z then
        Z := Z * X;
    else
        Z := X+Y;
        --#MIKA Z = 175 and Y = 37
    end if;
end Test;

end vsCodeExtensionTest;
```

```
------------------------------------------------------------
--               MIKA TEST INPUTS GENERATOR               --
--           https://github.com/echancrure/Mika           --
--                 http://www.midoan.com/                 --
------------------------------------------------------------
TEST NUMBER 1
CONSTRUCTED TEST INPUT
x = 138
y = 37
------------------------------------------------------------

```


Symbolic Execution
Variables are assigned symbolic values in order to build a boolean equation. A solver is then used to determine the validity of these equations
Generated test Inputs
The test inputs are those that the solver found to be true for the boolean equations formed form the variables symbolic values.
Stage 1
Stage 3
foo.pl
The generated prolog file is passed to the mika_ada_generator with the selected switch e.g exception or branch
Stage 2
Prolog Backtracking
Prologs inherent backtracking ability is used to return to the previous statement if the solver encounters a condition which proves to be false.
Stage 4

Results

The results of the research project were positive with the division by zero and the array indexing out of bounds additions generating test inputs. Although the parser does not have the type information needed to push further into more complex exceptions. Further research into performing these checks within the symbolic executor are the next phase.

Conclusion

The search for runtime errors in code is a very difficult task and not many tools have this functionality, even though the cost of verification and validation of software is very high [5]. Tools that could automate some of this process and give better confidence in the produced product would be very beneficial.

A great deal of learning took place involving the use of symbolic execution and the parsing of source code also. Along with a greater understanding of the Ada language.

Although the addition of the checks for runtime errors within the parser was too ambiguous to be a sustainable way of proceeding, an idea to move the checks into the symbolic executor where the type information of the elements from the source code is available seems promising for future research.

Acknowledgements & Contact

For further information on this please feel free to contact at C00231080@itcarlow.ie or derrybrennan@gmail.com

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

References

[1] Adacore.com. 2020. About Ada - Adacore. [online] Available at: https://www.adacore.com/about-ada [Accessed 14 November 2020].

[2] Meudec, C., 2021. echancrure/Mika. [online] GitHub. Available at: https://github.com/echancrure/Mika [Accessed 29 March 2021].

[3] Lesk, M, and Schmidt, E., Lex - A Lexical Analyzer Generator, Available at: http://dinosaur.compilertools.net/lex/index.html [Online], Accessed on: 27/10/2020

[4] Johnson, S., Yacc: Yet Another Compiler-Compiler, Available at: http://dinosaur.compilertools.net/yacc/index.html [Online], Accessed on: 27/10/2020

[5] Rommel, C., 2018. Controlling Costs with Software Language Choice. [online] Adacore.com. Available at: https://www.adacore.com/uploads/techPapers/Controlling-Costs-with-Software-Language-Choice-AdaCore-VDC-WP.PDF [Accessed 29 March 2021].