Ada Runtime Error Generator

Technical Document

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

Introduction

All code for the Ada runtime error generator section of the project can be found on GitHub at: https://github.com/derrymb/MikaRuntimeError

All code for the Mika extension for Visual Studio Code can be found on GitHub at: https://github.com/derrymb/Mika-Visual-Studio-Code-Extension

Below I will detail the most pretant code used in this project. Including additions made to existing files and new files created. Also a test file for Prolog used to familiarise myself with that language during initial research.

Package.Json

The definition file of the extension, this includes the name of the extension, the settings options of the extension, the commands defined within the extension and the dependencies of the extension also.

```
{
	"name": "mika-annotations-ada--js-",
	"displayName": "Mika Annotations Ada (js)",
	"description": "Annotates Ada code for test generation",
	"version": "1.0.0",
	"engines": {
		"vscode": "^1.53.0"
	},
	"categories": [
		"Other"
	],
	"activationEvents": [
		"onCommand:mika-annotations-ada--js-.adaannotations",
		"onCommand:mika-annotations-ada--js-.genTestInput"
	],
	"main": "./extension.js",
	"contributes": {
		"configuration": {
			"title": "mika-annotations-ada--js-",
			"properties": {
				"mika-annotations-ada--js-.mikaPath": {
					"type": "string",
					"default":
"%appdata%\\Roaming\\Midoan\\Mika\\bin",
					"description": "Enter in your the path to
your mike bin folder"
				},
				"mika-annotations-ada--js-.gnatPath": {
					"type": "string",
					"default": "C:\\GNAT\\2010\\bin",
                              "description": "Enter in your the path to
your GNAT installation"
                        }
                  }
            },
            "commands": [
                  {
                        "command":
"mika-annotations-ada--js-.adaannotations",
                        "title": "Mika Ada annotations"
                  },
                  {
                        "command":
"mika-annotations-ada--js-.genTestInput",
                        "title": "Mika Generate test inputs"
                  }
            ]
      },
      "scripts": {
            "lint": "eslint .",
            "pretest": "npm run lint",
            "test": "node ./test/runTest.js"
      },
      "devDependencies": {
            "@types/glob": "^7.1.3",
            "@types/mocha": "^8.0.4",
            "@types/node": "^12.11.7",
            "@types/vscode": "^1.53.0",
            "eslint": "^7.19.0",
            "glob": "^7.1.6",
            "mocha": "^8.2.1",
            "typescript": "^4.1.3",
            "vscode-test": "^1.5.0"
      }
}
```

Extension.js

The main file of the extension, where all the commands are defined and executed.

```
const vscode = require('vscode');
const fs = require('fs');
const glob = require('glob');
const cp = require('child_process');
const separator = "\\";
const NewMikaFolder = "SecretMikaFolder";
const MikaProcedure = "\nprocedure SecretMikaCall(ID : in Integer; E :
in Boolean) is\nbegin\n\tnull;\nend SecretMikaCall;\n";
const MikaComment = "--#MIKA 'enter the conditions you would like to
be met here'";
const PackageBody = "package body";
const NewLine = "\n";
const Dot = ".";
const secret_mika_function_call = "SecretMikaCall({args});\n";
const LINES =
"----------------------------------------------------------\n";
const TEST_NUMBER_STRING = "TEST NUMBER ";
const CONSTRUCTED_TEST_INPUT = "CONSTRUCTED TEST INPUT \n";
const MIKAHEADER = const MIKAHEADER =
`----------------------------------------------------------
--              MIKA TEST INPUTS GENERATOR              --
--           https://github.com/echancrure/Mika          --\n`;



/**
 * @param {vscode.ExtensionContext} context
 */

function copy_current_dir()  {
	var paths =
vscode.window.activeTextEditor.document.fileName.split(separator);
	var name = paths.pop();
	var fullPath = paths.join(separator);
	var mikaFolder = fullPath + separator + NewMikaFolder;
	if(!fs.existsSync(mikaFolder))
	{
		fs.mkdirSync(mikaFolder);
	}
	var dirPaths = fs.readdirSync(fullPath);
	for(let i = 0; i < dirPaths.length; i++)
	{
		if (dirPaths[i].includes(Dot))
		{
			fs.copyFileSync(fullPath + separator + dirPaths[i],
mikaFolder + separator + dirPaths[i]);
		}
	}
	return [name, mikaFolder];
}

function get_mika_comment(code) {
	var commentsAndLineNos = [];
	var subProgram;
	for(let i = 0; i < code.length; i++){
		let line = code[i].trim();
		if(line.toLowerCase().startsWith("procedure ") ||
line.toLowerCase().startsWith("function ")){
			subProgram = line.slice(line.indexOf(' '),
line.indexOf('(')).trim();
		}
		if(line.toLowerCase().startsWith("--#mika")) {
			commentsAndLineNos.push([line.slice(line.indexOf('
')).trim(),i+1]);
			break;
		}
	}
```

```
      return [commentsAndLineNos, subProgram];
}

function activate(context) {
      var editor = vscode.window.activeTextEditor;
      let disposable =
vscode.commands.registerCommand('mika-annotations-ada--js-.genTestInpu
t', function () {
            var [name, mikaFolder] = copy_current_dir();
            var nameMinusExt = name.substring(0,name.length-4);
            var lines = editor.document.getText().split(NewLine);
            var [mikaComments, subProgram] = get_mika_comment(lines);
            var package_body_line_number = -1;
            for(let i = 0; i<lines.length; i++) {
                  if(lines[i].toLowerCase().includes(PackageBody)) {
                        package_body_line_number = i + 1;
                        break;
                  }
            }

            var path = mikaFolder + separator + name;
            var textOfCopy =
fs.readFileSync(path).toString().split(NewLine);
            var textOffset = 0;
            const insert_at_position = (arr,pos,element) => {
                  if(pos == 0){
                        return  [element].concat(arr);
                  }
                  return
arr.slice(0,pos).concat([element]).concat(arr.slice(pos));
            };
            for(let i = 0; i < mikaComments.length; i++){
                  let [comment,line_number] = mikaComments[i];
                  textOfCopy =
insert_at_position(textOfCopy,line_number+(textOffset++),secret_mika_f
unction_call.replace("{args}",`${i+1},${comment}`));
            }
            textOfCopy =
insert_at_position(textOfCopy,package_body_line_number,MikaProcedure);
            textOfCopy = textOfCopy.join('\n');
            fs.writeFileSync(path,textOfCopy);
            var config =
vscode.workspace.getConfiguration("mika-annotations-ada--js-");
            var mp = config.mikaPath;
            var gp = config.gnatPath;
            if(fs.existsSync(mp) && fs.existsSync(gp)) {
                  let commands = [
                        `cd ${mikaFolder}`,
                        `${mp}${separator}mika_ada_parser.exe -M"${mp}"
-f"${gp}" -gnat05 -d ${nameMinusExt}`,
                        `cd
${mikaFolder}${separator}${nameMinusExt}_mika`,
                        `${mp}${separator}mika_ada_generator.exe
-M"${mp}" -S${subProgram} -Tquery -Cignored -d ${nameMinusExt}`
                  ];
                  cp.spawnSync(commands.join(' & '),{shell:true});
                  try{
                        var jsonFile =
glob.sync(`${mikaFolder}${separator}${nameMinusExt}_mika

${separator}${nameMinusExt}_*${separator}${nameMinusExt}.json`)[0];
                        if (jsonFile === undefined)
                              throw "Error";
                  }
                  catch(e) {
                        vscode.window.showErrorMessage("Mika failed to
generate test inputs");
                        return;
                  }
                  var text = fs.readFileSync(jsonFile);
                  var json = JSON.parse(text);
                  vscode.workspace.openTextDocument().then((a) => {
```

```
vscode.window.showTextDocument(a,{viewColumn:vscode.ViewColumn.Beside}
).then(e => {
                            e.edit(edit => {
                                 let outer_keys = Object.keys(json);
                                 let offset = 6;
                                 edit.insert(new vscode.Position(0, 0),
MIKAHEADER);
                                 for(let i=0;i<outer_keys.length;i++)
                                 {
                                      edit.insert(new
vscode.Position(i+(offset++), 0), LINES);
                                      let test_string_with_number =
TEST_NUMBER_STRING + (i+1) + "\n";
                                      edit.insert(new
vscode.Position(i+(offset++), 0), test_string_with_number);
                                      edit.insert(new
vscode.Position(i+(offset++), 0), CONSTRUCTED_TEST_INPUT);

Object.keys(json[outer_keys[i]]).forEach(function(key){
                                           edit.insert(new
vscode.Position(i+(offset++), 0),`${key} =
${json[outer_keys[i]][key]}\n`);
                                      });
                                      edit.insert(new
vscode.Position(i+(offset++), 0), LINES);
                                      edit.insert(new
vscode.Position(i+(offset++), 0),"\n");
                                 }
                                 })
                            });
                       }, (error) => {
                            console.error(error);
                            return;
                       });
                  fs.rmdirSync(mikaFolder, {recursive:true});
           }
```

```
		else {
			vscode.window.showErrorMessage("Mika or GNAT paths are 
invalid.");
		}

	});
	let disposable2 = 
vscode.commands.registerCommand('mika-annotations-ada--js-.adaannotati
ons', function () {
		var editor = vscode.window.activeTextEditor;
		editor.insertSnippet( new vscode.SnippetString(MikaComment, 
editor.selection.active));
	});
	context.subscriptions.push(disposable);
	context.subscriptions.push(disposable2);
}

function deactivate() {}

module.exports = {
	activate,
	deactivate
}
```

GitHubWebScraper.py

A web scraper written in python used to search GitHub repositories for suitable Ada code to test the projects software on. The Url variable can be changed to target a specific search and the cookie setting marked with ‘XXXXXX’ can be filled in with your own cookie settings to do the web scrape ethically.

```
import requests
from bs4 import BeautifulSoup
import time

url =
'https://github.com/search?l=Ada&p=1&q=ada+extension%3Aadb+language%3A
Ada+language%3AAda&ref=advsearch&type=Code'

headers = {
    'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86
Safari/537.36'
}

session = requests.Session()
session.cookies.set('dotcom_user','XXXXXX',domain='.github.com')
session.cookies.set('logged_in','yes',domain='.github.com')
session.cookies.set('user_session','XXXXXXXX',domain='github.com')
session.cookies.set('_device_id','XXXXXXXX',domain='github.com')
r = session.get(url,headers=headers)
myWebscraping = BeautifulSoup(r.text,'html.parser')

l = []
next_page = None
while (next_page := myWebscraping.find("a",{"class":"next_page"})) or
myWebscraping.find("span",{"class":"next_page disabled"}):
    for link in myWebscraping.find_all('a'):
        if (link :=
f"https://github.com/{link.get('href')}").endswith('.adb'):
            time.sleep(3)
            r = session.get(link)
            html = BeautifulSoup(r.text,'html.parser')
            code = html.find('div',{'itemprop':'text'}).text
            if 'with' not in code:
                print(f'Found valid code: {link}')
                l.append(link)
    if next_page == None:
        break
    next_page = f"https://github.com/{next_page.get('href')}"
    print(f'\t\tNavigating to {next_page}')
    next_page = session.get(next_page,headers=headers)
    myWebscraping = BeautifulSoup(next_page.text)
print('\t\tFinished Web Scraping, writing to file.')
template = "<a href='{url}'>{url}</a><br>"

l = [template.format(url=url) for url in l]

with open(r'C:\Users\XXXXX\Desktop\links.html','w') as f:
    for link in l:
        f.write(link)
print('\t\tDone!')
```

Ada.y

Ada.y is the Yet Another Compiler Compiler file used to parse Ada code supplied to Mika. This is a large file of over 5200 lines of code so below are areas where I made additions to it in support of the runtime error checking.

```
#define YYSTACK_SIZE 1000                //Parser generator constant

#define SAFETY 5                         //number of characters added
to malloc

#define COND 0

#define GATE 1

#define DECI 2

#define BRAN 3

#define RUNE 4

int condition_nb = 1;   //counter for the number of conditions

int gate_nb = 1;        //counter for individual gate

                        //(i.e. boolean operators : and, or, xor, not
and_then, or_else, not)

int decision_nb = 1;    //counter for the number of decisions

int branch_nb = 1;      //counter for the number of branches

int runtime_nb = 1;     //counter for the number of runtime error
checks

//values returned by lexer or the parser

%union {char *id;                           // for REF, PAC and ADA
most stuff

        struct id_kind_t id_kind;           // for identifiers and
strings in REF

        struct id_ref_t  id_ref;            // only for ADA
identifiers, character literals, string literals, operators
```

```
        struct id_decision id_deci;         // for relation and
expression_2 to indicate if we are within a decision

        struct id_subprogram_t id_subprogram;       //for subprograms
only

        struct true_line_column_t true_line_column; //used for
recording line and columns of detected bran,deci,cond, rune and gates

       }

/*can be a array access, a function/procedure call, a type conversion,
or a subtype_indication with index_contraint */

/*we will have to differentiate between many things*/

indexed_component : name '(' value_list ')'

                    {$$ =
malloc(SAFETY+strlen(tmp_s)+strlen($1)+strlen($1)+strlen($1)+strlen($3
)+strlen($3)+strlen($3)+56);

                     itoa(runtime_nb++, tmp_s, 10);

                     print_coverage_details(RUNE, tmp_s, current_unit,
yylineno, column+1);

                     strcpy($$, "indexed(");

                     strcat($$, $1);

                     strcat($$, ", [");

                     strcat($$, "rune(");

                     strcat($$, tmp_s);

                     strcat($$, ", ");

                     strcat($$, $3);

                     strcat($$, " > ");

                     strcat($$, "tic(");

                     strcat($$, $1);

                     strcat($$, ", last) || ");
```

```
                          strcat($$, $3);
                          strcat($$, " < ");
                          strcat($$, "tic(");
                          strcat($$, $1);
                          strcat($$, ", first) ,");
                          strcat($$, $3);
                          strcat($$, ")");
                          strcat($$, "])");
                          free($1);
                          free($3);
                         }
                       ;
//uses 'struct id_decision'
term    : factor                {$$ = $1;}
        | term multiplying_operator factor
                                {if (!strncmp($2, "String_", 7))
//it is a user defined operator
                                   {$$.id =
malloc(SAFETY+strlen($1.id)+strlen($2)+strlen($3.id)+15);
                                    strcpy($$.id, "indexed(");
                                    strcat($$.id, $2);
                                    strcat($$.id, ",[");
                                    strcat($$.id, $1.id);
                                    strcat($$.id, ", ");
                                    strcat($$.id, $3.id);
                                    strcat($$.id, "])");
```

```
                                       }
                                      else if(!strncmp($2, " / ", 3))
                                      {
                                        $$.id =
malloc(SAFETY+strlen(tmp_s)+strlen($1.id)+strlen($2)+strlen($3.id)+str
len($3.id)+14);
                                        itoa(runtime_nb++, tmp_s, 10);
                                        print_coverage_details(RUNE,
tmp_s, current_unit, yylineno, column+1);
                                        strcpy($$.id, $1.id);
                                        strcat($$.id, $2);
                                        strcat($$.id, "rune(");
                                        strcat($$.id, tmp_s);
                                        strcat($$.id, ", ");
                                        strcat($$.id, $3.id);
                                        strcat($$.id, " = 0, ");
                                        strcat($$.id, $3.id);
                                        strcat($$.id, ")");
                                      }
                                     else
                                       {$$.id =
malloc(SAFETY+strlen($1.id)+strlen($2)+strlen($3.id)+1);
                                        strcpy($$.id, $1.id);
                                        strcat($$.id, $2);
                                        strcat($$.id, $3.id);
                                       }
                                     free($1.id);
```

```
                                 free($2);

                                 free($3.id);

                                 $$.is_a_decision = 0;

                                }

        ;  //runtime Error check, counter for this,

//print cond, gate, deci or bran information to Fcond_ids file

void print_coverage_details(int type, char * number, struct unit_type
*unit, int line, int column)

{

  switch (type) {

    case COND : fprintf(Fcond_ids, "cond(");

                break;

    case GATE : fprintf(Fcond_ids, "gate(");

                break;

    case DECI : fprintf(Fcond_ids, "deci(");

                break;

    case BRAN : fprintf(Fcond_ids, "bran(");

                break;

    case RUNE : fprintf(Fcond_ids, "rune(");

                break;

    default : fprintf(stdout, "Mika ERROR: unknown type of coverage %i
in print_coverage_details", type);

              fflush(stdout);

              my_exit(31);

  }
  fprintf(Fcond_ids, "%s, '%s', '%s', '%s', '%s', %i, %i).\n", number,
unit->name, unit->filename, unit->suffix, unit->path, line, column);

}
```

Test.adb

Ada code written to test out custom types.

```
with Ada.Text_IO;

procedure test is
    subtype my_int is Integer range 1 .. 10;
    X : Integer;
    Y : my_int;
begin
    X:=2;
    Y:=5;
    if X < 5 then
        Ada.Text_IO.Put_Line("Hello");
    else
        Ada.Text_IO.Put_Line("Goodbye");
    end if;

    if Y = 5 then
        Ada.Text_IO.Put_Line("Custom types Baby!!!");
    end if;
end test;
```

Db.pl

Db.pl was a file written early on in my research phase of the project to try and familiarise myself with how prolog works and also its syntax. Although over the course of the project I never wrote prolog code myself there was a lot of looking and reading generated prolog files and my introduction to the language helped me to understand what was happening.

```
/*
listing: writes everything in the database
listing(predicate): writes all the atoms of that predicate
write: writes what is inside brackets to the screen
    e.g write('Hello World!').
nl: will print a new line
multiple statements: join multiple statements with a ','
    e.g write('Hello World!'), nl.*/

loves(romeo, juliet).
/*
predicate(fact): loves
atoms: romeo, juliet (atoms always start with lower case characters)
    atoms can contain uppercase characters, just must not start with
them
    and may also contain + - _ / * < > characters
*/

/*juliet loves romeo if romeo loves juliet*/
loves(juliet, romeo) :- loves(romeo, juliet).

/*
Variables: loves(romeo, X)
the variable X will return in this instance who loves romeo
variables always are UPPERCASE characters
*/

/*
Keep predicates grouped together
```

```
*/
male(derry).
male(michael).
male(willie).
male(james).
male(andy).

female(ana).
female(dora).
female(lisa).
female(zhe).
female(cliodna).
female(saoirse).

/*
Using variables: parent(X, zhe). will return the X's of parents of zhe
will print out 'no' once all X's are printed, stating that there are
no more

parent(X, dora), dances(X). 'Who is the parent of dora who also
dances'

parent(Y, dora), parent(X,Y). 'Who is the parent of dora Y, where Y is
the child of X' basically find the grandparent Y
parent(michael, X), parent(X,Y). 'Who is the child of michael X, where
X is the Parent of Y' basically find michaels grand children Y
*/

parent(derry, dora).
parent(derry, zhe).

parent(ana, dora).
parent(ana, zhe).

parent(michael, derry).
parent(cliodna, derry).
parent(willie, ana).
parent(lisa, ana).

sister(lisa, saoirse).
/*
parent(X, ana), sister(X,Y). find the parent of ana where Y is the
sister of X
*/

happy(derry).
happy(ana).
happy(zhe).
happy(james).

with_derry(ana).

near_water(ana).

swims(ana) :-
    happy(ana),
    near_water(ana).

swims(derry) :-
    happy(derry).

swims(derry) :-
    near_water(derry).

runs(derry) :- /* ':-' (rule) is the same as an if statement, e.g
derry runs if he is happy */
    happy(derry).

dances(ana) :-
    happy(ana),  /*can have multiple conditions joined together by ','
e.g ana dances IF ana is happy AND ana is with_derry*/
    with_derry(ana).
```

```
/*
male(X), female(Y).
this will list all the combinations of males and females
';' will print the next combination
'a' will print all the combinations together
'RET(return key)' will stop the listings
*/

does_ana_dance :- dances(ana),
    write('When Ana is happy and with Derry she dances').

get_grandchild :-
    parent(michael, X),
    parent(X,Y),
    write('Michaels grandchild is '),
    write(Y), nl.

/*
string formatting:
~w insert Variable
~s transpose a string
~n new line
*/
get_grandparent :-
    parent(X, dora),
    parent(X, zhe),
    format('~w ~s grandparent ~n', [X, "is the"]). % ~w = X, ~s = "is
the"

/*
Axiom (established fact)
Using grand_parent, we can grand_parent(dora,A). where A will be
dora's grandparents
Another example
hungry(X) :- human(X). if X is a human, then X is also hungry
*/
```

```
grand_parent(X,Y) :- % grand_parent(dora,A). X=dora Y=A(grandparent)
    parent(Z,X),     % Z=parent of X(dora)
    parent(Y,Z).     % Y(A)=parent of dora’s parent(Z)


human(george).
hungry(X) :- human(X). % hungry(george). will return true as all
humans are hungry

stabs(zhe, ana, dagger).
hates(derry, X) :- stabs(X, ana, dagger).

/*
anonymous variable
check for the existence of a predicate using an anonymous variable
male(_).
the '_' underscore character is used for the anon var
*/

what_grade(d) :-
    write('You need to study!').

what_grade(a) :-
    write('You are doing great!').

what_grade(b) :-
    write('Good job, nearly perfect').

what_grade(c) :-
    write('Keep working hard and you\'ll get there').

what_grade(Other) :-
    Grad = Other,
    format('You got a ~w ~s ~n', [Grad, "grade"]).


/*
```

```
OBJECTS && STRUCTURES
*/

owns(derry, pet(cat,honey)).

negative(X) :- X < 0.
positive(X) :- X > 0.
zero(X) :- X == 0.
notzero(X) :- X =\= 0.

range(Low, Low, High).
range(Out,Low,High) :- NewLow is Low+1, NewLow =< High, range(Out,
NewLow, High).
```