Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

SISTEMA DE VERIFICACIÓN AUTOMÁTICA Y ALEATORIA DE CÓDIGO EN LENGUAJE C

ANGELO JULIAN CERPA VILLAGRA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCION INFORMATICA

Noviembre 2017

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

SISTEMA DE VERIFICACIÓN AUTOMÁTICA Y ALEATORIA DE CÓDIGO EN LENGUAJE C

ANGELO JULIAN CERPA VILLAGRA

Profesor Guía: **Ismael Figueroa Palet**

Profesor Co-referente: **Iván Mercado Bermúdez**

Carrera**: Ingeniería de Ejecución en Informática**

Noviembre 2017

Dedicatoria

Dedico este trabajo de título, a mis padres y familia, tate y pepi que me formaron como persona
y a mi profesor que me ayudo a enfrentar este gran reto.

Índice

Lista de Figuras ... iii
Resumen ... iv
Abstract ... iv
1 Introducción ... 6
2 Descripción del Tema ... 7
3 Estado del Arte ... 8
3.1 Descripción General del Estado del Arte ... 8
3.2 Herramientas Utilizadas ... 9
3.2.1 Frama-C ... 9
3.2.2 ACSL ... 10
3.2.3 Frama-C E-ACSL Plug-in ... 10
3.2.4 Quickcheck4c ... 10
3.2.5 CodeRunner ... 10
4 Definición de Objetivos ... 12
4.1 Objetivo General ... 12
4.2 Objetivos Específicos ... 12
5 Esquema de Solución Propuesta ... 13
5.1 Descripción General ... 13
5.2 Diagrama General de la Solución ... 13
6 Catálogo de Problemas Seleccionados ... 15
6.1 Variables simples (Modulo 1) ... 15
6.2 Módulo 2 ... 15
6.3 Problemas Relevantes Sin Especificación Completa ... 16
7 Esquema de la Solución Específica ... 17
7.1 Declaración de Propiedades de Correctiud ... 17
7.2 Restricciones de las Variables ... 18
7.3 Implementación Alumno ... 18
7.4 Código Instrumentado ... 19
7.5 Archivo Json con Valores Relevantes de la Función ... 22
7.6 Función Checker ... 22

7.7 Función Final Verific y Ejecutable 23
8 Implementación 24
9 Alcances y Limitaciones 25
10 Conclusiones 26
11 Referencias 27

Lista de Figuras

Figura 2.1 Cono de la experiencia Edgar Dale 7
Figura 2.1 Escuela de ingeniería informática PUCV 7
Figura 3.1 Logo Frama-C...9
Figura 3.2 Ejemplo de pregunta en CodeRunner 11
Figura 5.1 Llamada al script verific 13
Figura 7.1 Propiedad*es* de correctitud en forma matemática 17
Figura 7.3 Propiedades de correctitud en E-ACSL.....17 **¡Error! Marcador no definido.**
Figura 7.4 Ejemplo de restricciones...........................18 **¡Error! Marcador no definido.**
Figura 7.5 Ejemplo implementación correcta ....18 **¡Error! Marcador no definido.**
Figura 7.6 Ejemplo implementación incorrecta..........19 **¡Error! Marcador no definido.**
Figura 7.7 Función encontrar letra incorrecta............20 **¡Error! Marcador no definido.**
Figura 7.7 Función encontrar letra correcta................20 **¡Error! Marcador no definido.**
Figura 7.8 Ejemplo implementación correcta ....20 **¡Error! Marcador no definido.**
Figura 7.9 Ejemplo función __gen_e_acsl_encontrarletra21 **¡Error! Marcador no definido.**
Figura 7.10 Ejemplo json ....22 **¡Error! Marcador no definido.**
Figura 7.11 Ejemplo funcion checker ....23 **¡Error! Marcador no definido.**
Figura 7.12 Ejemplo ejecutable que pasa 1000 casos de prueba23 **¡Error! Marcador no definido.**
Figura 7.13 Ejemplo de ejecutable con problemas en la implementación del alumno....23 **¡Error! Marcador no definido.**

Resumen

Este documento presenta el proyecto de software de verificación y testing “Verific” que pretende dar solución a problemas relacionados con los casos de prueba en la plataforma *CodeRunner* [1]. Se presentará herramientas que aportan a la resolución de este problema, objetivos, diagrama general de la solución, catálogo de problemas seleccionados, esquema de la solución específica y conclusiones. Se espera como resultado esperado el concluir con totalidad los objetivos presentes en el siguiente informe.

Palabras-claves: *pruebas basadas en propiedades, casos de prueba.*

Abstract

This paper presents the verification and testing software project "Verific" that tries to solve problems related to the test cases in the CodeRunner[1] platform. It will present tools that contribute to the resolution of this problem, objectives, general diagram of the solution, catalog of selected problems, schematic of the specific solution and conclusions. It is expected as an expected result, to fully meet the objectives presented in the following paper.

Keywords: *property-based testing, tests case.*

Glosario de Términos

Pruebas basadas en propiedades: consisten en especificar propiedades formales de correctitud, contra las cuales se comprueba si un código en particular las satisface o no.

Casos de prueba: Constan de una entrada y salida que se contrastan con las dadas por una aplicación o sistema de software y según esto se especifica la correctitud del programa según los casos probados.

1 Introducción

¿Cómo los alumnos aprenden?, Edgar Dale fue un pedagogo estadounidense que en 1946 formuló su famoso cono de la experiencia [2]. Él investigó las distintas formas de aprendizaje para analizar la profundidad de comprensión que se conseguía con cada una de ellas. Evidenció los resultados de su estudio en una pirámide del aprendizaje que muestra la efectividad de cada método en particular. Este cono se basa en 10 niveles de aprendizaje, los que son enumerados de menor a mayor impacto como: símbolos verbales; símbolos visuales; imágenes fijas, grabaciones y radio; películas; exposiciones; viajes de campo; demostraciones; representaciones dramáticas; experiencias simuladas y por ultimo experiencia directa.

En el contexto de la enseñanza de la Informática, un caso típico podría ser: un alumno de primer año que no siempre llega temprano a sus clases, la clase no va a su ritmo (muy rápida o muy lenta), que duda de sus pocos conocimientos sobre la materia por lo que no se atreve a preguntar al profesor, y que al volver a casa intenta probar lo aprendido, pero ya no puede preguntarle a nadie. Si el alumno necesita averiguar cómo se hace algo, busca en Internet, intenta adaptarlo pero no es exactamente lo que necesita y pierde tiempo en analizar código o le resulta muy complicado entenderlo. El tiempo en la sala de clases es acotado, y es donde el profesor debe repartir su atención a un curso generalmente numeroso. Si bien no se puede dar con una fórmula única de aprendizaje, nuestra propia experiencia nos dice claramente que en el hacer o interactuar es donde se obtienen mejores resultados de aprendizaje, y para quienes ya saben algo de programación no es ningún secreto que se aprende experimentando, fallando, investigando, probando y volviendo a fallar. Es necesario que la instancia de experimentación del alumno sea lo más provechosa posible, y es por eso que este trabajo busca refinar el entorno de pruebas para permitir un mejor aprendizaje y evitar frustraciones. En particular, este trabajo de título está centrado en el desarrollo de una herramienta para refinar problemas de programación utilizados en la asignatura Fundamentos de Programación, en el contexto de la plataforma CodeRunner. La herramienta en cuestión permite especificar propiedades formales de correctitud, las que se utilizan para la generación automática de casos de prueba, que finalmente otorgarán retroalimentación inmediata y automática a los estudiantes, ya que pueden ver con contraejemplos específicos en qué casos su código no está cumpliendo con la especificación requerida.

Figura 1.1 Cono de la experiencia de Edgard Dale [2]

2 Descripción del Tema

En el primer año de las carreras de Ingeniería Civil e Ingeniería en Ejecución, de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso (PUCV), se dicta la asignatura Fundamentos de Programación, en la que los alumnos deben desarrollar las competencias fundamentales de programación utilizando, entre otras herramientas, la plataforma CodeRunner. Existe el gran inconveniente de que la mayoría de las preguntas no tienen los casos de prueba necesarios, pudiendo ser que la implementación de código realizada por el alumno sea validada como correcta sin que esta lo esté del todo, y dada la gran cantidad de preguntas es costoso aumentar la cantidad de casos de prueba. Si bien el contexto en que se presenta este trabajo de título es dentro de la Escuela de Ingeniería Informática de la PUCV, pero es aplicable en cualquier centro educacional que utilice la plataforma CodeRunner.

Figura 2.1 Alumnos Escuela de Ingeniería Informática[3]

3 Estado del Arte

Se presenta la descripción general del estado del arte y las herramientas a utilizar.

3.1 Descripción General del Estado del Arte

Las *pruebas basadas en propiedades*, también conocidas por su nombre en inglés *property-based testing (PBT)*, consiste en especificar propiedades formales de correctitud, contra las cuales se comprueba si un código en particular las satisface o no. En vez de hacerlo de manera matemática mediante algún tipo de demostración, la validación implica contrastar la salida generada por un código con su declaración de salida esperada. Estas declaraciones se comprueban para muchas entradas diferentes generalmente con carácter aleatorio, las fallas en estas pruebas revelan problemas que podrían no haber sido advertidos en pruebas manuales. En el fondo, mediante la aplicación de muchas pruebas aleatorias se busca establecer un *convencimiento* de correctitud más que una *demostración* de correctitud.

Sobre la temática de property-based testing existen diferentes trabajos o estudios de cuales destacan:

- *Property-based testing for functional programs* [4] En la que se desarrolla la temática de pruebas basadas en propiedades desde la perspectiva de la programación funcional, se presenta la herramienta QuickSpec, que intenta inferir una ecuación de especificación de un programa funcional con la ayuda de las pruebas y también se desarrolla un algoritmo de caja negra para calificar automáticamente los programas de los estudiantes mediante pruebas, infiriendo para cada programa un conjunto de errores que contiene el programa
- *Property-based testing in Java* [5] En donde se evalúa el framework PropCheck tanto su implementación, desarrollo , generación de datos y contraste entre datos esperados y obtenidos.

En desarrollos que ocupan la técnica PBT el de mayor renombre es QuickCheck [6] que es una herramienta para probar programas del lenguaje Haskell automáticamente. El programador proporciona una especificación del programa, en forma de propiedades que deben satisfacer las funciones, y QuickCheck luego prueba que las propiedades se mantienen en un gran número de casos generados al azar.

Existen diversas implementaciones de QuickCheck para el lenguaje C como por ejemplo:

- *Quickcheck4c* [7] que provee pruebas automatizadas de propiedades con varios generadores de valor y destaca por su categorización del caso de prueba mediante etiquetas para estadísticas de cobertura.

- *Theft* [8] genera entradas y busca ejemplos donde la prueba falle. Si Theft encuentra fallos, también genera y prueba variantes más simples de la entrada, y luego reporta el ejemplo más sencillo encontrado.
- *Qc*[9] [8] que permite probar escenarios de pruebas y puede ejecutar automáticamente pruebas cuando el código cambia.

3.2 Herramientas Utilizadas

Basándonos en trabajos previos, las herramientas seleccionadas o necesarias para implementar una solución al problema son:

3.2.1 Frama-C

Es un framework colaborativo que posee diversas herramientas destinadas al análisis de código del lenguaje C, su enfoque permite desarrollar sobre los resultados ya alcanzados por otros programadores. Cuenta con herramientas para análisis estático del código fuente, que sintetiza información de un código sin ejecutarlo , con lo que se puede evaluar la construcción de un programa según la cantidad de comentarios por línea de código o la profundidad de las estructuras de control anidadas, también cuenta con herramientas cercanas a la búsqueda heurística de errores, que analizan en profundidad el código y permiten al usuario manipular las especificaciones funcionales, y probar que el código fuente satisfaga estas especificaciones.

Frama-C está organizado con una arquitectura plug-in (comparable a la del Gimp o Eclipse). Un núcleo común que centraliza la información y realiza el análisis, los complementos interactúan entre sí a través de las interfaces definidas por el kernel, esto posibilita la robustez de un desarrollo en Frama-C y permite un amplio espectro de funcionalidad.

Figura 3.1 Logo Frama-C[10]

3.2.2 ACSL

El lenguaje de especificación ANSI / ISO C (ACSL) es un lenguaje de especificación de comportamiento para programas C. El diseño de ACSL está inspirado en lenguaje de especificaciones para Java JML [11]. También hereda mucho del lenguaje de especificaciones del analizador de código fuente Caduceus [12] ya obsoleto, pues reemplazado por Frama-C. ACSL puede expresar una amplia gama de propiedades funcionales y la noción primordial en ACSL es el contrato de función con el que se especifica lo que se espera que realice una función.

3.2.3 Frama-C E-ACSL Plug-in

En general las propiedades escritas en ACSL se utilizan como entrada para sistemas de demostraciones, ya sean automáticas o asistidas, y no tienen efecto en la ejecución del software. Sin embargo, el lenguaje E-ACSL, mediante restricciones sintácticas, permite de traducir automáticamente un programa C con anotaciones ACSL, en otro programa que falla en tiempo de ejecución si se infringe una anotación o propiedad. Si no se infringe ninguna anotación, el comportamiento del nuevo programa es exactamente el mismo que el del programa original.

3.2.4 Quickcheck4c

Es un framework de pruebas basado en la propiedad, inspirado en QuickCheck. Su objetivo principal del proyecto es proporcionar una API adecuada sin la necesidad de demasiadas dependencias. Permite pruebas automatizadas de propiedades con varios generadores de valor; generadores de valor para tipos de base: long, int, double, float, boolean, char; generadores de valores para tipos de array: string, long, int double, float, boolean, char; posibilidad de agregar generadores de valores personalizados para tipos complejos.

3.2.5 CodeRunner

CodeRunner es un plug-in open source para Moodle de preguntas de programación que puede ejecutar código de programas enviados por los estudiantes con una amplia gama de preguntas de programación en muchos idiomas diferentes. Está destinado principalmente para su uso en cursos de programación de computadores aunque puede ser utilizado para calificar cualquier pregunta para la cual la respuesta sea de texto. Normalmente se utiliza en el modo de cuestionario adaptativo de Moodle; Los estudiantes desarrollan el código de respuesta a cada pregunta de programación y consiguen ver sus resultados del caso de prueba inmediatamente. A continuación, puede corregir su código y volver a enviar, por lo general con una pequeña penalización.

Question 1

Correct

Mark 1.00 out of 1.00

Flag question

Write a Python3 function *sqr(n)* that returns the square of its numeric parameter *n*.

For example:

Test	Result
print(sqr(-3))	9
print(sqr(11))	121

Answer:

```
1  def sqr(n):
2      return n * n
```

Check

Test	Expected	Got	
print(sqr(-3))	9	9	✓
print(sqr(11))	121	121	✓
print(sqr(-4))	16	16	✓
print(sqr(0))	0	0	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Figura 3.2 Ejemplo de pregunta en CodeRunner

4 Definición de Objetivos

Se describen los objetivos generales y específicos del proyecto, evidenciando la finalidad y enfoque a la solución que se propone.

4.1 Objetivo General

Implementar y validar una plataforma que permita la especificación y verificación, mediante casos de prueba aleatorios, de algoritmos básicos escritos en el lenguaje C enfocados a la asignatura fundamentos de programación.

4.2 Objetivos Específicos

1. Implementar un mecanismo de extracción del lenguaje E-ACSL
2. Generar condiciones ejecutables de validación, según las propiedades escritas en E-ACSL
3. Crear un catálogo de problemas de programación representativos de la asignatura, especificando sus propiedades de correctitud, y soluciones de referencia.
4. Integrar la plataforma de especificación y verificación con CodeRunner
5. Diseñar y aplicar una evaluación experimental de la plataforma integrada a CodeRunner.

5 Esquema de Solución Propuesta

Se presenta el esquema de la solución propuesta, con su descripción general y el diagrama de la solución general.

5.1 Descripción General

Se propone el desarrollo de *Verific*, una herramienta para la generación automática y aleatoria de casos de prueba, combinado con testing automático. La solución planteada contempla utilizar un lenguaje de especificación de programas en conjunto con una librería de *Property-based testing* que establece un escenario de entrada, ejecuta el código bajo pruebas aleatorias y, a continuación, comprueba la correctitud de la salida. Entonces la esencia de la solución propuesta es según dada la especificación, combinar la declaración de condiciones de correctitud con mecanismos de generación automática de tests de prueba, y así al definir un ejercicio de programación, el profesor o ayudante precise las propiedades y éstas se utilicen para generar casos de prueba, pudiendo saber automáticamente si las respuestas de los alumnos están correctas y con la posibilidad de abundantes casos de prueba.

Más específicamente se utilizará el lenguaje de especificación ACSL y una implementación de quichcheck para lenguaje C.

5.2 Diagrama General de la Solución

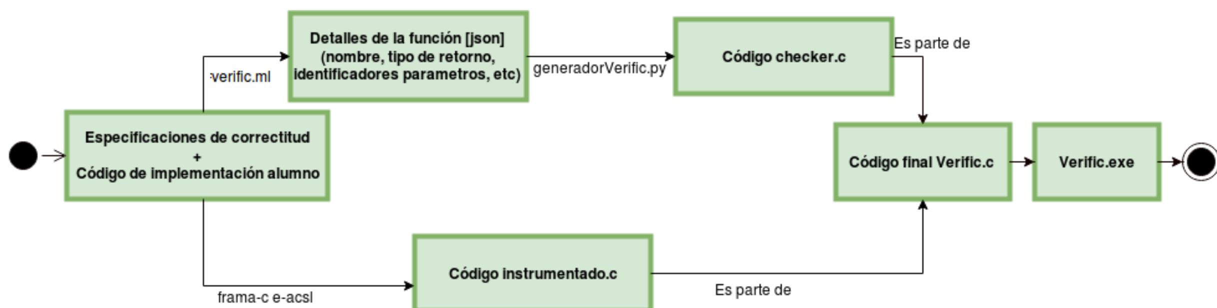

Figura 5.1 Diagrama general de la solución

La resolución del problema comienza con una cabecera de función de programa en lenguaje C y su especificación de propiedades en el lenguaje de especificación E-ACSL. Luego por un lado el plug-in "Verific" de Frama-C, a implementar en este trabajo, obtiene las características de las precondiciones del problema, es decir la cantidad y el tipo de dato de el o los argumentos que recibe esa función, luego mediante el framework quickcheck4c se genera el código que permite producir gran cantidad de casos de prueba. Por otro lado Frama-C E-ACSL plug-in genera un código instrumentado que verificará las especificaciones asociadas a

la cabecera de función, que detecta y aborta la subrutina en el caso de existir si se infringe una anotación. Por último al concatenar el código instrumentado, el código quickcheck4c y la cabecera de función con especificaciones se realiza la integración con la plataforma Coderunner que recibe el código del alumno.

6 Catálogo de Problemas Seleccionados

Según lo propuesto en el objetivo general se selecciona una muestra significativa de problemas, para evaluar en la plataforma CodeRunner, para validar la solución se trabajará solo en ejercicios relevantes para validar la solución, se dividieron los ejercicios en dos clasificaciones, una de ejercicios con variables simples y otra de ejercicios con arreglos. Coincide que en el módulo 1 de la asignatura se ven ejercicios con variables simples y en el módulo 2 de la asignatura se desarrollan ejercicios con arreglos.

6.1 Variables simples (Modulo 1)

Los contenidos del módulo 1 son: introducción al C, elementos básicos de C, estructuras de control, estructuras selectivas, estructuras iterativas, estilo de codificación y se desarrollan ejercicios con variables simples.

En donde se plantea la comprobación de las siguientes funciones:

1. `int duplicar(int n);` recibe un entero y retorna el entero duplicado.
2. `int esA(char letra);` recibe una letra y retorna 1 en el caso que la letra sea a y 0 en el caso que no lo sea.
3. `int maximo(int a,intb) ;` recibe dos enteros a y b , retornando el mayor de estos, si son iguales retorna cualquiera.
4. `int comparaAnd(bool a,bool b) ;` recibe dos boolean hace la comparacion logica "And" y la retorna.

6.2 Módulo 2

Los contenidos del módulo 2 son: vectores estáticos y operaciones, cadenas estáticas y operaciones, vectores multidimensionales estáticos y operaciones, registros y operaciones, definición de Nuevos tipos de Datos.

En donde se plantea la comprobación de las siguientes funciones:

1. `int buscarIndice (int n,int* arr, int v);` recibe un arreglo de enteros, su largo y un entero del cual se busca su posicion, si no se encuentra en el arreglo retornar -1
2. `int encontrarLetra(int n, char* arr, char v);` recibe un arreglo de caracteres, su largo y un caracter del cual se busca su posicion, si no se encuentra en el arreglo retornar -1.
3. `int estaOrdenadoInt (int * a,int n),` recibe un arreglo de enteros y su largo, retorna 1 si este está ordenado de menor a mayor o 0 en el caso contrario
4. char letraMaxima(int n, char * arr) , recibe un arreglo de enteros y su largo,retorna la letra de mayor valor según la tabla ascci.

6.3 Problemas Relevantes Sin Especificación Completa

Dentro del lenguaje C, nos encontramos con un problema, teniendo dos variables de tipo float, no siempre la multiplicación de a*b sea completamente igual a la multiplicación de b*a lo que plantea un inconveniente a la hora de contrastar el resultado del codigo del alumno con el resultado obtenido al aplicar la propiedad, ya que dependería de qué variable se premultiplica y cual se postmultiplica lo que matemáticamente no es correcto al hablar de numeros reales.

E-ACSL explica en su manual [referencia al manual https://frama-c.com/download/e-acsl/e-acsl.pdf] que los números reales exactos son difíciles de implementar y que por lo tanto no soportan esta precisión en esos tipos de datos.

Es por esto que una posible solución a este dilema seria: en los casos que sea necesario ocupar estos tipos de datos, realizar la comparación con un margen de error que puede ser definible y que debiese depender de la cantidad de variables que correspondan a valores de números reales y a sus valores máximos o mínimos, por lo tanto, en este caso la comparación no se realizaría comparando la igualdad entre dos números reales obtenidos, sino que comparando que el valor del error entre lo obtenido y lo esperado sea menor que el error soportado.

7 Esquema de la Solución Específica

Para ejemplificar se tomará el proceso de la solución para la función `int encontrarLetra (int n,char * arr,char` que se debe desarrollar según al siguiente enunciado:

Desarrolle una función en C que recibe como parámetros el largo del arreglo, el arreglo y un carácter a buscar. Si el carácter a buscar está dentro del arreglo la función debe retornar el índice de la primera aparición del carácter de izquierda a derecha, en el caso de no encontrar el carácter la función debe retornar -1.

La cabecera de la función es `int encontrarLetra (int n,char * arr,char v)`.

Primeramente se crea el archivo `encontrarLetra.i` que debe contener la declaración de propiedades de correctitud , las restricciones de las variables y el código implementado por el alumno, al procesar con verific este archivo se generará el ejecutable `encontrarLetra.exe` que al ser ejecutado arroja si el la implementación del alumno fue correcta, en el caso de pasar todos los casos de prueba o en el caso contrario con qué valores falla la implementación del alumno.

Dentro de la carpeta Verific se debe ejecutar el script verific-script.sh , entregandole como argumento el archivo.i que se desea comprobar.

Ejemplo:

```
$ ./verific-script.sh encontrarLetra.i
```

Figura 7.1 Llamada al script verific

7.1 Declaración de Propiedades de Correctiud

Analizando las propiedades de correctitud de forma matemática nos encontramos con:

$$\forall\, i \in Z / \{0 \leq i \leq n-1\}$$
$$\text{Resultado}= i \rightarrow \exists \text{ Arreglo}[i] == \text{v}$$
$$\text{Resultado} = -1 \rightarrow \nexists \text{Arreglo}[i] == \text{v}$$

Figura 7.2 Propiedad de correctitud en forma matemática

Lo que escrito en el lenguaje E-ACSL sería de la forma:

```
/*@ requires n > 0 && \valid(arr+(0..n-1));
@ ensures -1 <= \result <= n-1;
@ behavior success:
@  ensures \result >= 0 ==> arr[\result] == v;
@ behavior failure:
@  ensures \result == -1 ==>
@  \forall integer k; 0 <= k < n ==> arr[k] != v;
@*/
```

Figura 7.3 Propiedades de correctitud en E-ACSL

Las propiedades de correctitud deben ser escritas entre comentarios comentarios especiales de la forma:
`/*@ @*/.`

La notación `requires` nos indica las precondiciones necesarias en este caso, que n sea mayores a 0 y que existan las posiciones de memoria válidas para el arreglo denotado con la cláusula `\valid`, dentro de `\valid` se validan los índices desde 0 hasta n-1.

`behaviour` nos indica que existe más de un caso posible dentro del árbol de soluciones, en este caso hay dos `behaviour`, por un lado `behaviour success`, que se encarga de contralar el caso en que exista la letra dentro del arreglo de caracteres , por otro lado `behaviour failure`, que controla el caso en donde el carácter no se encuentre en el arreglo.

En el caso de `behaviour success`, se asegura con la cláusula ensures que si el resultado es mayor que 0 entones en la posición \result se encontrará el carácter v.

En el caso de `behaviour failure`, se asegura que si el resultado es -1 entonces para todo k entero que pertenezca al intervalo 0<.=k<n entonces se cumple siempre que arreglo[k] es distinto del carácter v;

7.2 Restricciones de las Variables

Las restricciones de las variables deben ir entre comentarios especiales de la forma /*## *##/ en el archivo nombreDeArchivo.i . Se debe tener en cuenta la cabecera de la función `int encontrarLetra (int n,char * arr,char v)` y describir la restricción para cada parámetro de la función.

En este caso las restricciones son de la forma:

```
/*##{"restriction":["5 to 6","6","0 to 128"]}  ##*/
```

Figura 7.4 Ejemplo de restricciones

En donde se escribe en formato json la restricción de cara variable, la primera restricción es “5 to 6” para que genere siempre el valor 5 en el largo del arreglo, luego “6” , para la generación del arreglo de char de largo 5 y por ultimo la generación del carácter entre el rango 0 a 128.

7.3 Implementación Alumno

Se debe ingresar en el archivo encontrarLetra.i la implementación del alumno, esta implementación puede ser correcta o incorrecta, el caso de una implementación correcta puede ser:

```c
int encontrarletra( int n  , char * arr,char v) {
  int i,j;

   for(i=0;i<n;i++)
        {
                if(arr[i]== v)
                        return i;

        }
   return -1;

}
```

Figura 7.4 Ejemplo implementación correcta

Una implementación incorrecta, en el caso de confundir el retorno del índice con el retorno de la posición puede ser :

```c
int encontrarletra( int n  , char * arr,char v) {
  int i,j;

   for(i=0;i<n;i++)
        {
                if(arr[i]== v)
                        return i+1;

        }
   return -1;

}
```

Figura 7.4 Ejemplo implementación incorrecta

7.4 Código Instrumentado

Para el caso de encontrarLetra.i al correr Frama-C E-ACSL en se genera el código instrumentado, creando un archivo de la nombre encontrarLetra.verific.instr, el código instrumentado abortará el proceso en el caso de que la función programada por el alumno no cumpla con alguna de las condiciones de correctitud declaradas anteriormente.

El código instrumentado en este caso tiene declaraciones de estructuras necesarias, las declaraciones de correctitud y el código del alumno en comentarios y como funciones sustanciales posee su propia versión de la función `int encontrarLetra (int n,char * arr,char v)` que es escrita según la función implementada por el alumno.

En el caso de haber seguido la implementación que anteriormente se mencionó como correcta esta función queda de la forma:

```
65 int encontrarletra(int n, char *arr, char v)
66 {
67   int __retres;
68   int i;
69   i = 0;
70   while (i < n) {
71     if ((int)*(arr + i) == (int)v) {
72       __retres = i;
73       goto return_label;
74     }
75     i ++;
76   }
77   __retres = -1;
78   return_label: return __retres;
79 }
```

Figura 7.5 Función encontrar letra según una implementación correcta por parte del alumno

En el caso de haber seguido la implementación que anteriormente se mencionó como correcta esta función queda de la forma:

```
65 int encontrarletra(int n, char *arr, char v)
66 {
67   int __retres;
68   int i;
69   i = 0;
70   while (i < n) {
71     if ((int)*(arr + i) == (int)v) {
72       __retres = i+1;
73       goto return_label;
74     }
75     i ++;
76   }
77   __retres = -1;
78   return_label: return __retres;
79 }
80
```

Figura 7.5 Función encontrar letra según una implementación incorrecta por parte del alumno

También dentro del código instrumentado se genera la función

`int __gen_e_acsl_encotrarletra(int n , cahr *arr,char v)` que hará uso de la función `encontrarletra(int n, char * arr , char v)`

mencionada anteriormente, sea en el caso de implementación correcta o incorrecta `__gen_e_acsl_encotrarletra` será de la forma:

```
 92 int __gen_e_acsl_encontrarletra(int n, char *arr, char v)
 93 {
 94   int __gen_e_acsl_at_6;
 95   char __gen_e_acsl_at_5;
 96   char *__gen_e_acsl_at_4;
 97   char __gen_e_acsl_at_3;
 98   char *__gen_e_acsl_at_2;
 99   long long __gen_e_acsl_at;
100   int __retres;
101   __gen_e_acsl_at_6 = n;
102   __gen_e_acsl_at_5 = v;
103   __gen_e_acsl_at_4 = arr;
104   __gen_e_acsl_at_3 = v;
105   __gen_e_acsl_at_2 = arr;
106   __gen_e_acsl_at = (long long)n;
107   __retres = encontrarletra(n,arr,v);
108   {
109     int __gen_e_acsl_and;
110     int __gen_e_acsl_implies;
111     int __gen_e_acsl_implies_2;
112     if (-1 <= __retres) __gen_e_acsl_and = __retres <= __gen_e_acsl_at - 1LL;
113     else __gen_e_acsl_and = 0;
114     __e_acsl_assert(__gen_e_acsl_and,(char *)"Postcondition",
115                     (char *)"encontrarletra",
116                     (char *)"-1 <= \\result <= \\old(n) - 1",2);
117     if (! (__retres >= 0)) __gen_e_acsl_implies = 1;
118     else {
119       int __gen_e_acsl_valid_read;
120       __gen_e_acsl_valid_read = __e_acsl_valid_read((void *)(__gen_e_acsl_at_2 + __retres),
121                                                     sizeof(char),
122                                                     (void *)__gen_e_acsl_at_2,
123                                                     (void *)(& __gen_e_acsl_at_2));
124       __e_acsl_assert(__gen_e_acsl_valid_read,(char *)"RTE",
125                       (char *)"encontrarletra",
126                       (char *)"mem_access: \\valid_read(__gen_e_acsl_at_2 + __retres)",
127                       4);
128       __gen_e_acsl_implies = *(__gen_e_acsl_at_2 + __retres) == __gen_e_acsl_at_3;
129     }
130     __e_acsl_assert(__gen_e_acsl_implies,(char *)"Postcondition",
131                     (char *)"encontrarletra",
132                     (char *)"\\result >= 0 ==> *(\\old(arr) + \\result) == \\old(v)",
133                     4);
134     if (! (__retres == -1)) __gen_e_acsl_implies_2 = 1;
135     else {
136      int __gen_e_acsl_forall;
137      int __gen_e_acsl_k;
138      __gen_e_acsl_forall = 1;
139      __gen_e_acsl_k = 0;
140      while (1) {
141        if (__gen_e_acsl_k < __gen_e_acsl_at_6) ; else break;
142        {
143          int __gen_e_acsl_valid_read_2;
144          __gen_e_acsl_valid_read_2 = __e_acsl_valid_read((void *)(__gen_e_acsl_at_4 + __gen_e_acsl_k),
145                                                           sizeof(char),
146                                                           (void *)__gen_e_acsl_at_4,
147                                                           (void *)(& __gen_e_acsl_at_4));
148          __e_acsl_assert(__gen_e_acsl_valid_read_2,(char *)"RTE",
149                          (char *)"encontrarletra",
150                          (char *)"mem_access: \\valid_read(__gen_e_acsl_at_4 + __gen_e_acsl_k)",
151                          7);
152          if (*(__gen_e_acsl_at_4 + __gen_e_acsl_k) != __gen_e_acsl_at_5)
153            ;
154          else {
155            __gen_e_acsl_forall = 0;
156            goto e_acsl_end_loop1;
157          }
158        }
159        __gen_e_acsl_k ++;
160      }
161      e_acsl_end_loop1: ;
162      __gen_e_acsl_implies_2 = __gen_e_acsl_forall;
163    }
164    __e_acsl_assert(__gen_e_acsl_implies_2,(char *)"Postcondition",
165                    (char *)"encontrarletra",
166                    (char *)"\\result == -1 ==>\n(\\forall integer k; 0 <= k < \\old(n) ==> *(\\old(arr) + k) != \\old(v))",
167                    6);
168    return __retres;
169  }
170 }
171 
172 
```

Figura 7.6 Ejemplo función __gen_e_acsl_encontrarletra

7.5 Archivo Json con Valores Relevantes de la Función

Para el caso en cuestión se genera el archivo json llamado `encontrarLetra.verfic.json` y que contiene:

```
1 {"name": "encontrarletra", "arg_types": ["int", "char *", "char"], "arg_names": ["n", "arr", "v"], "return": "int"}
```

Figura 7.7 Ejemplo json

Este archivo se genera a partir de la cabecera de la función y sirve para la creación de la función checker que genera valores aleatorios.

7.6 Función Checker

Mediante el framework quickcheck4c , con los valores obtenidos en el archivo json y las restricciones de las variables se genera el archivo de nombre `encontrarLetra.verifi.checker` que genera los valores aleatorios y contiene código en c de la forma:

```
1 #include "/home/angelo/Escritorio/Verific/quickcheck4c.h"
2 #include <stdio.h>
3 #include <string.h>
4 #include <signal.h>
5
6 char *failureMessage;
7
8 static void handleAbort(int signal_number) {
9   if(failureMessage){
10     printf("Los valores con que fallo el test son: %s \n", failureMessage);
11   }
12    exit(1);
13  }
14 QCC_TestStatus encontrarletra_check (QCC_GenValue **vals,int len,QCC_Stamp ** stamp ){
15  int n = *QCC_getValue(vals, 0, int*);
16  char * arr = QCC_getValue(vals, 1, char ** );
17  char v = *QCC_getValue(vals, 2, char*);
18  char *msg_n = vals[0]->show(vals[0]->value, vals[0]->n);
19  char *msg_arr = vals[1]->show(vals[1]->value, vals[1]->n);
20  char *msg_v = vals[2]->show(vals[2]->value, vals[2]->n);
21  failureMessage = (char*) realloc(failureMessage, sizeof (msg_n) +  sizeof (msg_arr) +  sizeof (msg_v) +  100);
22  sprintf(failureMessage," n: %s  arr: %s  v: %s  " ,msg_n  ,msg_arr  ,msg_v  );
23  __gen_e_acsl_encontrarletra( n , arr , v );
24  signal(SIGABRT, &handleAbort);
25  return 1 ;
26 }
27
28 QCC_GenValue* QCC_genIntR0() {
29  return QCC_genIntR(5,6);
30 }
31 QCC_GenValue* QCC_genArrayCharL1() {
32  return QCC_genArrayCharL(6);
33 }
34 QCC_GenValue* QCC_genChar2() {
35  return QCC_genChar(0,128);
36 }
37
38 int main() {
39   QCC_init(0);
40   printf("Testing encontrarletra \n" );
41   QCC_testForAll(1000, 1000, encontrarletra_check, 3, QCC_genIntR0 ,  QCC_genArrayCharL1 ,  QCC_genChar2 );
42
43 return 0;
44 }
```

Figura 7.8 Ejemplo función checker

Es necesario destacar que si la función `__gen_e_acsl_encotrarletra` presente en la línea 23 del archivo , encuentra un caso en el que no corresponde con las propiedades de correctitud, esta enviará el mensaje de "abort" que será tomado por la función de la línea 24 signal(SIGABRT, &handleAbort) que imprimirá los valores que provocaron esta interrupción

7.7 Función Final Verific y Ejecutable

Al concatenar el código instrumentado con la función checker se obtiene el código final en el archivo `encontrarLetra.verific.c` que al ser compilado con el gcc se obtiene el ejecutable `encontrarLetra.verific.exe`, este archivo al ser ejecutado nos arrojará dos tipos de respuestas.

En el caso de que la implementación del alumno sea correcta según las especificaciones de correctitud se evidenciará que pasó los casos de prueba exitosamente de forma:

```
./encontrarLetra.verific.exe
Testing encontrarletra
1000 test passed (0)!
```

Figura 7.9 Ejemplo de ejecutable que pasa 1000 casos de prueba

En el caso de que la implementación del alumno no cumpla con alguna de las especificaciones de correctitud y falle con algún caso de prueba se evidenciará de la forma:

```
./encontrarLetra.verific.exe
Testing encontrarletra
Postcondition failed at line 4 in function encontrarletra.
The failing predicate is:
\result >= 0 ==> *(\old(arr) + \result) == \old(v).
Los valores con que fallo el test son:  n: 5  arr: ['K', '=', 'o', 'F', 's', '{']  v: 's'
```

Figura 7.10 Ejemplo de ejecutable con problemas en la implementación del alumo

En este caso falla el test ya que retorna la posición y no el índice

8 Implementación

El proyecto se implementó utilizando los lenguajes de programación: C , Python y Ocaml además de trabajar con el lenguaje de scripting Bash y el lenguaje de especificación de especificación E-Acsl.

Es necesario destacar que para el correcto funcionamiento de Verfic es necesaria la instalación previa del framework frama-c from opam, python 3 y el compilador gcc gnu. Todo esto está destacado en en el archivo de instalación del Verific.

Existiendo diferentes tareas a realizar por Verfic, primeramente, el encargado de generar json de la especificación es el archivo "verific.ml" en lenguaje Ocaml, luego, la generación de código en C es realizada por el archivo "generador.py" la generación de la versión instrumentada del código se realiza a través del plug in E-Acls que pertenece al framework Frama-C, siguiendo, la función checker es generada mediante el framework Quickchec4c por último se concatena mediante comandos de bash y se crea el ejecutable mediante el compilador gcc.

El script que permite realizar esta seguidilla de instrucciones es "verific-script.sh", cumpliendo con cada una de las etapas de la solución controlando también que se ejecute cada instrucción sin errores, en el caso de existir un error, se detiene la ejecución del scipt y se muestra un mensaje evidenciando en cual instrucción existió un problema.

9 Alcances y Limitaciones

Se logró realizar los objetivos primordiales que tienen realcion con la verficacion y testing aleotorio según propiedades de correctitud dando un termino éxitoso al proyecto.

Sin embargo dentro de los objetivos esperados, por dificultad y tiempo asociados no se logró realizar dos objetivos secundarios:

1. Integrar la plataforma de especificación y verificación con CodeRunner
2. Diseñar y aplicar una evaluación experimental de la plataforma integrada a CodeRunner

Es necesario destacar que las espeficiaciones de propiedades debe ser codificada de forma correcta, de lo contrario se estará comprobando el convencimiento de correctitud de algo que no corresponda, lo que llevaria a errores en los resultados arrojados por Verific.

Otra limitacion es tambien en el uso de valores reales exactos , los que es un caso recurrente de errores en el lenguaje C, ademas que no esta soportado por uno de los frameworks base de esta investigacion E-ACSL por lo que no es posible generar el codigo instrumentado de manera correcta para estos tipos de datos.

10 Conclusiones

Gracias a los conocimientos adquiridos a lo largo de nuestra carrera y a la experiencia adquirida de forma personal se logró llevar a cabo un proyecto de utilidad para el área de la investigación educación y enseñanza de la programación.

Actualmente no existe una herramienta que apoye compruebe en tiempo real la ejecución de un programa en C, según sus declaraciones de correctitud. Los estudiantes en el proceso de aprendizaje actual pueden verse, quizás frustrados al no saber bajo qué condiciones su código no responde correctamente y al tener pruebas estáticas se favorece malas prácticas de programación al corregir un código para un fallo en particular, pueden no estar considerando dar una solución general.

Con este proyecto se pretende ayudar a todos los estudiantes que deseen someterse a este tipo de pruebas y así beneficiar su aprendizaje, exponiéndolo a pruebas que si bien pueden ser más rigurosas, son transparentes ya que se puede saber con qué valores se produce un error. Generar un proyecto con las utilidades mencionadas representa un gran reto, ya que podría materializarse ampliar el alcance de este y que ser útil en otros campos de investigación.

Si bien queda como trabajo pendiente la integración con CodeRunner y la evaluación experimental de la herramienta integrada lograron los objetivos primordiales del proyecto realizando el testing automático y aleatorio con retroalimentación.

Es necesario destacar que para el funcionamiento más simplificado de esta herramienta son necesarios cambios en el lenguaje de especificación E-ACSL que actualmente no soporta todas las especificaciones del lenguaje ACSL y que algunas de estas harían más fácil la especificación de propiedades de correctitud. También es necesario dar solución a un problema propio del lenguaje C, el uso de variables de los tipos de datos float en donde al realizar operaciones que incluyen multiplicación o división de variables con estos tipos de datos, no siempre se realizan de forma conmutativa, una buena forma de análisis de correctitud para este caso podría ser una que se base en desigualdades teniendo en cuenta un margen de error que debe ser estudiado en relación a la cantidad de variables de estos tipos a utilizar y sus valores máximos o mínimos.

La comprobación de funciones mediante testing automático y aleatorio según sus propiedades de correctitud es aplicable a otros lenguajes de programación iniciales además de C, por lo que es posible un trabajo que extienda este proyecto, que contemple solo propiedades de correctitud , que estas propiedades se traspasen a un código intermedio abstracto para luego ser definido en cual lenguaje se desea contrastar con la solución del alumno.

Ha sido una gran experiencia implementar esta herramienta ya que sirvió para obtener nuevos conocimientos no solo de forma teórica sino también de forma práctica.

11 Referencias

[1](*1932), Methods for Analyzing the Content of Motion Pictures (en inglés)* .Disponible en *http://www.brocku.ca/MeadProject/Payne_sundry/Dale_1932.html* .Recuperado el 12 de julio de 2017.

[2]*CodeRunner (en inglés)* .Disponible en *http://coderunner.org.nz/* .Recuperado el 12 de julio de 2017.

[3]N. Smallbone, «Property-based testing for functional programs ,» Chalmers University of Technology and Göteborg University,Göteborg, Sweden, 2011

[4]A.Abdelghany, «Property-based testing for functional in Java ,» https://www.cs.nuim.ie/courses/desem/sites/default/files/PropertyBasedTesting.pdf. Recuperado el 12 de julio de 2017.

[5]*QuickCheck (en inglés)* .Disponible en *http://www.cse.chalmers.se/%7Erjmh/QuickCheck/* .Recuperado el 12 de julio de 2017.

[6]*Quickcheck4c (en inglés)* .Disponible en https://github.com/nivox/quickcheck4c .Recuperado el 12 de julio de 2017.

[7]*theft (en inglés)* .Disponible en *https://github.com/silentbicycle/theft* .Recuperado el 12 de julio de 2017.

[8]*qc (en inglés)* .Disponible en https://github.com/mcandre/qc .Recuperado el 12 de julio de 2017.

[9]*The Java Modeling Language (en inglés)* .Disponible en http://www.eecs.ucf.edu/~leavens/JML//index.shtml .Recuperado el 12 de julio de 2017.

[10] *Caduceus (en inglés)* .Disponible en *http://caduceus.lri.fr/* .Recuperado el 12 de julio de 2017.