SI1001 Teoría de la Computación
Frama-C y su plugin WP

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2025-2

Frama-C y su plugin WP

Preliminares

- El texto guía para esta sección es (Blanchard 2024).
- La numeración (de secciones, definiciones, teoremas, figuras, páginas, etc.) en esta sección corresponde a la numeración en el texto guía.
- Los archivos fuentes de esta sección están en el directorio `src/frama-c`.

Frama-C y su plugin WP

Descripción

- Frama-C (*FRAmework for Modular Analysis of* C *code*) es un plataforma para análisis de programas escritos en C creada por la CEA LIST y el Inria.
- ACSL (*ANSI/ISO C Specification Language*) (pronunciado «Axel») es el lenguaje de especificación para ANSI/ISO C usado por Frama-C.
- WP (*Weakest Precondition*) es un plugin para realizar verificación formal de programas en Frama-C.

Frama-C y su plugin WP

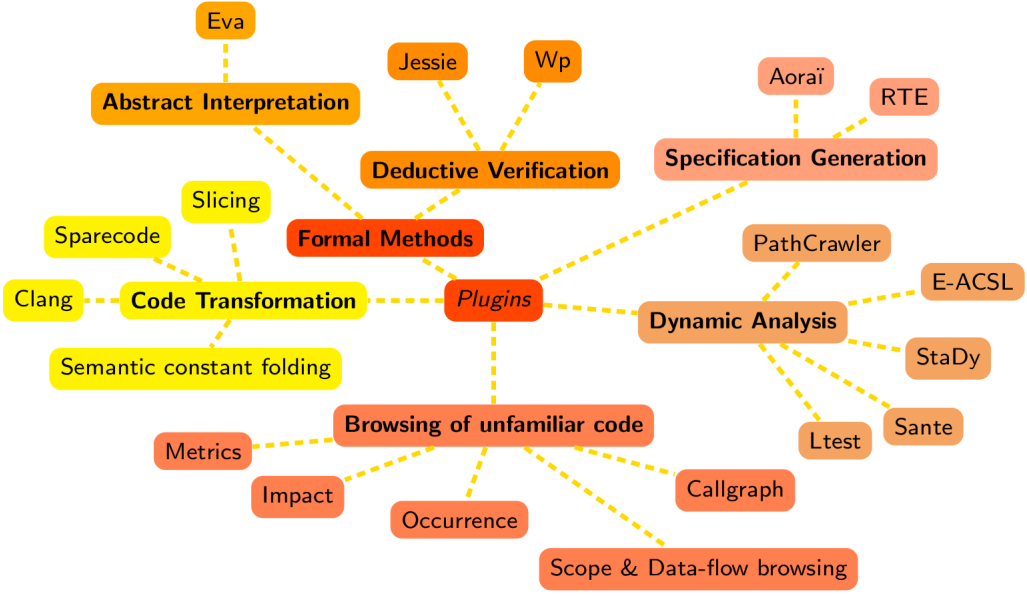

Figura (pág. 13).

Frama-C y su plugin WP

Contratos para funciones

*«The goal of a **function contract** is to state the properties of the input that are expected by the function, and in exchange the properties that will be assured for the output.»* (pág. 22)

Los contratos para funciones están compuestos de **pre-condiciones** y **post-condiciones**, las cuales se escriben en ACSL.

Ejemplo

Calcular el valor absoluto de un número entero (`abs-1.c`).

```
1  int abs(int val) {
2    if ( val < 0 ) return -val;
3    return val;
4  }
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Adicionamos una primera post-condición escrita en ACSL (`abs-2.c`).

```
1  /*@
2    ensures \result >= 0;
3  */
4  int abs(int val) {
5    if ( val < 0 ) return -val;
6    return val;
7  }
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Adicionamos una segunda post-condición escrita en ACSL (abs-3.c).

```
1  /*@
2    ensures \result >= 0;
3    ensures (val >= 0 ==> \result == val) && (val < 0 ==> \result == -val);
4  */
5  int abs(int val) {
6    if ( val < 0 ) return -val;
7    return val;
8  }
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Ejecutamos las instrucciones y demostramos las post-condiciones utilizando el plugin WP.

```
$ cd src/frama-c
$ frama-c-gui abs-3.c
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

¿Es nuestro programa libre de errores? ¿Qué acerca de errores en tiempo de ejecución?

Ejemplo (continuación)

¿Es nuestro programa libre de errores? ¿Qué acerca de errores en tiempo de ejecución?

El pluging RTE (*runtime errors*) adiciona anotaciones al programa para evitar errores en tiempo de ejecución (*integer overflow*, *invalid pointer dereferencing*, división por cero, etc.)

(continua en la próxima diapositiva)

Ejemplo (continuación)

La anotación adicionada por el pluging RTE es debido a la representación de números en el computador. Como es señalado por *The GNU C Library Reference Manual* para la versión 2.36:

> *Most computers use a two's complement integer representation, in which the absolute value of* `INT_MIN` *(the smallest possible* **`int`** *) cannot be represented; thus,* `abs (INT_MIN)` *is not defined.*

(continua en la próxima diapositiva)

Ejemplo (continuación)

Si un número entero es representado con n bits usando la representación de complemento a dos los valores que se pueden representar están en el intervalo $[-2^{n-1}, 2^{n-1} - 1]$:

- Un `int` de C en GCC es representado con 32 bits (en mi computador), es decir, los valores que se pueden representar son $[-2^{31}, 2^{31} - 1] = [-2147483648, 2147483647]$.
- Un `Int` de Haskell en GHC es representado con 64 bits (en mi computador), es decir, los valores que se pueden representar son $[-2^{63}, 2^{63} - 1] = [-9223372036854775808, -9223372036854775807]$.

(continua en la próxima diapositiva)

Ejemplo (continuación)

Programa que muestra el problema empleando la función `abs` de la *GNU C library* (`src/glibc.c`).

```
1  #include <limits.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main() {
6    printf("Maximum int (INT_MAX, 2^31 - 1): %d\n", INT_MAX);
7    printf("Minimum int (INT_MIN, -2^31): %d\n", INT_MIN);
8    printf("Absolute value of INT_MIN (error): %d\n", abs(INT_MIN));
9  }
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Compilar y ejecutar el programa con las siguientes instrucciones:

```
$ cd src
$ make glibc-abs
gcc-15.1.0 -Wall -Wextra -Wpedantic -Werror -std=c23 \
  -o glibc-abs glibc-abs.c
$ ./glibc-abs
Maximum int (INT_MAX, 2^31 - 1): 2147483647
Minimum int (INT_MIN, -2^31): -2147483648
Absolute value of INT_MIN (error): -2147483648
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Programa que muestra el problema empleando la función `abs` de GHC (`src/ghc-abs.hs`).

```
1  main :: IO ()
2  main = do
3    let maxInt = maxBound :: Int
4        minInt = minBound :: Int
5    print $ "Maximum Int (maxBound :: Int, 2^63 - 1): " ++ show maxInt
6    print $ "Minimum Int (minBound :: Int, -2^63): " ++ show minInt
7    print $ "Absolute value of (minBound :: Int) (error): " ++
8      show (abs minInt)
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Compilar y ejecutar el programa con las siguientes instrucciones:

```
$ cd src
$ make ghc-abs
ghc -Wall -Werror -o ghs-abs ghc-abs.hs
$ ./ghc-abs
"Maximum Int (maxBound :: Int, 2^63 -1): 9223372036854775807"
"Minimum Int (minBound :: Int, -2^63): -9223372036854775808"
"Absolute value of (minBound :: Int) (error): -9223372036854775808"
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Adicionamos la pre-condición asociada a la representación de los números enteros (`abs-4.hs`).

```
1  #include <limits.h>
2
3  /*@
4    requires INT_MIN < val;
5    ensures \result >= 0;
6    ensures (val >= 0 ==> \result == val) && (val < 0 ==> \result == -val);
7  */
8  int abs(int val) {
9    if ( val < 0 ) return -val;
10   return val;
11 }
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

Ejecutamos Frama-C con las instrucciones

```
$ cd src/frama-c
$ frama-c-gui -wp -wp-rte abs-4.c
```

(continua en la próxima diapositiva)

Ejemplo (continuación)

También podemos verificar que las llamadas a la función `abs` satisfagan las pre-condiciones (`abs-5.hs`).

```
1 void foo(int a) {
2   int b = abs(42);
3   int c = abs(-42);
4   int d = abs(a);       // Warning: "a" may be INT_MIN
5   int e = abs(INT_MIN); // False: the parameter is INT_MIN
6 }
```

(continua en la próxima diapositiva)

La función `main`

La función `main` es el punto de entrada del programa. Por esta razón al contexto de su evaluación `Frama-C` adiciona el valor inicial de la variables globales.

Ejemplo

Una función `main` (`main.c`).

```
1  int h = 42;
2
3  //@ requires 0 <= h <= 100 ;
4  int main() {
5    //@ assert h == 42 ;
6  }
7
8  //@ requires 0 <= h <= 100 ;
9  void foo() {
10   //@ assert h == 42 ;
11 }
```

(continua en la próxima diapositiva)

Frama-C y su plugin WP

Ejemplo (continuación)

Ejecutamos Frama-C con las instrucciones

```
$ cd src/frama-c
$ frama-c-gui -wp -wp-rte main.c
```

Apuntadores

El libro (Kernighan y Ritchie 1988) comienza la presentación de los apuntadores de la siguiente manera:

> *A pointer is a variable that contains the address of a variable. Pointers are much used in* C*, partly because they are sometimes the only way to express a computation, and partly because they usually lead to more compact and efficient code than can be obtained in other ways... (pág. 83)*
>
> *Pointers have been lumped with the* `goto` *statement as a marvelous way to create impossible-to-understand programs. This is certainly true when they are used carelessly, and it is easy to create pointers that point somewhere unexpected. With discipline, however, pointers can also be used to achieve clarity and simplicity. (pág. 83)*

Apuntadores

En C:

- el operador unario `&` da la dirección de un objeto (variable, función, etc) y
- el operador unario `*` es el operador de desreferenciación *«dereferencing operator»*, que cuando aplicado a un apuntador, accede al objeto apuntado por el apuntador.

Ejemplo

```
1  int x = 1;
2  int y = 2;
3  int* p;  /* ip is a pointer to int */
4
5  p = &x;  /* p now points to x */
6  y = *p;  /* y is now 1 */
7  *p = 0;  /* x is now 0 */
```

Paso de parámetros a una función

El paso de parámetros a una función puede ser **por valor** *«call by value»* o **por referencia** *«call by reference»*.

- Paso por valor: La función recibe el valor de la variable que se pasa y crea una copia local de ésta. En consecuencia, no es posible modificar la variable pasada.
- Paso por referencia: La función recibe la dirección de la variable que se pasa. En consecuencia, los cambios realizados por la función o el método se realizarán sobre la variable pasada.

En C utilizamos apuntador para pasar los argumentos por referencia.

Ejemplo

Contrato para un función recibiendo los argumentos por referencia (swap-1.c).

```
1  /*@
2    ensures *a == \old(*b) && *b == \old(*a);
3  */
4  void swap(int* a, int* b) {
5    int tmp = *a;
6    *a = *b;
7    *b = tmp;
8  }
```

(continua en la próxima diapositiva)

Ejemplo

Debemos verificar que los apuntadores sean válidos (`swap-2.c`).

```
1  /*@
2    requires \valid(a) && \valid(b);
3    ensures  *a == \old(*b) && *b == \old(*a);
4  */
5  void swap(int* a, int* b) {
6    int tmp = *a;
7    *a = *b;
8    *b = tmp;
9  }
```

(continua en la próxima diapositiva)

Ejemplo

¿Por qué falla el contrato? (swap-3.c).

```
1  int h = 42;
2
3  int main() {
4    int a = 37;
5    int b = 91;
6
7    //@ assert h == 42;
8    swap(&a, &b);
9    //@ assert h == 42;
10 }
```

(continua en la próxima diapositiva)

Frama-C y su plugin WP

Ejemplo

Debemos especificar los efectos colaterales *«side effects»* de la función (`swap-4.c`).

```
1  /*@
2    requires \valid(a) && \valid(b);
3    assigns *a, *b;
4    ensures  *a == \old(*b) && *b == \old(*a);
5  */
6  void swap(int* a, int* b) {
7    int tmp = *a;
8    *a = *b;
9    *b = tmp;
10 }
```

Referencias


Allan Blanchard (2 de nov. de 2024). Introduction to C program proof with Frama-C and its WP plugin. Recent version from the GitHub repository. 2 de nov. de 2024. URL: https://github.com/AllanBlanchard/tutoriel_wp (vid. pág. 2).


Brian W. Kernighan y Dennis M. Ritchie (1988). The C Programming Language. 2.ª ed. Sofware Series. Prentice-Hall, 1988 (1978) (vid. pág. 24).