Programación III

Informe de la Práctica 1

Verificación Formal con SPARK 2014

Curso 2020/2021

1. Nombre y apellido de los miembros del equipo.

Sergio Teodoro Castellano Betancor

1. Listado enumerado con el nombre y tipo (procedimiento/función) de los procedimientos y funciones verificados.

*(1) function addTenXTimes*

*(2) procedure fromNegativeToZero,*

*(3) procedure median*

*(4) procedure substractTenAddTen.*

1. Tabla que muestra qué características de SPARK se han utilizado para verificar formalmente cada uno de los procedimientos y funciones.

*El número de cada columna se corresponde con el número asignado a los métodos en el apartado (2). Si necesitas más columnas añade más columnas a las tablas de esta plantilla.*

*En cada casilla solamente hay que marcar con ‘X’ (en el centro de la casilla) si la verificación de este método utiliza la característica de SPARK indicada en el margen derecho de esa fila* ***y su valor no es NULL ni True****.*

*El uso de Contract\_Cases es opcional.*

***Para que tu práctica puntúe debes tener al menos una X en cada una de las filas (excepto en la fila de Contract\_Cases, que es opcional)***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| Global | X | X |  |  |
| Depends | X | X | X | X |
| Pre | X | X | X | X |
| Post | X | X |  | X |
| Contract\_Cases |  |  | X |  |
| ‘Result | X |  |  |  |
| ‘Old |  |  |  | X |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| for all |  |  |  | X |
| for some |  | X |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| Loop\_Variant |  | X |  |  |
| Loop\_Invariant | X | X |  | X |
| ‘Loop\_Entry | X |  |  | X |

Número de tests unitarios hechos para comprobar cada procedimiento y función. De nuevo, el número de cada columna se corresponde con el número asignado a los métodos en el apartado (2). Si necesitas más columnas añade más columnas a esta tablas.

**El número mínimo de tests por cada ejercicio es 3**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| Número de tests | 4 | 5 | 4 | 6 |

1. Cabecera completa (con su contrato) y cuerpo de cada uno de los procedimientos y funciones verificados formalmente (incluyendo el comentario que describe su comportamiento, y manteniendo la numeración del apartado (2)).

*En cada procedimiento/función debes especificar si debe verificarse con un* ***nivel de verificación mínimo****. En caso de no especificar nada se PRESUPONE que se verifica con nivel 0.*

***Utiliza fuente de letra pequeña para facilitar la lectura del código SPARK. Respeta también el sangrado del código.***

**Cuerpo 1.**

package body Pkg\_addTenXTimes with SPARK\_Mode is

function addTenXTimes (X : Natural) return Positive is

Res : Positive;

begin

if X = 0 then

return N;

end if;

Res := N;

for i in 1 .. X loop

Res := Res + N;

pragma Loop\_Invariant(Res = Res'Loop\_Entry \* i + N);

end loop;

return Res;

end addTenXTimes;

end Pkg\_addTenXTimes;

**Cabecera 1.**

package Pkg\_addTenXTimes with SPARK\_Mode is

N : Positive := 10;

function addTenXTimes (X : Natural) return Positive

-- Add 'N' to the number 10 X times.

with

Global => (Input => N),

Depends => (addTenXTimes'Result => (X, N)),

Pre => X < Natural'Last/N and then N = 10,

Post => addTenXTimes'Result = N \* X + 10;

end Pkg\_addTenXTimes;

**Cuerpo 2.**

package body Pkg\_fromNegativeToZero with SPARK\_Mode is

procedure fromNegativeToZero (Vec : in out T\_Vec) is

i : Positive := 1;

begin

while i <= Vec'Last loop

if Vec(i) < 0 then

Vec(i) := zero;

end if;

pragma Loop\_Invariant(i in 1 .. Vec'Last);

pragma Loop\_Variant(Increases => i);

pragma Loop\_Invariant(if Vec(i) < 0 then

Vec(i) = zero);

i := i + 1;

end loop;

end fromNegativeToZero;

end Pkg\_fromNegativeToZero;

**Cabecera 2.**

package Pkg\_fromNegativeToZero with SPARK\_Mode is

type T\_Vec is array (Positive range <>) of Integer;

zero : Natural := 0;

procedure fromNegativeToZero (Vec : in out T\_Vec)

-- Replace the negative numbers of the array passed by parameter with 'zero'.

with

Global => (input => zero),

Depends => (Vec => (Vec, zero)),

Pre => Vec'Length > 0 and then Vec'First = 1 and then Vec'Last < Positive'Last,

Post => (for some i in Vec'Range => Vec(i) in 0 .. Integer'Last);

end Pkg\_fromNegativeToZero;

**Cuerpo 3.**

package body Pkg\_Median with SPARK\_Mode is

procedure Median (Vec : T\_Vec; Res : out Integer) is

begin

if Vec'Length = 1 then

Res := Vec(Vec'First);

else

Res := Vec((Vec'Length/2));

end if;

end Median;

end Pkg\_Median;

**Cabecera 3.**

package Pkg\_Median with SPARK\_Mode is

type T\_Vec is array (Positive range <>) of Integer;

procedure Median (Vec : T\_Vec; Res : out Integer)

-- Find the median of a given vector and store the result in 'Res'.

with

Global => null,

Depends => (Res => Vec),

Pre => Vec'Length > 0 and then Vec'First = 1 and then Vec'Last = Vec'Length,

Contract\_Cases => ((Vec'Length = 1) => Res = Vec(1),

(Vec'Length > 1) => Res = Vec((Vec'Length/2)));

end Pkg\_Median;

**Cuerpo 4.**

package body Pkg\_substractTenAddTen with SPARK\_Mode is

procedure substractTenAddTen (Vec : in out T\_Vec) is

begin

if Vec'Length = 1 then

Vec(1) := Vec(1) - 10;

else

for i in 1 .. Vec'Length/2 loop

Vec(i) := Vec(i) - 10;

pragma Loop\_Invariant(for all j in 1 .. i =>

Vec(j) = Vec'Loop\_Entry(j) - 10);

end loop;

for j in (Vec'Length/2) + 1 .. Vec'Last loop

Vec(j) := Vec(j) + 10;

pragma Loop\_Invariant(for all k in (Vec'Length/2) + 1 .. j =>

Vec(k) = Vec'Loop\_Entry(k) + 10);

end loop;

end if;

end substractTenAddTen;

end Pkg\_substractTenAddTen;

**Cabecera 4.**

package Pkg\_substractTenAddTen with SPARK\_Mode is

type T\_Vec is array (Positive range <>) of Integer;

procedure substractTenAddTen (Vec : in out T\_Vec)

-- Subtract 10 from the elements of the first half of a given array.

-- Then add 10 to the other half.

with

Global => null,

Depends => (Vec => Vec),

Pre => Vec'Length > 0 and then Vec'First = 1 and then Vec'Last < Positive'Last

and then

(if Vec'Length = 1 then

Vec(1) >= Integer'First + 10

else

(for all i in 1 .. Vec'Length/2 =>

(if Vec(i) < 0 then

Vec(i) >= Integer'First + 10))

and then

(for all j in (Vec'Length/2) + 1 .. Vec'Last =>

(if Vec(j) > 0 then

Vec(j) <= Integer'Last - 10))),

Post => (if Vec'Length = 1 then

Vec(1) = Vec'Old(1) - 10

else

(for all i in 1 .. Vec'Length/2 =>

Vec(i) = Vec'Old(i) - 10)

and then

(for all j in (Vec'Length/2) + 1 .. Vec'Last =>

Vec(j) = Vec'Old(j) + 10));

end Pkg\_substractTenAddTen;

Entrega adicional

1. (Video) Cada equipo debes subir un video de un máximo de 10 minutos explicando la verificación formal de su código: precondiciones, postcondiciones, invariantes.

El vídeo se encuentra en la entrega que cuenta con dos archivos: Entrega.zip y Práctica3.mp4 ya que solo puedo subir dos archivos de 50MB o menos.