



Taller de Especificación Formal Especificativa++

Prof: José Luis Sierra Rodríguez



Introducción

- ESPECIFICA++ significa (Executable SPECIFICAtions in C++).
- ESPECIFICA++ permite utilizar un subconjunto de C++ para describir especificaciones.
- Las especificaciones resultantes serán expresiones C++:
 - Pueden ejecutarse, lo mismo que cualquier otra expresión de C++.
 - Pueden validarse utilizando casos de prueba, lo mismo que cualquier programa C++.
- ESPECIFICA++ permite utilizar los mismos mecanismos que se utilizan para desarrollar programas en el desarrollo de especificaciones formales:
 - Construir incrementalmente las especificaciones.
 - Probarlas utilizando casos de prueba significativos, para “asegurarnos” de que realmente están expresando lo que queremos que expresen.



Cuantificación universal

- La cuantificación universal en ESPECIFICA++ permite expresar (como el resto de cuantificaciones en ESPECIFICA++) cuantificaciones universales en las que las variables cuantificadas varían en rangos finitos de valores (v.g., un intervalo de enteros).
- Este tipo de cuantificaciones son las que habitualmente aparecen en la especificación formal de propiedades de programas
- Ejemplo:
 - Todos los elementos del vector almacenado en las **n** primeras posiciones de **a** son positivos:
$$\forall i: 0 \leq i < n: a[i] > 0$$
 - Todos los valores del vector son distintos:
$$\forall i, j: 0 \leq i < n \wedge 0 \leq j < n \wedge i \neq j: a[i] \neq a[j]$$

Cuantificación universal

■ **epp_forall**(*<tipo variables>*,
 (*<lista de variables cuantificadas>*),
 <valor inicial>, } **Rango de variación**
 <valor final>,
 <restricciones adicionales>,
 <predicado cuantificado>)

- *<tipo variables>*: Tipo del rango de valores en los que varían las variables cuantificadas (normalmente **int**).
- *<lista de variables cuantificadas>*: Lista de las variables que se cuantifican, separadas por comas (,). **Importante**: la lista siempre debe encerrarse entre paréntesis, incluso aunque haya una única variable.
- *<valor inicial>* y *<valor final>*: Definen el rango en el que varían las variables cuantificadas.
- *<restricciones adicionales>*: Expresión booleana (*predicado*) que determina qué valores de variables son válidas.
- *<predicado cuantificado>*: Expresión booleana que caracteriza la propiedad que tiene que cumplirse.

- Si no hay ninguna asignación de valores a variables que satisfagan *<restricciones adicionales>*, el valor es **true**.



Cuantificación universal

- Ejemplo:

- Todos los elementos del vector almacenado en las **n** primeras posiciones de **a** son positivos:

- Formalización: $\forall i: 0 \leq i < n: a[i] > 0$

- Expresión en ESPECIFICA++

```
epp_forall(int, (i), 0, n-1, true, a[i]>0)
```

- Todos los valores del vector son distintos:

- Formalización: $\forall i, j: 0 \leq i < n \wedge 0 \leq j < n \wedge i \neq j: a[i] \neq a[j]$

- Expresión en ESPECIFICA++

```
epp_forall(int, (i,j), 0, n-1, i != j, a[i]!=a[j])
```



Cuantificación existencial

- Expresión de cuantificaciones existenciales con variables variando en rangos finitos de valores.
- Tienen la misma forma que las cuantificaciones universales:
- **epp_exists**(*<tipo variables>* ,
 (*<lista de variables cuantificadas>*),
 <valor inicial> ,
 <valor final> ,
 <restricciones adicionales> ,
 <predicado cuantificado>)
- Los argumentos tienen exactamente la misma forma, significado y propósito que los de **epp_forall**
- Si no hay ninguna asignación de valores a variables que satisfagan *<restricciones adicionales>*, el valor es **false**.



Cuantificación existencial

- Ejemplo:

- El vector tiene al menos un positivo:

- Formalización: $\exists i: 0 \leq i < n: a[i] > 0$
- Expresión en ESPECIFICA++

```
epp_exists(int, (i), 0, n-1, true, a[i]>0)
```

- El vector contiene valores distintos:

- Formalización: $\exists i, j: 0 \leq i < n \wedge 0 \leq j < n \wedge i \neq j: a[i] \neq a[j]$
- Expresión en ESPECIFICA++

```
epp_exists(int, (i,j), 0, n-1, i != j, a[i]!=a[j])
```



Sumatorio

- Suma de los valores que toma una expresión en la que aparecen las variables cuantificadas, para todas aquellas asignaciones de valores a variables que cumplen las restricciones especificadas.
- **epp_sum**(*<tipo variables>* ,
 (*<lista de variables cuantificadas>*),
 <valor inicial> ,
 <valor final> ,
 <restricciones adicionales> ,
 <tipo expresión> ,
 <expresión cuantificada>)
- Los cinco primeros argumentos tienen la misma forma, significado y propósito que los de **epp_forall** y **epp_exists**
- Hay un nuevo argumento: el tipo de la expresión cuantificada:
 <tipo expresión>
- *<expresión cuantificada>* es la expresión que indica los valores que se suman.

Si no hay ninguna asignación de valores a variables que satisfagan *<restricciones adicionales>*, el valor es 0.



Sumatorio

- Ejemplos:

- Suma de los elementos del vector:

- Formalización: $\sum i: 0 \leq i < n: a[i]$
- Expresión en ESPECIFICA++

```
epp_sum(int, (i), 0, n-1, true, int, a[i])
```

- Suma de los productos de pares de elementos del vector:

- Ejemplo: si el vector es (1, 2, 2, 2, 5), el resultado debe ser $1 \times 2 + 1 \times 2 + 1 \times 2 + 1 \times 5 + 2 \times 2 + 2 \times 2 + 2 \times 5 + 2 \times 2 + 2 \times 5 + 2 \times 5 = 53$
- Formalización: $\sum i, j: 0 \leq i < n \wedge 0 \leq j < n \wedge i < j: a[i] \times a[j]$
- Expresión en ESPECIFICA++

```
epp_sum(int, (i,j), 0, n-1, i < j, int, a[i]*a[j])
```

- Producto de los valores que toma una expresión en la que aparecen las variables cuantificadas, para todas aquellas asignaciones de valores a variables que cumplen las restricciones especificadas.
- **epp_prod**(*<tipo variables>*,
 (*<lista de variables cuantificadas>*),
 <valor inicial>,
 <valor final>,
 <restricciones adicionales>,
 <tipo expresión>,
 <expresión cuantificada>)
- Los argumentos tienen la misma forma, significado y propósito que los de **epp_sum**
- Si no hay ninguna asignación de valores a variables que satisfagan *<restricciones adicionales>*, el valor es 1.

- Ejemplos:

- Producto de los elementos del vector:

- Formalización: $\prod i: 0 \leq i < n: a[i]$
- Expresión en ESPECIFICA++

```
epp_prod(int, (i), 0, n-1, true, int, a[i])
```

- Producto de las sumas de pares del vector:

- Formalización: $\prod i,j: 0 \leq i < n \wedge 0 \leq j < n \wedge i < j: a[i] + a[j]$
- Expresión en ESPECIFICA++

```
epp_prod(int, (i,j), 0, n-1, i < j, int, a[i]+a[j])
```



Maximización

- Máximo de los valores que toma una expresión en la que aparecen las variables cuantificadas, para todas aquellas asignaciones de valores a variables que cumplen las restricciones especificadas.
- **epp_max**(*<tipo variables>* ,
 (*<lista de variables cuantificadas>*),
 <valor inicial> ,
 <valor final> ,
 <restricciones adicionales> ,
 <tipo expresión> ,
 <expresión cuantificada>)
- Los argumentos tienen la misma forma, significado y propósito que los de **epp_sum** y **epp_prod**
- **Importante:** si no hay ninguna asignación de valores a variables que satisfagan el predicado en *<restricciones adicionales>*, la ejecución de este tipo de expresión levanta la excepción EEmptyRange

- Ejemplos:

- Máximo de los elementos del vector:

- Formalización: $\max i: 0 \leq i < n: a[i]$
- Expresión en ESPECIFICA++

```
epp_max(int, (i), 0, n-1, true, int, a[i])
```

- Máximo de los productos de pares del vector:

- Formalización: $\max i,j: 0 \leq i < n \wedge 0 \leq j < n \wedge i < j: a[i] * a[j]$
- Expresión en ESPECIFICA++

```
epp_max(int, (i,j), 0, n-1, i < j, int, a[i]*a[j])
```



Minimización

- Mínimo de los valores que toma una expresión en la que aparecen las variables cuantificadas, para todas aquellas asignaciones de valores a variables que cumplen las restricciones especificadas.
- **epp_min**(*<tipo variables>*,
 (*<lista de variables cuantificadas>*),
 <valor inicial>,
 <valor final>,
 <restricciones adicionales>,
 <tipo expresión>,
 <expresión cuantificada>)
- Mismo formato que **epp_max**
- Si no se encuentra ninguna asignación de valores a variables que satisfaga *<restricciones adicionales>*, se levanta la excepción EEmptyRange

- Ejemplos:

- Mínimo de los elementos del vector:

- Formalización: $\min i: 0 \leq i < n: a[i]$
- Expresión en ESPECIFICA++

```
epp_min(int, (i), 0, n-1, true, int, a[i])
```

- Mínimo de los productos de pares del vector:

- Formalización: $\min i,j: 0 \leq i < n \wedge 0 \leq j < n \wedge i < j : a[i] * a[j]$
- Expresión en ESPECIFICA++

```
epp_min(int, (i,j), 0, n-1, i < j, int, a[i]*a[j])
```



Cuenta

- Número de posibles asignaciones de valores a variables que cumplen las restricciones especificadas, y el predicado de selección indicado.
- **epp_count**(*<tipo variables>*,
 (*<lista de variables cuantificadas>*),
 <valor inicial>,
 <valor final>,
 <restricciones adicionales>,
 <predicado de selección>)
- Mismo formato que **epp_forall** y **epp_exists**
- Dado que esta cuantificación no involucra una expresión, no es necesario especificar el tipo de dicha expresión.
- *<predicado de selección>* es la condición que tienen que cumplir las asignaciones a variables que se contabilizan.
- Si no hay ninguna asignación de variables que cumplan *<restricciones adicionales>* y *<predicado de selección>*, el valor es 0.

- Ejemplos:

- Número de negativos en el vector:

- Formalización: $\# i: 0 \leq i < n: a[i] < 0$
- Expresión en ESPECIFICA++

```
epp_count(int, (i), 0, n-1, true, a[i]<0)
```

- Número de productos negativos de parejas de elementos del vector :

- Formalización: $\# i,j: 0 \leq i < n \wedge 0 \leq j < n \wedge i < j : a[i] * a[j] < 0$
- Expresión en ESPECIFICA++

```
epp_count(int, (i,j), 0, n-1, i < j, a[i]*a[j]<0)
```



Implicación y doble implicación

- Implicación: **epp_imp**(P_0 , P_1). Expresa $P_0 \rightarrow P_1$
- Ejemplo: $a[i] > 0 \rightarrow a[j] = 1$ puede expresarse como

`epp_imp(a[i]>0, a[j]==1)`

- Implicación: **epp_equiv**(P_0 , P_1). Expresa $P_0 \leftrightarrow P_1$
- Ejemplo: $a[i] > 0 \leftrightarrow a[j] = 1$ puede expresarse como

`epp_equiv(a[i]>0, a[j]==1)`

Combinación de predicados y expresiones

- Todas las construcciones anteriores producen expresiones C++:
 - **epp_forall** y **epp_exists** expresiones de tipo **bool** (predicados)
 - **epp_sum**, **epp_prod**, **epp_max** y **epp_min** expresiones del tipo de la expresión cuantificada (tipo que se indica explícitamente en la construcción).
 - **epp_count** expresiones de tipo entero
- Son expresiones C++, como otras cualesquiera \Rightarrow pueden combinarse libremente entre sí, y con otras expresiones C++, siempre y cuando se respeten las restricciones del sistema de tipos de C++.
- Ejemplo:
 - El vector tiene, al menos un 0, o al menos un 1, no tiene valores negativos, y tiene más de 4 elementos:

```
(epp_exists(int,(i),0,n-1,true,a[i]==0) ||  
  epp_exists(int,(i),0,n-1,true,a[i]==1) )  
    &&  
  epp_forall(int,(i),0,n-1,true,a[i]>=0)  
    &&  
  (n>4)
```



Abstracción

- Pueden utilizarse funciones C++ para abstraer predicados y expresiones
- Ejemplo:

```
bool contiene(int a[], int n, int v) {  
    return epp_exists(int,(i),0,n-1,true,a[i]==v);  
}  
  
bool todos_no_negativos(int a[], int n) {  
    return epp_forall(int,(i),0,n-1,true,a[i]>=0);  
}
```

- Ahora, el predicado anterior puede escribirse como:
(contiene(a,n,0) || contiene(a,n,1)) && todos_no_negativos(a,n) && n>0

Operacionalización

- Las cuantificaciones se transforman **invocaciones a funciones** que las *evalúan*.
- La evaluación se realiza mediante bucles anidados que:
 - Generan todas las posibles asignaciones de valores a variables en el rango de variación.
 - Comprueban si las asignaciones cumplen las restricciones adicionales exigidas.
 - ... y, en este caso, las tienen en cuenta para *calcular* el resultado.
- Ejemplo:

epp_sum(int, (i,j), 0, n-1, i < j, int, a[i]*a[j])

Se genera una función de evaluación

```
int evalua_sum(int a[], int n) {  
    int resul = 0;  
    for(int i=0; i <= n-1; i++) {  
        for(int j=0; j <= n-1; j++) {  
            if(i<j) {  
                resul += a[i]*a[j];  
            }  
        }  
    }  
    return resul;  
}
```

La cuantificación se traduce en una invocación a dicha función

evalua_sum(a,n)

Operacionalización

- De esta forma, ESPECIFICA++ genera automáticamente implementaciones de algoritmos (no necesariamente eficientes... de hecho, la mayoría son muy ineficientes, implementaciones de **fuerza bruta**) que evalúan las cuantificaciones.
- Aún así, estas **malas** implementaciones de **fuerza bruta** seguirán siendo suficientes para probar las especificaciones con casos de prueba pequeños (pero significativos).
- Las funciones de evaluación no se generan como funciones con nombre, sino que se utiliza un mecanismo incorporado a partir de la versión 11 de C++ que permite definir y utilizar **funciones anónimas: expresiones lambda**.
- De esta forma, una traducción más exacta del ejemplo anterior sería:

```
[&]() {  
    int resul = 0;  
    for(int i=0; i <= n-1; i++) {  
        for(int j=0; j <= n-1; j++) {  
            if(i<j) {  
                resul += a[i]*a[j];  
            }  
        }  
    }  
    return resul;  
}()
```

Operacionalización

- En realidad, no se genera una función de evaluación para cada cuantificación, sino que se reutilizan implementaciones genéricas que sirven para cada tipo de cuantificación.
- Traducción *real*/del ejemplo anterior:

```
epp_sum(int, (i,j), 0, n-1, i < j, int, a[i]*a[j])
```



```
[&]() {  
    int i, j;  
    std::function<int()> _init = [&]() {return 0; };  
    std::function<int()> _end = [&]() {return n - 1; };  
    std::function<bool()> _filter = [&]() {return i < j; };  
    std::function<int()> _val = [&]() {return a[i] * a[j]; };  
    return _epp::do_qaexp_launcher<int, int, _epp::SUM>(_init, _end,  
                                                    _filter, _val, i, j); }();
```

- Todas las definiciones necesarias para utilizar ESPECIFICA++ están en un archivo .h: `especificapp.h`
- Para utilizar ESPECIFICA++ basta incluir dicho archivo.
- Programa típico utilizado para refinar y probar una especificación:

```
#include "especificapp.h"

#include <iostream>

using namespace std;

// El tramo comprendido entre i y j es un tramo constante
bool tramo_cte(int a[], int i, int j) {
    return epp_forall(int, (k), i, j, true, a[i] == a[k]);
}

// Longitud del tramo constante más largo
int lon_tc_mas_largo(int a[], int n) {
    return epp_max(int, (i, j), 0, n - 1, i <= j && tramo_cte(a, i, j), int, (j - i) + 1);
}

// resul contiene la longitud del tramo cte más largo
bool lmtc(int a[], int n, int resul) {
    return (resul == lon_tc_mas_largo(a, n));
}
```




Uso

```
// PROGRAMA DE PRUEBA

const int N = 20; // numero máximo de elementos

bool lee_caso(int a[], int & n, int & resul) {
    cin >> n;
    if (n != -1) {
        for (int i = 0; i < n; i++) {
            cin >> a[i];
        }
        cin >> resul;
        return true;
    }
    else {
        return false;
    }
}

bool ejecuta_caso() {
    int a[N];
    int n;
    int resul;
    if (lee_caso(a, n, resul)) {
        cout << std::boolalpha << lmtc(a, n, resul) << endl;
        return true;
    }
    else {
        return false;
    }
}

int main() {
    while (ejecuta_caso());
    return 0;
}
```

Uso

- Ejemplo de uso de ESPECIFICA++ para refinar y probar la anotación de un algoritmo iterativo:

```
...
#include <cassert>

...
int lon_max_tc(int a[], int tam_a, int n) {
    /*PRE:*/ assert(0 < n && n <= tam_a);

    int i = 1;
    int len = 1;
    int resul = 1;

    /*INV:*/ assert(lmtc(a, i, resul));
    /*INV:*/ assert(ult_cons_comp(a, i, len));
    /*INV:*/ assert(0 < i <= n);
    /*COTA:*/ assert((n - i) >= 0);
    while (i < n) {
        if (a[i] == a[i - len]) {
            i++;
            len++;
            if (len > resul) resul = len;
        }
        else { i++; len = 0; }

        /*INV:*/ assert(lmtc(a, i, resul));
        /*INV:*/ assert(ult_cons_comp(a, i, len));
        /*INV:*/ assert(0 < i <= n);
        /*COTA:*/ assert((n - i) >= 0);
    }
    /*POS:*/assert(lmtc(a, n, resul));
    return resul;
}
```

}



Ejemplos ESPECIFICA++

- Predicado que, dado un vector de enteros almacenados en las n primeras posiciones de a , establezca que el máximo del vector no aparece repetido en dicho vector

```
int num_veces(int a[], int n, int v) {  
    return epp_count(int,(i),0,n-1,true,a[i] == v);  
}
```

```
int maximo(int a[], int n) {  
    return epp_max(int,(i),0, n - 1,true,int,a[i]);  
}
```

```
num_veces(a, n, maximo(a, n)) == 1
```



Ejemplos ESPECIFICA++

- Predicado que, dado un vector de enteros almacenados en las n primeras posiciones de a , y una variable s , establezca que s es la mayor de las sumas de los valores de los tramos estrictamente decrecientes del vector.

```
int suma(int a[], int i, int j) {  
    return epp_sum(int, (u), i, j, true, int, a[u]);  
}
```

```
auto ted(int a[], int i, int j) {  
    return epp_forall(int, (u), i + 1, j, true,  
                      a[u] < a[u - 1]);  
}
```

```
auto mayor_suma_ted(int a[], int n) {  
    return epp_max(int, (i, j), 0, n - 1,  
                   i <= j && ted(a, i, j), int, suma(a, i, j));  
}
```

```
s == mayor_suma_ted(a, n)
```



Ejemplos ESPECIFICA++

- Predicado que, dado un vector de enteros almacenados en las n primeras posiciones de a , establezca que hay un tramo constante en el vector cuya suma es igual al máximo elemento del vector

```
int tcte(int a[], int i, int j) {  
    return epp_forall(int, (u), i + 1, j, true, a[u] == a[i]);  
}
```

```
bool hay_tcte_con_suma_maxima(int a[], int n) {  
    return epp_exists(int, (i, j), 0, n - 1,  
        i <= j && tcte(a, i, j), suma(a, i, j) == maximo(a, n));  
}
```

```
hay_tcte_con_suma_maxima(a, n)
```