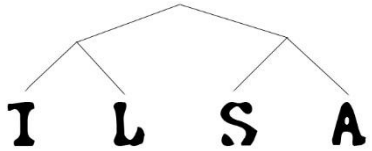
ESPECIFICA++, un DSL para el desarrollo de la competencia en especificación formal en materias de programación

Proyectos INNOVA-DOCENCIA 2024-2025

Mejora del Desarrollo de Competencias Específicas en Materias de Programación Mediante la Construcción de Especificaciones Ejecutables (Proy. Nº 152)

José-Luis Sierra-Rodríguez, Mercedes Gómez-Albarrán, Ana-María González-de-Miguel, Marta López-Fernández, Antonio Sarasa-Cabezuelo



Introducción

- **Programación: materia troncal en enseñanza universitaria de la informática.**
- **Competencia básica en programación: desarrollo de programas correctos**
- **Necesidad de competencia básica en métodos formales.**

Introducción

- **ESPECIFICA++ (Executable SPECIFICAtions in C++): DSL embebido en C++ para la especificación de algoritmos mediante lógica de predicados.**
- **Las fórmulas en ESPECIFICA++ son ejecutables**

Introducción

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

Cuantificaciones en ESPECIFICA++

- Las **cuantificaciones** en ESPECIFICA++ involucran **variables** cuantificadas que varían en **rangos finitos** de valores (v.g., un intervalo de enteros).
- Este tipo de cuantificaciones son las que habitualmente aparecen en la **práctica** durante 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]>0

Cuantificación universal

- **epp_forall**(*<tipo variables>*,
  (*<lista de variables cuantificadas>* **)**,
  <valor inicial>,
  <valor final>,
  <restricciones adicionales>,
  <predicado cuantificado>)

  Rango de variación

  - *<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.

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])
```

Otras cuantificaciones

- Cuantificación **existencial**.
  - Ejemplo: El vector contiene valores distintos:
    - Formalización: $\exists$ i,j: **0 ≤ i < n** $\wedge$ **0 ≤ j < n** $\wedge$ i≠j: a[i] ≠ a[j]
    - Expresión en ESPECIFICA++

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

- **Sumatorio**.
  - Ejemplo: Suma de los productos de pares de elementos del vector:
    - Formalización: Σ i,j: 0 ≤ i < n $\wedge$ 0 ≤ j < n $\wedge$ i<j : a[i] × a[j]
    - Expresión en ESPECIFICA++

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

- **Productorio**.
  - Ejemplo: Producto de las sumas de pares del vector:
    - Formalización: Π i,j: 0 ≤ i < n $\wedge$ 0 ≤ 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])
```

Otras cuantificaciones

- **Minimización**.
  - Ejemplo: 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])
```

- **Maximización**.
  - Ejemplo: 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])
```

- **Cuenta**.
  - Ejemplo: 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)
```

- **Doble 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 ⇒ 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 en **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 utilizan **expresiones lambda** (incorporadas en C++ a partir de la versión 11)
- 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); }();
```

Uso

- 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;
}
```

Uso

- ESPECIFICA++ permite plantear al estudiantado problemas de especificación de naturaleza muy similar a los problemas de diseño de algoritmos
  - Enunciado:

1) Dado un vector de enteros, se dice que una secuencia de valores consecutivos de **v** es un *tramo par* cuando todos los valores son números pares. Especifica un predicado que, dado un vector almacenado en las **n** primeras posiciones del array **a** ($n \geq 0$), sea cierto si y sólo sí en el vector hay, al menos, un tramo par y el valor de la variable **l** coincide con la longitud del tramo par más largo de dicho vector.

Ejemplos

n	Valores en las *n* primeras posiciones de **a**	Valor de **l**	Valor del predicado
10	2, 4, 5, 7, 9, 6, 8, 10, 12, 14	5	**CIERTO**
5	2, 4, 5, 8, 10	2	**CIERTO**
5	2, 4, 5, 8, 10	3	**FALSO**
3	1, 3, 5	0	**FALSO**

(*) El valor en el último caso es **FALSO** porque no hay tramo par.

Uso

– Plantilla de solución:

```
#include "especificapp.h"

#include <iostream>

using namespace std;

bool lon_tp_mas_largo(int a[],  int n, int l) {
          // DEFINE AQUI EL PREDICADO PEDIDO. PUEDES
          // DEFINIR Y UTILIZAR, ASI MISMO, LOS PREDICADOS
          // Y EXPRESIONES AUXILIARES QUE CONSIDERES OPORTUNOS
}


// PROGRAMA DE PRUEBA: NO MODIFICAR

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

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

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

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

Uso

- Problemas de especificación en DomJudge

TALLER ESPECIFICACION preliminary results - not fi

Filter

RANK	TEAM	SCORE	C1	C2	C3	C4	C5	C6	EJ1	EJ10	EJ2	EJ3	EJ4	EJ5	EJ6	EJ7	EJ8	EJ9
1	FAL FAL22	16 195206	17267 1 try	26713 1 try	26747 1 try	26766 1 try	26813 4 tries	27366 2 tries	2538 1 try	5568 1 try	2612 3 tries	2633 1 try	2672 1 try	5295 2 tries	5489 1 try	5505 2 tries	5527 1 try	5535 1 try
2	FAL10	15 162446	18281 1 try	18296 1 try	18311 1 try	18317 1 try	18342 1 try	4 tries	7089 1 try	7084 1 try	7089 2 tries	7083 2 tries	7082 2 tries	7082 1 try	7081 1 try	7081 1 try	7085 1 try	7083 1 try
3	FAL05	12 102517	16479 1 try	16485 1 try	1 try	3 tries			6676 3 tries	7124 1 try	6849 3 tries	6848 2 tries	6847 2 tries	6846 3 tries	6921 1 try	7066 4 tries	7073 1 try	7083 1 try
4	FAL43	10 91423	28647 6 tries						6993 1 try		6911 4 tries	6921 1 try	6930 1 try	6935 1 try	6961 1 try	6974 1 try	6988 1 try	7003 1 try
5	FAL02	9 71656	2 tries	14486 1 try					7035 2 tries	7202 1 try	7 tries	7120 1 try	7127 1 try	7154 1 try	7163 1 try	7171 1 try	7178 1 try	4 tries

ESPECIFICA++, un DSL para el desarrollo de la competencia en especificación formal en materias de programación

Proyecto INNOVA-DOCENCIA 2024-2025

Mejora del Desarrollo de Competencias Específicas en Materias de Programación Mediante la Construcción de Especificaciones Ejecutables (Proy. Nº 152)

José-Luis Sierra-Rodríguez, Ana-María González-de-Miguel, Marta López-Fernández, Antonio Sarasa-Cabezuelo

Grupo de Investigación en Ingeniería del Lenguaje, Software y Aplicaciones

Mercedes Gómez-Albarrán,

Grupo de Investigación en Aplicaciones de la Inteligencia Artificial