Curso de Doctorado:
Lenguajes Integrados Multiparadigma

Pascual Julián Iranzo

Pascual.Julian@uclm.es

Universidad de Castilla – La Mancha

Departamento de Informática

Ciudad Real (España)

Presentación: Objetivos y características del curso

Objetivos

- Que el alumno conozca los fundamentos, características y aplicaciones de los lenguajes declarativos integrados.
- Estudiar los mecanismos operacionales y la semántica asociados a las diferentes propuestas de integración de los lenguajes lógicos y funcionales.
- Revisar las técnicas de implementación.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

2.- Sistemas ecuacionales.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

2.- Sistemas ecuacionales.

3.- Sistemas de reescritura.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

2.- Sistemas ecuacionales.

3.- Sistemas de reescritura.

4.- Narrowing y estrategias de narrowing.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

2.- Sistemas ecuacionales.

3.- Sistemas de reescritura.

4.- Narrowing y estrategias de narrowing.

5.- Residuación.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.

2.- Sistemas ecuacionales.

3.- Sistemas de reescritura.

4.- Narrowing y estrategias de narrowing.

5.- Residuación.

6.- Taxonomía de las técnicas de implementación.

Presentación: Objetivos y características del curso

Indice

1.- Introducción.
2.- Sistemas ecuacionales.
3.- Sistemas de reescritura.
4.- Narrowing y estrategias de narrowing.
5.- Residuación.
6.- Taxonomía de las técnicas de implementación.
7.- Curry: un ejemplo de lenguaje integrado.

Presentación: Objetivos y características del curso

Método de Evaluación

- Realizar y exponer un trabajo relacionado con los contenidos teóricos del curso.
- Los trabajos serán individuales.
- Tiempo de exposición entre 20 minutos (mínimo) y 30 minutos.

Indice

Computadores y Lenguajes Imperativos

- Los computadores modernos presentan una organización y funcionamiento basado en el modelo de von Neumann.

- Conduce a un modelo de cómputo en el que:

  Los procesadores ejecutan las intrucciones secuencialmente

  $\Longrightarrow$ Lenguajes dificiles de paralelizar.

Computadores y Lenguajes Imperativos

- Los computadores modernos presentan una organización y funcionamiento basado en el modelo de von Neumann.

- Conduce a un modelo de cómputo en el que:

  Hay una separación entre el tipo de informaciones que almacena la MC (instrucciones / datos)

  $\Longrightarrow$ repercusión sobre el diseño de estos lenguajes.

Computadores y Lenguajes Imperativos

- Los computadores modernos presentan una organización y funcionamiento basado en el modelo de von Neumann.

- Conduce a un modelo de cómputo en el que:

  Se introduce el concepto de estado de la computación (registros de la UCP + MC como un conjunto de palabras).

Computadores y Lenguajes Imperativos

- Los recursos expresivos de los lenguajes imperativos pueden verse como abstracciones de los componentes de la máquina de von Neumann o de sus operaciones elementales:

  variables $\Longleftrightarrow$ celdas de la MC / registros

Computadores y Lenguajes Imperativos

- Los recursos expresivos de los lenguajes imperativos pueden verse como abstracciones de los componentes de la máquina de von Neumann o de sus operaciones elementales:

  registro (o estructura) / arrray $\Longleftrightarrow$ conjunto contiguo de celdas de la MC

Computadores y Lenguajes Imperativos

- Los recursos expresivos de los lenguajes imperativos pueden verse como abstracciones de los componentes de la máquina de von Neumann o de sus operaciones elementales:

  nombres de variables $\Longleftrightarrow$ direcciones de celdas de la MC / registros

Computadores y Lenguajes Imperativos

- Los recursos expresivos de los lenguajes imperativos pueden verse como abstracciones de los componentes de la máquina de von Neumann o de sus operaciones elementales:

  instrucciones de control $\Longleftrightarrow$ intrucciones de salto condicional o incondicional del lenguaje máquina

Computadores y Lenguajes Imperativos

- Los recursos expresivos de los lenguajes imperativos pueden verse como abstracciones de los componentes de la máquina de von Neumann o de sus operaciones elementales:

  instrucción de asignación $\Longleftrightarrow$ intrucciones LOAD+STORE o MOVE del lenguaje máquina

Computadores y Lenguajes Imperativos

- La instrucción de asignación resulta ser representativa del cuello de botella de von Neumann y nos obliga a pensar en términos de trasiego de información entre celdas de memoria.

- la instrucción de asignación separa la programación en dos mundos: Expresiones / Instrucciones.

  direcciones = expresión.

Lenguajes Imperativos vs. Declarativos

Características de los Lenguajes Imperativos (Resumen)

PROGRAMA	Transcripción de un algoritmo
INSTRUCCIONES	Instrucciones máquina
MODELO DE CÓMPUTO	Máquina de estados
VARIABLES	Referencias a memoria

Lenguajes Imperativos vs. Declarativos

Programación Imperativa

- "*algoritmos + estructuras de datos = programas*" [Wirth]
- **Ejemplo:** Concatenación de dos listas.

Lenguajes Imperativos vs. Declarativos

Programación Imperativa

```
#include <stdio.h>
#include <stdlib.h>
typedef struct nodo {
        char dato;
        struct nodo *enlace;
} LISTA;
void mostrar(LISTA *ptr);
void insertar(LISTA **ptr, char elemento);
LISTA *crear_lista();
LISTA *concatenar(LISTA *ptr1, LISTA *ptr2);
```

Programación Imperativa

```
void main()
  {
        LISTA *l1, *l2, *lis = NULL;
        l1 = crear_lista();
        l2 = crear_lista();
        lis = concatenar(l1, l2);
        printf("\n La nueva lista enlazada es: ");
        mostrar(lis);
  }
```

Lenguajes Imperativos vs. Declarativos

Programación Imperativa

```
void mostrar(LISTA *ptr)
  {
        while(ptr != NULL)
          {
                printf("%c",ptr->dato);
                ptr = ptr->enlace;
          }
        printf("\n");
  }
```

Programación Imperativa

```
void insertar(LISTA **ptr, char elemento)
  {
        LISTA *p1, *p2;
        p1 = *ptr;
        if (p1 == NULL) {
            p1 = malloc(sizeof(LISTA));
            if (p1 != NULL) {
                p1->dato = elemento;  p1->enlace = NULL;  *ptr = p1;
            }
        } else { . . . }
  }
```

Lenguajes Imperativos vs. Declarativos

Programación Imperativa

```
LISTA *crear_lista()
  {
        LISTA *lis = NULL;
        char elemento;
        printf("\n Introduzca elementos: ");
        do {
            elemento = getchar();
            if(elemento != '\n') insertar(&lis, elemento);
        } while(elemento != '\n');
        return lis;
  }
```

Lenguajes Imperativos vs. Declarativos

Programación Imperativa

```
LISTA *concatenar(LISTA *ptr1, LISTA *ptr2)
  {
        LISTA *p1;
        p1 = ptr1;
        while(p1->enlace != NULL) p1 = p1->enlace;
        p1->enlace = ptr2;
        return ptr1;
  }
```

Programación Imperativa

Este programa ilustra las características de los lenguajes imperativos:

- Es una secuencia de instrucciones que son ordenes a la máquina que operan sobre un estado no explícitamente declarado.

Programación Imperativa

Este programa ilustra las características de los lenguajes imperativos:

- Para entender el programa debemos ejecutarlo mentalmente (cómo cambian los contenidos de las variables y otras estructuras en la MC).

Programación Imperativa

Este programa ilustra las características de los lenguajes imperativos:

- Es necesario gestionar explícitamen la MC (llamada al sistema `malloc()` + variables de tipo puntero).

Programación Imperativa

Este programa ilustra las características de los lenguajes imperativos:

- Rigidez (sólo concatena listas de caracteres).

Programación Imperativa

Este programa ilustra las características de los lenguajes imperativos:

- La lógica y el control están mezclados, lo que dificulta la verificación formal del programa.

Lenguajes Declarativos

Uso de una cierta lógica como lenguaje de programación:

- Un programa es un conjunto de fórmulas lógicas que resultan ser la especificación del problema que se pretende resolver, y

- la computación se entiende como una forma de inferencia o deducción en dicha lógica.

Lenguajes Declarativos

Los principales requisitos que debe cumplir la lógica empleada son:

- disponer de un lenguaje que sea suficientemente expresivo;
- disponer de una semántica operacional (un mecanismo de cómputo que permita ejecutar los programas);

Lenguajes Declarativos

Los principales requisitos que debe cumplir la lógica empleada son:

- disponer de una semántica declarativa que permita dar un significado a los programas de forma independiente a su posible ejecución;
- resultados de corrección y completitud.

Características de los Lenguajes Declarativos (Resumen)

PROGRAMA	Especificación de un problema
INSTRUCCIONES	fórmulas lógicas
MODELO DE CÓMPUTO	Inferencias lógicas
VARIABLES	Variables lógicas

Lenguajes Imperativos vs. Declarativos

Programación Declarativa

- El programador especifica *qué* debe computarse más bien que *cómo* deben realizarse los cómputos.
- "*programa = lógica + control*" [Kowalski]

Programación Declarativa

- El componente lógico determina el significado del programa mientras que el componente de control solamente afecta a su eficiencia.
- la tarea de programar consiste en centrar la atención en la *lógica* dejando de lado el *control* al sistema.

Ventajas de la Programación Declarativa

- Estilo de programación de muy alto nivel y control automático.
- Mayor poder expresivo.
- Mayor productividad, programas más pequeños y fáciles de mantener.

Programación Lógica

- Se basa en (fragmentos de) la lógica de predicados: lógica de cláusulas de Horn (HCL).
- Define relaciones mediante cláusulas

$$\mathcal{A} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_2 \quad \text{(implicaciones)}$$

Programación Lógica: Ejemplo

- Concatenación de dos listas.

$$
\begin{array}{lcl}
app([\,], X, X) & \leftarrow & \\
app([X|X_s], Y, [X|Z_s]) & \leftarrow & app(X_s, Y, Z_s)
\end{array}
$$

Programación Lógica: Ejemplo

- Concatenación de dos listas.

$$
\begin{array}{lcl}
app([\,], X, X) & \leftarrow & \\
app([X|X_s], Y, [X|Z_s]) & \leftarrow & app(X_s, Y, Z_s)
\end{array}
$$

- **Lectura declarativa**:

  La concatenación de la lista vacía $[\,]$ y otra lista X es la propia lista X.

Programación Lógica: Ejemplo

- Concatenación de dos listas.

$$
\begin{array}{lcl}
app([\,], X, X) & \leftarrow & \\
app([X|X_s], Y, [X|Z_s]) & \leftarrow & app(X_s, Y, Z_s)
\end{array}
$$

- **Lectura declarativa**:

  La concatenación de $[X|X_s]$ e Y es la lista que resulta de añadir el primer elemento X de $[X|X_s]$ a Z_s, si Z_s es la concatenación de X_s e Y.

Programación Lógica: Ejemplo

- Concatenación de dos listas.

$$
\begin{array}{lcl}
app([\,], X, X) & \leftarrow & \\
app([X|X_s], Y, [X|Z_s]) & \leftarrow & app(X_s, Y, Z_s)
\end{array}
$$

- **Lectura operacional**:

  Para concatenar dos listas $[X|X_s]$ e Y primero es preciso resolver el problema de concatenar el resto X_s de la primera lista, a la segunda Y.

Programación Lógica: Ejemplo

- Concatenación de dos listas.

$$
\begin{array}{lcl}
app([\,], X, X) & \leftarrow & \\
app([X|X_s], Y, [X|Z_s]) & \leftarrow & app(X_s, Y, Z_s)
\end{array}
$$

- **Lectura operacional**:

  La concatenación de la lista vacía $[\,]$ y otra lista X es un problema ya resuelto.

Programación Lógica: Ejemplo

Principales características de este programa:

- Gestión automática de la memoria
- Mecanismo de cómputo que permite una búsqueda indeterminista (*built-in search*) de soluciones:

Programación Lógica: Ejemplo

Principales características de este programa:

- Gestión automática de la memoria

- Mecanismo de cómputo que permite una búsqueda indeterminista (*built-in search*) de soluciones:

  Responde a diferentes objetivos (sin necesidad de efectuar ningún cambio en el programa),

Programación Lógica: Ejemplo

Principales características de este programa:

- Gestión automática de la memoria
- Mecanismo de cómputo que permite una búsqueda indeterminista (*built-in search*) de soluciones:

  Computa con datos parcialmente definidos,

Programación Lógica: Ejemplo

Principales características de este programa:

- Gestión automática de la memoria
- Mecanismo de cómputo que permite una búsqueda indeterminista (*built-in search*) de soluciones:

  Relación de entrada/salida no está fijada de antemano.

Programación Lógica: Semántica operacional

- Resolución SLD (regla de computación φ)

$$\frac{(\mathcal{G} \equiv \leftarrow \mathcal{Q}_1 \wedge \mathcal{A}' \wedge \mathcal{Q}_2),\ \ \varphi(\mathcal{G}) = \mathcal{A}',\ \ \mathcal{C} \equiv (\mathcal{A} \leftarrow \mathcal{Q}) \ll \Pi,\ \ \sigma = mgu(\mathcal{A},\mathcal{A}')}{\mathcal{G} \stackrel{\sigma}{\Longrightarrow}_{SLD} \leftarrow \sigma(\mathcal{Q}_1 \wedge \mathcal{Q} \wedge \mathcal{Q}_2)}$$

- Procedimiento de prueba por refutación SLD:

  regla de computación $+$ regla de ordenación $+$ regla búsqueda.

Programación Funcional

- Se basa en el concepto de función (matemática) y su definición mediante ecuaciones, que constituyen el programa.
- Computación = reducción determinista de expresiones (funcionales) para obtener un valor.
- Aproximaciones: ecuacional; algebraica; λ-cálculo.

Programación Funcional: Ejemplo

- Concatenación de dos listas.

$$data\ [t]\ =\ [\,]\ |\ [t : [t]]$$
$$app\ ::\ [t] \rightarrow [t] \rightarrow [t]$$

$$app\ [\,]\ x\ =\ x$$
$$app\ (x : x_s)\ y\ =\ x : (app\ x_s\ y)$$

Programación Funcional: Ejemplo

Principales características de este programa:

- Necesidad de fijar el perfil de la función (dominio $+$ rango) $\Longrightarrow$ idea de *tipo de datos*

  [Lenguajes *fuertemente basados en tipos*]

- Se ha declarado la estructura de datos lista:

Programación Funcional: Ejemplo

Principales características de este programa:

- Necesidad de fijar el perfil de la función (dominio + rango) $\Longrightarrow$ idea de *tipo de datos*

  [Lenguajes *fuertemente basados en tipos*]

- Se ha declarado la estructura de datos lista:

  "[ ]" y ":" son los *constructores del tipo*

Programación Funcional: Ejemplo

Principales características de este programa:

- Necesidad de fijar el perfil de la función (dominio $+$ rango) $\Longrightarrow$ idea de *tipo de datos*

  [Lenguajes *fuertemente basados en tipos*]

- Se ha declarado la estructura de datos lista:

  t es una *variable de tipo* $\Longrightarrow$ Polimorfismo

Programación Funcional:
Otras características de las funciones (matemáticas)

- Transparencia referencial: la salida viene determinado exclusivamente por la entrada.

$\Longrightarrow$

  - No efectos laterales;
  - Permite el razonamiento ecuacional (substitución de iguales por iguales);
  - Cómputos deterministas.

Programación Funcional:
Otras características de las funciones (matemáticas)

- Composición de funciones:
  - Permite la construcción de programas mediante el empleo de funciones primitivas o previamente definidas por el usuario;
  - Refuerza la modularidad de los programas.

Programación Funcional: Orden superior

- Empleo de las funciones como “ciudadados de primera clase”.

- **Ejemplo**: Funciones de Curry

$$(+) \;::\; Integer \rightarrow Integer \rightarrow Integer$$
$$> \; 2 \; + \; 3$$
$$5$$

(El resultado es un *Integer*)

Programación Funcional: Orden superior

- Empleo de las funciones como “ciudadados de primera clase”.

- **Ejemplo**: Funciones de Curry

$$(+) \;::\; Integer \rightarrow Integer \rightarrow Integer$$
$$> \; (2+)$$
$$***Expression \; : (2+)$$ (El resultado es una funció
$$***Of\ type \quad : Integer \rightarrow Integer$$

Programación Funcional: Orden superior

- Empleo de las funciones como “ciudadados de primera clase”.

- **Ejemplo**: Las funciones se pueden pasar como argumentos

$$
\begin{aligned}
&map \;::\; (t_1 \rightarrow t_2) \rightarrow [t_1] \rightarrow [t_2] \\
&map\ f\ [\,] \;=\; [\,] \\
&map\ f\ (x : x_s) \;=\; (f\ x) : (map\ f\ x_s)
\end{aligned}
$$

Programación Funcional: Semántica operacional

- β–reducción (aproximación clásica: λ–cálculo)

$$(\lambda x.\lambda y.+\ x\ y)\ 3\ 2 \rightarrow_\beta (\lambda y.+\ 3\ y)\ 2 \rightarrow_\beta +\ 3\ 2 \rightarrow_\delta 5$$

- Estrategias de evaluación:
  - función de selección para redexes: *leftmost* vs. *rightmost*; *innermost* vs. *outermost*.
  - orden aplicativo (voraz, *call by value*): *leftmost innermost*.
  - orden normal (perezosa, *call by name*): *leftmost outermost*.

Programación Funcional: Semántica operacional

- Reescritura (*aproximación algebraica*)

$$\frac{(l = r) \ll \Pi, \quad (\exists p)(\exists \sigma).\sigma(l) = t|_p}{t \quad \rightarrow_\sigma \quad t[\sigma(r)]_p}$$

- Estrategias de evaluación:
  - función de selección para redexes: *leftmost* vs. *rightmost*; *innermost* vs. *outermost*.
  - *leftmost innermost*; *leftmost outermost*; *parallel outermost*; *parallel outermost*; ...

Programación Lógica vs. Funcional: Diferencias

Programación Lógica	Programación Funcional
PROGRAMA: Conjunto de cláusulas que defi - nen relaciones	PROGRAMA: Conjunto de ecuaciones que de- fi nen funciones.
SEMÁNTICA OPERACIONAL: Resolución SLD (unifi cación)	SEMÁNTICA OPERACIONAL: Reducción (ajuste de patrones)
SEMÁNTICA DECLARATIVA: Teoría de modelos (Modelo mí- nimo)	SEMÁNTICA DECLARATIVA: Algebraica (Algebra inicial) / De- notacional

Programación Lógica vs. Funcional: Diferencias

Primer orden	Orden superior
Indeterminismo; Variables lógicas; Datos parcialmente especifi cados; E/S adireccional	Determinismo; Sin variables lógicas; Datos completamente especifi cados; E/S direccional
Sin tipos	Tipos y polimorfi smo
No estructuras infi nitas	Estructuras infi nitas
No evaluación perezosa	Evaluación perezosa

Ventajas de los Lenguajes Funcionales respecto a los Lógicos

- Funciones de orden superior
  - mejor abstracción
  - mejor modularización
- Evaluación eficiente
  - reducción determinista
  - evaluación perezosa

Ventajas de los Lenguajes Lógicos respecto a los Funcionales

- Búsqueda de soluciones indeterminista.
- Mayor potencia expresiva (e.g. variables extra; información parcial; inversibilidad).
- Aplicaciones en investigación operativa, BD e IA.
- Programación con restricciones (*constraint solving*).

Motivación: Objetivo

- Cada estilo tiene algo que ofrecer al otro:
  - los lenguajes lógicos poseen la lógica de la unificación y de las variables lógicas.
  - los lenguajes funcionales poseen la lógica de la igualdad y de las funciones.

- **Objetivo:** combinar las mejores características de los lenguajes lógicos y funcionales.

Motivación: Ventajas (esperadas) de los Lenguajes Lógico-Funcionales

- Potencia expresiva de un lenguaje lógico
- Eficiencia de un lenguaje funcional
- Aplicaciones de ambos campos

Motivación: Problemas con Prolog

- Evitar ciertas características impuras de Prolog:
  - Indeterminismo incontrolado: ineficiencia y ramas infinitas.
  - Control *ad hoc*: corte (rojo — no declarativo).
  - E/S no declarativa.

Motivación: Problemas con Prolog

- La integración de funciones produce:
  - Indeterminismo incontrolado: ineficiencia y ramas infinitas.
  - Control *ad hoc*: corte (rojo — no declarativo).
  - E/S no declarativa.

Motivación: Problemas con Prolog

- La integración de funciones produce:
  - Mejor eficiencia: determinismo.
  - Control *ad hoc*: corte (rojo — no declarativo).
  - E/S no declarativa.

Motivación: Problemas con Prolog

- La integración de funciones produce:
  - Mejor eficiencia: determinismo.
  - Control automático: corte dinámico (declarativo).
  - E/S no declarativa.

Motivación: Problemas con Prolog

- La integración de funciones produce:
  - Mejor eficiencia: determinismo.
  - Control automático: corte dinámico (declarativo).
  - E/S declarativa.

Aproximaciones a la Integración: funcional $+ \Rightarrow$ lógico

- Sintaxis funcional: ecuaciones (condicionales?)

$$s = t \leftarrow s_1 = t_1 \wedge \ldots \wedge s_n = t_n$$

  - Los predicados son funciones booleanas.
  - Las fórmulas atómicas son ecuaciones $\mathcal{A} = \texttt{true}$.

Aproximaciones a la Integración: funcional $+ \Rightarrow$ lógico

- **Objetivo**: Añadir variables lógicas y no determinismo a un lenguaje funcional

- Semánticas operacionales:
  - Narrowing = reducción + unificación.

  - Residuación = reducción + espera hasta que las funciones esten suficientemente instanciadas.

Aproximaciones a la Integración: lógico $+ \Rightarrow$ funcional

- Sintaxis lógica con características funcionales: Cláusulas de Horn con igualdad
  - $\mathcal{A} \leftarrow s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n$
  - $s = t \leftarrow s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n$

Aproximaciones a la Integración: lógico $+ \Rightarrow$ funcional

- **Objetivo**: Añadir igualdad a un lenguaje lógico, Funciones definidas, Tipos de Datos.

- Semánticas operacionales:
  - Flattening = desanidar funciones + SLD.
  - Resolución SLDE = SLD + E-unificación.
  - Resolución SLD + nuevas reglas para las funciones (Residuación o narrowing).

Lenguajes

Mecanismo Operacional	Lenguajes
Narrowing (reescritura con unifi cación)	ALF, BABEL, SLOG
Flattening (Traducción a Prolog)	K-LEAF, EUROPA
SLDE-resolución (resolución con unifi cación semántica)	LPG
Residuación (Resolución con congelación de ecuaciones y reescritura)	ESCHER, Le Fun, Oz
Weakly needed narrowing	$\mathcal{TOY}$
(Weakly) Needed narrowing + Residuación	Curry

Bibliografía

- Hanus M., 1994. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628.

- Moreno–Navarro J.J., 1995. Programación Lógica y Programación funcional. En *Estudios sobre la programación Lógica y sus aplicaciones*, Universidad de Santiago de Compostela, pp. 35–77.