Programación Declarativa

Tema 2

Francisco José Correa Zabala
fcorrea@eafit.edu.co

Departamento de informática y Sistemas.
Universidad Eafit.

Medellín, Colombia. febrero de 2013

Objetivos

Reconocer argumentos fundamentales de la programación declarativa.

Presentar los argumentos que sustentan y definen la programación declarativa.

Establecer diferencias y similitudes entre los paradigmas presentados.

Extender los aprendizajes para ganar fortalezas para el desarrollo de software en programación orientada a objetos.

Agenda

1. Tareas
2. Nociones básicas del paradigma declarativo.
3. Comparación con el paradigma imperativo.
4. Sistema Formal.
5. Enfoques de la semántica.

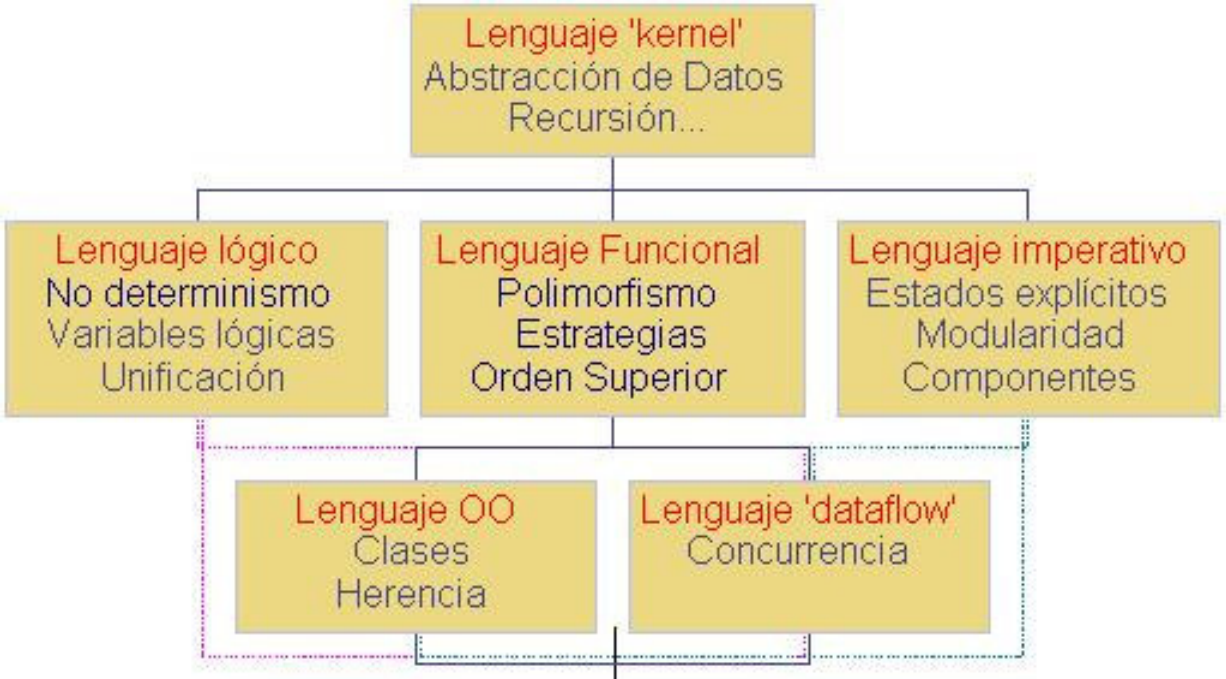
Discusión de las tareas

1. Investigar aspectos básicos o fundamentales de un paradigma que sean diferentes o complementarios a los presentados del curso. Una hoja. Se califica con exposición.

Uso de los paradigmas: tres aproximaciones

- **Programación como un oficio.**
  - Se estudia en un paradigma único y con un único lenguaje.
  - Es contraproducente: por ejemplo, aprender programación concurrente en Java –monitores, memoria compartida. . . – puede llevar a la conclusión errónea de que la concurrencia es siempre tan complicada y costosa como en Java.
- **Programación como una rama de las matemáticas.** O bien se estudia en un lenguaje 'ideal', restringido (Dijkstra) o resulta demasiado teórico, alejado de la práctica.
- **Programación en términos de conceptos.** Estudiar un conjunto de conceptos semánticos y estructuras de implementación en términos de las cuales se describen de forma natural diferentes lenguajes y sus implementaciones.

Importancia de los conceptos



✓ Estos conceptos (y muchos otros) ponen en evidencia lo que es común a los distintos lenguajes.

✓ Además, se pueden describir sin hablar de ningún lenguaje en particular.

✓ Por ello, en adelante, ignoraremos las diferencias ligadas a la sintaxis concreta.

Porque estudiar paradigmas (1/2)

★ Entender el diseño e implementación de los lenguajes.

- ✓ **Sintaxis.** Define las reglas de construcción de los programas.
- ✓ **Semántica.** Establece el significado de los programas.
- ✓ **Implementación.** Presenta cómo se ejecutan.
- ✓ **Pragmática.** Establece los aspectos prácticos de su uso.

★ Elegir el lenguaje más apropiado para una aplicación dada.

- ✓ Imperativo: Pascal/Fortran — **Aplicaciones científicas**
- ✓ OO: C++/Java — **Programación de sistemas**
- ✓ Funcional: Haskell/LISP/ML — **Aplicaciones simbólicas**
- ✓ Lógico: Prolog/Mercury . . .

Porque estudiar paradigmas (2/2)

- ★ Facilitar el aprendizaje de un nuevo lenguaje.
- ★ Simular *características en* lenguajes que carecen de ellas.
- ★ Mejorar la habilidad de desarrollar algoritmos eficientes.
- ★ Mejorar el uso del lenguaje de programación disponible.
- ★ Aumentar el vocabulario del programador.
- ★ Facilitar el diseño de un nuevo lenguaje.
- ★ Un lenguaje es a la vez una ayuda y un limitante.
- ★ Lo que se viene es la integración de paradigmas

Ahora, se trata de no aprender un lenguaje concreto sino los principios generales de los mismos. *. . . Pero en la práctica puede ser otra cosa*

Paradigmas de programación

Imperativa	Declarativa
Programa:	
Transcripción de un algoritmo	Especificación del problema mediante algún formalismo
Instrucciones:	
Ordenes a la máquina	Fórmulas en alguna lógica
Modelo de Computación:	
Máquina de estados (Variables)	Maquina de inferencias
Variables del programa:	
Referencias a memoria	Lógicas

Programación Declarativa

Programa Imperativo

Declare X, Y Reales

Lea X

Lea Y

$Z := X * Y$

Muestre Z

Programa Declarativo

$1 * X = X.$

$(X + 1) * Y = (X * Y) + Y$

- ✓**Son ordenes a la máquina**
- ✓**Para entenderlo: seguimiento**
- ✓**Lógica y control mezclados.**
- ✓**Gestión de memoria.**
- ✓**Secuencia de Cómputos.**

- ♣ **Especifica el *qué*.**
- ♣ **Lógica: Significado.**
- ♣ **Un programa: teoría formal.**
- ♣ **Computación: Inferencia**
- ♣ **Datos parcialmente definidos.**
- ♣ **Relación I/O.**
- ♣ **Fácil de Paralelizar**

Ejemplo . . .

Ejemplo: Longitud de una lista

Programa imperativo

```
function length (L: list): nat
B:bool;
faux:list;
B:= is_empty(L);
if B then
    return  0;
else
    aux:=tail(L);
    return  1+length(aux)
end if
endfunction;
```

Paradigma imperativo

- Establece el *cómo* proceder.
- Lenguajes orientados a enunciados (instrucciones).
- El concepto básico es el estado de la máquina, el cual se define por los valores de las variables involucradas y que se almacenan en la memoria.
- Las instrucciones suelen ser secuenciales y el programa consiste en construir los estados de la máquina necesarios para llegar a la solución.
- Este modelo procede del hardware de la máquina convencional.
- Eficiente, difícil de modificar y verificar, con efectos laterales.

Ejemplo: Longitud de una lista

Programa funcional

$$\texttt{length}(\texttt{nil}) = 0$$
$$\texttt{length}(\texttt{E} : \texttt{L}) = \texttt{length}(\texttt{L}) + 1$$

Programa lógico

$$\texttt{length}(\texttt{nil}, 0)$$
$$\texttt{length}(\texttt{E.L}, \texttt{N}) \Leftarrow \texttt{length}(\texttt{L}, \texttt{M}) \wedge \texttt{N} = \texttt{M} + 1$$

Paradigma Declarativo

- Establece lo que se desea.
- Fácil de verificar y modificar, conciso y claro.
- Paradigma funcional:
  - Definición recursiva de funciones.
  - Basado en las matemáticas (λ-cálculo).
  - Orden-superior.
- Paradigma Lógico:
  - Basado en reglas: “Si C entonces A”.
  - Simple y potente (fórmulas lógicas).
  - Computación: inferencias lógicas.

Paradigma Orientado a Objetos

- Objeto: estado + operaciones.
- Concepto de clase, instancia, subclases y herencia.
- Elementos fundamentales: abstracción, encapsulamiento, modularidad y jerarquía.

Programación Declarativa

KOWALSKI Y COLMENAURE (1972) introducen la idea de que la **lógica** puede utilizarse como **lenguaje de programación**.

Idea 1 L. de prog. lógica $\equiv$ L. de especificación $\equiv$ L. de programación

Idea 2 Programa $=$ lógica $+$ Control

Lógica Relaciona el "QUE" (Especificación)

Control Establece el "COMO": secuencia de pasos y gestión de memoria. (Implementación)

Idea 3 Prog. declarativa $\equiv$ Demostr. teoremas $+$ extracción de resp.

prog. lógica ≡ especificación ≡ programa

Especificación:

$$\texttt{fib(0)} = \texttt{suc(0)}$$
$$\texttt{fib(suc(0))} = \texttt{suc(0)}$$
$$\texttt{fib(suc(suc(X)))} = \texttt{fib(suc(X))} + \texttt{fib(X)}$$

Programa:

$$\texttt{fib(X)} = \texttt{fib\_aux(suc(0), suc(0), X)}$$
$$\texttt{fib\_aux(A, B, 0)} = \texttt{A}$$
$$\texttt{fib\_aux(A, B, suc(X))} = \texttt{fib\_aux(B, A + B, X)}$$

```
fib(suc(0))   →  fib_aux(suc(0),suc(0),suc(0))
              →  fib_aux(suc(0),suc(suc((0)),0))
              →  suc(0)

fib(suc(suc(0)))  →  fib_aux(suc(0),suc(0),suc(suc(0)))
                  →  fib_aux(suc(0),suc(suc((0)),suc(0)))
                  →  fib_aux(suc(suc(0)),suc(suc(suc(0))),0)
                  →  suc(suc(0))
```

Programa = lógica + control

La programación declarativa provee un nivel más alto de programación:

- √ Semántica mas sencilla.
- √ Control automático.
- √ Más fácil de paralelizar.
- √ Mayor potencia expresiva.
- √ Menor tamaño del código.
- √ Mayor productividad.
- √ Mejor mantenimiento.

Lenguajes imperativos

- √ Eficiencia.
- √ Modularidad.
- √ Herramientas de compilación separadas.
- √ Herramientas de depuración de errores.
- √ Han sido exitosos en algunas áreas.

Prog. declarativa $\equiv$ Demostr. teoremas $+$ extracción de resp.

$\texttt{sum}(\texttt{0}, \texttt{X}, \texttt{X})$.

$\texttt{sun}(\texttt{X}, \texttt{Y}, \texttt{Z}) \Leftarrow \texttt{sum}(\texttt{X} - \texttt{1}, \texttt{Y}, \texttt{Z} - \texttt{1})$.

Demostración de teoremas:

?- $\texttt{sum}(\texttt{3}, \texttt{4}, \texttt{7})$.

yes.

Extracción de respuestas:

?- $\texttt{sum}(\texttt{3}, \texttt{2}, \texttt{W})$.

$\texttt{W} = \texttt{5}$.

?- $\texttt{sum}(\texttt{X}, \texttt{Y}, \texttt{4})$.

$\texttt{X} = \texttt{0}, \texttt{Y} = \texttt{4}$;

$\texttt{X} = \texttt{1}, \texttt{Y} = \texttt{3}$;

$\texttt{X} = \texttt{2}, \texttt{Y} = \texttt{2}$;

$\texttt{X} = \texttt{3}, \texttt{Y} = \texttt{1}$;

no.

Escribir Programas

La característica fundamental de la programación declarativa es el uso de la lógica como lenguaje de programación.

Un programa es una teoría formal en una cierta lógica. Una computación es una forma de inferencia o deducción en dicha lógica.

Los requisitos que debe cumplir la lógica empleada son:

- $\surd$ Disponer de un lenguaje expresivo.
- $\surd$ Un mecanismo de cómputo que permita ejecutar programas.
- $\surd$ Significado de los programas de forma independiente a la ejecución.
- $\surd$ Resultados de corrección y completitud.

Programación Declarativa

LÓGICA

L. ecuacional

L. clausal

L. heterogénea

L. de géneros ordenados

L. modal

L. temporal

. . .

ESTILO

Funcional

Relacional

Tipos

Herencia

S.B.C.

Concurrencia

Sistema Formal

1. Lenguaje formal : permite expresar enunciados o fórmulas sintácticamente correctas.

2. Semántica : proporciona significado o interpretación a dichas formulas: Noción de verdad, significado de los sentencias y expresiones que se originan en el lenguaje.

3. Método de cálculo : forma de generar nuevas expresiones en el lenguaje: reglas de inferencia, demostraciones, . . .

Sistema formal

"El estudio de un sistema abarca tanto la componente sintáctica como la semántica, así como la relación entre ambas". Carnap

1. Sintaxis:

   a) Lenguaje formal: conjunto de enunciados sintácticamente correctos: alfabeto, reglas y definiciones.

   b) Cálculo deductivo: Reglas para obtener fbf nuevas: axiomas, reglas de inferencia y concepto de deducción y demostración.

2. Semántica: significado e interpretación. Noción de verdad

Propiedades: Consistencia, Corrección, Completitud y decidibilidad.

Sistema formal: Propiedades

1. **Completitud:** Si todo fórmula que “se sigue lógicamente” de un conjunto de premisas (es válida según la noción de verdad), es demostrable en el sistema a partir de dicho conjunto de premisas.

2. **Corrección:** Si toda fórmula demostrable a partir de un conjunto de premisas, “se sigue lógicamente” de ellas.

3. **Decidibilidad:** Si existe un procedimiento finito para comprobar si una formula es o no válida.

4. **Consistencia:** Cuando de él no se desprenden contradicciones.

Enfoque de la Semántica

- **Operacional:**
  - Interprete para el lenguaje
  - Primero define una máquina abstracta y el significado de cada construcción: son acciones en la máquina.
  - Lo hace ejecutable
- **Axiomático:** Significado en términos de axiomas, verifica el estado de la máquina tras su ejecución

Enfoques de la Semántica

- **Declarativo**: El significado es en términos de una estructura matemática.
  - **Teoría de modelos**. Consecuencias lógicas. Programa (interpretación y modelo, relaciones)
  - **Algebraica**. Teoría de categorías. Programa (álgebra y modelo, ecuaciones)
  - **Punto fijo**. Funciones recursivas.
  - **Denotacional**. Teoría de dominios de Scott.

Enfoques de la Semántica

Teoría de modelos. El programa es un conjunto de fórmulas que definen relaciones:

$\texttt{sum}(\texttt{0}, \texttt{X}, \texttt{X}).$
$\texttt{sun}(\texttt{X}, \texttt{Y}, \texttt{Z}) \Leftarrow \texttt{sum}(\texttt{X} - \texttt{1}, \texttt{Y}, \texttt{Z} - \texttt{1})$

El programa se interpreta con las nociones de **interpretación** y de **modelo** de un conjunto de formulas.

El significado de un programa es el conjunto de sus consecuencias lógicas.

$\texttt{sum}(\texttt{0}, \texttt{0}, \texttt{0}).$
$\texttt{sum}(\texttt{0}, \texttt{30}, \texttt{30}).$
$\texttt{sum}(\texttt{2}, \texttt{3}, \texttt{5}).$
$\vdots$

Enfoques de la Semántica

Algebraica. Por ejemplo los lenguajes ecuacionales de primer orden.

Un programa es un conjunto de ecuaciones que definen funciones.

$\mathtt{sum}(\mathtt{0}, \mathtt{X}) = \mathtt{X}.$
$\mathtt{sun}(\mathtt{X} + \mathtt{1}, \mathtt{Y}) = \mathtt{sum}(\mathtt{sum}(\mathtt{X}, \mathtt{Y}), \mathtt{1})$

Se interpreta haciendo uso de las nociones **álgebra** y **modelo** de un conjunto de ecuaciones.

El significado de un programa es el conjunto de clases de equivalencias inducidas por las ecuaciones.

$$\begin{aligned} [0] &= \{0, 0+0, 0+0+0, \ldots\} \\ [1] &= \{1, 0+1, 0+0+1, \ldots, 1+0, 1+0+0, \ldots\} \end{aligned}$$

Enfoques de la Semántica

Punto fijo. Comúnmente se utiliza como enlace para demostrar la equivalencia entre las diferentes caracterizaciones semánticas de un mismo programa.

Se define como una función sobre el programa.

El significado del programa es el menor punto fijo de dicha función.

En general, dicho operador se presenta como una función T, a la que aplicamos la siguiente definición.

Sea H un conjunto y $T : 2^H \rightarrow 2^H$ un operador monótono.

El menor punto fijo, *lfp*, de T es el menor subconjunto I de H tal que $T(I) \subseteq I$. Así, $T(I) \subseteq I \Rightarrow lfp(T) \subseteq I$.

Enfoques de la Semántica

Denotacional. Teoría de dominios de Scott.

Se requiere definir:

✓Las categorías o dominios sintácticos

✓Las categorías o dominios semánticos

✓funciones de evaluación semántica

✓Las ecuaciones semánticas

Es muy rica y permite dar cuenta de:

✓Computaciones que no terminan

✓orden superior

Ventajas de los Lenguajes Formales

La especificación formal es ejecutable.

Se puede estudiar matemáticamente sus propiedades: consistencia, completitud, equivalencia entre semánticas, . . .

Se puede transformar mecánicamente y estudiar la corrección de dicha transformación.

Dada una especificación formal y la semántica del lenguaje se puede estudiar la corrección y completitud del programa respecto de la especificación.