Paradigmas de lenguajes de programación

Departamento de Computación, FCEyN, UBA

29 de enero de 2018

Cátedra y modalidad

- Modalidad: Clases teóricas, prácticas y laboratorio.
- Horarios: Lunes, Miércoles y Jueves de 17 a 22.

Teoría

- Hernán Melgratti.

Práctica

- Christian Roldán (JTP), Iván Arcuschin, Marín Jedwabny

Recursos

Bibliografía

- Textos: no hay un texto principal, se utilizan varios, referencias en página web
- Publicaciones relacionadas
- Diapositivas de teóricas y prácticas

Página web

- Información al día del curso, consultar periódicamente y leer al menos una vez todas las secciones

Mailing list

- ¡Hacer todas las preguntas y consultas que quieran!

Paradigma

Marco filosófico y teórico de una escuela científica o disciplina en la que se formulan teorías, leyes y generalizaciones y se llevan a cabo experimentos que les dan sustento

Fuente: Merriam-Webster[1]

[1] *A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated*

Lenguajes de Programación

- lenguaje usado para comunicar instrucciones a una computadora
- instrucciones describen cómputos que llevará a cabo la computadora
- computacionalmente completo si puede expresar todas las funciones computables

Paradigmas de Lenguaje de Programación

Marco filosófico y teórico en el que se formulan soluciones a problemas de naturaleza algorítmica

- Lo entendemos como
  - un estilo de programación
  - en el que se escriben soluciones a problemas en términos de algoritmos
- Ingrediente básico es el modelo de cómputo
  - la visión que tiene el usuario de cómo se ejecutan sus programas

Objetivos del curso

Conocer los *pilares conceptuales* sobre los cuales se erigen los lenguajes de programación de modo de poder

- Comparar lenguajes
- Seleccionar el más adecuado para una determinada tarea
- Usar las herramientas adecuadamente
- Prepararse para lenguajes/paradigmas futuros

Nos centraremos en

- los paradigmas:
  - *imperativo*
  - funcional
  - orientado a objetos
  - lógico
- Hay otros: concurrente, eventos, continuaciones, probabilístico, quantum.
- ¡La distinción a veces no es clara!

Enfoque del curso

1. Conceptos
   - Introducción informal de conceptos a través de ejemplos.
2. Fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Aspectos de un Lenguaje

- Sintaxis
- Semántica

Aspectos de un Lenguaje

- Sintaxis
  - descripción del conjunto de secuencias de símbolos considerados como programas válidos
  - teoría de lenguajes formales bien desarrollada, comenzando a mediados de 1950 con Noam Chomsky como pionero
  - notación BNF (y EBNF) ampliamente utilizada; desarrollada por John Backus para Algol 58, modificada por Peter Naur para Algol 60
  - amplio abanico de herramientas para generar analizadores léxicos y parsers a partir de notación formal como BNF
- Semántica

Aspectos de un Lenguaje

- Sintaxis
- Semántica
  - descripción del significado de instrucciones y expresiones
  - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas)
  - ¿Por qué semántica formal?
  - Enfoques: axiomático, *operacional*, denotacional

Enfoque del curso

1. Conceptos
   - Introducción informal de conceptos a través de ejemplos.
2. Fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Aspectos de un Lenguaje

- Sintaxis
- Semántica
- Sistema de Tipos
  - propósito: prevenir errores en tiempo de ejecución
  - en general, requiere anotaciones de tipo en el código fuente
  - ejemplos: evitar sumar booleanos, aplicar función a número incorrecto de argumentos.
  - análisis de tipos en tiempo de compilación: chequeo de tipos estático
  - análisis de tipos en tiempo de ejecución: chequeo de tipos dinámico

Algunos temas cubiertos

- Lambda cálculo
  - Sintaxis y ejemplos de programación
  - Sistemas de tipos e inferencia
  - Semántica operacional
- Resolución
  - En lógica proposicional
  - En lógica de primer orden
  - SLD
- Programación orientada a objetos
  - Sistemas de tipos, herencia como subtipado

Imperativo

¿Qué hace este programa?

```
int main (){
  int r,s,n;
  n = 1;
  r = 1;
  s = 0;
  while (n<11) {
    r = r * n;
    s = s + r;
    n = n + 1;
  };
  printf ("%d",s);
}
```

Ayuda: imprime 4037913

Estado global + asignación + control de flujo

Evaluación

- Computación expresada a través de modificación reiterada de estado global (o memoria)
- Variables como abstracción de celdas de memoria
- Resultados intermedios se almacenan en la memoria
- Repetición basada en iteración

Imperativo

¿Qué hace este programa?

```
int main (){
  int r,s,n;
  n = 1;
  r = 1;
  s = 0;
  while (n<11) {
    r = r * n;
    s = s + r;
    n = n + 1;
  };
  printf ("%d",s);
}
```

Ayuda: imprime 4037913

Evaluación

- Eficiente
  - Modelo ejecución - Arquit. $= 0$
- Bajo nivel de abstracción
  - Especif - Implement $= \infty$
- Matemática/lógica de programas compleja
  - Operacional vs denotacional

Funcional

```
sum (map fact [1..n])

fact 0 = 1
fact n = n * fact (n-1)
```

Evaluación

- Computación expresada a través de la aplicación y composición de funciones
- No hay un estado global
- Resultados intermedios (salida de las funciones) son pasados directamente a otras funciones como argumentos
- Repetición basada en recursión
- Expresiones tipadas

Funcional

```
sum (map fact [1..n])

fact 0 = 1
fact n = n * fact (n-1)
```

Evaluación

- Alto nivel de abstracción
  - Especificación ejecutable: *Sumar resultado de aplicar factorial a la lista de números de 1 a n*
- Declarativo
- Matemática de programas elegante
- Razonamiento algebraico posible
  - $e + e \simeq 2 * e$
  - $a == b \simeq b == a$
- Ejecución lenta (?)

Lógico

```
fact(0,1).
fact(N,Res) :- M is N-1, fact(M,Aux), Res is N*Aux.
lFact([],N,N).
lFact([H|T],M,N) :- fact(M,H), M1 is M+1, lFact(T,M1,N).
sumaFact(X,N) :- lFact(L,1,N), sumList(L,X).
```

Evaluación

- Programas son predicados
- Computación expresada a través de "proof search"
- No hay estado global
- Resultados intermedios son pasados a través de unificación
- Repetición basada en recursión

Lógico

```
fact(0,1).
fact(N,Res) :- M is N-1, fact(M,Aux), Res is N*Aux.
lFact([],N,N).
lFact([H|T],M,N) :- fact(M,H), M1 is M+1, lFact(T,M1,N).
sumaFact(X,N) :- lFact(L,1,N), sumList(L,X).
```

Evaluación

- Alto nivel de abstracción
  - Especificación ejecutable
- Declarativo
- Fundamento lógico robusto
  - Resolución
- Ejecución lenta

Orientado a Objetos

```
Numero << factorial

self = 0 ifTrue: [^ 1].
self > 0 ifTrue: [^ self * (self - 1) factorial].
self error: 'Not valid for negative integers'

Numero << sumaFactoriales

^((1 to: self) collect: [:x | x factorial]) sum
```

Evaluación

- Computación a través del intercambio de mensajes entre objetos
- Dos enfoques: basados en clases o por prototipos.

Orientado a Objetos

Evaluación

- Alto nivel de abstracción
  - Objetos
  - Mensajes
- Arquitectura extensible
  - Polimorfismo de subtipos
  - Binding dinámico
- Matemática de programas compleja

Top five: "Great Works in Programming Languages", B.Pierce

- C.A.R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-580 and 583, October 1969.
- Peter J. Landin. The next 700 programming languages. Communications of the ACM, 9(3):157-166, March 1966.
- Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 17:348-375, August 1978.
- Gordon Plotkin. Call-by-name, call-by-value, and the λ-calculus. Theoretical Computer Science, 1:125-159, 1975.
- John C. Reynolds. Towards a theory of type structure. In Colloque sur la Programmation, Paris, France, volume 19 of Lecture Notes in Computer Science, pages 408-425. Springer-Verlag, 1974.

Vanguardia

Conferencias Europeas

- The European Joint Conferences on Theory and Practice of Software (ETAPS)
  - Foundations of Software Science and Computation Structures (FOSSACS)
  - Fundamental Approaches to Software Engineering (FASE)
  - European Symposium on Programming (ESOP)
  - International Conference on Compiler Construction (CC)
  - Tools and Algorithms for the Construction and Analysis of Systems (TACAS)
- International Colloquium on Automata, Languages and Programming (ICALP)
- Computer Science Logic

Vanguardia

Conferencias ACM

- Principles of Programming Languages (POPL)
- International Conference on Functional Programming (ICFP)
- Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)
- Programming Language Design and Implementation (PLDI)
- Principles and Practice of Parallel Programming (PPOPP)

Recordando Haskell

- Valores vs Expresiones
- Transparencia referencial
- Tipos básicos (Int, Bool, Float) y compuestos (pares, listas)
  Los tipos tiene asociados operaciones que no tienen significado para otros tipos
- A toda expresión bien formada se le puede asignar un tipo que sólo depende los componentes de la expresión (strong-typing).
- Las funciones: ->
- Programa Funcional: Conjunto de ecuaciones.
- Evaluación: aplicar ecuaciones (orientadas de izquierda a derecha)
- Evaluación Estricta y no Estricta.
- Depende del orden de evaluación del lenguaje: Aplicativo (Eager) o Normal (Lazy)
  Haskell implementa Normal.
- Polimorfismo párametrico.

Recordando (?) Haskell

- Alto orden (funciones como valores).
- Currificación.
- Esquemas de recursión (map, filter, fold).