Introducción

Modelos Formales de Computación

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Paradigmas de Programación: Lenguajes y Modelos

LENGUAJES

- Un lenguaje de programación es un lenguaje implementable con su sintáxis y semántica definidas formalmente.

  Ejemplo: un compilador y un intérprete son definiciones formales que usan otro lenguaje de programación como método de descripción.

- Un lenguaje de programación se compone de diferentes características las cuales pueden clasificarse examinando la semántica: “esenciales” y “azúcar sintáctico”.

Paradigmas de Programación: Lenguajes y Modelos

MODELOS

- Idea 1: Un modelo de computación (MC) de un lenguaje de programación es un modelo que trata de capturar los aspectos del lenguaje para poder razonar formalmente sobre ellos.

  Cuestión 1: ¿Cualquier modelo me sirve para un lenguaje específico?

  No, por ejemplo cualquier computación determinista puede expresarse en una Máquina de Turing (MT), pero este modelo formal no nos ayudará a razonar sobre la correción de un programa concreto escrito en un lenguaje específico porque las MT son muy generales y alejadas de la semántica de un lenguaje específico.

- Idea 2: Un MC trata de capturar los aspectos esenciales del lenguaje o de una implementación del lenguaje.

Paradigmas de Programación: Lenguajes y Modelos

MODELOS

- Definición: Un MC es un sistema formal que define un lenguaje y cómo las sentencias del mismo se ejecutan en una máquina abstracta.
- Lenguajes y modelos están muy relacionados:
  - el lenguaje se construye sobre su modelo de computación (o viceversa) o al mismo tiempo
- Un modelo razonable es el que puede usarse para resolver muchos problemas, tiene técnicas de razonamiento prácticas y directas y puede implementarse fácilmente.

Paradigmas de Programación: Lenguajes y Modelos

PARADIGMA

- Un paradigma de programación es un conjunto de principios para programar un computador.
- Cada paradigma soporta un conjunto de conceptos que lo hacen apropiado para determinado tipo de aplicaciones.

  POO → gran número de abstracciones organizadas en una jerarquía

  PL → transformar o navegar por estructuras simbólicas complejas de acuerdo a reglas lógicas
- Pueden haber muchos lenguajes pertenecientes a un paradigma

  Java y Prolog

Paradigmas de Programación: Lenguajes y Modelos

- El Modelo de Computación hace precisa la noción imprecisa de Paradigma de Programación.

- Cuando un lenguaje se basa en varios modelos de computación decimos que es MULTIPARADIGMA.

  Por ejemplo, **Erlang** es funcional, concurrente y soporta programación distribuida tolerante a fallos; o **Microsoft F#** es imperativo, funcional, OO, concurrente y con reflexión.

Paradigmas de Programación: Lenguajes y Modelos

- Cada MC se basa en un lenguaje “núcleo” (kernel).
- Añadir un nuevo concepto a un MC permite nuevas formas de expresión (lo que puede simplificar los programas) pero puede hacer mas complicado el razonamiento.

  Por ejemplo, añadir asignación destructiva/variables mutables (estado explícito) al modelo de la programación funcional nos permite expresar cualquier técnica de la programación orientada a objetos. Sin embargo, es mas fácil razonar sobre programas funcionales que sobre orientados a objetos

Definiendo un Lenguaje de Programación

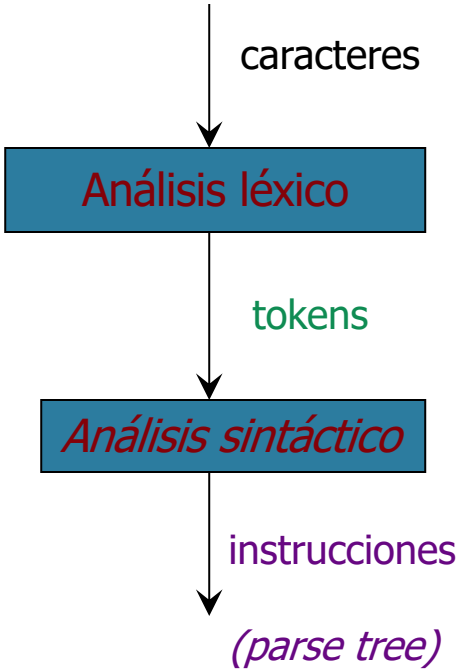
- Sintaxis (gramáticas)
- Semántica (significado)

Sintaxis

- Define **qué** es un programa legal, es decir, un programa que puede ser ejecutado por una máquina.
- La sintaxis se define mediante reglas gramaticales.
- Una gramática define **cómo** construir las sentencias (**instrucciones**) del lenguaje combinando las palabras (**tokens**).
- Las reglas gramaticales describen los tokens y las instrucciones.

Sintaxis

- Una instrucción es una secuencia de tokens.
- Un token es una secuencia de caracteres.
- Analizador Léxico: un programa que reconoce una secuencia de caracteres y produce una secuencia de tokens.
- Parser (Analizador Sintáctico): un programa que reconoce una secuencia de tokens caracteres y produce una secuencia de instrucciones que se representan como árboles (parse tree).



Sintaxis

Extended Backus-Naur Form

- EBNF (Extended Backus-Naur Form) es una notación para definir gramáticas para lenguajes de programación.
- Tipos de símbolos:
  - Terminales: tokens
  - No Terminales: secuencias de tokens que se representan por una regla gramatical

$$\langle \text{nonterminal} \rangle \quad ::= \quad \langle \text{rule body} \rangle$$

Sintaxis

REGLAS GRAMATICALES

- ⟨digit⟩ ::= 0 | 1 | 2 | 3 | 5 | 6 | 7 | 8 | 9 |
- ⟨integer⟩ ::= ⟨digit⟩ { ⟨digit⟩ }
- ⟨statement⟩ **::=** skip **|** ⟨expression⟩ ‘=‘ ⟨expression⟩
  | if ⟨expression⟩ then ⟨statement⟩
  { elseif ⟨expression⟩ then ⟨statement⟩ **}**
  [else ⟨statement⟩] end **|** …
- ⟨expression⟩ **::=** ⟨variable⟩ **|** ⟨integer⟩ **|** …

Semántica

- Define **qué** hace un programa cuando se ejecuta.
- La semántica debe ser simple y debe permitir al programador razonar sobre los programas (corrección, tiempo de ejecución, y uso de la memoria)
- ¿Cómo definir la semántica para un lenguaje con el que construir sistemas complejos?
- Definir la semántica de un lenguaje kernel que tenga en cuenta sólo los aspectos básicos del lenguaje y un traductor de la sintaxis del lenguaje de programación al lenguaje kernel.

LENGUAJE KERNEL

⟨s⟩ ::=	skip	Empty statement
	⟨s⟩ $_1$ ⟨s⟩ $_2$	Statement sequence
	local ⟨x⟩ **in** ⟨s⟩ **end**	Declaration
	⟨x⟩ = ⟨y⟩	Variable-variable binding
	⟨x⟩ = ⟨v⟩	Variable-value binding
	if ⟨x⟩ **then** ⟨s⟩ $_1$ **else** ⟨s⟩ $_2$ **end**	Conditional
	case ⟨x⟩ **of** ⟨pattern⟩ **then** ⟨s⟩ $_1$ **else** ⟨s⟩ $_2$ **end**	Pattern matching
	{ ⟨x⟩ ⟨y⟩ $_1$ ... ⟨y⟩ $_n$}	Procedure application

⟨v⟩ ::= ⟨number⟩ | ⟨record⟩ | ⟨procedure⟩

⟨number⟩ ::= ⟨int⟩ | ⟨float⟩

⟨record⟩ , ⟨pattern⟩ ::= ⟨literal⟩ | ⟨literal⟩ (⟨feature⟩ $_1$: ⟨x⟩ $_1$ ... ⟨feature⟩ $_n$: ⟨x⟩ $_n$)

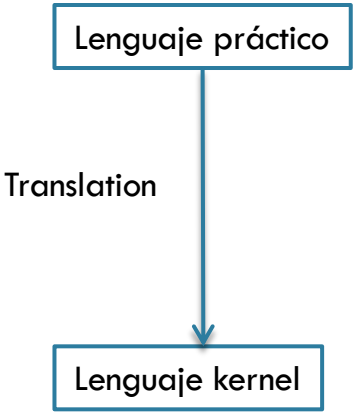
⟨procedure⟩ ::= **proc** { $ ⟨x⟩ $_1$ ... ⟨x⟩ $_n$} ⟨s⟩ **end**

⟨literal⟩ ::= ⟨atom⟩ | ⟨bool⟩

⟨feature⟩ ::= ⟨atom⟩ | ⟨bool⟩ | ⟨int⟩

⟨bool⟩ ::= **true** | **false**

LENGUAJE KERNEL




```
fun {Sqr X} X*X end
B = {Sqr {Sqr A}}
```

- Proporciona abstracciones útiles para el programador
- Puede extenderse con otras abstracciones

```
proc {Sqr X Y}
   { * X X Y}
end
local T in
   {Sqr A T}
   {Sqr T B}
end
```

- Fácil de comprender y razonar con él
- Tiene una semántica formal precisa

Semántica

Aproximaciones a la semántica formal

- La **semántica denotacional** y la teoría de dominios ve los programas como simples objetos matemáticos, centrándose en su comportamiento de entrada-salida y abstrayéndose del flujo de control.
- La **semántica axiomática** se basa en usar lógicas de programas (como la lógica de Hoare) para razonar sobre los programas.
- La **semántica operacional** describe el comportamiento de los programas por medio de máquinas abstractas.
- El **cálculo de procesos** se centra en el comportamiento de sincronización y comunicación de sistemas concurrentes.

Principales Paradigmas de Programación

- Imperativo
- Funcional
- Lógico
- Orientado a Objetos
- Concurrente

El Modelo de Computación Imperativo

- La computación consiste *en efectuar cambios y responder a la información almacenada en una memoria*.
- La computación puede describirse en términos de una *secuencia* de *pasos* determinados por los *estados de la memoria*.
- Matemáticamente, esta forma de computación puede caracterizarse en términos de la *máquina de Turing*.
- Qué es y qué no es computable puede derivarse a partir de las capacidades de la máquina de Turing.

La Programación Imperativa

- El modelo imperativo es la base de la forma en la que se construyen los computadores (*arquitectura Von Neumann*).
- Muchos lenguajes de programación ampliamente conocidos incluyen un subconjunto de características *imperativas*.
- Los algoritmos expresados imperativamente pueden analizarse en términos de cuántos pasos básicos realizan y cuánta memoria requieren (lo que ha dado lugar a estudiar la *complejidad computacional*).

El Modelo de Computación Funcional

- La computación consiste *en aplicar funciones a datos*.
- Todo puede construirse con funciones las cuales pueden combinarse de forma compleja.
- Matemáticamente, las funciones y sus comportamiento puede modelizarse con el *lambda cálculo*.
- Otros modelos de computación: lógica combinatoria y los sistemas de reescritura.

La Programación Funcional

- Los lenguajes de PF tienen una sintaxis simple pero son muy expresivos: *alta productividad*.
- Pueden modularizarse ya que *carece de efectos laterales*.

- La PF introduce ciertas ideas de programación:
  - declarativa (sin *control explícito, aunque con variantes*).
  - sin una "arquitectura de máquina" explícita (modelo de memoria).

Laboratorio con Haskell

El Modelo de Computación Lógico

- La computación consiste *en razonar sobre qué se deduce del enunciado de un problema*.
- En un problema computacional, los datos y restricciones pueden ser expresados en lógica.
- Computar el resultado es inferir que algo se sigue lógicamente de una descripción de los aspectos relevantes del mundo.
- Una forma particular de enunciados lógicos, *las cláusulas de Horn*, tienen una interpretación computacional natural.

La Programación Lógica

- Los programas lógicos también tienen una sintaxis simple.

- Heredan la modularidad natural de los programas funcionales, pero pueden modelizar relaciones además de funciones.

- La PL introduce ciertas ideas de programación:
  - declarativa (sin control explícito, aunque con variantes).
  - datos parcialmente definidos
  - flexibilidad en qué argumentos son de entrada o salida.

Laboratorio con Prolog

El Modelo de Computación Orientado a Objetos

- La computación consiste *en la interacción entre distintos participantes que pueden tener un estado propio*.
- Capturar las características esenciales:
  - **objetos** (colecciones de atributos y métodos)
  - ***crear y destruir objetos***
  - **usar métodos** de un objeto (llamada a *funciones* o *paso de mensajes*).
  - **alterar atributos y métodos** en un objeto.
- Modelos de computación: cálculo de objetos, álgebra de procesos, cálculo de sistemas comunicantes

La Programación Orientada a Objetos

- Programación con **encapsulación**, **estado explícito** y **herencia**.
- Soportado por la abstracción ***clase***.
- Un programa OO describe el comportamiento de un sistema en términos de sus constituyentes (objetos).
- Cada objeto tiene datos internos y la capacidad de actuar (cambiar) esos datos.
- Los objetos solo interactúan a través de mensajes.

El Modelo de Computación Concurrente

- La computación consiste *en la ejecución simultánea de varias instrucciones (al mismo tiempo)*.

- Características de los sistemas concurrentes (opuestas al modelo secuencial)
  - No terminación
  - No determinismo
  - Interferencia

- Modelos de computación: redes de petri, conducido por los datos (hilos, thread), sistemas de transición, streams (productor/consumidor), conducido por la demanda (lazy), cálculo de procesos.

La Programación Concurrente

- Varias aproximaciones:
  - estado compartido (Java)
  - pase de mensajes (Erlang)

- En un programa concurrente, cada secuencia de operaciones (*thread*) se ejecuta de forma secuencial pero los hilos se comunican e interfieren entre sí.
- La interferencia está sujeta a restricciones impuestas por las operaciones de sincronización.