Análisis Comparativo de Lenguajes (cod. 1956)

Ciencias de la Computación

Dpto. de Computación - FCEFQyN - UNRC

Departamento de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto

2016

Presentación de la Asignatura

Teóricos

Ariel Gonzalez.
Miércoles de 11 a 14hs., Aula 18 Pab.4I

Prácticos

Comisión 1:
Lunes de 9 a 12 hs., Lab. 102 Pab. 2 y Viernes de 10 a 12 hs., Lab. 102 Pab. 2.
Comisión 2:
Martes de 16 a 19hs, Lab. 102 Pab. 2 y Jueves de 14 a 16 hs., Lab. 101 Pab. 2.

Docentes de los prácticos

Prof. Maria M. Novaira - Lic. Valeria Bengolea - Mg. Ariel Gonzalez - Prof. Angeli Sandra
Ay. de Segunda: a confirmar

Presentación de la Asignatura

Respecto a las Consultas

- La secretaría del departamento no brindará información a cerca de los contenidos y horarios de la materia.
- No se responderán consultas telefónicas en ningúm momento.
- No se reponderán consultas fuera de los horarios establecidos.
- Toda información referida a la materia, se encontrará exclusivamente en el sitio web de la asignatura.

Presentación de la Asignatura

Modalidad y Calendario

El régimen de regularización de la materia, exige la aprobación de 2 exámenes parciales con sus respectivos recuperatorios.

- Primer Parcial: Viernes 30-09
- Primer Recuperatorio: Jueves 06-10.
- Segundo Parcial: Viernes 10-11.
- Segundo Recuperatorio: Jueves 17-11.
- Los exámenes deberán aprobarse con nota mínima: 5 (cinco).

Temario

- Estudio de los Conceptos y Principios usados en el diseño de lenguajes de programación
- Análisis de características y comparación de lenguajes
- Paradigmas de programación (imperativo, orientada a objetos, funcional y logica)
- Concurrencia
- Semántica de lenguajes de programación

Objetivos

- Mayor capacidad de análisis y comparación de lenguajes
- Conocimiento de detalles de implementación
- Mejor utilización de los lenguajes
- Mejor utilización de herramientas de desarrollo

Bibliografía y herramientas a utilizar

Bibliografía

- *Programming Language Concepts and Paradigms*. David Watt
- *Programming Languages: Design and Implementation. 2nd Ed.* Terrence Pratt, Marvin Zelkowitz
- *Concepts, Techniques and Models of Computer Programming*. Peter Van Roy and Seif Haridi
- Tutoriales y manuales de los lenguajes a utilizar

Lenguajes de programación a utilizar

- Imperativos: Pascal, C, Python, etc.
- Funcionales: Lisp, Haskell, etc.
- Orientados a objetos: C++, Java, Python, etc.
- Lógicos: Prolog
- Multiparadigma: Oz
- Concurrencia: Java y Erlang

Un poco de historia

Años	Evolución
1951-55	Assembly, Lenguajes de expresiones (A0)
1956-60	Fortran, Algol 58, COBOL, Lisp
1961-65	Snobol, Algol 60, COBOL 61, Jovial, APL
1966-70	Fortran 66, COBOL 65, Algol 68, Simula, BASIC, PL/I
1971-75	Pascal, COBOL 74, C, Scheme, Prolog
1976-80	Smaltalk, Ada, Fortran 77, ML
1981-85	Turbo Pascal, Smaltalk 80, Ada 83, Postcript
1986-90	Fortran 90, C++, SML, Haskell
1991-95	Ada 95, TCL, Perl, Java, Ruby, Python
1996-00	Ocaml, Delphy, Eiffel, JavaScript
2000-	D, C#, Fortran 2003, Starlog, TOM, . . .

Diseño de lenguajes de programación

Qué es la programación?

Diseñar y utilizar (componer) abstracciones para obtener un objetivo.

Objetivo principal

Permitir definir ***abstracciones*** para describir *procesos computacionales* en forma **modular**.

- *Abstracción procedural*: abstrae *computaciones*.
- *Abstracción de datos*: abstrae la representación (implementación) de los datos.

Lenguajes de Programación - Propiedades requeridas

- **Universal**: cada problema computable debería tener una solución programable en el lenguaje.
- **Natural:** con respecto a su dominio de aplicación. Por ejemplo, un lenguaje orientado a problemas numéricos debería ser muy rico en tipos numéricos, vectores y matrices.
- **Implementable**: debería ser posible escribir un interprete o compilador en algún sistema de computación.
- **Eficiente: en recursos**
- **Simple**
- **Uniforme**
- **Legible**
- **Seguro**

Sintaxis y semántica

- **Sintaxis**: tiene que ver con la *forma* que adoptan los programas.
- **Semántica**: tiene que ver con el *significado* de los programas, es decir de su *comportamiento* cuando son ejecutados.

Definiciones formales de sintaxis

Gramáticas

Gramáticas: conjunto de *reglas* o *producciones* que especifican cadenas válidas de palabras del lenguaje

- *regulares*: las reglas son de la forma
  $NonTerminal \rightarrow Terminal\ NonTerminal \mid Terminal$
  Equivalentes a las *expresiones regulares*
- *libres de contexto y BNFs*: permiten definir frases estructuradas. Las reglas son de la forma:
  $NonTerminal \rightarrow (NonTerminal \mid Terminal)^*$

Autómatas

Formalismos de aceptación de cadenas de un lenguaje.

- *finitos*: aceptan lenguajes regulares
- *pila*: aceptan lenguajes libres de contexto

Gramáticas libres de contexto

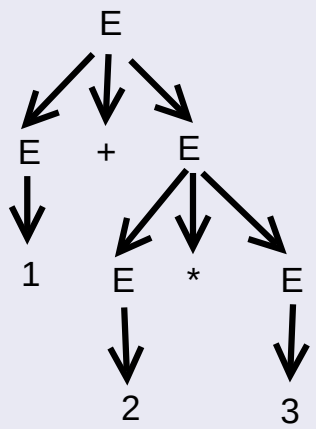
Gramática libre de contexto

$$G =< N, T, S, P >$$

- N: conjunto de símbolos ***no terminales (variables)***
- T: conjunto de símbolos***terminales***
- P: conjunto de producciones de la forma $\subseteq N \times (N \times T)^*$
- S: símbolo de comienzo.

Ejemplo de una gramática libre de contexto (expresiones aritméticas):
E -> E + E | E - E | E * E | E / E | (E) | D
D -> 0 | 1 | . . . | 9

Árbol sintáctico o de derivación



Un árbol de derivación para la cadena *1+2*3*
La gramática es *ambigua*.

Extended Backus-Naur Form (EBNF)

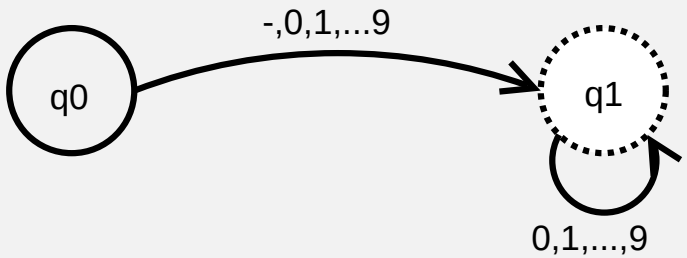
- Provee una forma más cómoda de describir gramáticas libres de contexto.
- La parte derecha de una producción es reemplazada por una expresión regular:
  - Parte opcional:
    $N \rightarrow \ldots [\text{opcional}] \ldots$
  - Repeticiones:
    $N \rightarrow \ldots (0 \text{ o más veces}) * \ldots$
    $N \rightarrow \ldots (1 \text{ o más veces}) + \ldots$
- Permite la implementación inmediata de reconocedores (parsers):
  1. Cada *no terminal* N es un procedimiento.
  2. El cuerpo del procedimiento reconoce *tokens* (terminales) e invoca a otros procedimientos.

Reconocedores de lenguajes (parsers)

- El *parser* implementa un reconocedor (ej:autómata pila) para una gramática libre de contexto.
- El parser se apoya en el *scanner* para reconocer los símbolos terminales (*tokens*).
- El scanner reconoce un lenguaje regular, por lo que implementa un autómata finito.
- A las características *dependientes del contexto* (ej: uso de un identificador dentro de su alcance) las resuelve el *analizador semántico*.
- Existen herramientas para generar código fuente de reconocedores de lenguajes a partir de la especificación de la gramática. Ej: *lex* y *yacc*.

Definiciones formales de sintaxis (cont.)

- Expresión regular que acepta números enteros: $(-)?[0-9]+$
- Autómata finito que acepta un lenguaje regular equivalente:



(el estado q_1 es un estado final)
Ejercicio: definir una gramática regular equivalente.

Lenguajes de Programación

Sintaxis

Definición de las frases *legales* del lenguaje.
Especificada por una gramática o EBNF.

Semántica

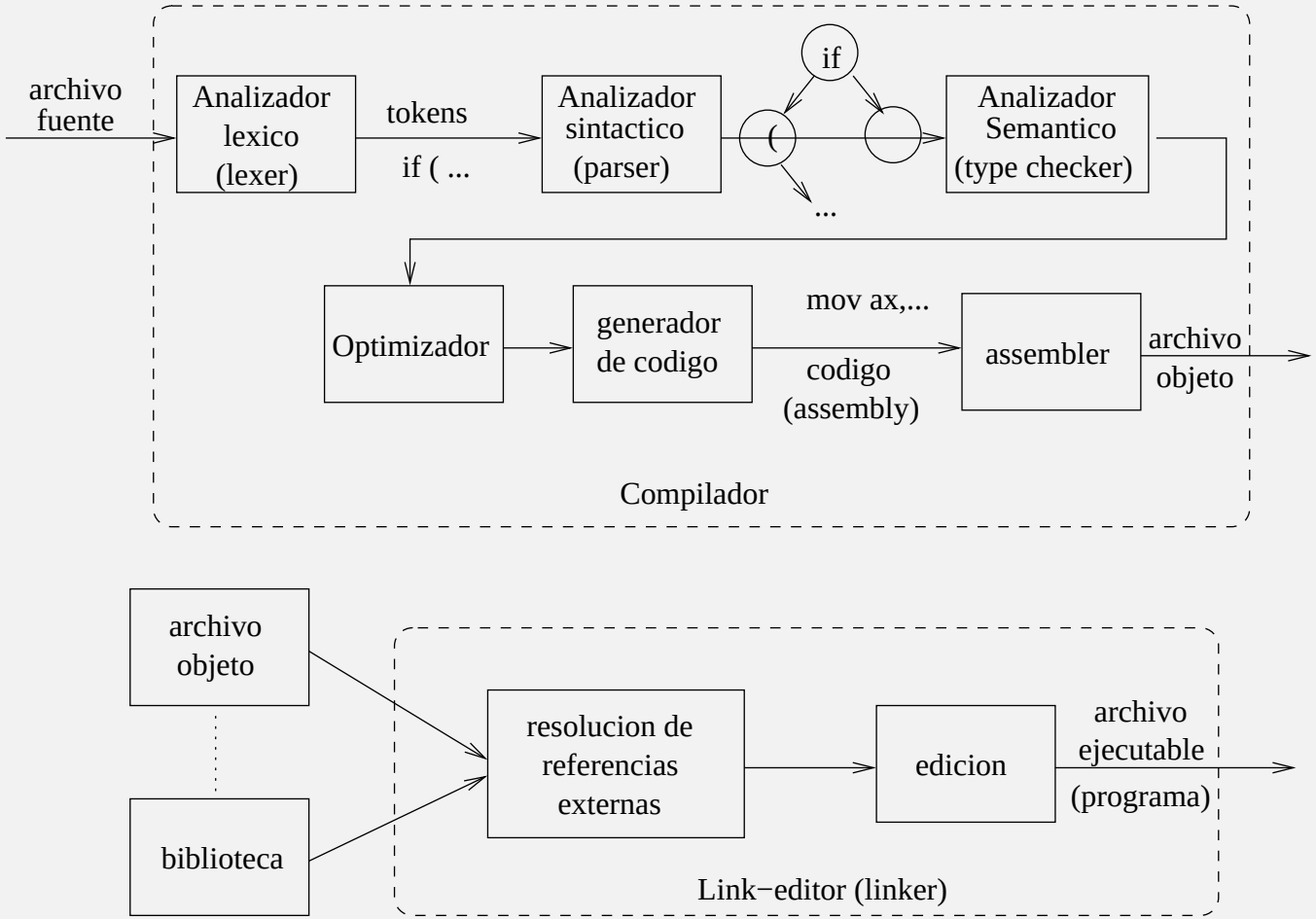
Significado de las frases del lenguaje.

- **Operacional**: cómo se ejecutan las sentencias en una máquina abstracta.
- **Denotacional**: define una sentencia como una función sobre un dominio abstracto.
- **Axiomática**: se definen sentencias como una relación entre estados de entrada y salida.
- **Lógica**: se definen las sentencias como un modelo de una teoría lógica.

Procesadores de lenguajes de programación

- **Compilador**: traduce un programa *fuente* a código assembly u objeto (archivo binario enlazable)
- **Linker o enlazador** combina archivos objetos y bibliotecas, resolviendo referencias externas y genera un *programa* (archivo binario ejecutable)
- **Intérprete**: a partir del código fuente genera una *representación interna* la cual es *evaluada o ejecutada* por el mismo intérprete(Basic,Haskel,ML,SmallTalk, etc)
- **Ejecutor o máquinas virtuales**: es posible generar código de una máquina vitual (no hardware real), ej: JVM. El código es ejecutado por medio de un *ejecutor o intérprete* (ej: COBOL, Java)

El proceso de compilación



Archivos binarios

Archivos objeto

Generalmente contienen diferentes tipos de datos:

- **Block Started by Symbol (BSS)**: área de datos estática inicializada en cero.
- **Text (code) segment**: contiene instrucciones de máquina. Generalmente de tamaño fijo y sólo lectura.
- **Data segment**: área estática que contiene los valores de las variables globales inicializadas en el programa.
- **Tabla de símbolos**: mapping de identificadores a sus direcciones de memoria (o en blanco en caso que sean referencias externas, es decir símbolos definidos en otros módulos).

Archivos binarios (cont.)

Archivos bibliotecas (libraries)

Colección de subprogramas (funciones, procedimientos, datos) relacionados.
No tienen una dirección de comienzo.

- **Estáticas**: componentes de la biblioteca son *insertados* en cada programa que la utilize. Ventajas y Desventajas.
- **Dinámicas**: los componentes se cargan durante la ejecución de un programa. Ventajas y Desventajas.

Archivos ejecutables (programas)

Son el resultado del enlazado de los diferentes archivos objetos y bibliotecas que componen un programa.
Generalmente contiene segmentos como los archivos objeto.

Capítulo 2

Lenguajes y modelos de programación

Características de los Modelos o Paradigmas de programación.

Modelos de computación

Programación

- Un *modelo de computación*: lenguaje y semántica de ejecución.
- *Técnicas de programación*.
- *Técnicas de **razonamiento** sobre programas.*

Modelos (o paradigmas)

- **Declarativo**: sin estado (memoryless)
  - Programación funcional
  - Programación lógica
- **Imperativo**: con estado (las computaciones dependen de su estado interno además de sus parámetros)
  - Componentes
  - Programación orientada a objetos
- **Concurrencia**: procesos como actividades independ.

Modelo declarativo

Ventajas

- Claridad y Simplicidad.
- Razonamiento Modular.

Desventajas

- Dificultad para modelar ciertos problemas, ej. operaciones de entrada-salida.
- Rendimientos menores respecto a programaas equivalentes con estado.

Transparencia Referencial (definición)

Analizar el ejemplo:
x := 1; $\{x = 1\}$ y := f(x) + x; $\{x = 1 \wedge y = 2 + 1\}$

Modelo con Estados

Ventajas

- Facilidad para solucionar ciertos problemas.
- Eficiencia.

Desventajas

- Perdida de razonamiento al estilo ecuacional.
- Perdida de razonamiento Modular.

Lenguajes

Lenguajes Declarativos

- *Funcionales*: Haskell, LISP, SCHEME, ML y sus derivados (Ocaml, etc.).
- *Relacionales*: Prolog y sus derivados (micro-PROLOG)
- *Memoria de Asignación única*: subconjunto de OZ.

Lenguajes con estado

- *Procedurales*: Pascal, C, Fortran, etc.
- Orientados a Objetos: JAVA, C++, etc.

Clasificación de los Elementos de un Lenguaje de Programación

- *Valores*: literales y agregados.
- *Declaraciones*: de variables, constantes, de Tipos, ...?
- *Comandos*: básicos, de secuencia o bloques, Saltos, de Selección o Condicionales, Iteración
- *Operadores*.
- *Expresiones*.

Tipos de Datos

Definición

*Un **tipo de datos** es una descripción de un conjunto de valores junto con las operaciones que se aplican sobre ellos.*

Cada valor corresponde a un *tipo de datos*.

Clasificación

- *elementales*: los números y literales.
- *estructurados*:

  Por el tipo de sus elementos que contienen:
  - **homogeneos**
  - **heterogeneos**

  Por sus características de manejo:
  - **estáticos**
  - **dinámicos**
  - **semi-dinámicos**

Chequeo de Tipos

Clasificación y Características

- Tipado estático. El chequeo se realiza en tiempo de compilación.
- Tipado dinámico. El chequeo se realiza en tiempo de ejecución.

Sistemas de tipos

- *sistema de tipos fuertes*: impide que se ejecute una operación sobre argumentos de tipo no esperado.
- *sistema de tipos débil*: realiza conversiones de tipos (*cast*). Ejemplo.
- *Polimorfismo (muchas formas)*:operaciones aceptan argumentos de una familia de tipos.
- *Tipos dependientes*: ej. los arreglos dependen de un tipo base.
- *Sistema de tipos Seguro*: un lenguaje fuertemente tipado se dice que tiene un sistema de tipos Seguro. No permite operaciones que producirían condiciones inválidas.

Declaraciones, ligadura y ambientes

Una declaración relaciona (liga) un identificador con una o mas entidades. Ej. **const Pi=3.141517**;

- Ligadura estática. Durante la compilación.
- Ligadura dinámica. Durante la ejecución.

Una declaración de alguna entidad *alcanza* una cierta porción en un programa (bloques).
Un *ambiente* (o *espacio de nombres*) es un cjto. de ligaduras.

Alcanzabilidad

- *Alcance estático*. Las sentencias de un bloque se ejecutan en el ambiente de su declaración.
- *Alcance dinámico*. Las sentencias de un bloque se ejecutan en el ambiente de su ejecución. No permite analizar un bloque de forma modular.

Equivalencia de Tipos

- Equivalencia Estructural. Requiere que dos tipos tengan la misma estructura.
- Equivalencia por nombre. Las dos expresiones de tipo deben tener el mismo nombre.

Excepciones

Definición

Es un evento que ocurre durante la ejecución como producto de alguna operación inválida, y requiere la ejecución de código fuera del flujo normal de control.

Un lenguaje debe proveer mecanismos para el manejo de excepciones (*exception handling*).