Diseño de lenguajes de programación

Que es un Lenguaje de Programación?

Herramienta fundamental de un desarrollador.

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.

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, . . .

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

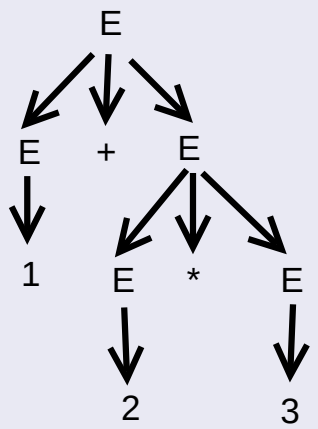
Gramática libre de contexto

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

- 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)

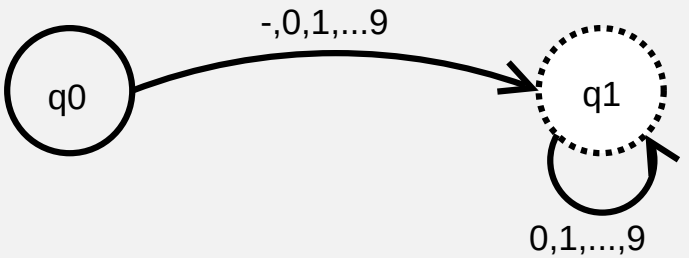
- Provee una forma más cómoda de describir gramáticas libres de contexto.
- La parte derecha de una producción es reemplazada por una expresión regular:
  - Parte opcional:
    $N \rightarrow \ldots [\text{opcional}] \ldots$
  - Repeticiones:
    $N \rightarrow \ldots (0 \text{ o más veces}) * \ldots$
    $N \rightarrow \ldots (1 \text{ o más veces}) + \ldots$
- 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

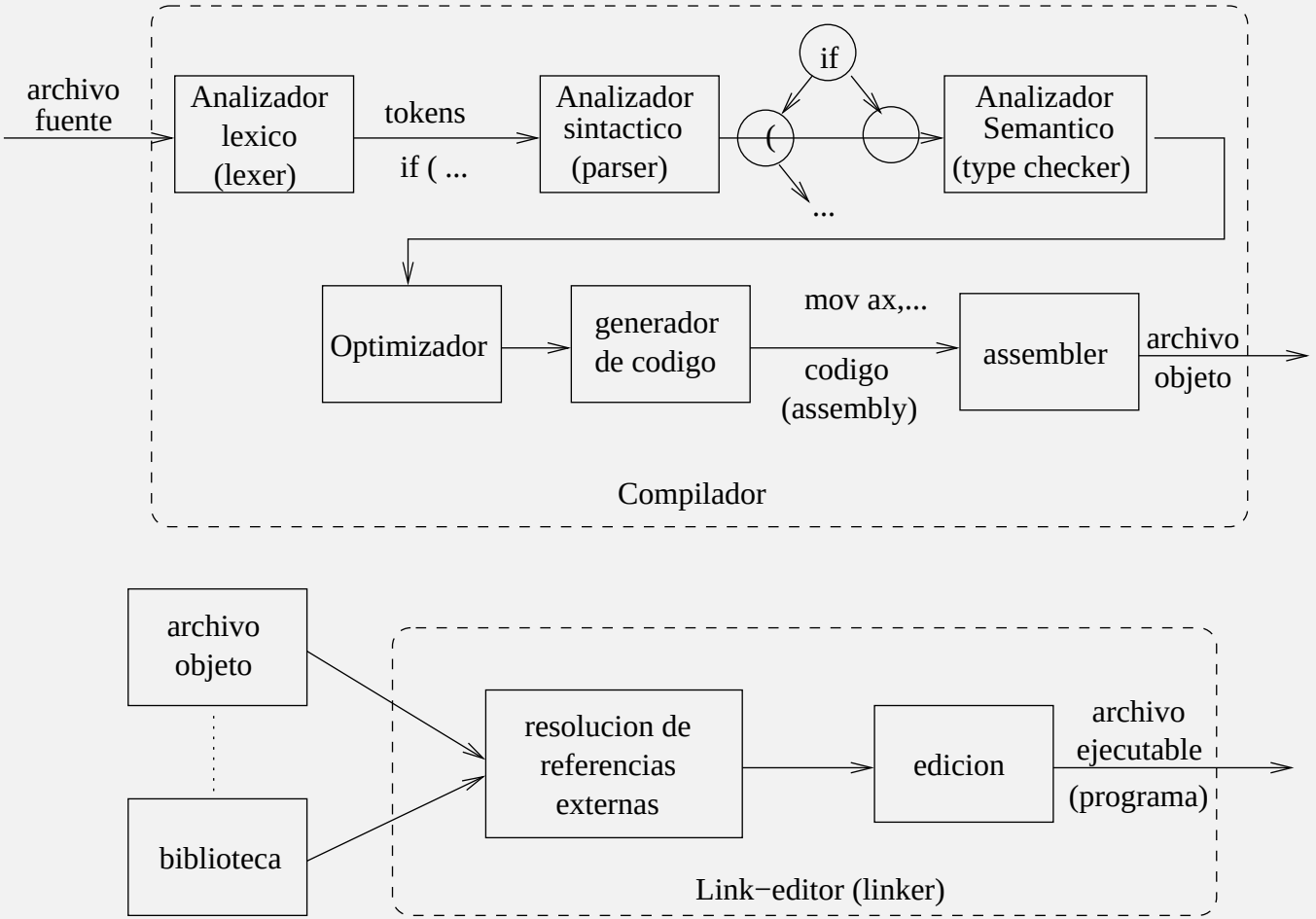
Significado de las frases del lenguaje.

- **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.