ANÁLISIS SINTÁCTICO

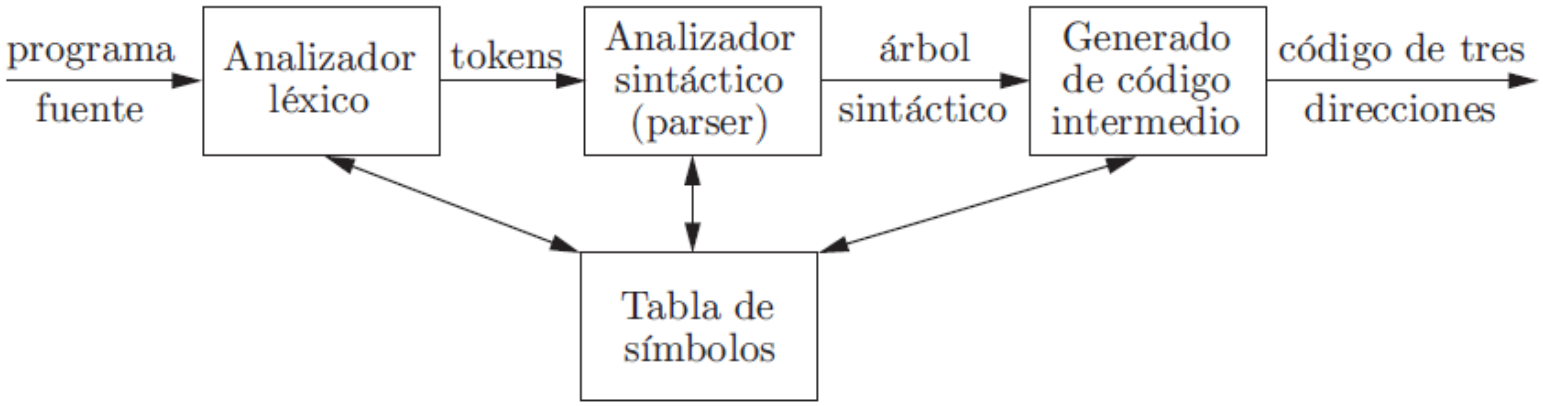
MTRO. MIGUEL ÁNGEL ROMO MARTÍNEZ

ANÁLISIS SINTÁCTICO

- Trabaja con una gramática de contexto libre y *genera el árbol sintáctico* que reconoce su sentencia de entrada. Las categorías gramaticales del análisis léxico se relacionan con los terminales de la gramática.

INTRODUCCIÓN

- Utilizando algunos meta-compiladores, la fase de análisis de un compilador descompone un programa fuente en piezas componentes y produce una representación interna, a la cual se le conoce como código intermedio. La fase de síntesis traduce el código intermedio en el programa destino.
- El front-end se muestra en el esquema y el back-end lo implementa la herramienta con la que se encuentre trabajando.



CONTENIDO

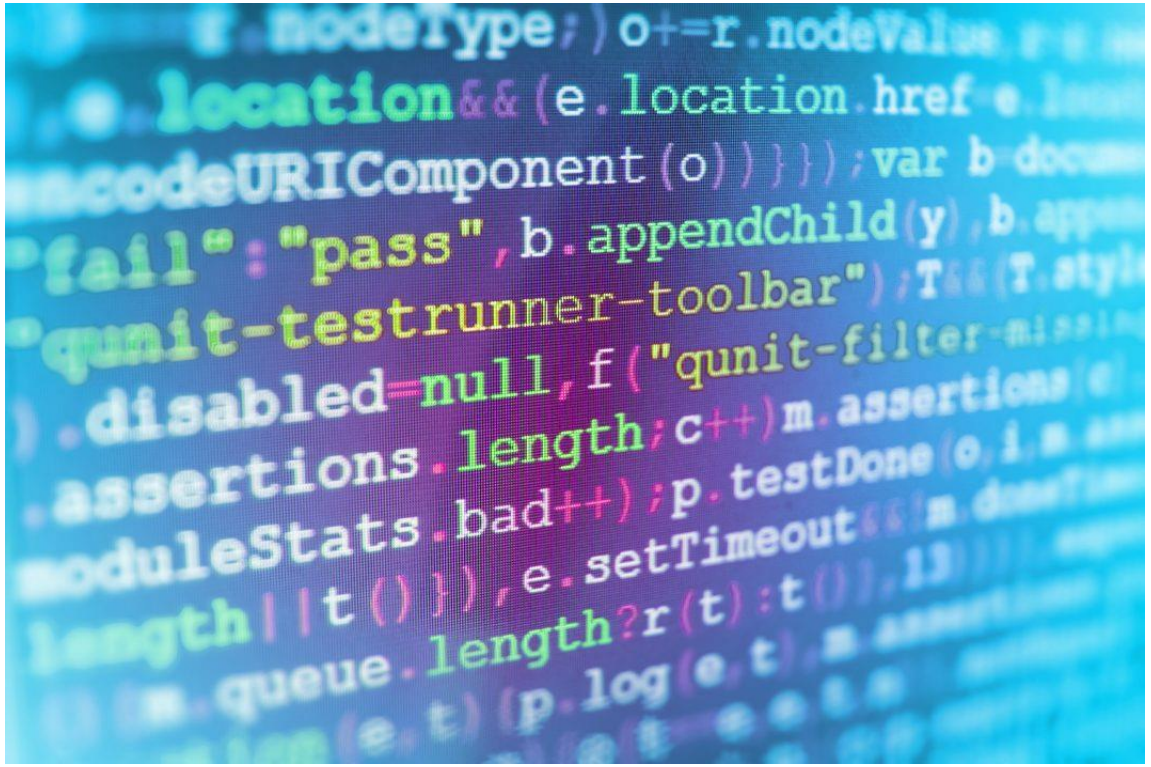
Gramática libre de contexto

BNF, EBNF y JCUP

Árbol de análisis sintáctico

GRAMÁTICA LIBRE DE CONTEXTO

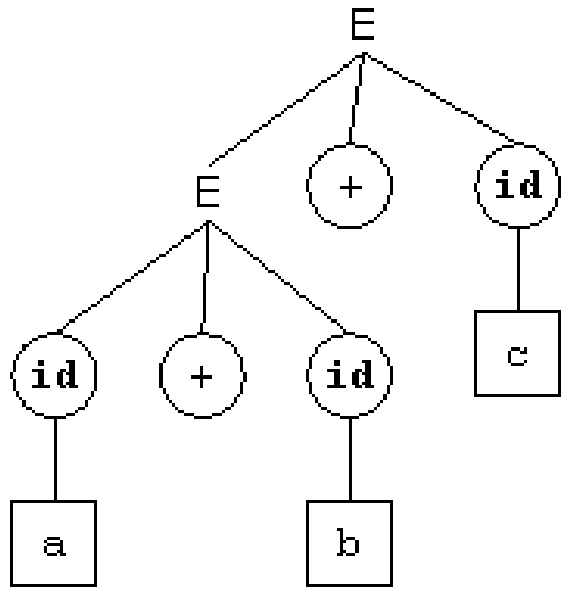
- Las gramáticas libres de contexto son **gramáticas formales** compuestas por un conjunto de símbolos **terminales**, los **tokens**, un conjunto de símbolos **no terminales**, de los cuales uno es el **axioma**, y un conjunto de **producciones**. Las producciones están compuestas por un símbolo no terminal en la parte izquierda de la misma y una secuencia de símbolos terminales y no terminales (o la cadena vacía) en la parte derecha (describen todas las cadenas posibles en un lenguaje formal dado). Las producciones pueden ir aplicándose sin tener en cuenta el contexto (el código específico).



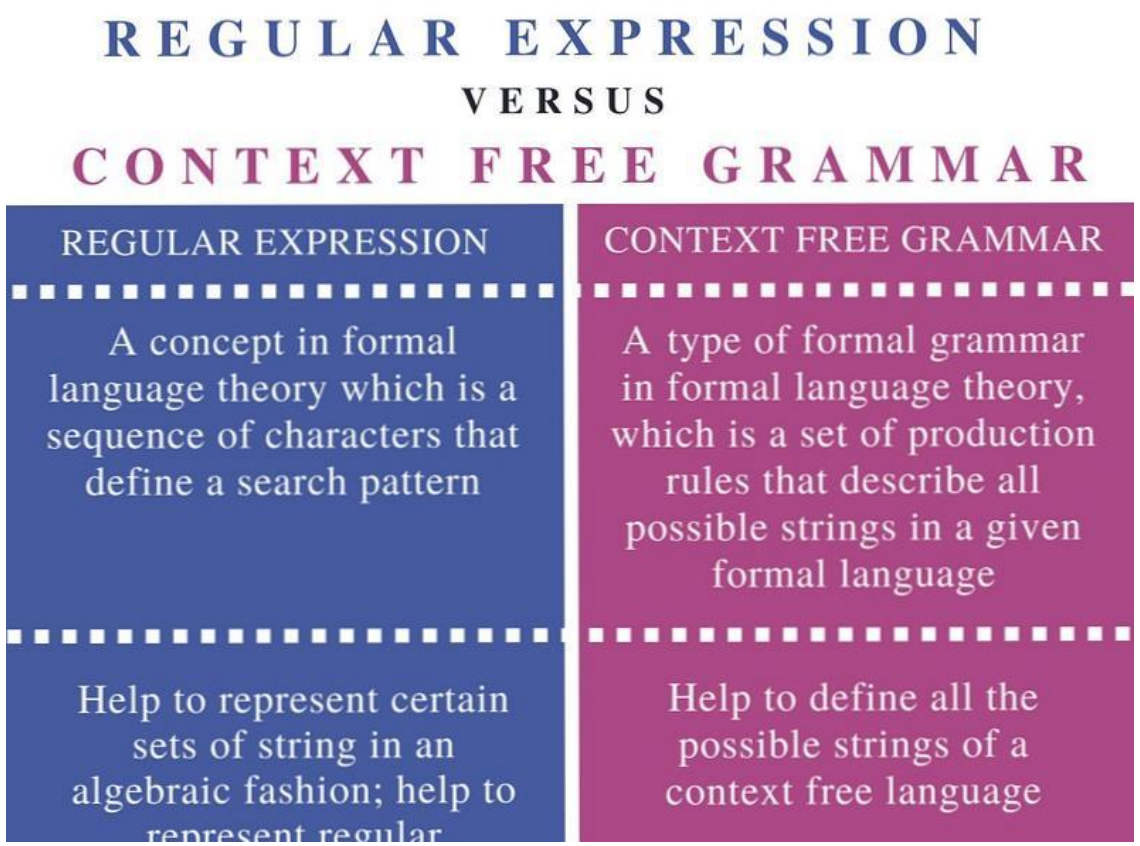
GRAMÁTICA LIBRE DE CONTEXTO DOS EJEMPLOS

- Un **ejemplo** de gramática libre de contexto para **expresiones aritméticas** es el siguiente:
- Terminales = {int, float, +, -, *, /}
  No terminales = {EXP, CONST, OP}
  Axioma = EXP
- Reglas de producción:
  EXP -> CONST OP CONST
  OP -> + | - | * | /
  CONST -> int | float

- Un segundo **ejemplo,** de una expresión algebraica utilizando una representación gráfica.



GRAMÁTICA LIBRE DE CONTEXTO VS EXPRESIONES REGULARES



NOTACIÓN DE BAKUS-NAUR (BNF)

- La Notación de Bakus-Naur (BNF) es una notación para representar gramáticas de contexto libre creada por John Bakus y Peter Naur en la década de los 50 y usada en generadores de analizadores sintácticos clásicos como Yacc (*Another Compiler-Compiler / UNIX*). La sintaxis de la misma, descrita en la propia notación, es la siguiente:
- Metasímbolos:

  ::= se define como

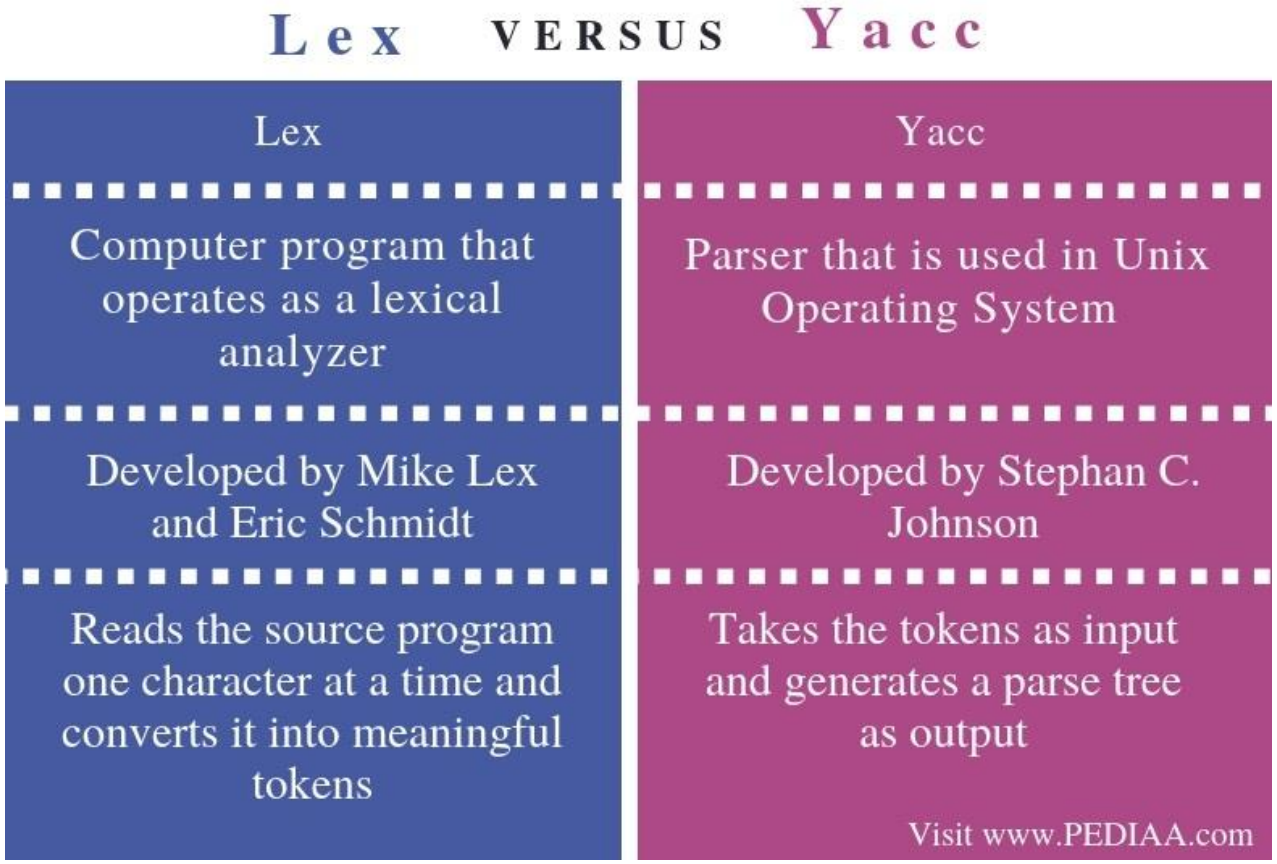
  | or

  { } repetición

  [ ] opcional

- • Los terminales entre comillas, por ejemplo: 'if', '5'

LEX VS YACC


Lex VERSUS Yacc
Lex
Computer program that operates as a lexical analyzer
Developed by Mike Lex and Eric Schmidt
Reads the source program one character at a time and converts it into meaningful tokens
Yacc
Parser that is used in Unix Operating System
Developed by Stephan C. Johnson
Takes the tokens as input and generates a parse tree as output
Visit www.PEDIAA.com

EJEMPLO DE BNF

- Gramática para representar números con decimales

  numero ::= entPos | entPos '.' entPos

  entPos ::= digito | digito entPos → Ciclo recursivo (tantas veces como aparezca)

  digito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

E-BNF

- **Extended BNF**: En algunos lenguajes de programación se utiliza el BNF mejorado, algunos símbolos utilizados son similares a los que se usan para expresiones regulares:

  ? Opcional

  * 0 a n veces

  + 1 a n veces

EJEMPLO ANTERIOR CON E-BNF

- BNF

numero ::= entPos | entPos '.' entPos

entPos ::= digito | digito entPos

digito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

- E-BNF

numero ::= digito + ('.' digito +) ?

digito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

BNF and EBNF Versions of an Expression Grammar

BNF:

```
<expr> → <expr> + <term>
       | <expr> - <term>
       | <term>
<term> → <term> * <factor>
       | <term> / <factor>
       | <factor>
<factor> → <exp> ** <factor>
         | <exp>
<exp> → (<expr>)
      | id
```

EBNF:

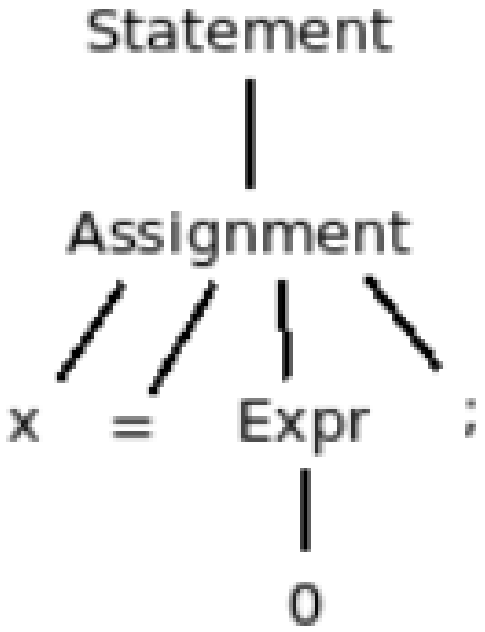
```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
<factor> → <exp> { ** <exp>}
<exp> → (<expr>)
      | id
```

JCUP

- JCUP herramienta de Java tipo LALR (Look-Ahead Left to Right) parser (analizador), que hace reconocimiento bottom-up.
- Es decir reconoce de izquierda a derecha primero las hojas y luego el nodo padre. En estructuras de datos el recorrido sería en Inorden: subárbol izquierdo, raíz, subárbol derecho.

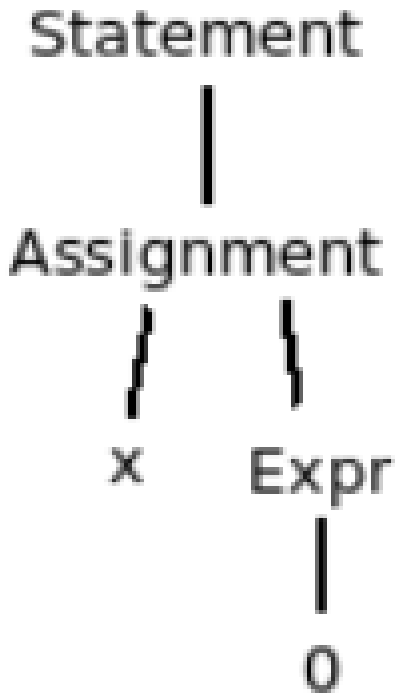
ÁRBOL SINTÁCTICO CONCRETO

- Es un árbol el cual representa con sus nodos el lenguaje tal y como ha sido reconocido por el Analizador Sintáctico.



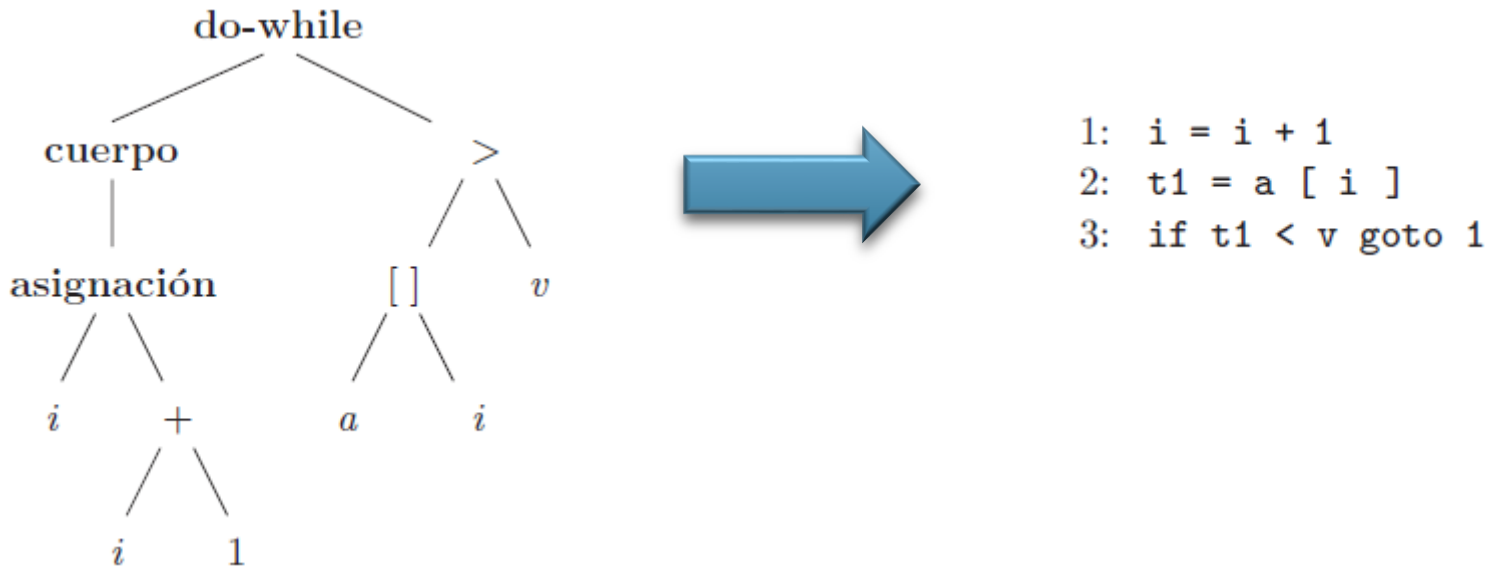
ÁRBOL SINTÁCTICO ABSTRACTO

- Es un árbol el cual representa el lenguaje fuente sin incluir ciertas construcciones sintácticas no esenciales como puntos, puntos y coma o paréntesis.



ANÁLISIS SINTÁCTICO –> CÓDIGO INTERMEDIO

- Algunos compiladores combinan el análisis sintáctico y la generación de código intermedio en un solo componente. Es decir, a partir del árbol sintáctico abstracto, se genera una representación interna.



GRACIAS

MAROMOM@UP.EDU.MX

EXPLICA EN UN PDF CON TUS PALABRAS:

GRAMÁTICA LIBRE DE CONTEXTO, BNF, EBNF, JCUP, YACC Y LOS TIPOS DE ARBOLES SINTÁCTICOS