1) Análisis Sintáctico

Wednesday, September 4, 2019 2:03 PM

Gramáticas Libres de Contexto

Análisis Sintáctico

- Universal
  - Se puede implicar en cualquier gramática libre de contexto
  - Es muy lento
- Descendente
  - Primero que apareció
  - Construye árbol sintáctico de raíz a hojas
  - Derivaciones a la izquierda
  - Tipos
    - Recursivo
    - LL(k) (Predictive) → L - Left; L - Left (Derivación izquierda); K - Núm. de símbolos que debe adelantar la cadena (Preanálisis)
- Ascendente
  - SLR o LR(0) //Derecha
  - LR(1) //Más popular
  - LALR // Más usado

Nota GCC en un inicio era LALR pero por Bison regresaron a Recursivo

Recuperación de errores

// Ejemplos

```
int x, y ;
y = 3.1416 + 'c' ;
```

- Son los que se deben a la estructura del programa

//Recuperación

a) Modo pánico

b) Gramáticas con producciones de error
   // Todos los posibles errores que puede generar el programador

c) Corrección de errores a nivel frase

c) Corrección de errores a nivel frase
// Frase → Línea de código

d) Corrección de errores a nivel global

Gramáticas Libres de Contexto

G ≜ Gramática | Son un conjunto de reglas que definen el lenguaje

→ G(N, Σ, S, P)

- N ≜ Símbolos no terminales
- Σ ≜ Alfabeto o símbolos terminales
- S ≜ Símbolo inicial y $S \in N$

$N \cap \Sigma = \phi$ // Vacío, no debe existir algo

- P ≜ Conjunto de producciones

// Forma de las producciones

$A \rightarrow \alpha$ | $A \in N$ | $\alpha \in (N \cup \Sigma)^*$

A: Encabezado, α: Cuerpo

NOTA
- Letras griegas son cadenas de terminales y no terminales
- Letras minúsculas son símbolos
- Las cadenas formadas por símbolos terminales se les conoce como sentencia

---

// Ejemplo

$E \rightarrow TE'$
$E' \rightarrow +TE'$
$E' \rightarrow \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT'$
$T' \rightarrow \varepsilon$
$T \rightarrow$)

- $N = \{E, E', T, T', F\}$
  No terminales ≜ Encabezados
- $\Sigma = \{+, *, id, (,)\}$
  // Son los tokens a generar
- ε ...

$T' \to \varepsilon$
$F \to id$
$F \to E$

// Son los tokens a generar
• S = E

// Propiedad

$A \to \alpha_1, A_1 \to \alpha_2, A_2 \to \alpha_3, \ldots, A \to \alpha_n \Longrightarrow A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$

∴ Reducción

$E \to TE'$
$E' \to +TE' | \varepsilon$
$T \to FT'$
$T' \to *FT' | \varepsilon$
$F \to id | E$

Derivación

• Es el proceso de generar las cadenas que pertenecen al lenguaje definidos en la gramática.

① En cada paso de la derivación solo se puede sustituir un sólo símbolo

• Existen 2 tipos de derivación
  1) Por derecha
  2) Por izquierda

• Ejemplo sea: Derivar $w = id + (id * id)$

$E \to TE'$
$E' \to +TE' | \varepsilon$
$T \to FT'$
$T' \to *FT' | \varepsilon$
$F \to id | E$

// Derivación por la izquierda

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow idT'E'$

izq

Escogemos esta debido a la entr

$\Rightarrow id + T' E' \Rightarrow id + F T' E' \Rightarrow$

$\Rightarrow id + (E)\, T' E' \Rightarrow id + (TE') T' E' \Rightarrow$

$\Rightarrow id + (F T' E')\, T' E' \Rightarrow$

$\Rightarrow id + (id\, T' E')\, T' E' \Rightarrow$

$\Rightarrow id + (id * F\, T' E') T' E' \Rightarrow$

$\Rightarrow id + (id * id\, T' E') T' E' \Rightarrow$

$\Rightarrow id + (id * id\, \varepsilon\ E') T' E' \Rightarrow$

$\Rightarrow id + (id * id)\, \varepsilon\ E'$

$\Rightarrow id + (id * id)$

∴ Al haberse generado sabes que pertenece al lenguaje

// Derivación por la derecha $id + (id * id)$

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to id \mid (E)$

$E \Rightarrow TE' \Rightarrow T + TE' \Rightarrow$

$\Rightarrow T + T\, \varepsilon \Rightarrow T + FT' \Rightarrow$

$\Rightarrow T + F \Rightarrow T + (E) \Rightarrow T + (TE')$

$\Rightarrow T + (T) \Rightarrow T + (FT') \Rightarrow$

$\Rightarrow T + (F * FT') \Rightarrow T + (F * F) \Rightarrow T + (F * id)$

$\Rightarrow T + (id * id) \Rightarrow FT' + (id * id) \Rightarrow F + (id * id) \Rightarrow$

$\Rightarrow id + (id * id)$

// Diseñar gramática de la declaración de variables en lenguaje C de enteras/flotantes/Char

Declaración - D
Tipo dato - T
Lista de identificadores - L

- $D \to T\ L\ ;$
- $T \to int \mid Float \mid char$
- ~~$L \to id \mid id, L \mid id = V \mid id\ L$~~ → Restructurar
- $L \to I \mid I, L$
- $I \to id\ A$

- $A \rightarrow =V \mid B \mid \varepsilon$
- $B \rightarrow [num]B \mid \varepsilon$
- $V \rightarrow$ num | caracter

Recursividad Izquierda

- $A \rightarrow A\alpha \mid \beta$

Recursividad Derecha

- $A \rightarrow \alpha A \mid \beta$

$\beta, \alpha \overset{\Delta}{=}$ Cadenas de terminales, no terminales y la cadena vacía

// Diseñar una gramática para las expresiones aritméticas

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \% E \mid (E)$
$\mid num \mid id$

NOTA: Es una gramática ambigüa

Ejemplo | Cadena | id + id * id

$E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E \Rightarrow$

$\Rightarrow id + id * id$

1) Gramaticas ambiguaa

Monday, September 9, 2019 2:21 PM

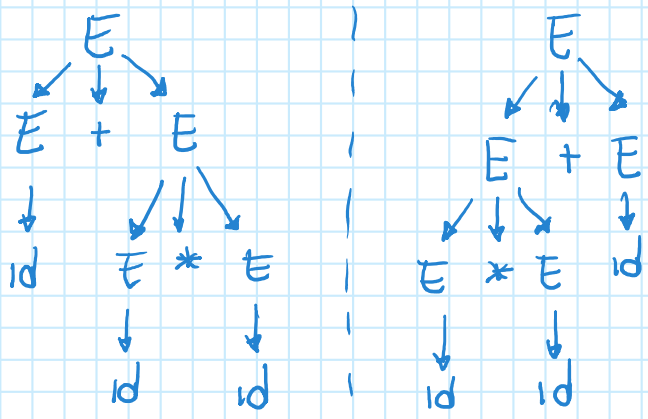
Árbol Sintáctico o Árbol de derivación

- Es la representación gráfica de una derivación.

Derivación → Se conoce que símbolo se fue derivando

Árbol → No se puede saber que tipo de derivación fue usada

- La raíz es el símbolo inicial de la gramática
- Cada nodo del árbol puede contener sólo un símbolo gramatical (Tanto un terminal como un no terminal)



// No representan lo mismo respecto a la entrada

// No dan el mismo resultado

// Son de la misma cadena

- La gramática es ambigua si tenemos diferentes árboles para una misma cadena
  - → Si es recursiva por la derecha y por la izquierda es ambigua

Pasos para eliminar recursividad

Pasos para eliminar recursividad

1) Ordenar de menor a mayor la jerarquía de los operadores

1] + -
2] * / %
3] ( )

2) A cada nivel de presedencia asignar un No Terminal

+ - E
* / % T
( ) F

3) Para los operadores binarios, seleccionar la asociatividad que mejor se les acomode

3.1) Asociatividad Izquierda → Recursividad Izquierda

3.2) Asociatividad Derecha → Recursividad Derecha

4) Los operadores unarios no pueden cambiar el tipo de asociatividad

Recursividad Izquierda: $A \rightarrow A\alpha \mid \beta$

E + -
T * / %

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid T\%F \mid F$

T * / %

F ( )

$T \to T*F \mid T/F \mid T\%F \mid F$

$F \to (E)$

NOTA: Todas las no relacionadas se colocan en el nivel máximo de presedencia

Recursiva por la Izq

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid T\%F \mid F$

$F \to (E) \mid num \mid id$

Recursiva por la Der

$E \to T+E \mid T-E \mid T$

$T \to F*T \mid F/T \mid F\%T \mid F$

$F \to (E) \mid num \mid id$

NO OLVIDAR

// Hacer una gramática ambigua para el Leng. de las exp. regulares que se pueda formar con los símbolos "a" y "b" utilizando los op +, *, ?, disy, concate y ().

$E \to id\ opUn \mid (id\ opUn)$

$opUn \to + \mid * \mid ?$

$id \to a\ id \mid b\ id \mid a \mid b$

1) EBNF

Wednesday, September 11, 2019 12:02 PM

- Condiciones que provoca que una gramática sea ambigüa
  - No se especifica la prescdencia y asociatividad en la gramática
  - Ambigüedad por "suspensión-else"
    - "if" anidadas
  - Ambigüedad a partir de producciones especiales
- Es ambigüa si su lenguaje contiene una sentencia ambigüa
  - Si se puede derivar la sentencia con 2 o más derivaciones distintas

---

- $E \to E^+ \mid E^* \mid E? \mid E|E \mid$
  - $\mid$
  - concatenación
  - $* \; + \; ?$
  - $()$

$E \to E"|"C \mid C$

$C \to CO \mid O$

$O \to O^* \mid O^+ \mid O? \mid O$

$P \to (E) \mid a \mid b$

//Por la derecha

$D \to C"|"D \mid C$

$C \to OC \mid O$

//Recuerda

- No es lo mismo el menos que la negación
- (1) Suma / Menos

C → OC | O

O → O* | O+ | O? | P

P → (O) | a | b

① Suma / Menos

② Divi / Multi /

③ Negación

④ ( )

NOTACIÓN EBNF

→ ::= <No terminal>

$A \to A\alpha \mid \beta$

$A \to \beta\{\alpha\}$ α^*

Recursividad Derecha

Gramáticas Regulares → Gramáticas Libres de Contexto

- $A \to \alpha A \mid \beta$

  $A \to \{\alpha\}\beta$ | $\{\alpha\} = \alpha^*$

- $A \to \alpha \mid \varepsilon$

  $A \to [\alpha]$ | $[\alpha] = \alpha?$

- $A \to \alpha\beta \mid \alpha$

  $A \to \alpha[\beta]$

- $A \to \alpha\beta \mid \beta$

  $A \to [\alpha]\beta$

- $A \to \alpha\beta\gamma \mid \alpha\gamma$

  $A \to \alpha[\beta]\gamma$

//Ejemplo

$A \to A\alpha \mid \beta$ | $A \to \{\alpha\}\beta$

① $E \to E + T \mid T$ | $E \to T\{+T\}$

(α = $+T$, β = T)

② $T \to T * F \mid F$ | $T \to F\{*F\}$

(2) $T \to T * F \mid F$ (α = $* F$, β = F) | $T \to F \{* F\}$

(3) $F \to (E) \mid id$ | $F \to (E) \mid id$

//No hay alfas porque no hay recursividad

$A \to \beta \{\alpha\}$

$D \to C "|" D \mid C$ (α = $C "|" D$, β = C) | $D \to C \{"|" D\}$

$C \to O C \mid O$ (α = $O C$, β = O) | $C \to O \{O\}$

$O \to O^* \mid O^+ \mid O? \mid P$ (α_1 = O^*, α_2 = O^+, α_3 = $O?$, β = P) | $O \to P \{* \mid + \mid ?\}$

$P \to (D) \mid a \mid b$ | $P \to (D) \mid a \mid b$ //Queda igual

//Por la derecha

$D \to \{"|" D\} C$ | $O \to P \{* \mid + \mid ?\}$ (¿?)

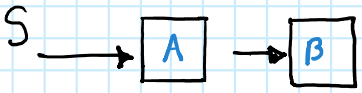
$C \to \{O\} O$ | $P \to (D) \mid a \mid b$

Es unario NO CAMBIA

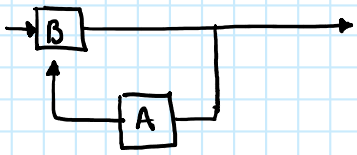
Diagrama de Sintaxis (EBNF)

- ○ ≙ Terminal
- □ ≙ No terminal
- → ← ↑↓ Sob flechas verticales y horizontales

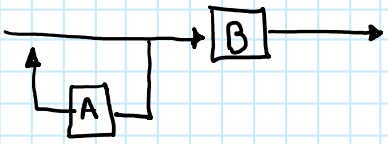
- $S \rightarrow AB$



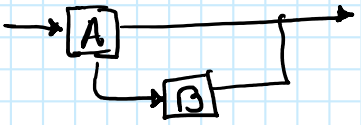
- $S \rightarrow B\{A\}$



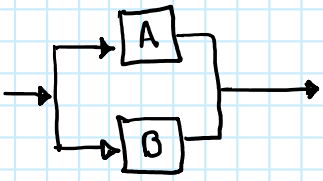
- $S \rightarrow \{A\}B$



- $S \rightarrow A[B]$

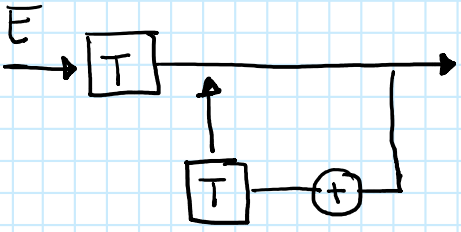


- $S \rightarrow A|B$

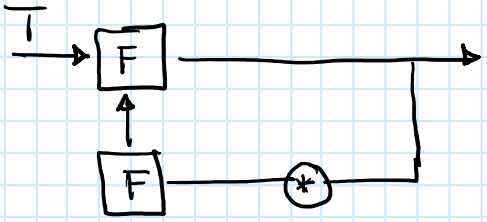


Ejercicio

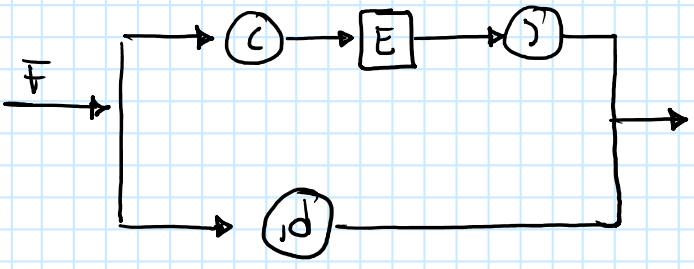
- $E \rightarrow T\{+T\}$



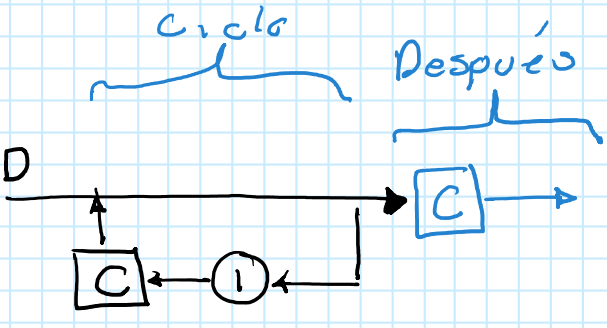
- $T \rightarrow F\{*F\}$



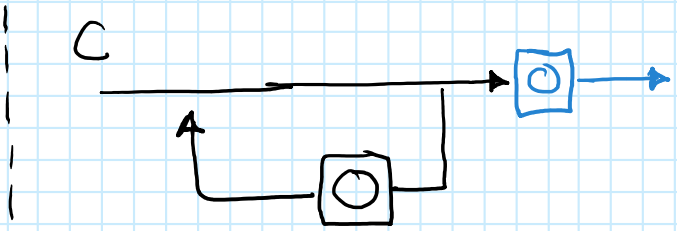
- $F \rightarrow (E)|id$



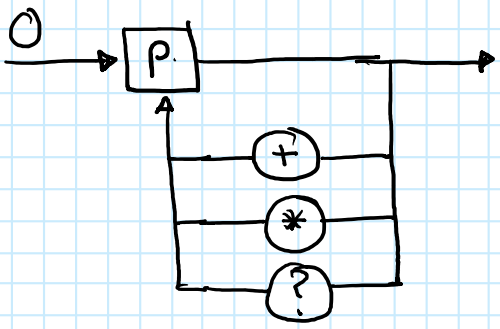
- $D \rightarrow \{C\ "i"\}C$



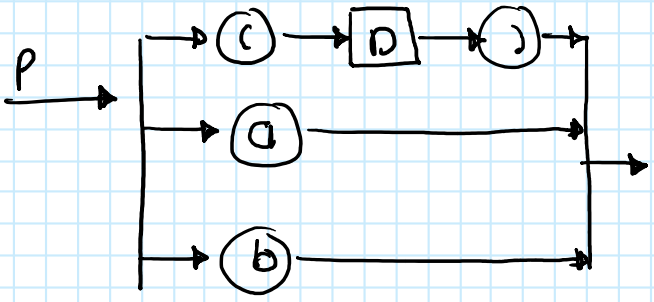
- C → {O} O



- O → P {+ | * | ?}



- P → (D) | a | b



1) Tokens.h | EBNF

Wednesday, September 18, 2019 1:24 PM

- #ifndef TOKENS_H
  #define TOKENS_H

```
#define MAS 1
#define MUL 2
#define ID 3
#define LPAR 4
#define RPAR 5
#endif
```

- Analizador Sintáctico

```
#include "tokens.h"
into TOKEN;
void E(){
 T();
 while(TOKEN==MAS){
  eat(MAS);
  T();
 }
}

void T(){
 F();
 while(TOKEN==MUL){
  eat(MUL);
  F();
 }
}

void F(){
 switch(TOKEN){

  case LPAR:
   eat(LPAR);
   E();
   eat(RPAR);
  break;

 case ID:
  eat(ID);
 break:
```

```
in main(...){
.
.
.
init(arv[1];
return 0;
}


void init(char *){
 yyin = fopen( s, "r");
 if(yyin)
 TOKEN = yylex();
 E();
 if(TOKEN==0){
  printf("String was accepted");
 }
 else{
  error();
 }
}

void eat(){
if(TOKEN ==token)
 TOKEN=yylex();
else
 error();

}
```

NOTA

También se le conoce como "parser"

```
    case ID:
      eat(ID);
    break;

    default:
      error();

    }
  }
```

```
}

void error(){
  printf("...", yylineno);
}
```

• Analizador Léxico

```
%{
 include <studio.h>
 include "tokens.h"
}%

%option noyywrap
%option yylineno
id [a-z A-Z][a-zA-Z0-9]*

%%

"+" {return MAS;}
"-" {return MUL;}
"(" {return LPAR;}
")" {return RPAR;}
{id} {return ID;}
[ \t\n] { }
. {printf("Error lexicon \n");}

%%
```

NOTA

```
extern int yylineno;
//Variable de otro archivo .c
```

COMPILACIÓN

```
flex lexer.l
gcc lex.yy.c parser.c -o -lfl
```

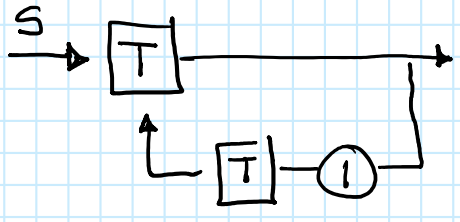
▸ Ejercicios BNF

→ $A \to A\alpha \mid B$; $A \to B\{\alpha\}$

• $S \to S\ "i"\ T \mid T$

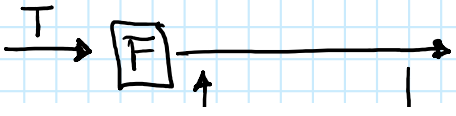
α β

↳ $S \to T\ \{"i"\ T\}$

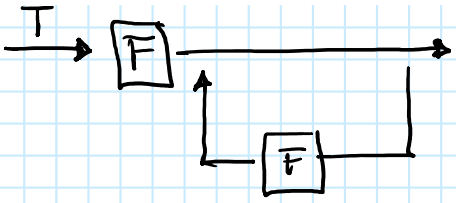


• $T \to T\ F \mid F$

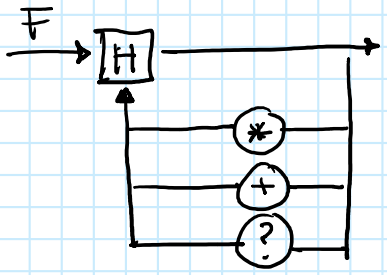
α β



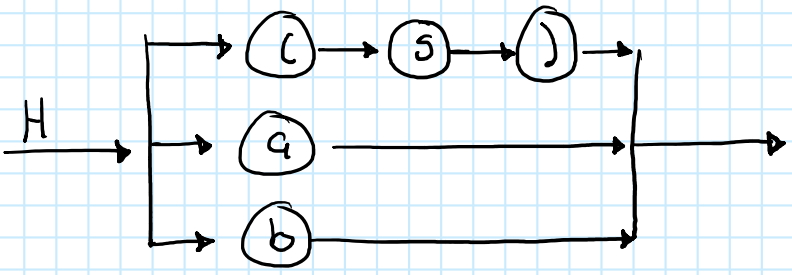
- $T \rightarrow T\ F \mid F$ (α = F, β = F)
  - $T \rightarrow F\{F\}$



- $F \rightarrow F * \mid F + \mid F ? \mid H$ (α_1 = *, α_2 = +, α_3 = ?, β = H)
  - $F \rightarrow H\{* \mid + \mid ?\}$



- $H \rightarrow (S) \mid a \mid b$



// función para "S"

```
void S(){
  T();
  while(TOKEN == "+"){
     eat("+");
     T();
  }
}
```

```
void T(){
   F();
   while(TOKEN == ? ){
      F();
   }
}
// Back-Track
```

Todavía no lo sabemos ...

2) Syntactic Analysis Down

Wednesday, September 18, 2019 2:48 PM

- No pueden ser recursivas por la izquierda
- No pueden ser ambiguas
  - Recursivo
  - Predictivo { LL(k)

$E \to T\,E'$

$E' \to +\,T\,E' \mid \varepsilon$

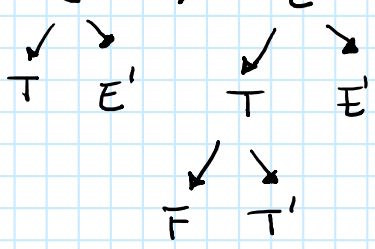
$T \to F\,T'$

$T' \to *\,F\,T' \mid \varepsilon$

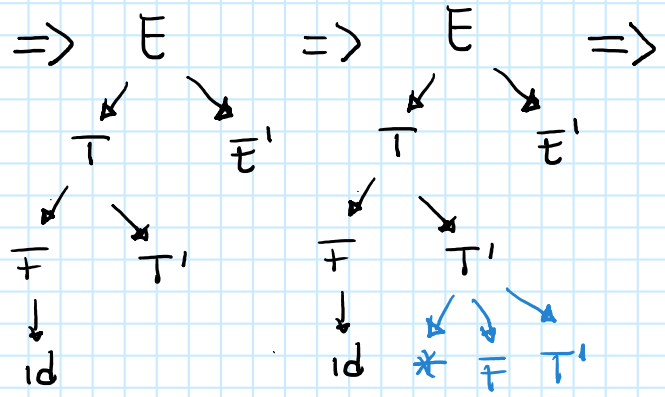
$F \to (E) \mid id$

$\omega =$ "id * id"

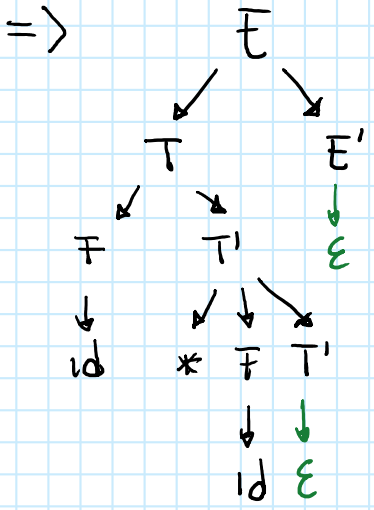
$E \overset{lm}{\Rightarrow} E \overset{lm}{\Rightarrow} E \Rightarrow$



$\Rightarrow$



$\Rightarrow$



Eliminación de la recursividad izq

• $A \to A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$

Eliminación:

$A \to \beta_1 A' | \beta_2 A' | \dots | \beta_m A'$

$A' \to \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon$

Ejemplo 1

$E \to E + T | T$ (α = $+T$, β = T)

$\begin{cases} F \to T E' \\ E' \to +T E' | \varepsilon \end{cases}$

$T \to T * F | F$ (α = $*F$, β = F)

$\begin{cases} T \to F T' \\ T' \to *F T' | \varepsilon \end{cases}$

$F \to (E) | id \longrightarrow F \to (E) | id$

Directo

▶ Factorización

$A \to \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_m$

Factorización:

$A \to \alpha A' | \gamma_1 | \gamma_2 | \dots | \gamma_m$

$A' \to \beta_1 | \beta_2 | \dots | \beta_n$

$E \to E + T | T$ (β = E, α = $+T$, T)

$\begin{cases} E \to T E' \\ E' \to +E | \varepsilon \end{cases}$ (β = $+E$)

$T \to T * F | F$

$\begin{cases} T \to F T' \\ T' \to *T | \varepsilon \end{cases}$

- Factor común

$F \to (E) | id \longrightarrow F \to (E) | id$

Directo

Ejercicio 1

Eliminar recursividad izq, ambigüedad

$A \to \beta A'$
$A' \to \alpha_1 A' \mid \varepsilon$

- $P \to DS$
- $D \to \underbrace{DTL;}_{\alpha} \mid \underbrace{\varepsilon}_{\beta}$

$$\begin{cases} D \to D' \\ D' \to TL;D' \mid \varepsilon \end{cases} \quad \begin{cases} D \to TL;D \mid \varepsilon \end{cases}$$

//Recursivo Derecha

- $T \to int \mid float$
- $L \to \underbrace{id, L}_{\alpha} \mid \underbrace{id}_{\beta}$

$$\begin{cases} L \to id\ L' \\ L' \to ,L' \mid \varepsilon \end{cases}$$

- $S \to SS \mid if(B)S \mid if(B)S\ else\ S \mid while(B)s \mid \{s\} \mid id = E;$

NOTA | //Para ambigüedad necesita al menos 2 de S(algo)S

// Factor común

$S \to if(B)\ S\ S' \mid while(B)\ s \mid \{s\} \mid id = E;$

$S' \to \varepsilon \mid else\ S$

$A \to \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$
$A' \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

// Recursividad

$S \to if(B)\ S\ S'\ S'' \mid while(B)\ s\ S'' \mid \{s\}\ S'' \mid id = E;\ S''$

$S' \to \varepsilon \mid else\ S$

$S'' \to \underbrace{SS''}_{\alpha} \mid \varepsilon \mid$

$A \to \beta A'$
$A' \to \alpha_1 A' \mid \varepsilon$

- $E \to E+G \mid E-G \mid G$ // Recursividad

  $E \to GE'$

  $E' \to +GE' \mid -GE' \mid \varepsilon$

- $G \to G*F \mid G/F \mid G\%F \mid F$ // Recursividad

  $G \to FG'$

  $G' \to *FG' \mid /FG' \mid \%FG' \mid \varepsilon$

- $F \to (E) \mid id \mid num$

- $B \to B||B \mid B\&\&B \mid !B \mid ERE$

  // Sin ambigüedad

  $A \to A\alpha \mid B$

  $B \to B||B_1 \mid B_1$

  $B_1 \to B_1 \&\& B_2 \mid B_2$

  $B_2 \to !B \mid ERE$

  // Recursividad

  $A \to \beta A'$

  $A' \to \alpha_1 A' \mid \varepsilon$

- $B \to B_1 B'$

  $B' \to ||B_1 B' \mid \varepsilon$

- $B_1 \to B_2 B_1'$

  $B_1' \to \&\& B_2 B_1' \mid \varepsilon$

- $B_2 \to !B \mid ERE$

- $R \to == \mid != \mid >= \mid <= \mid < \mid >$

→ Juntando todo

$P \to DS$

$D \to TL;D \mid \varepsilon$

$T \to int \mid float$

$L \to id\,L'$

$L' \to ,L \mid \varepsilon$

$S \to if(B)\,SS'S'' \mid while(B)\,SS'' \mid \{S\}S'' \mid id = E;\,S''$

$S' \to \varepsilon \mid else\,S$

$S'' \to SS'' \mid \varepsilon$

$E \to GE'$

$E' \to +GE' \mid -GE' \mid \varepsilon$

$G \to FG'$

$G' \to *FG' \mid /FG' \mid \%FG' \mid \varepsilon$

$F \to (E) \mid id \mid num$

$B \to B_1 B'$

$B' \to \|\, B_1 B' \mid \varepsilon$

$B_1 \to B_2 B_1'$

$B_1' \to \&\& B_2 B_1' \mid \varepsilon$

$B_2 \to !B \mid ERE$

$R \to == \mid != \mid >= \mid <= \mid < \mid >$

2) First y follow

Monday, September 23, 2019 2:28 PM

First

a) If $X \in \Sigma$

$First(x) = \{x\}$

b) If $X \to Y_1 Y_2 \dots Y_k$ $X \in N$ (No terminales)

$First(x) = First(x) \cup First(Y_1)$

From $i = 2$ to k of 1 by 1:

if $Y_1 \dots Y_{i-1}$ son todos anulables:

$First(x) = First(x) \cup First(Y_i)$

if $Y_1 \dots Y_k$ son todos anulables:

$First(x) = First(x) \cup \{\varepsilon\}$

NOTA:

A es anulable

$A \to \varepsilon$

$A \to Y_1 Y_2 \dots Y_k$

Todos anulables

if Y_1 es anulable

$First(x) = First(x) \cup First(Y_2)$

if Y_1 y Y_2 son anulables

$First(x) = First(x) \cup First(Y_3)$

Follow ← Sólo Terminales y no contiene ε

a) Si es el símbolo inicial

$Follow(A) = Follow(A) \cup Follow\{\$\}$

b) if $B \to \alpha A \beta$ ← Estamos calculando a "A"

$Follow(A) = Follow(A) \cup First(\beta)$ ← Puede estar formada de $Y_1 Y_2 \dots$

c) if $B \to \alpha A$ o ($B \to \alpha A \beta$ y β es anulable)

$Follow(A) = Follow(A) \cup Follow(B)$

Ejemplo

$S \to ABC$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$
$C \to c \mid \varepsilon$

	Anulable	FIRST a	b	c	ε	FOLLOW a	b	c	$
S	✓	✓	✓	✓	✓				✓
A	✓	✓			✓	✓	✓	✓	
B	✓		✓		✓				
C	✓			✓	✓				

* S tiene producciones anulables ∴ Anulable
* S no aparece en alguna producción

- First(BC) = {b, c, ε}
  - Anulable ∴ Cumple "C"
  - ∴ Fin Archivo → Follow $

---

① Ejercicio

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid id$

	Anulables	FIRST +	*	id	(	)	ε	FOLLOW +	*	id	(	)	$
E				✓	✓							✓	✓
E'	✓	✓					✓					✓	✓
T				✓	✓			✓				✓	✓
T'	✓		✓				✓	✓				✓	✓
F				✓	✓			✓	✓			✓	✓

FOLLOW
E = $
E' → Agregamos todo el de E
T → Follow ∪ E' ∴ +
Pero como es anulable agregamos Follow(E)
Segundo parte (E') → + T E' ya lo agregamos
(→ Follow(E')

segunda parte (E) → T ε

↳ Hacemos Follow (E')

T' → Follow (T)

Segunda parte Follow (T)

F → First (T')

Follow (T)

// Segunda parte

Follow (T')

→ Tabla de análisis Sintáctico LL(1)

	+	*	id	(	)	$
E			E→TE'	E→TE'		
E'	E'→+TE'				E'→ε	E'→ε
T			T→FT'	T→FT'		
T'	T'→ε	T'→*FT'			T'→ε	T'→ε
F			F→id	F→(E)		

E → T E'
E' → +T E' | ε
T → F T'
T' → *F T' | ε
F → (E) | id

• Tomaremos producción a producción (Tiene 8)

• E → T E'

First(TE') = { id, (}

• E' → +TE'

First(+TE') = { + }

• E' → ε

First(ε) = { ε } ∴ Follow(E') = {), $ }

First(ε) = {ε} ∴ Follow(E') = {), $}

- T → FT'

  First(FT') // F no es anulable ∴ Comenzamos con F

  ↳ = {id, (}

- T' → *FT'

  First(*FT') = {*}

- T' → ε

  First(ε) = {ε} ∴ Follow(T') = {+,), $}

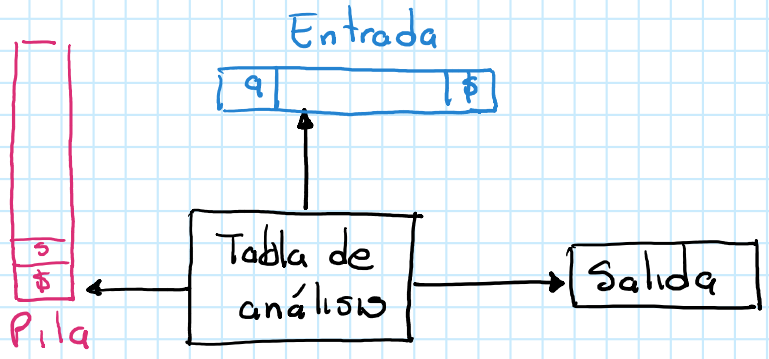
- F → (E)

  First((E)) = {(}

- F → id

  First(id) = {id}

Análisis Sintácticos



Pila	Entrada	Acción
E $	id + id * id $	E → TE'

Top stack

// Hacemos pop

E $	id + id * id $	E → TE'
// Hacemos pop		
TE' $	id + id * id $	T → FT'
FT'E' $	id + id * id $	F → id
idT'E' $	id + id * id $	avanzar
// Hacemos pop otack y llamas yylex		
T'E' $	+ id * id $	T' → ε
E' $	+ id * id $	E' → + TE'
+TE' $	+ id * id $	avanzar
TE' $	id * id $	T → FT'
FT'E' $	id * id $	F → id
idT'E' $	id * id $	avanzar
T'E' $	* id $	T' → * FT'
*FT'E' $	* id $	avanzar
FT'E' $	id $	F → id
idT'E' $	id $	avanzar
T'E' $	$	T' → ε
E' $	$	E' → ε
$	$	ACEPTAR //

Top Stack

Pseudo-Código

Construir la Tabla

```
for each (producción A → α ε G ){
    for each (a ∈ First(α)) and (a ≠ ε){
        M[A, a] = A → α
    }
```

```
if (ε ∈ First(α)) Then {
    for each (b ∈ Follow(A)) {
        M[A, b] = A → α    // M es la tabla
    }
}
}
```

Análisis Sintáctico

$ ≜ Fin archivo

X: es el apuntador TOP STACK
t: es el token actual
M: es la tabla de análisis

```
while (X ≠ $) do {
    if (M[X, t] = X → Y1 Y2 ... Yn) {
        pop()
        n = |Y1 Y2 ... Yn|  // Longitud cadena
        for (i = n, 1, -1)
            push(Yi)
    }
    else if (M[X, t] = error) {
        error()  // Función errores sintácticos
    }
    else if (X ∈ Σ and X = t) {  // X ≜ Terminal
        pop()
        yylex()
    }
    else  // X ∈ Σ and X ≠ t {
        error()
    }
}
if (X = $ and X = t) {
    aceptar()
```

```
if (x = $ and x = ε){
    aceptar ()
}
else {
    error ()
}
```

▸ **Ejercicio**

A → BC
B → aD | ε
C → bE | ε
D → c
E → d

	Anulable	FIRST a	b	c	d	ε	FOLLOW a	b	c	d	$
A	✓	✓				✓					✓
B	✓	✓				✓			✓		✓
C	✓		✓			✓				✓	✓
D				✓					✓		✓
E					✓					✓	✓

// TABLA DE ANÁLISIS SINTÁCTICO

	a	b	c	d	$
A					
B					
C					
D					
E					

- A → B C

  First (BC) = {

- B → aD | ε

  - First (aD) = { a }
  - First (ε) = { ε }

  ∴ Follow (ε) = {

  // Tomamos (A) → B C

2) Análisis Sintáctico

Wednesday, October 16, 2019 1:48 PM

Pila	Entrada	Acción	
$0	()()()$	r2	r(S → ε)
$01	()()()$	d2	
$012	)()()$	r2	r(S → ε)
$0123	)()()$	d5	
$01235	)()()$	r2	
$012356	)()()$	d7	
$0123567	)()()$	r1	r(S → S(S))
$0123	)()()$	d5	
$01235	)()()$	r2	
$012356	)()()$	d7	
$0123565	)()()$	r1	r(S → S(S))
$0123	)()$	d4	
$01234	)()$	r1	r(S → S(S))
$01	)()$	d2	
$012	)$	r2	
$0123	)$	d4	
$01234	$	r1	r(S → S(S))
$01	$	Aceptar	

3) LL(1) - LR(1)

miércoles, 9 de octubre de 2019 12:40 p. m.

1. Si $A \to \alpha_1 | \alpha_2 | \dots | \alpha_n$ entonces

$First(\alpha_1) \cap First(\alpha_2) \cap First(\alpha_3) \dots First(\alpha_n) = \phi$

2. Si $\varepsilon \in First(A)$ entonces

$First(A) \cap Follow(A) = \phi$

// Algoritmos

a) $A \to \alpha \cdot a\beta, b$ $goto(I_i, a) = I_j$

Acción $[i, a]$ = desplazar j

b) $A \to \alpha \cdot, a$

Acción $[i, a]$ = reducir $(A \to \alpha)$

// Cuando está al final de la producción del α

Ejercicio

Análisis Sintáctico LL(1)

	a	(	)	$
A	A → a	A → (A)		

Pila	Entrada	Acción	
A $	((a))$	A → (A)	
(A)$	((a))$	avanzar	
→ A)$	(a))$	A → (A)	→ "Pop"
(A))$	a))$	avanzar	
→ A))$	a))$	A → a	

a)) $	a)) $	avanzar
)) $	)) $	avanzar
) $	) $	avanzar
$	$	aceptar

// Con la tabla de la actividad (Hoja siguiente)

(c) Elabore la tabla de análisis LR(1)

Edo	ACCIÓN				GOTO
	a	(	)	$	A
0	d2	d3			1
1				Acepta	
2				r2	
3	d5	d6			4
4			d7		
5			r2		
6	d5	d6			8
7				r1	
8			d9		
9			r1		

Pila	Entrada	Acción
$0	((a)) $	d3
$03	(a)) $	d6
$036	a)) $	d5
$0365	)) $	r(A → a)
$036 8	)) $	d9
$03689	) $	r(A → (A))
$034	) $	d7
$0347	$	r(A → (A))
$01	$	aceptar

Lo

- Desplazar = Push Pila = Recorrer apuntador entrada

→ "Pop pila"

0) A' → A
1) A → (A)
2) A → a

NOTA

① Tipo de análisis sintáctico LR(1)

② Bison → LALR

Ejercicio

	a	(	)	$
A	A → a	A → (A)		

Pila	Entrada	Acción
A$	(((a))$	A → (A)
(A)$	((a))$	avanzar
A)$	(a))$	A → (A)
(A))$	a))$	avanzar
A))$	))$	A → a
a))$	a))$	avanzar
))$	))$	avanzar
)$	)$	avanzar
	$	error

(c) Elabore la tabla de análisis LR(1)

Edo	ACCIÓN				GOTO
	a	(	)	$	A
0	d2	d3			1
1				Acepta	
2				r2	
3	d5	d6			4
4			d7		
5			r2		
6	d5	d6			8
7				r1	
8			d9		
9			r1		

Pila	Entrada	Acción
$0	((a))$	d3
$03	((a))$	d6
$036	(a))$	d6
$0366	a))$	d5
$03665	))$	r(A → a)
$03668	))$	d9
$036689	)$	r(A → (A))
$0368	)$	d9
$03689	$	error

d?

- Desplazar
  = Push Pila
  = Recorer apuntador entrada

→ "Pop pila"

0) A' → A
1) A → (A)
2) A → a

// Código

```
void A(){
    if (token == '(' ){
        eat('(');
        A();
        eat(')');
    }else if (token == 'a'){
        eat('a');
```

```
} else if (token == 'a')
    eat('a');
  else {
    error();
  }
```

---

① Obtener tabla LL(1) y LR(1)

$A \to A\alpha \mid \beta$

- $S \to S(S) \mid \varepsilon$ | $S' \to S$; $S \to (S)S \mid \varepsilon$

$\therefore S \to (S)S \mid \varepsilon$

// First y Follow

SIMBOLOS	Anulable	(	)	ε	(	)	$
S	✓	✓		✓		✓	✓

// Tabla de análisis LL(1)

SIMBOLOS	(	)	$
S	$S \to (S)S$	$S \to \varepsilon$	$S \to \varepsilon$

$S \to (S)S \mid \varepsilon$

- $S \to (S)S$

  $First((S)S) = \{ (\}$

  $S \to \varepsilon$

  $First(\varepsilon) = \{\varepsilon\}$

  $\therefore Follow(S) = \{), \$ \}$

// Tabla LR(1)

// Parte de goto

// Tomar la aumentada

$S' \to S$

$S \to (S)S' \mid \varepsilon$

cerradura$(S' \to S) = \{ S' \to \cdot S, \$ \quad First(\$) = \{\$\}$

$S \to \cdot (S)S', \$$

$S \to \cdot, \$ \} = I_0$

$goto(I_0, S) = \{ S \cdot, \$ \} = I_1$

$goto(I_0, ( ) = \{ S \to (\cdot S)S, \$ \quad First(( ) = \{ (\}$

$S \to \cdot (S)S,)$

$S \to \cdot, \$) \} = I_2$

$goto(I_2, S) = \{ (S \cdot)S', \$ \} = I_3$

$goto(I_2, ( ) = \{ S \to (\cdot S)S,) \quad First(( ) = \{ (\}$

$S \to \cdot (S)S,$

$S \to \cdot,) \} = I_4$

$goto(I_3,)) = \{ S \to (S) \cdot S, \$$

$S \to \cdot (S)S, \$$

$S \to \cdot, \$ \} = I_5$

$goto(I_4, S) = \{ S \to (S \cdot)S,) \} = I_6$

$goto(I_4, ( ) = \{ S \to (\cdot S)S,) ¿?$

$S \to \cdot (S)S,)$

$S \to \cdot,) \} = I_4$

$goto(I_5, S) = \{ S \to (S)S \cdot, \$ \} = I_7$

$goto(I_5, ( ) = \{ \ldots \} = I_2$

$goto(I_6,)) = \{ S \to (S) \cdot S,) \quad First( )) = \{) \}$

$S \to \cdot (S)S,)$

goto(I_6,) = { S → (S)·S ,) First(1)={)}
S → ·(S)S ,)
S → · ,) } = I_8

goto(I_8, S) = { S → (S)S· ,) } = I_9

goto(I_8, ( ) = {...} = I_2 I_4

// Tabla LR(1)

EDO	(	)	$	Goto S
0	d2			1
1			Aceptar	
2	d4			3
3		d5		
4	d4			6
5	d2			7
6		d8		
7			r_1	
8	~~d2~~ d4			9
9		r_1		

0) S' → S
1) S → (S)S | ε

// Hacer "cerradura" y LR

S' → S
S → S(S) | ε

cerradura(S' → S) = { S' → ·S , $
S → ·S(S), $/(// First(() = {(}
S → · , $/(} = I_0

goto(I_0, S) = { S' → S·, $ Aceptor
S → S·(S), $/(} = I_1

goto(I_1, ( ) = { S → S(·S), $/) // Preguntar

goto(I_1, c) = { S → S (· S), $/) //Preguntar
S → · S(S),)/c
S → · ,)/c } = I_2

reducir

goto(I_2, S) = { S → S (S ·), $/c d?
S → S · (S),)/c } = I_3

goto(I_3,)) = { S → S (S) · , $/X c } = I_4

goto(I_4, () = { S → S (· S),)/c
S → · S (S),)/c
S → · ,)/c } = I_5

reducir

goto(I_5, S) = { S → S (S ·),)/c
S → S · (S),)/c } = I_6

goto(I_6,)) = { S → S (S) · ,)/c } = I_7

goto(I_6, c) = { ... } = I_5

// TABLA LR(1)

1) S → (S)
2) S → ε

EDO	c	)	$	Goto S
0	r2		r2	1
1	d2		Aceptar	
2	r2	r2		3
3	d5	d4		
4	r1		r1	
5	r2	r2		6
6	d5	d7		
7	r2	r2		

3) Syntactic Analysis Up | Cerradura - LR(1)

lunes, 7 de octubre de 2019 01:16 p. m.

- Recursiva por Izquierda

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

- Recursiva por la derecha

$E \to T+E \mid T$
$T \to F*T \mid F$
$F \to (E) \mid id$

// No afecta la recursividad pero sí afecta la ambigüedad

NOTA: Podría afectar si no está factorizada
↳ // Si sólo es uno, no se factoriza

// Ejercicio (1)

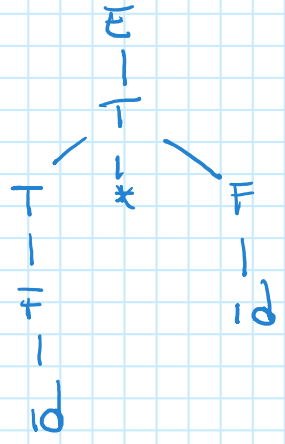
$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

// Cadena entrada
$\omega =$ "id * id"

- Derivar
- Reducir

$E \Rightarrow T \Rightarrow T*F \Rightarrow$
$\Rightarrow T*id \Rightarrow F*id \Rightarrow id*id$

// Árbol



// Ejercicio (2)

// Forma de "Elementos punteados"

G' { $S' \to S$; $S \to AB$; $A \to a \mid \varepsilon$; $B \to b \mid \varepsilon$ }

// Forma de "Elementos punteados"

$A \to \alpha \cdot B \beta, a$

Cerradura ({ $S' \to \cdot S, \$$ })

$= \{ S' \to \cdot S, \$$ — First($\$$) = { $\$$ } — $\varepsilon \$ = \$$

$S \to \cdot AB, \$$ — First($B\$$) = { $b, \$$ }

$A \to \cdot a, b/\$$

$A \to \cdot, b/\$ \} = I_0$

// Es para omitir escribir 2 veces — $A \to \cdot a, b$ y $A \to \cdot a, \$$ } $A \to \cdot a, b/\$$

// No ponemos "ε"

goto $(I_0, S) = \{ S' \to S \cdot, \$ \} = I_1$

goto $(I_0, A) = \{ S \to A \cdot B, \$$ — First($\$$) = { $\$$ }

// Cerradura $B \to \cdot b, \$$

$B \to \cdot, \$ \} = I_2$

goto $(I_0, a) = \{ A \to a \cdot, b/\$ \} = I_3$

// I_0 se acabó, pasamos a I_1, pero este tiene fin de archivo

goto $(I_2, B) = \{ S \to AB \cdot, \$ \} = I_4$

goto $(I_2, b) = \{ B \to b \cdot, \$ \} = I_5$

goto (I_2, b) = { B → b·, $ } = I_5

//Ejercicio (3)

S → SS+ | SS* | a

↓

S' → S
S → SS+ | SS* | a

// Podemos omitir recursividad

S → a S'
S' → ε | S + S' | S * S'

Cerradura ({ S' → ·S, $ })

= { S' → ·S, $

First($) = { $ }

S → ·SS+, $/a

First(S+ $) = { a } //Agregamos "/a"

S → ·SS*, $/a

First(S* $) = { a }
First(S* a) = { a }

S → ·a, $/a# } = I_0

se aplicaría first pero da igual

goto (I_0, S) = { S' → S·, $

S → S·S+, $/a

First(+ $) = { + }
First(+ a) = { + }

S → S·S*, $/a

First(* $) = { * }

S → ·SS+, +/*/a

First(* a) = { * }

S → ·SS*, +/*/a

S → ·a, +/*/a } = I_1

goto (I_0, a) = { S → a·, $/a } = I_2

$goto(I_0, a) = \{ S \to a \cdot, \$/a \} = I_2$

$goto(I_1, S) = \{ S \to SS \cdot +, \$/a$ } // Terminal ∴ No necesita First
$S \to SS \cdot *, \$/a$
$S \to S \cdot S+, +/*/a$
$S \to S \cdot S*, +/*/a$

$S \to \cdot SS+, +/*/a$
$S \to \cdot SS*, +/*/a$
$S \to \cdot a, +/*/a$ $\} = I_3$

$goto(I_1, a) = \{ S \to a \cdot, +/*/a \} = I_4$

$goto(I_3, +) = \{ S \to SS+ \cdot, \$/a \} = I_5$

$goto(I_3, *) = \{ S \to SS* \cdot, \$/a \} = I_6$

$goto(I_3, a) = I_4$

$goto(I_3, S) = \{ S \to SS \cdot +, +/*/a$
$S \to SS \cdot *, +/*/a$
$S \to S \cdot S+, +/*/a$
$S \to S \cdot S*, +/*/a$
$S \to \cdot SS+, +/*/a$
$S \to \cdot SS*, +/*/a$
$S \to \cdot a, +/*/a$ $\} = I_7$

- $goto(I_7, +) = \{ S \to SS+ \cdot, +/*/a \} = I_8$
- $goto(I_7, *) = \{ S \to SS* \cdot, +/*/a \} = I_9$

$goto(I_7, a) = I_4$

$goto(I_7, S) = I_7$

// Ejercicio (4)

$S' \to S$
$S \to +SS \mid *SS \mid a$

cerradura ({S' → ·S, $}) = { S' → ·S, $
S → · +SS, $
S → · *SS, $
S → · a, $ } = I_0

// No hay first de alguno ya que tenemos los terminales

goto (I_0, S) = {S → S'·, $} = I_1

goto(I_0, +) = {S → +·SS, $ First (S $) = {+, *, a}
S → · +SS, +/*/a
S → · *SS, +/*/a
S → · a, +/*/a } = I_2

goto(I_0, *) = { S → *·SS, $ First (S $) = {+, *, a}
S → · +SS, +/*/a
S → · *SS, +/*/a
S → · a, +/*/a } = I_3

goto(I_0, a) = {a·, $} = I_4

// TABLA LR(1)

① S' → S·, $ // En el estado de Fin de archivo | 2.c

② S_1 no tiene nada
↳ A → α·, a | REDUCIR | 2.b

3) Ejercicios | Cerradura - Tabla LR(1)

jueves, 10 de octubre de 2019 08:25 p. m.

$A' \to A$

$A \to (A) \mid a$

cerradura($\{ A' \to \cdot A, \$ \} = \{ A' \to \cdot A, \$$
$A \to \cdot (A), \$$
$A \to \cdot a, \$ \} = I_0$

goto $(I_0, A) = \{ A' \to A \cdot, \$ \} = I_1$

goto $(I_0, a) = \{ A \to a \cdot, \$ \} = I_2$

goto $(I_0, ($ $) = \{ A \to (\cdot A), \$$ // First( ) $\$) = \{) \}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_3$

goto $(I_3, A) = \{ A \to (A \cdot),) \} = I_4$

goto $(I_3, a) = \{ A \to a \cdot,) \} = I_5$

goto $(I_3, ($ $) = \{ A \to (\cdot A),)$ // First( )) $= \{) \}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_6$

goto $(I_4,)$ $) = \{ A \to (A) \cdot, \$ \} = I_7$

goto $(I_6, A) = \{ A \to (A \cdot),) \} = I_8$

goto $(I_6, a) = \{ A \to a \cdot,) \} = I_5$

goto $(I_6, ($ $) = \{ \ldots \} = I_6$

goto $(I_8,)$ $) = \{ A \to (A) \cdot,) \} = I_9$

(c) Elabore la tabla de análisis LR(1)

Edo	ACCIÓN				GOTO
	a	(	)	$	A
0	d2	d3			1
1				Acepta	
2				r2	
3	d5	d6			4
4			d7		
5			r2		
6	d	d6			8
7				r1	
8			d9		
9			r1		

Aceptación (Forma) $S' \rightarrow S., \$$

//Reglas

0) $A' \rightarrow A$

1) $A \rightarrow (A)$

2) $A \rightarrow a$

REPASO 1

Sunday, October 20, 2019 6:33 PM

Temas examen

a) Derivaciones { Derecha / Izquierda
b) Árboles derivación
c) Eliminar ambigüedad
d) Notación EBNF
e) Diagramas de sintaxis

Teoría Análisis Sintáctico

- Existen 3 tipos de análisis sintáctico
  1) Universal → Aplica a cualquier gramática
     → Ej lento
  2) Descendentes → Va de raíz a hojas
     → Derivaciones a la izquierda
     → Tipos → Recursivo
       → LL(k)
  3) Ascendente → SLR
     → LR
     → LALR (Bison)
- Recuperación errores
  a) Modo panico
  b) Gramáticas con producciones de error
  c) Correcciones → Por frase
     → De forma global

Derivación

// Derivación por izquierda

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to * FT' \mid \varepsilon$
$F \to id \mid (E)$

- Cadena de entrada
  $\omega = id + (id * id)$

$E \Rightarrow \underline{T}E' \Rightarrow FT'E' \Rightarrow idT'E' \Rightarrow id\,\varepsilon E' \Rightarrow$
$\Rightarrow id + TE' \Rightarrow id + FT'E'$ $id + (E)T'E' \Rightarrow$
$\Rightarrow id + (TE')T'E' \Rightarrow id + (FT'E')T'E' \Rightarrow$
$\Rightarrow id + (idT'E')T'E' \Rightarrow id + (id * FT'E')T'E' \Rightarrow$
$\Rightarrow id + (id * idT'E')T'E' \Rightarrow id + (id * id\,\varepsilon\,\varepsilon)\,\varepsilon\,\varepsilon$ //

// Derivación por la derecha

$E \Rightarrow T\underline{E'} \Rightarrow T + TE' \Rightarrow T + T\,\varepsilon \Rightarrow T + FT'$

Formulario

- Derivación Izquierda: $E \to \underline{E}T$
- Derivación Derecha: $E \to E\underline{T}$
- Factorizar Derecha: $A \to \alpha A \mid \beta$
- Factorizar Izquierda: $A \to A\alpha \mid \beta$
- EBNF (Derecha): $A \to \{\alpha\}\beta$
- EBNF (Izq): $A \to \beta\{\alpha\}$

○ ≙ Terminal □ ≙ No terminal

$\Rightarrow T + F \not{\varepsilon} \Rightarrow T + (E) \Rightarrow T + (T\,E') \Rightarrow$

$T + (T \not{\varepsilon}) \Rightarrow T + (F\,T') \Rightarrow T + (F * F\,T') \Rightarrow T + (F * F \not{\varepsilon})$

$\Rightarrow T + (F * id) \Rightarrow T + (id * id) \Rightarrow F\,T' + (id * id)$

$\Rightarrow F \not{\varepsilon} + (id * id) \Rightarrow id + (id * id)$ //

Gramáticas Ambiguas y Arboles Sintacticos

- Gramática es ambigua si tenemos diferentes árboles para una misma cadena (Recursivo por derecha y por izquierda)
- Es ambigua si contiene alguna sentencia ambigua

// Eliminar Recursividad

1) Ordenar menor to mayor jerarquía de operadores
   1] + -
   2] * / %
   3] ( )
2) Asignar a cada nivel un NO TERMINAL
3) Considere operadores binarios

E + -	$E \to E+T \mid E-T \mid T$
T * / %	$T \to T*F \mid T/F \mid T\%F \mid F$
F ( )	$F \to (E) \mid num \mid id$

Restantes

$A \to A\alpha \mid \beta$ Izquierda

Derecha

$E \to T+E \mid T-E \mid T$

$T \to F*T \mid F/T \mid F\%T \mid T$

$F \to (E) \mid num \mid id$

Restantes

$A \to \alpha A \mid \beta$ Derecha

// Ejercicios

① $S \to SSS' \mid a$; $S' \to + \mid *$

// Eliminar recursividad izquierda

$S \to aT$

$T \to SS'T \mid \varepsilon$

$S' \to + \mid *$

② $S \to SS+ \mid SS* \mid a$

// Eliminar R. izquierda

$S \to aT$

$T \to S+T \mid S*T$

③ $E \to E^+ \mid E^* \mid E? \mid E|E$

|	E
Concatenación	A
+ * ?	B
( )	C

Parte de ejercicios

① $S \to SSS' \mid a$ (α = SS', β = a)

$A \to A\alpha \mid \beta$ // Recursiva Izquierda

$S \to aT$

$T \to SS'T \mid \varepsilon$

② $E \to E^+ \mid E^* \mid E? \mid E|E$

// Jerarquía

// Recursivo por derecha

$A \to \alpha A \mid \beta$

$E \to A|E \mid A$

$A \to BA \mid B$

$B \to B^+ \mid B^* \mid B? \mid C$

$C \to (E) \mid a \mid b$

// Recursivo por Izq

$A \to A\alpha \mid \beta$

$E \to E|A \mid A$

$A \to AB \mid B$

$B \to B^* \mid B^+ \mid B? \mid C$

$C \to (a) \mid a \mid b$

③ Obtener EBNF

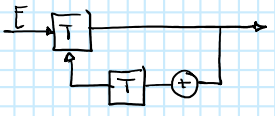
Derecha $A \to \{\alpha\}\beta$; Izq $A \to \beta\{\alpha\}$

// Se tomó la derecha y se uso EBNF Izq

- $E \to A|E \mid A$ (α = $|E$, β = A) ; $E \to A\{|E\}$
- $A \to BA \mid B$ (α = A, β = B) ; $A \to B\{A\}$

④ EBNF y Diagrama sintáctico

- $E \to T\{+T\}$



- $S \to S|T \mid T$; S

③ $E \to E^+ | E^* | E? | E|E |$

1 concatenación	E
* + ?	C
( )	O
	P

// Por la derecha $(A \to \alpha A | \beta)$

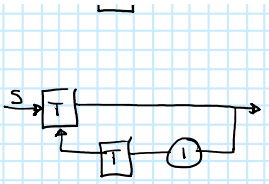
$E \to C\ "|"\ E \mid C$
$C \to OC \mid O$
$O \to O^+ \mid O^* \mid O? \mid P$
$P \to (E) \mid a \mid b$

// Por la izquierda $(A \to A\alpha \mid \beta)$

$E \to E\ "|"\ C \mid C$
$C \to CO \mid O$
$O \to O^+ \mid O^* \mid O? \mid P$
$P \to (E) \mid a \mid b$

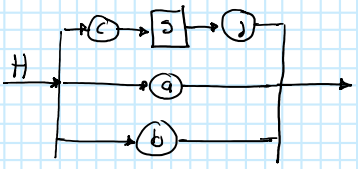
- $S \to S\ ;\ T \mid T$ (α = $;\ T$, β = T)

$\therefore S \to T\{;T\}$



- $H \to (S) \mid a \mid b$

$\therefore H \to (S) \mid a \mid b$



▸ Notación EBNF

- Recursiva por Derecha: $A \to \alpha A \mid \beta$
  - $A \to \{\alpha\}\beta$
- Recursiva por Izquierda: $A \to A\alpha \mid \beta$
  - $A \to \beta\{\alpha\}$

// Ejercicios

① Por la izquierda

$D \to C\ "|"\ D \mid C$ (α = $"|"\ D$, β = C)

$D \to C\{"|"D\}$

¿? $C \to O\ C \mid O$ (α = O, β = O)

$C \to O\{O\}$

$O \to O^* \mid O^+ \mid O? \mid P$ (α = O^*, O^+, $O?$; β = P)

$O \to P\{* \mid + \mid ?\}$

$P \to (D) \mid a \mid b$

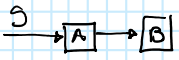
$P \to (D) \mid a \mid b$

▸ Diagrama de Sintaxis (EBNF)

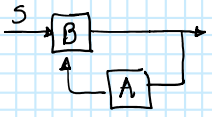
□ ≙ No terminal

○ ≙ Terminal

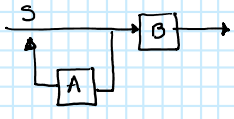
- $S \to AB$



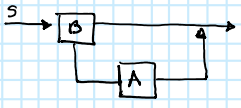
- $S \to B\{A\}$ //Izquierda



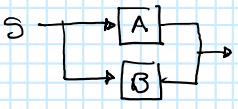
- $S \to \{A\}B$ // Derecha



- $S \to B[A]$

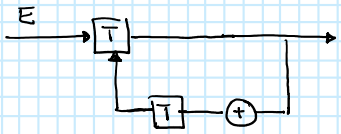


- $S \to A \mid B$

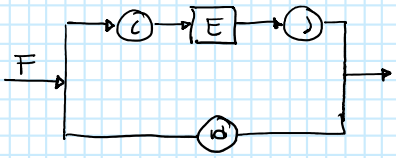


Ejercicios

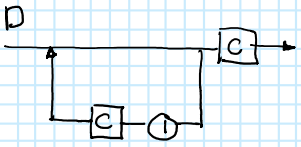
① $E \rightarrow T\{+T\}$



② $F \rightarrow (E) \mid id$



③ $D \rightarrow \{C\ "1"\}C$

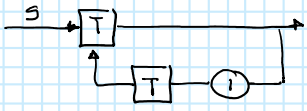


Ejercicios EBNF y Diagramas Sintácticos

① $A \rightarrow A\alpha \mid \beta$

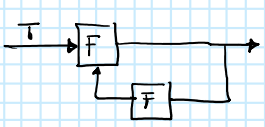
- $S \rightarrow S\ |\ T \mid T$ (α: $|\ T$, β: T)

  $\therefore S \rightarrow T\{|\ T\}$



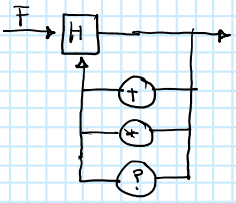
- $T \rightarrow T\ F \mid F$ (α: F, β: F)

  $\therefore T \rightarrow F\{F\}$



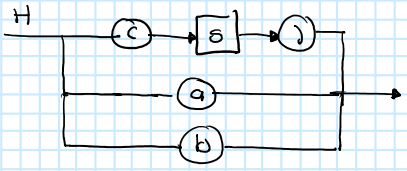
- $F \rightarrow F* \mid F+ \mid F? \mid H$ (α: $*$, $+$, $?$; β: H)

  $\therefore F \rightarrow H\{*\mid+\mid?\}$



- $H \rightarrow (S) \mid a \mid b$

  $\therefore H \rightarrow (S) \mid a \mid b$



REPASO 2

Monday, October 21, 2019 10:06 AM

Temas: ① Análisis sintáctico Recursivo
② Análisis Sintáctico LL(1)
2.1] Eliminar Recursividad Izquierda
2.2] Factorizar
Checar Follow → 2.3] First / Follow
2.4] Tabla LL(1)
2.5] Análisis LL(1)

Introducción

- El análisis sintáctico decendente
  ① No puede ser recursivo por izquierda
  ② No puede ser ambigüo
- Existen del tipo
  ① Recursivo ② LL(k)

Eliminación Recursividad izquierda

- Reglas: $A \to A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_n$
- Eliminación:
$$A \to \beta_1 A' | \beta_2 A' | \dots | \beta_n A'$$
$$A' \to \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A' | \varepsilon$$

// Ejemplos

$E \to E+T | T$ (α: $+T$, β: T)

$$E \to T\,E'$$
$$E' \to +T\,E' | \varepsilon$$

$T \to T*F | F$

$$T \to F\,T'$$

$T \to T * F / F$; $T \to F T'$, $T' \to * F T' / \varepsilon$

$F \to (E) / id$; $F \to (E) / id$

Factorización

- Regla: $A \to \alpha \beta_1 / \alpha \beta_2 / \ldots / \alpha \beta_n$
- Factorización: $A \to \alpha A' / \gamma_1 / \gamma_2 / \ldots / \gamma_n$, $A \to \beta_1 / \beta_2 / \ldots / \beta_n$

// Ejemplo

$E \to E + T / T$; $E \to T E'$, $E' \to E + / \varepsilon$

$T \to T * F / F$; $T \to F T'$, $T' \to T * / \varepsilon$

$F \to (E) / id$

// Ejercicios

// Recursividad

$D \to D T L ; / \varepsilon$; $D \to \not{\varepsilon} D'$, $D' \to D T L ; D' / \varepsilon$

$L \to id , L / id$; $L \to id L'$, $L' \to , L' / \varepsilon$

② Recursividad y Factorización

// Factorización

$S \to SS \mid if(B)S \mid if(B)S\,else\,S \mid while(B)S \mid \{S\} \mid id = E;$

(α: $if(B)S$, α: $if(B)S$, β: $else\,S$)

$S \to SS \mid if(B)S\,S' \mid while(B)S \mid \{S\} \mid id = E;$

(α: S; β: $if(B)S\,S'$, $while(B)S$, $\{S\}$, $id = E;$)

$S' \to else\,S \mid \varepsilon$

omitimos → directo

// Recursividad Izquierda

$A \to A\alpha \mid \beta$

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

$S \to if(B)SS' \mid while(B)SS'' \mid \{S\}S'' \mid id = E;S''$

$S'' \to SS'' \mid \varepsilon$

$S' \to else\,S \mid \varepsilon$

First / Follow

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid id$

Anulables	First: +	First: *	First: (	First:)	First: id	Follow: ε	Follow: +	Follow: *	Follow: id	Follow: (	Follow:)	Follow: $
E			✓	✓							✓	✓
E'	✓		✓		✓						✓	✓
T			✓	✓			✓				✓	✓
T'	✓		✓		✓		✓				✓	✓
F			✓	✓			✓	✓			✓	✓

Follow(T) | First(E') = { +, ε }

Follow(T) = { E' }

Follow(E) = {), $ }

Follow(E') = { Follow(E) = {), $ } }

Follow(T') | Follow(T) = { +,), $ }

Follow(F) | First(T') { *, ε }
| Follow(T) { +,), $ }
| Follow(T') { ¿? }

Tabla de análisis LL(1)

	+	*	id	(	)	$
E			E → TE'	E → TE'		
E'	E' → +TE'				E' → ε	E' → ε
T						
T'						
F						

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | id

// 8 producciones

- E → TE'
  First(TE') = { (, id }
- E' → +TE'
  First(+TE') = { + }
- E' → ε
  First(ε) = { ε } ∴ Follow(E') = {), $ }

// Basarse en First and Follow

Propiedades → Gramática LL(1)

① Si A → α1 | α2 | ... | αn
then

① Si $A \to \alpha_1 | \alpha_2 | \dots | \alpha_n$
then
$First(\alpha_1) \cap First(\alpha_2) \dots \cap First(\alpha_n) = \Phi$

② Si $\varepsilon \in$ al First de A
then
$First(A) \cap Follow(A) = \Phi$

REPASO 2

Wednesday, October 23, 2019 11:32 AM

$S \to SS+ \mid SS* \mid a$

$A \to A\alpha \mid \beta$; $A \to \beta A'$; $A' \to \alpha A' \mid \varepsilon$

- Factorizar a la izquierda

$S \to SSS' \mid a$

$S' \to + \mid *$

- Recursividad izquierda

$S \to S\,SS' \mid a$

$S \to aT$

$T \to SS'T \mid \varepsilon$

$S' \to + \mid *$

- First / Follow

Símb	Anulable	First +	First *	First a	First ε	Follow +	Follow *	Follow a	Follow $
S				✓		✓	✓	✓	✓
T	✓			✓		✓	✓	✓	✓
S'		✓	✓			✓	✓	✓	✓

Follow(S) = { First(S') = {+, *} y $ por ser inicial }

Follow(T) = { Follow(S) = {+, *, $} }

Follow(S') = { First(T) = {a} ; Follow(T) = {+, *, $} }

- Probar que es LL(1)

→ T cumple con 1ra propiedad

T no tiene ε ∴ Cumple con 2° propiedad

T no tiene ε ∴ Cumple con 2° propiedad

→ S' First(+) ∩ First(*) = $\emptyset$ ∴ cumple

S' no tiene ε ∴ Cumple

→ T First(SS'T) ∩ First(ε) = $\emptyset$ ∴ cumple

T tiene ε ∴ Pero igual cumple la propiedad
First(A) ∩ Follow(A)

- Tabla de análisis LL(1)

	a	+	*	$
S	S → aT			
T	T → SS'T	T → ε	T → ε	T → ε
S'		S' → +	S' → *	

S → aT
T → SS'T / ε
S' → + / *

- S → aT
  First(aT) = { a }
- T → SS'T
  First(SS'T) = { a }
- T → ε
  First(ε) = { ε }
  ∴ Follow(T) = { +, *, $ }
- S' → +
  First(+) = { + }
- S' → *
  First(*) = { * }

- $S \to SS+ \mid SS* \mid a$

α α β

$A \to \beta A'$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_3 A'$

// Recursividad

$S \to aS'$

$S' \to S+S' \mid S*S' \mid \varepsilon$

$S \to aT$

$T \to S+T \mid S*T \mid \varepsilon$

α α

// Factorizar

$S \to aT$

$T \to SS' \mid \varepsilon$

$S' \to +T \mid *T$

// First / Follow

	Anulable	First a	First +	First *	First ε	Follow a	Follow +	Follow *	Follow $
S		✓				✓	✓	✓	✓
T	✓	✓			✓	✓	✓	✓	✓
S'			✓	✓		✓	✓	✓	✓

- Follow(S) = { First(S') = { +, * } and $ }

~~Follow(T) = { }~~

- Follow(T) = { Follow(S) = { +, *, $ } }
- Follow(S') = { Follow(T) = { +, *, $ } }

- $S \to SS+ \mid SS* \mid a$

α α β

- $U \to UU+ \mid UU \mid UU* \mid b$

α α α

- $S \to \underbrace{SS}_{\alpha}\underbrace{+}_{\beta} \mid \underbrace{SS}_{\alpha}\underbrace{*}_{\beta} \mid a$

// Factorizar

$S \to SSS' \mid a$

$S' \to + \mid *$

// Recursividad Izq

$S \to \underbrace{SS}_{\alpha}+ \mid \underbrace{SS}_{\alpha}* \mid \underbrace{a}_{\beta}$

$S \to aS'$

$S' \to S+S' \mid S*S' \mid \varepsilon$

- $U \to \underbrace{UU}_{\alpha}+ \mid \underbrace{UU}_{\alpha} \mid \underbrace{UU}_{\alpha}* \mid b$

//Factorizar

$U \to UU \mid b$

$U' \to + \mid * \mid \varepsilon$

//Rec. Izq

Parser

$S \to TS'$
$S' \to +S \mid \varepsilon$
$T \to FT'$
$T' \to T \mid \varepsilon$
$F \to aF'$
$F' \to *F' \mid \varepsilon$

```
void S(){
    T()
    SP()
}

void SP(){
    if(t=="+"){
        eat(+)
        S()
    }
}

void T(){
    F()
    TP()
}

void TP(){
    if(t=='a'){
        T()
    }
}

void F(){
    eat(a)

void FP(){
    if(t==*){
```

```
void F(){
    eat(a)
    FP()
}
```

```
void FP(){
    if (t == *){
        eat(*)
        FP()
    }
}
```

REPASO 3

Wednesday, October 23, 2019 2:21 PM

Temas: Análisis LR

① Autómata LR(1)
② Tabla LR(1)
③ Análisis LR

Método "Elementos Punteados"

// Expandir gramática | $S' \to S$ | $S \to AB$ | $A \to a \mid \varepsilon$ | $B \to b \mid \varepsilon$

cerradura($\{S' \to \cdot S, \$\}$)

$= \{ S' \to \cdot S, \$$ First($\$$) = {$\$$}
$S \to \cdot AB, \$$ First($B\$$) = {$b, \$$}
$A \to \cdot a, b/\$$
$A \to \cdot, b/\$ \} = I_0$

goto$(I_0, S) = \{ S' \to S \cdot, \$ \} = I_1$
goto$(I_0, A) = \{ S \to A \cdot B, \$$ First($\$$) = {$\$$}
$B \to \cdot b, \$$
$B \to \cdot, \$ \} = I_2$
goto$(I_0, a) = \{ A \to a \cdot, b/\$ \} = I_3$

goto$(I_2, B) = \{ S \to AB \cdot, \$ \} = I_4$
goto$(I_2, b) = \{ B \to b \cdot, \$ \} = I_5$

Cerradura y Tabla LR(1)

$A' \to A$
$A \to (A) \mid a$

cerradura$(\{A' \to \cdot A, \$\}) = \{ A' \to \cdot A, \$$
$A \to \cdot (A), \$$
$A \to \cdot a, \$ \} = I_0$

goto$(I_0, A) = \{ A' \to A \cdot, \$ \} = I_1$
goto$(I_0, a) = \{ A \to a \cdot, \$ \} = I_2$
goto$(I_0, ($ $) = \{ A \to (\cdot A), \$$ // First$(\,)) = \{)\}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_3$

goto$(I_3, A) = \{ A \to (A \cdot),) \} = I_4$
goto$(I_3, a) = \{ A \to a \cdot,) \} = I_5$
goto$(I_3, ($ $) = \{ A \to (\cdot A),)$ // First$(\,)) = \{)\}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_6$

// Termina cuando son los mismos o en el próximo sale lo mismo

Ejercicio Previo

$S' \to S$
$S \to AB$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$

cerradura$(S' \to \cdot S, \$) =$
$\{ S' \to \cdot S, \$$ First($\$$) = {$\$$}
$S \to \cdot AB, \$$ First($B\$$) = {b}
$A \to \cdot a, \$, b$
$A \to \cdot, \$, b \} = I_0$

goto$(I_0, S) = \{ S' \to S \cdot, \$ \} = I_1$ // Aceptación
goto$(I_0, a) = \{ A \to a \cdot, \$, b \} = I_2$
goto$(I_0, A) = \{ S \to A \cdot B, \$$ First($\$$) = {$\$$}
$S \to \cdot b, \$$
$S \to \cdot, \$ \} = I_3$
goto$(I_3, B) = \{ S \to AB \cdot, \$ \} = I_4$
goto$(I_3, b) = \{ S \to b \cdot, \$ \} = I_5$

AUTOMATA

$A' \to A$
$A \to (A) \mid a$

Cerradura$(A' \to \cdot A) = \{$
$A' \to \cdot A, \$$ First($\$$) = {$\$$}
$A \to \cdot (A), \$$
$A \to \cdot a, \$ \} = I_0$

goto$(I_0, A) = \{ A' \to A \cdot, \$ \} = I_1$ // Acepta
goto$(I_0, ($ $) = \{ A \to (\cdot A), \$$ First$(A)) = \{)\}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_2$
goto$(I_0, a) = \{ A \to a \cdot, \$ \} = I_3$
goto$(I_2, A) = \{ A \to (A \cdot), \$ \} = I_4$
goto$(I_2, a) = \{ A \to a \cdot,) \} = I_5$
goto$(I_2, ($ $) = \{ A \to (\cdot A),)$ First$(A)) = \{)\}$
$A \to \cdot (A),)$
$A \to \cdot a,) \} = I_6$
goto$(I_4,)) = \{ A \to (A) \cdot, \$ \} = I_7$

// termina cuando son los mismos o en el próximo sale lo mismo

goto(I_4,)) = { A → (A)·, $ } = I_7

goto(I_6, A) = { A → (A·),) } = I_8 | goto(I_6, a) = I_5

goto(I_6, () = {...} = I_6

goto(I_8,)) = { A → (A)·,) } = I_9

A → ·a,) } = I_6

goto(I_4,)) = { A → (A)·, $ } = I_7

goto(I_6, A) = { A → (A·),) } = I_8

goto(I_6, () = I_6

goto(I_6, a) = I_5

goto(I_8,)) = { A → (A)·,) } = I_9

// Tabla LR(1)

Edo	a	(	)	$	GOTO A
0	d2	d3			1
1				Acepta	
2			r2		
3	d5	d6			4
4			d7		
5			r2		
6	d5	d6			8
7				r1	
8			d9		
9			r1		

Acepta el primer fin y conque terminal

r ← Reescribir

A → Se usa el previo terminal

// Ejercicio de Pila

Gramática

0) $A' \to A$

1) $A \to (A)$

2) $A \to a$

PILA	ENTRADA	ACCIÓN
$0	((a)) $	d3
$03	(a)) $	d6
$036	a)) $	d5
$0365	)) $	r2 // r(A → a)
$036 8	)) $	d9
$0368 9	) $	r1 // r → (A)
$034	) $	d7
$0347	$	r1 // r → (A)
$01	$	Aceptar //

un elemento

// Es el que va goto en 6

• TABLA LR(1)

0) $A' \to A$

1) $A \to (A)$

2) $A \to a$

ENTRADA	(	a	)	$	GOTO A
I_0	d2	d3			d1
I_1				Acepta	
I_2	d6	d5			d4
I_3				r2	
I_4			d7		
I_5			r2		
I_6	d6	d5			d8
I_7				r1	
I_8			d9		
I_9			r1		

PILA	ENTRADA	ACCIÓN
$0	((a)) $	d2
$02	(a)) $	d6
$026	a)) $	d5
$0265	)) $	r2 // A → a
$0268	)) $	d9
$02689	) $	r1 // A → (A)
$024	) $	d7
$0247	$	r1 // A → (A)
$01	$	Acepta

Gramática

$S' \to S$

$S \to +SS \mid *SS \mid a$

Cerradura($S' \to \cdot S, \$$) = {

$S \to \cdot +SS, \$$

$S \to *SS, \$$

$S \to a, \$$ } = I_0

goto(I_0, S) = { $S' \to S\cdot, \$$ } = I_1 // Acepta

goto(I_0, +) = { $S \to +\cdot SS, \$$ Firsts($S\$$) = {*, +, a}

$S \to \cdot + SS, *, +, a$
$S \to \cdot * SS, *, +, a$
$S \to \cdot a, *, +, a \} = I_2$

$goto(I_0, *) = \{ S \to * \cdot SS, \$ \quad First(S\$) = \{*, +, a\}$
$S \to \cdot + SS, *, +, a$
$S \to \cdot * SS, *, +, a$
$S \to \cdot a, *, +, a \} = I_3$

$goto(I_0, a) = \{ S \to a \cdot, \$ \} = I_4$

... continua

$S' \to S$

$S \to SS+ \mid SS* \mid a$

$cerradura(S' \to \cdot S) = \{$
$S' \to \cdot S, \$ \quad First(\$) = \{\$\}$
$S \to \cdot SS+, \$/a \quad First(S+\$) = \{a\}$
$S \to \cdot SS*, \$/a$
$S \to \cdot a, \$/a \} = I_0$

$goto(I_0, S) = \{ S' \to S \cdot, \$ \quad First(\$) = \{\$\}$
$S \to S \cdot S+, \$/a \quad First(S+) = \{+\}$
$S \to S \cdot S*, \$/a \quad First(S*) = \{*\}$
$S \to \cdot SS*, +/*/a$
$S \to \cdot SS+, +/*/a$
$S \to \cdot a, +/*/a \} = I_1$

$goto(I_0, a) = \{ S \to a \cdot, \$/a \} = I_2$
$goto(I_1, a) = \{ S \to a \cdot, +/*/a \} = I_3$
$goto(I_1, S) = \{ S \to SS \cdot +, \$/a$
$S \to SS \cdot *, \$/a$
$S \to S \cdot S*, +/*/a \quad First(S*) = \{*\}$
$S \to S \cdot S+, +/*/a \quad First(S+) = \{+\}$
$S \to \cdot SS*, +/*/a$
$S \to \cdot SS+, +/*/a$
$S \to \cdot a, +/*/a \} = I_4$

$goto(I_4, a) = \{ I_3 \}$
$goto(I_4, +) = \{ S \to SS+ \cdot, \$/a \} = I_5$
$goto(I_4, *) = \{ S \to SS* \cdot, \$/a \} = I_6$
$goto(I_4, S) = \{ S \to SS \cdot *, +/*/a$
$S \to SS \cdot +, +/*/a$
$S \to S \cdot S*, +/*/a \quad First(S*) = \{*\}$
$S \to S \cdot S+, +/*/a \quad First(S+) = \{+\}$
$S \to \cdot SS*, +/*/a$
$S \to \cdot SS+, */+/a$
$S \to \cdot a, +/*/a \} = I_7$

$goto(I_7, *) = \{ S \to SS+, +/*/a \} = I_8$
$goto(I_7, +) = \{ S \to SS*, +/*/a \} = I_9$