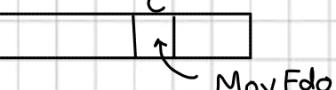


```

    {
        SjAux.Id = NumElemSj++  

        // Transición SjAct
        :
        Q.Add(SjAux);  

        C.Add(SjAux);
    }
    else
    {
        // Poner la transición
        //
        
    }
}
return Afld.Conv;
}

```

SEGUNDO PARCIAL

Análisis Sintáctico

Para llevar a cabo el análisis sintáctico de una cadena σ se hace uso de gramáticas. Únicamente consideraremos gramáticas libres de contexto.

Recordatorio:

Una gramática G es una cuatupla

$$(V_T, V_N, S, \Phi)$$

donde:

V_T : es un conjunto finito no vacío de símbolos llamados símbolos terminales

V_N : Conjunto finito no vacío de símbolos llamados símbolos no terminales

$S \in V_N$: llamado símbolo inicial de la gramática.

Φ : Conjunto de elementos llamados reglas gramaticales.

$$\Phi : V_N \rightarrow (V_N \cup V_T)^*$$

para las gramáticas libres de contexto.

Una gramática G define un lenguaje \mathcal{L}

donde:

$$\mathcal{L} = \{\sigma \mid S \xrightarrow{*} \sigma \text{ y } \sigma \in V_T^*\}$$

Con esto, una cadena σ pertenece al lenguaje definido por una gramática G si:

$$\sigma \in \mathcal{L}(G)$$

Ejemplo: Considerando la gramática $G(V_T, V_N, S, \Phi)$ donde

$$V_T = \{+, -, *, /, (,), \text{num}\}$$

$$V_N = \{E, T, F\}$$

$$S = E$$

$$\begin{aligned} \Phi &= \{E \rightarrow E + T, E \rightarrow E - T, E \rightarrow T, T \rightarrow T * F, T \rightarrow T / F, T \rightarrow F, F \rightarrow (E), F \rightarrow \text{num}\} \\ &= \{E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid T / F \mid F, F \rightarrow (E) \mid \text{num}\} \end{aligned}$$

Algunas cadenas $\sigma \in \mathcal{L}(G)$ son:

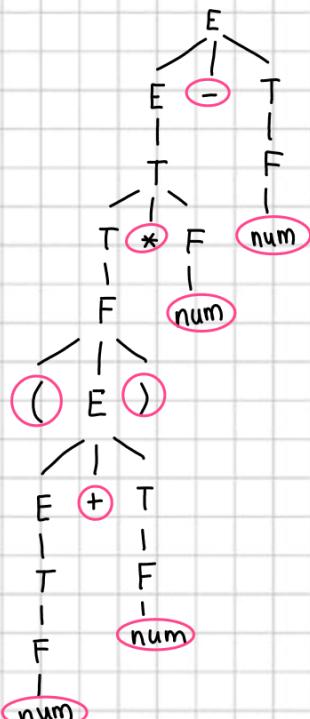
$$\begin{aligned} S &= E \Rightarrow E + T \Rightarrow E - T + T \\ &\Rightarrow T - T + T \Rightarrow F - T + T \\ &\Rightarrow (E) - T + T \\ &\Rightarrow (E + T) - T + T \\ &\Rightarrow (T + T) - T + T \\ &\Rightarrow (F + T) - T + T \\ &\Rightarrow (\text{num} + T * F) - T + T \\ &\Rightarrow (\text{num} + F * F) - T + T \\ &\Rightarrow (\text{num} + \text{num} * F) - T + T \\ &\Rightarrow (\text{num} + \text{num} * \text{num}) - T + T \\ &\stackrel{+}{\Rightarrow} (\text{num} + \text{num} * \text{num}) - \text{num} + \text{num} \end{aligned}$$

$$S = E \Rightarrow E + T \Rightarrow T + T \stackrel{+}{\Rightarrow} \underline{\text{num} + \text{num}}$$

Un analizador sintáctico determina si una cadena σ puede obtenerse por medio de derivaciones sucesivas, partiendo del símbolo inicial de G .

El proceso de derivación se puede representar por medio de un árbol de derivación.

Ejemplo: Determine si $\sigma = (\text{num} + \text{num}) * \text{num} - \text{num} \therefore \sigma \in \mathcal{L}(G)$



Tipos de analizadores sintácticos

> Descendentes: Se busca obtener el árbol de derivación para una cadena σ , a partir del símbolo inicial de G .
 $\sigma \in V^*$

> Ascendentes: En este tipo de analizadores el árbol se construye de las hojas (Símbolos Terminales), hacia la raíz (Símbolo inicial de G)

Los símbolos terminales representan las clases léxicas (Analizador Léxico - Tokens).

Analizadores Descendentes

Para aplicar análisis sintáctico descendente se requiere que:

- 1) G sea libre de contexto
- 2) G no sea ambigua
- 3) G no sea recursiva por la izquierda.

Recursión por la izquierda

Una gramática G es recursiva por la izquierda si \exists al menos una regla en la que el primer símbolo del lado derecho es igual al del lado izquierdo.

Ejemplo: la gramática con reglas.

$$\begin{aligned} E &\rightarrow E+T \mid E-T \mid T \\ T &\rightarrow T*T \mid T/F \mid F \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

es recursiva por la izquierda ya que:

$$\begin{aligned} E &\rightarrow E+T, \quad E \rightarrow E-T \\ T &\rightarrow T*T, \quad T \rightarrow T/F \end{aligned}$$

por lo tanto a G no se le puede aplicar análisis sintáctico descendente.

Eliminación de Recursión por la Izquierda

Consideremos las siguientes reglas gramaticales.

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

Con la recursión por la izquierda

El siguiente conjunto de reglas definen el mismo lenguaje:

$$\begin{aligned} A &\rightarrow \beta_1 A^y \mid \beta_2 A^y \mid \dots \mid \beta_m A^y \\ A^y &\rightarrow \alpha_1 A^y \mid \alpha_2 A^y \mid \dots \mid \alpha_n A^y \mid \epsilon \end{aligned}$$

Ejemplo: Eliminar la recursión izquierda de

$$\begin{aligned} E &\rightarrow E+T \mid E-T \mid T \\ T &\rightarrow T*T \mid T/F \mid F \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

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

$\begin{matrix} \square & \square & \square & \square \\ A & A & \alpha_1 & A & \alpha_2 & \beta_1 \end{matrix}$

aplicando el procedimiento

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid -TE' \mid E \end{aligned}$$

$$\begin{aligned} T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid /FT' \mid \epsilon \end{aligned}$$

Análisis Sintáctico Descendente

Considerando la sig. gramática con reglas:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid -TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid /FT' \mid \epsilon \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

construiremos un analizador por Descenso Recursivo

Procedimiento:

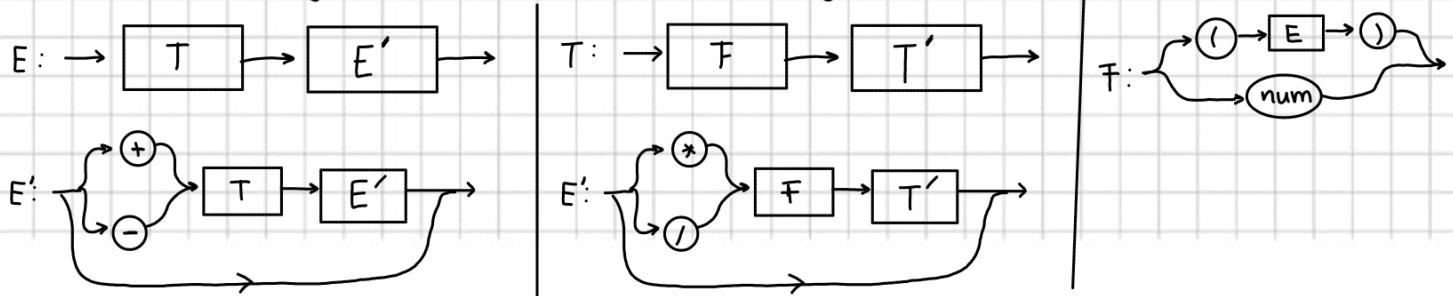
Se define una función booleana para cada símbolo no terminal de G, en este caso tenemos:

```
bool E( )  
{  
};
```

```
bool E'( )  
{  
};
```

```
bool F( )  
{  
};
```

Se simplifican las reglas gramaticales por medio de diagramas de sintaxis



El código asociado a cada función se obtiene de la siguiente manera:

Se analiza el lado derecho, de izq a derecha, $\alpha_1 \alpha_2 \dots \alpha_n$ asociado al lado izquierdo A, (i.e.) $A \rightarrow \alpha_1 \alpha_2 \dots \alpha_n$

Si $\alpha_i \in V_T$, se pide un token al analizador léxico. Si el token corresponde a la clase léxica de α_i , entonces se analiza α_{i+1} , en caso contrario la función termina y regresa false.

Si $\alpha_i \in J_N$, Se llama a la función α ; si α_i regresa verdadero entonces se analiza α_{i+1} , en caso contrario termina la función A con el valor falso.

Si se llega al análisis de α_n y es verdadero, entonces A es verdadero, en caso contrario es falso.

Construyamos las funciones para la gramática de ejemplo:

```
bool SintacExprAritm()
```

```
{
    int Token;
    if (E())
    {
        Token = Lexic.yylex();
        if (Token == 0)
            return true;
    }
    return false;
}
```

```
bool Ep()
```

```
{
    int Token;
    Token = Lexic.yylex();
    if (Token == Tokens.SUMA || Token == Tokens.RESTA)
    {
        if (T())
            if (Ep())
                return true;
            return false;
    }
    Lexic.UndoToken();
    return true;
}
```

```
bool T()
```

```
{
    if (F())
        if (Tp())
            return true;
        return false;
}
```

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE'|-TE'|E \\ T &\rightarrow FT' \\ F' &\rightarrow *FT'|/FT'|E \\ T &\rightarrow (E)|\text{num} \end{aligned}$$

```
bool Tp()
```

```
{
    int Token;
    Token = Lexic.yylex();
    if (Token == Tokens.PROD || Token == Tokens.DIV)
    {
        if (F())
            if (Tp())
                return true;
            return false;
    }
    Lexic.UndoToken();
    return true;
}
```

```
bool E()
```

```
{
    if (T())
        if (Ep())
            return true;
        return false;
}
```

V_T	Token
+	10
-	20
*	30
/	40
(50
)	60
NUM	70

```
static class Tokens
```

```
{  
    static int SUMA = 10;  
    static int RESTA = 20;  
    static int PROD = 30;  
    static int DIV = 40;  
    static int PAR_I = 50;  
    static int PAR_D = 60;  
    static int NUM = 70;  
}
```

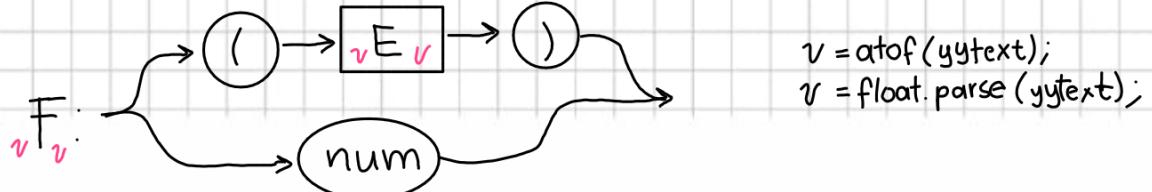
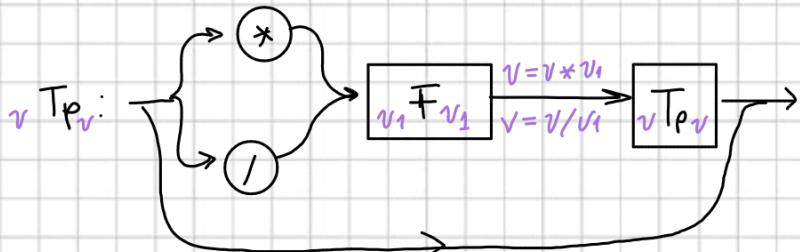
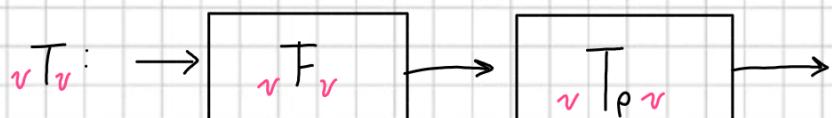
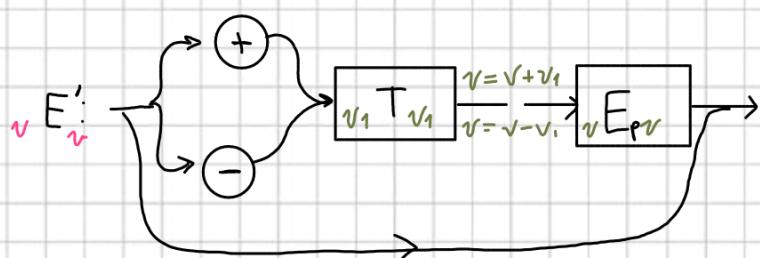
```
bool F()  
{  
    int Token;  
    Token = Lexic.yylex();  
    switch (Token)  
    {  
        case Tokens.PAR_I :  
            if (E())  
            {  
                Token = Lexic.yylex();  
                if (Tokens==PAR_D)  
                    return true;  
            }  
            break;  
        case Tokens.NUM:  
            return true;  
    }  
    return false;  
}
```

Atributos en un analizador descendente recursivo

Los Atributos se utilizan para llevar a cabo el análisis semántico, en el ejemplo anterior los atributos se pueden utilizar para:

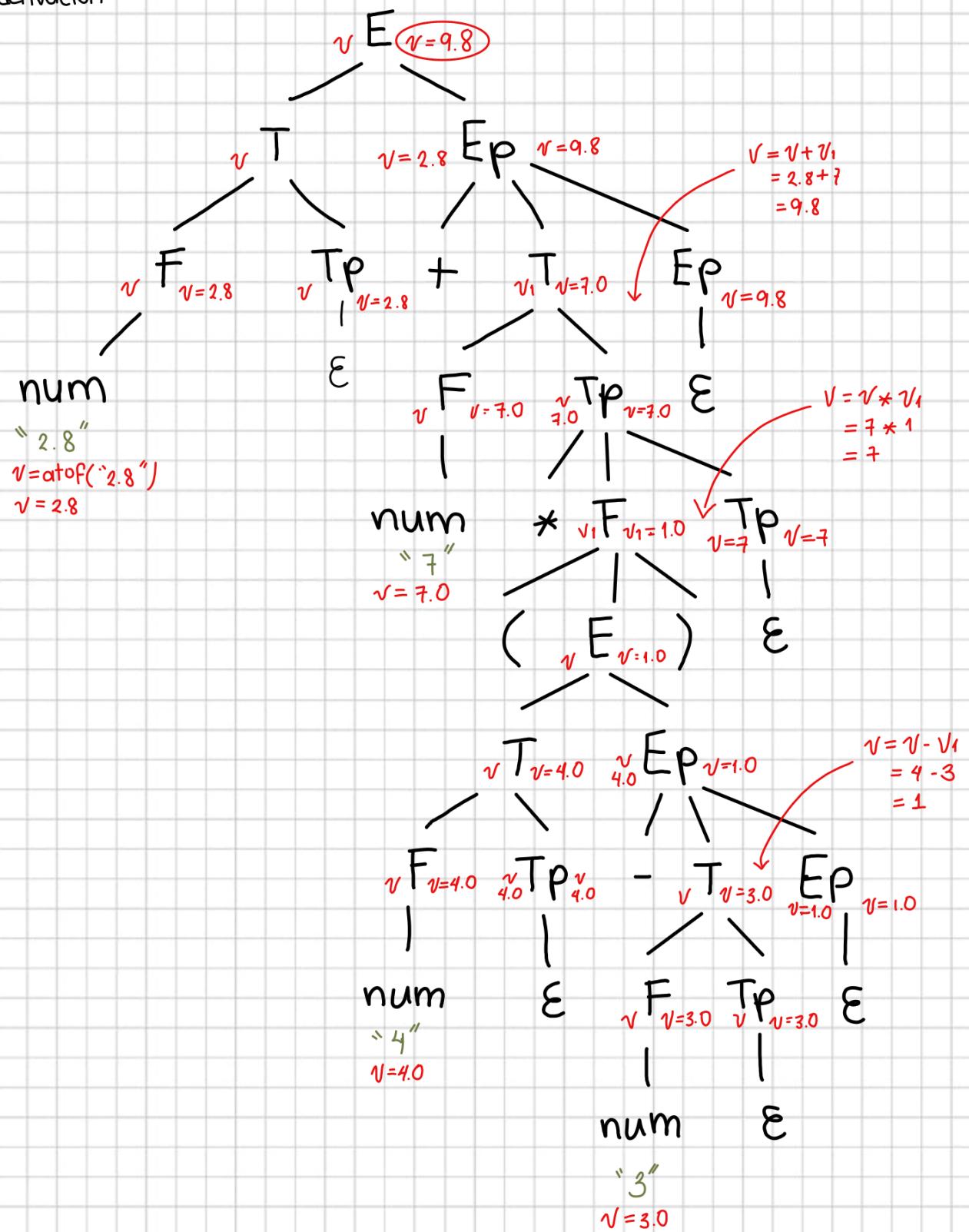
- a) Evaluar la expresión
 - b) Convertir a postfijo la expresión
 - c) Obtener código de nivel intermedio asociado a la expresión.
- etc.

Aplicaremos los atributos para evaluar la expresión:

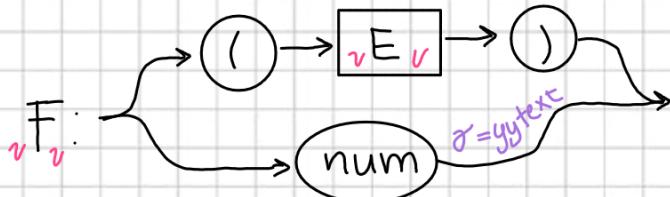
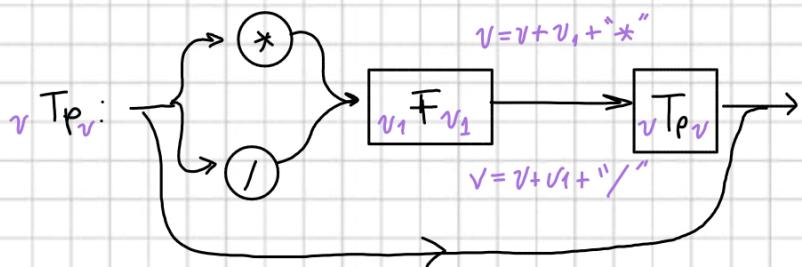
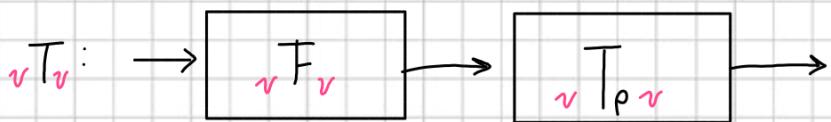
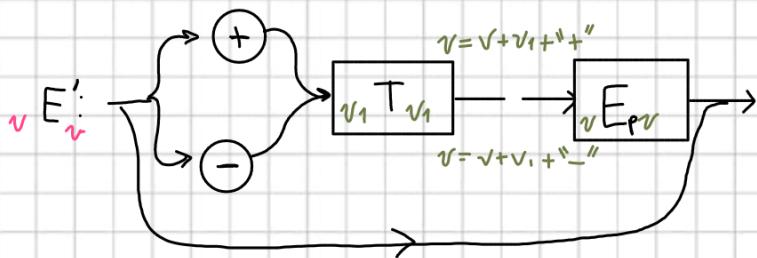


Mostraremos este procedimiento considerando $\sigma = 2.8 + 7 * (4 - 3)$

Árbol de derivación



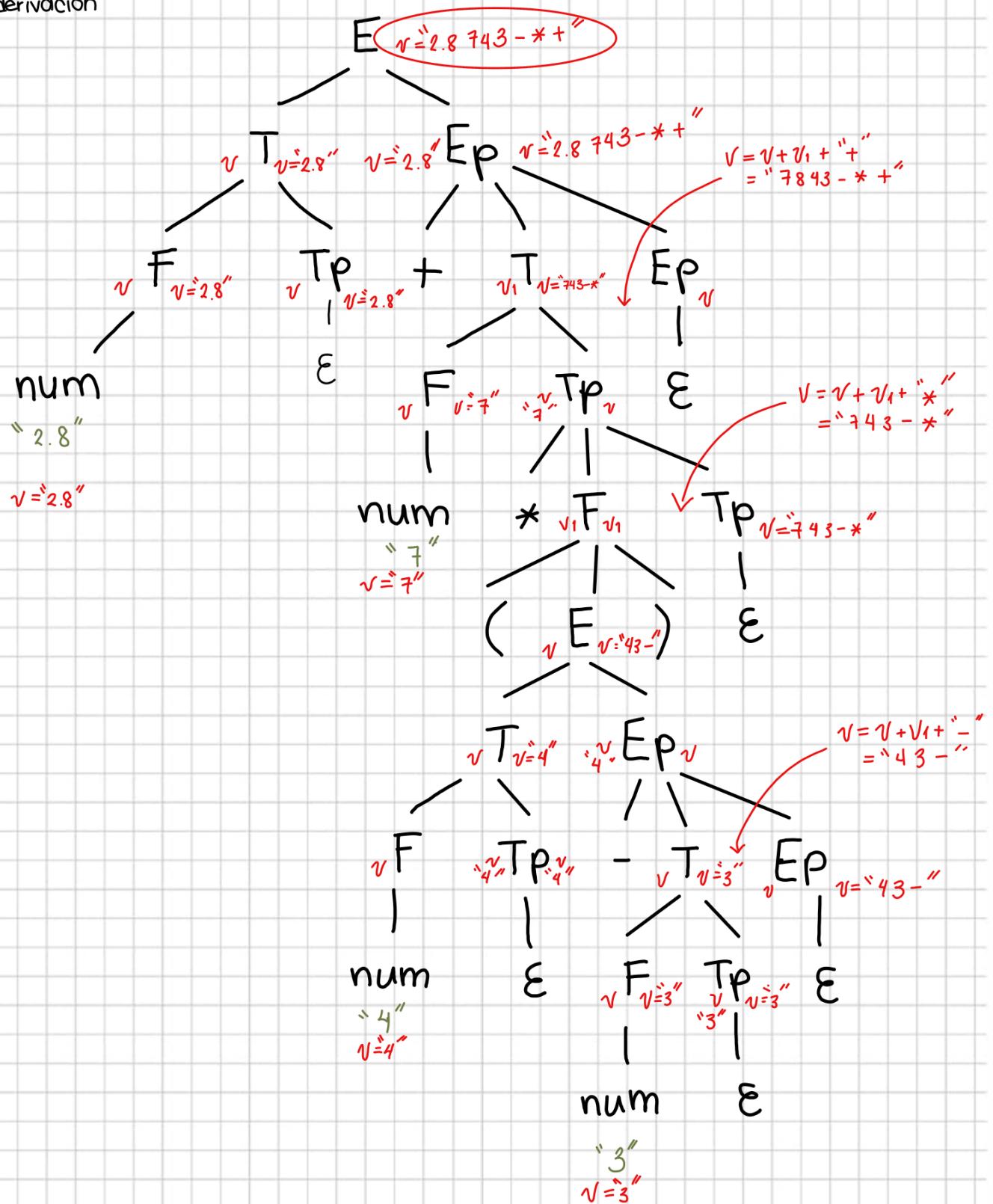
Aplicaremos los atributos para convertir la expresión:



$v = \text{atof}(yytext);$
 $v = \text{float.parse}(yytext);$

Mostraremos este procedimiento considerando $\sigma = 2.8 + 7 * (4 - 3)$

Árbol de derivación



Gramática para una calculadora científica

$$\begin{aligned} E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * P \mid T / P \mid P \\ P &\rightarrow P \wedge F \mid F \\ F &\rightarrow (E) \end{aligned}$$

$$\left. \begin{array}{l} | \sin(E) \\ | \cos(E) \\ | \tan(E) \\ | \text{ATAN}(E) \\ | \text{ASEN}(E) \\ | \text{ACOS}(E) \\ | \text{LOG}(E) \\ | \text{LN}(E) \\ | \text{EXP}(E) \\ | \text{PI} \\ | \text{NUM} \end{array} \right\} | \text{FUNC}(E)$$

Eliminemos recursión por la izquierda

$$\begin{aligned} E &\rightarrow TE_p \\ E_p &\rightarrow + T E_p \mid - T E_p \mid \epsilon \\ T &\rightarrow P T_p \\ T_p &\rightarrow * P T_p \mid / P T_p \mid \epsilon \\ P &\rightarrow F P_p \\ P_p &\rightarrow \wedge F P_p \mid \epsilon \\ F &\rightarrow (E) \end{aligned}$$

$$\left. \begin{array}{l} | \text{FUN}(E) \\ | \text{PI} \\ | \text{NUM} \end{array} \right.$$

Clases Léxicas

	e.r	token
SUMA	+	10
RESTA	-	20
PROD	*	30
DIV	/	40
POT	^	50
FUNC (soion c o s t a n : e x p)		60
PAR_I		70
PAR_D		80
PI		90
NUM		100

\ signo

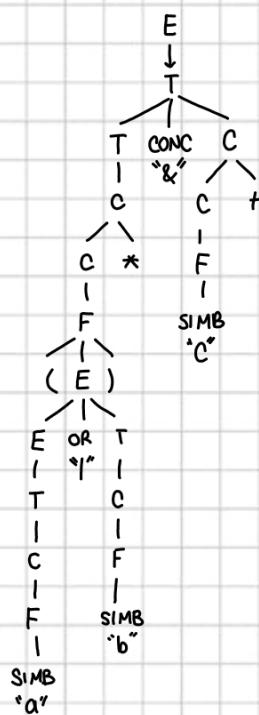
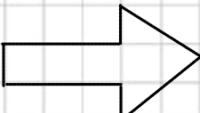
Construcción de un AFN a partir de una e.r., utilizando descenso recursivo

Gramática para el lenguaje de las e.r.

$E \rightarrow E$	OR	$T \mid T$
$T \rightarrow T$	CONC	$C \mid C$
$C \rightarrow C$	+	$C \ast \mid C ? \mid F$
$F \rightarrow (E)$		SIMB

Obtener el árbol de derivación para
 $\sigma = (a \mid b)^* \& C^+$

(SIMB OR SIMS) * CONC SIMB +



Eliminemos la recursión por la izquierda, para poder construir el analizador sintáctico por descenso recursivo.

$E \rightarrow T E'$
 $E' \rightarrow \text{OR } T E' \mid \varepsilon$
 $T \rightarrow C T'$
 $T' \rightarrow \text{conc } C T' \mid \varepsilon$
 $C \rightarrow F C'$
 $C' \rightarrow + C' \mid * C' \mid ? C' \mid \varepsilon$
 $F \rightarrow (E) \mid \text{SIMB} \mid [\text{SIMB} - \text{SIMB}]$

Tokens

Claase léxica	E.R	Token
OR	' '	10
CONC	'&'	20
CERR_POS	'+'	30
CERR_KLEEN	'*''	40
OPC	'?''	50
PAR_I	'(''	60
PAR_D	')'	70
SIMB	todos los caracteres	80
CORCH_I	'[''	90
CORCH_D	')'	100
GUION	'-'	110

```

bool E (AFN f)
{
    if (T(f))
        if (Ep (f))
            return true;
        else
            return false;
    else
        return false;
}

bool Ep (AFN f)
{
    int token;
    AFN f2 = new AFN ();
    Token = Lexic.yylex ();
    if (Token == 10)
    {
        if (T(f2))
            return true;
        else
            return false;
    }
}

bool T (AFN f)
{
    if (C(f))
        if (Tp (f))
            return true;
        else
            return false;
}

```

```

bool Tp (AFN f)
{
    int token;
    AFN f2 = new AFN();
    Token = Lexic.yylex();
    if (Token == 20)
    {
        if (C(f2))
        {
            f = f. CONC (f2);
            if (Tp (f))
                return true;
        }
        return false;
    }
    Lexic.UndoToken();
    return true;
}

```

```

bool C (AFN f)
{
    if (F(f))
        if (Cp(f))
            return true;
    return false;
}

```

```

bool Cp (AFN f)
{
    int token;
    AFN f2 = new AFN();
    Token = Lexic.yylex();
    switch (Token)
    {
        case 30:
            f.CerrPos();
            break;
        case 40:
            f.CerrKleen();
            break;
        case 50:
            f.Opc();
            break;
        default:
            Lexic.UndoToken();
            return true;
    }
    return Cp(f);
}

```

```

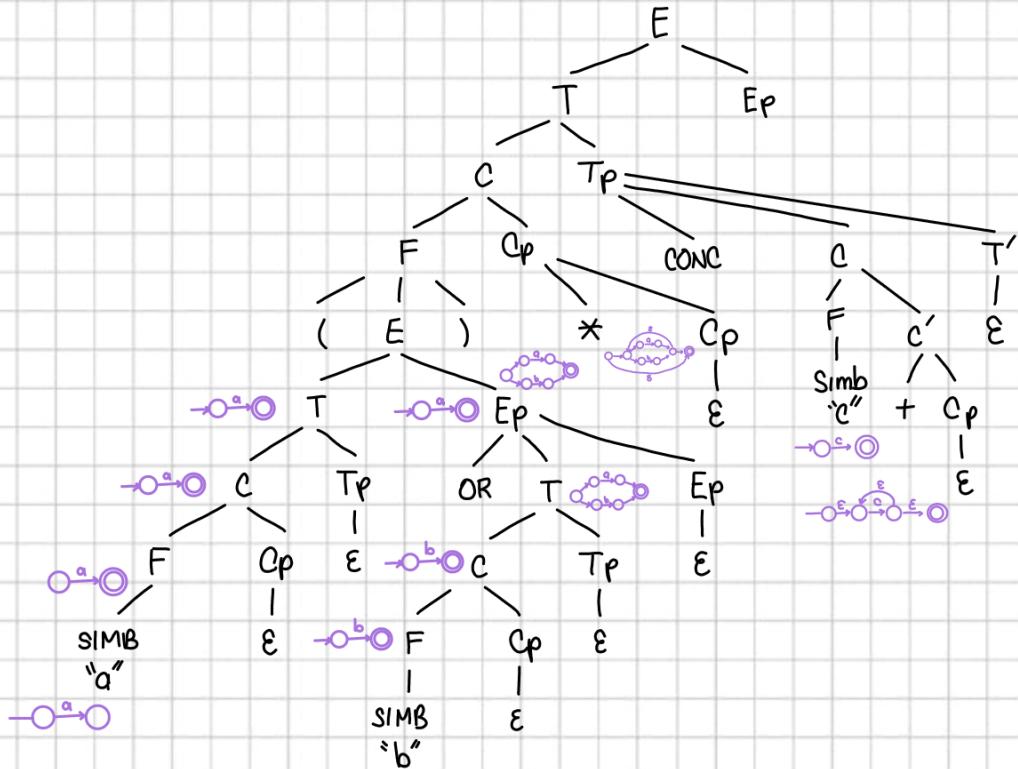
bool F (AFN f)
{
    char Simb1, Simb2;
    int Token;
    Token = Lexic.yylex();
    switch (Token)
    {
        case 60:
            if (E(f))
            {
                Token = Lexic.yylex();
                if (Token == 70)
                    return true;
            }
            return false;
        case 80:
            Simb1 = Lexic.yytext[0];
            f.CrearBasico(Simb1);
            return true;
        case 90:
            Token = Lexic.yylex();
            if (Token == 80)
            {
                Simb1 = Lexic.yytext[0];
                Token = Lexic.yylex();
                if (Token == 110)
                {
                    Token = Lexic.yylex();
                    if (Token == 80)
                    {
                        Simb2 = Lexic.yytext[0];
                        Token = Lexic.yylex();
                        if (Token == 100)
                        {
                            f.CrearBasico(Simb1, Simb2);
                            return true;
                        }
                    }
                }
            }
            return false;
    } // cierra switch
    return false;
}

```

Considerando:

$$\sigma = (a|b)* \& c +$$

mostraremos el proceso de construcción del AFN



Análisis Sintáctico Descendente LL(1)

El análisis LL(1) procesa la cadena de entrada σ , de izquierda (L) a derecha.

Se obtiene una derivación por la izquierda (L), se requiere de la información de 1 token para determinar la regla gramatical a utilizar en el proceso de derivación.

En general se pueden tener analizadores LL(k), $k \geq 1$, k entero.

Para construir analizadores LL(1) requerimos de 2 operaciones

- 1) First(α), donde $\alpha \in (V_N \cup V_T)^*$
- 2) Follow(A), donde $A \in V_N$

La operación First(α) es un conjunto de símbolos terminales $\cup \{\epsilon\}$ de los símbolos terminales con los que pueden iniciar las derivaciones de α .

La operación Follow(A) son los símbolos terminales que puede aparecer inmediatamente después de A en un proceso de derivación.

Ejemplos: Considerando las reglas gramaticales:

$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow + TE' \mid - TE' \mid \epsilon \\ T &\rightarrow F T' \\ T' &\rightarrow * FT' \mid / FT' \mid \epsilon \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

Obtener:

- 1) First(ϵ) = { ϵ }
- 2) First(+TE' $F T'$) = {+}
- 3) First(TE') = {(, num)}



$$4) \text{First}(E'T'F) = \{+, -, *, /, (, \text{num}\}$$

+TE' -TE' E *FT' /FT' E (E) num

$$5) \text{First}(E'T') = \{+, -, *, /, \epsilon\}$$

+TE' -TE' E *FT' /FT' E

Reglas para calcular el First de $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$ $\alpha_i \in V_N \cup V_T \cup \{\epsilon\}$

1) Si $\alpha = \epsilon$ entonces

$$\text{First}(\alpha) = \{\epsilon\}$$

2) Si $\alpha_1 \in V_T \Rightarrow$

$$\text{First}(\alpha) = \{\alpha_1\}$$

3) Si $\alpha_1 \in V_N$ y $\alpha_1 \rightarrow \beta$ en una regla de α_1 entonces

$$\text{First}(\beta) \subseteq \text{First}(\alpha_1)$$

4) Si $\alpha_1 \in V_N$ y $\epsilon \in \text{First}(\alpha_1)$ entonces se considera

$$\text{First}(\alpha) = \text{First}(\alpha_1) - \{\epsilon\}$$

$$\text{First}(\alpha_2 \alpha_3 \dots \alpha_n) \subseteq \text{First}(\alpha)$$

5) Si $\epsilon \in \epsilon \in \text{First}(\alpha_i) \forall i=1, 2, \dots, n$ entonces $\epsilon \in \text{First}(\alpha)$

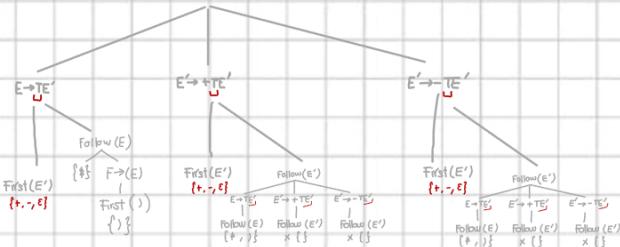
Follow(A), $A \in V_N$

Usaremos $\$$ para denotar el fin de una cadena

Ejemplos:

$$1) \text{Follow}(E) = \{ \), \$ \}$$

$$2) \text{Follow}(T) = \{ +, -,$$



Reglas para calcular el Follow(A) $A \in V_N$

$$\text{Follow}(A) = \{ \}$$
 inicialmente

1) Si A es el simbolo inicial de G entonces agregar $\$$ a Follow(A)

2) Si $B \rightarrow \alpha A \gamma^l$ con $\alpha, \gamma^l \in (V_N \cup V_T)^*$

2.1) Si $\gamma^l = \epsilon$ o $\epsilon \in \text{First}(\gamma^l)$ entonces $\text{Follow}(B) \subseteq \text{Follow}(A)$ y $\text{First}(\gamma^l) - \epsilon \subseteq \text{Follow}(A)$

Ejemplos: Calculemos los First de Todos los lados derechos:

#regla

- 1) First(TE') = { (, num }
- 2) First(+TE') = { + }
- 3) First(-TE') = { - }
- 4) First(ε) = { ε }
- 5) First(FT') = { (, num }
- 6) First(*FT') = { * }
- 7) First(/FT') = { / }
- 8) First(ε) = { ε }
- 9) First((ε)) = { () }
- 10) First(num) = { num }

11

12

13

14

15

16

17

18

19

20

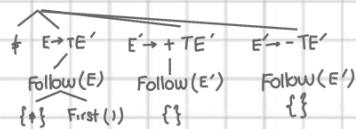
donde haga ϵ en el first se procede a calcular el Follow del lado izquierdo. En este caso hubo ϵ en el first de las reglas 4 y 8.

$$E' \rightarrow E \quad \gamma_4$$

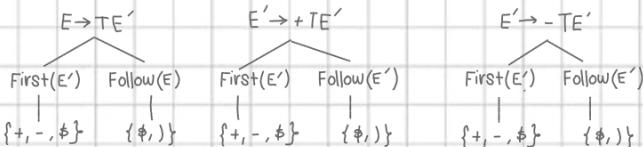
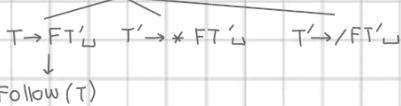
$$T' \rightarrow E \quad \gamma_8$$

Se calcula el Follow de los lados derechos de esas reglas.

$$a) \text{Follow}(E') = \{ \$,) \}$$



$$\text{Follow}(T') = \{ +, -, \$ \}$$



$$\begin{aligned} E &\rightarrow T \quad E' \quad \textcircled{1} \\ E' &\rightarrow +TE' \quad \textcircled{2} \mid -TE' \quad \textcircled{3} \mid \epsilon \quad \textcircled{4} \\ T &\rightarrow F \quad T' \quad \textcircled{5} \\ T' &\rightarrow *FT' \quad \textcircled{6} \mid /FT' \quad \textcircled{7} \mid \epsilon \quad \textcircled{8} \\ F &\rightarrow (\quad E) \quad \textcircled{9} \mid \text{num} \quad \textcircled{10} \end{aligned}$$

Creación de tabla de análisis sintáctico LL(1)

Gramática

$$\begin{aligned} E &\rightarrow T \quad E' \quad \textcircled{1} \\ E' &\rightarrow +TE' \quad \textcircled{2} \mid -TE' \quad \textcircled{3} \mid \epsilon \quad \textcircled{4} \\ T &\rightarrow F \quad T' \quad \textcircled{5} \\ T' &\rightarrow *FT' \quad \textcircled{6} \mid /FT' \quad \textcircled{7} \mid \epsilon \quad \textcircled{8} \\ F &\rightarrow (\quad E) \quad \textcircled{9} \mid \text{num} \quad \textcircled{10} \end{aligned}$$

La tabla tendrá como filas a los símbolos $V_n \cup V_T \cup \{\$\}$ y como columnas se tendrá a $V_T \cup \{\$\}$, para esta gramática tendremos

	+	-	*	/	()	num	\$
E					1, TE'		1, TE'	
E'	2, +TE'	3, -TE'				4, ε		4, ε
T					5, FT'		5, FT'	
T'	8, ε	8, ε	6, *FT'	7, /FT'		8, ε		8, ε
F					9, (E)		10, num	
+	pop							
-		pop						
*			pop					
/				pop				
(pop			
)						pop		
num							Accept	
\$								

Se analiza cada regla

Regla #1

$$E \rightarrow TE'$$

se calcula el first del lado derecho

$$\text{First}(TE') = \{ (, num \}$$

$$\Rightarrow \text{Tabla } [E, \{ (, num \}] = 1, TE'$$

Regla #2

$$E' \rightarrow +TE'$$

$$\text{First}(+TE') = \{ + \}$$

$$\Rightarrow \text{Tabla } [E', \{ + \}] = 2, +TE'$$

Regla #3

$$E' \rightarrow -TE'$$

$$\text{First}(-TE') = \{ - \}$$

$$\Rightarrow \text{Tabla } [E', \{ - \}] = 3, -TE'$$

Regla #4

$$E' \rightarrow \epsilon$$

$$\text{First}(\epsilon) = \{\epsilon\}$$

Como $\epsilon \in \text{First}$ entonces

se calcula el Follow del lado izq.

$$\text{Follow}(E') = \{ \), + \}$$

$$\Rightarrow \text{Tabla}[E', \{ \), + \} \cup \{ \epsilon \} - \{ \) \}] = 4, E'$$

Regla #5

$$T \rightarrow FT', \text{First}(FT') = \{ (, \text{num} \}$$

$$\Rightarrow \text{Tabla}[T, \{ (, \text{num} \}] = 5, FT'$$

Regla #7

$$T \rightarrow /FT', \text{First}(/FT') = \{/ \}$$

$$\Rightarrow \text{Tabla}[T, \{ / \}] = 7, /FT'$$

Regla #8

$$T' \rightarrow \epsilon$$

$$\text{First}(\epsilon) = \{\epsilon\} \Rightarrow \text{hay que calcular Follow}(T')$$

$$\text{Follow}(T') = \{ +, -,), + \}$$

$$\text{First}(\epsilon) - \{\epsilon\} \cup \text{Follow}(T') = \{ +, -,), + \}$$

$$\Rightarrow \text{Tabla}[T', \{ +, -,), + \}] = 8, \epsilon$$

Regla #6

$$T \rightarrow *FT', \text{First}(*FT') = \{ * \}$$

$$\Rightarrow \text{Tabla}[T, \{ * \}] = 6, *FT'$$

Definición de la tabla LL(1)

La tabla LL(1) es una relación con dominio $(V_N \cup V_T \cup \{ \# \}) \times (V_T \cup \{ \# \})$

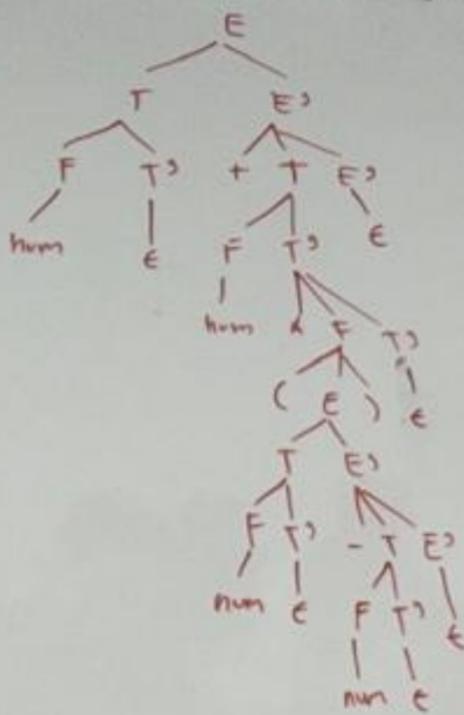
donde

$$\text{Tabla}(\alpha, \beta) = \begin{cases} \text{pop}, & \text{si } \alpha = \beta \text{ y } \alpha \in V_T \\ i, \gamma^*, & \text{si } \alpha \in V_N \text{ y } \alpha \rightarrow \gamma^* \text{ es la regla } \#i \text{ y } \beta \in \text{First}(\gamma^*) \\ i, \gamma^*, & \text{si } \alpha \in V_N \text{ y } \alpha \rightarrow \gamma^* \text{ es la regla } \#i \text{ y } \beta \in \text{Follow}(\alpha) \\ \text{acept}, & \text{si } \alpha = \beta \text{ y } \alpha = \# \\ \text{error}, & \text{en otro caso} \end{cases}$$

Ejemplo: Analizar $\sigma = 2.8 + 7 * (-5 - 4)^*$ usando la tabla LL(1)

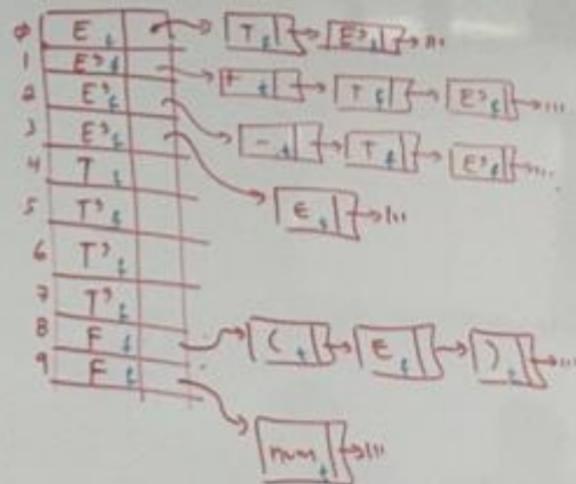
Pila	σ	Acción	
$\#E$	num + num * (num - num) $\#$	1, TE'	$\#E'T')E'T'num$
$\#E'T$	num + num * (num - num) $\#$	5, FT'	$\#E'T')E'T'$
$\#E'T'F$	num + num * (num - num) $\#$	10, num	$\#E'T')E'$
$\#E'T'num$	num + num * (num - num) $\#$	pop	$\#E'T')$
$\#E'T'$	+ num * (num - num) $\#$	8, ϵ	$\#E'T'$
$\#E'$	+ num * (num - num) $\#$	2, + TE'	$\#E'$
$\#E'T+$	+ num * (num - num) $\#$	pop	$\#$
$\#E'T$	num * (num - num) $\#$	5, FT'	$num) \#$ pop
$\#E'T'F$	num * (num - num) $\#$	10, num	$) \#$ 8, ϵ
$\#E'T'num$	num * (num - num) $\#$	pop	$) \#$ 4, ϵ
$\#E'T$	* (num - num) $\#$	6, *FT'	$) \#$ pop
$\#E'T'F*$	* (num - num) $\#$	pop	$\$$ 8, ϵ
$\#E'T'F$	(num - num) $\#$	9, (E)	$\$$ 4, ϵ
$\#E'T')E$	(num - num) $\#$	pop	$\$$ aceptar
$\#E'T')E$	num - num) $\#$	1, TE'	
$\#E'T')E'T$	num - num) $\#$	ϵ , FT'	
$\#E'T')E'T'F$	num - num) $\#$	10, num	
$\#E'T')E'T'num$	num - num) $\#$	pop	
$\#E'T')E'T'$	- num) $\#$	8, ϵ	
$\#E'T')E'$	- num) $\#$	3, - TE'	
$\#E'T')E'T-$	- num) $\#$	pop	
$\#E'T')E'T$	num) $\#$	5, ET'	
$\#E'T')E'T'F$	num) $\#$, num	

la última columna nos proporciona la info para el símbolo de desglose.



Implementación:

Las reglas gramaticales las representaremos en un arreglo de listas



Clase Gramática:

int NumReglas;

LadoIzq[] Reglas;

Conjunto<String> VN;

Conjunto<String> VT;

SimbolG SimbIn;

Operación First

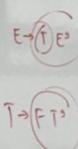
Conjunto<SimbolG> First(lista<SimbolG> l)

{ Conjunto<SimbolG> R = new Conjunto<SimbolG>();
R.Clear();

if(l[0].EsTerminal)

{ R.Add(l[0]);

} return R;



Requerimos una clase SimbolG que contendrá:

```

for(int i=0 ; i<_NumReglas ; i++)
{
    if(Reglas[i].SimbIzq.NombSimb == l[0].NombSimb)
        Clas
    {
        R.Union(First(Reglas[i].LadoDerecho));
        if(R.contains(epsilon))
        {
            if(L.count() == 1)
                return R;
            R.Union(First(&Subluta{l, l.count()-1})));
        }
    }
    return R;
}

```

Conjunto<SimbolG> Follow(SimbolG s)

```

int j;
{ Conjunto<SimbolG> R = new Conjunto<SimbolG>();
Conjunto<SimbolG> Ares = new Conjunto<SimbolG>();
R.Clear();
if( S.EsTerminal )
    return R;

```

// Buscar s en los lados derechos

for(int i=0 ; i<_NumReglas ; i++)
{

j=Reglas[i].LadoDerecho.IndexOf(s);

if(j == -1)
 continue;

if(j == (Reglas[i].LadoDerecho.Count() - 1))
 if(S == Reglas[i].SimbIzq)
 continue;

R.Union(Follow(Reglas[i].SimbIzq));

continue;

0 E → TE'

1 E' → +TE'

2 E' → -TE'

3 E' → ε

4 T → FT'

5 T' → xFT'

6 T' → /FT'

7 T → ε

8 F → (E)

9 F → num

Requerimos una clase SimbolG
que contendrá:

// Hay elementos después de S en la lista i | int NumReglas;

Aux = First(Reglas[i]).LadoDerecho.Sublist(j > Reglas[i].LadoDerecho.count() - j);

if (Aux tiene ε)

{ Aux = Aux - {ε};

R.Union(Aux);

R.Union(Follow(Reglas[i].SimbIzaq)));
continue;

}

R.Union(Aux);

}

return R;

}

Close Gramatica:

int NumReglas;

,

Generación del arreglo de Reglas

El llenado del arreglo de reglas lo realizaremos utilizando desenfoque recursivo. Para esto necesitamos una gramática que nos defina el lenguaje de las gramáticas

Gramática de Gramáticas

$$G \rightarrow \text{Reglas}$$

$$\text{Reglas} \rightarrow \text{Regla} ;$$

$$| \text{Reglas} \text{ Regla} ;$$

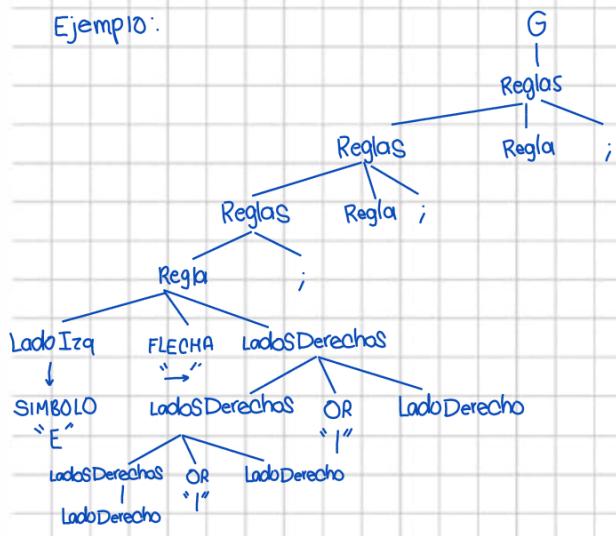
$$\text{Regla} \rightarrow \text{LadoIzq FLECHA LadosDerechos}$$

$$\text{LadoIzq} \rightarrow \text{SIMBOLO}$$

$$\text{LadosDerechos} \rightarrow \text{Lado Derecho}$$

$$| \text{LadosDerechos OR Lado Derecho}$$

Ejemplo:



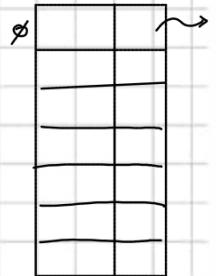
$$E \rightarrow E + T \mid E - T \mid T;$$

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

$$F \rightarrow (E) \mid \text{num};$$

$$\text{Lado Derecho} \rightarrow \text{SecSimbolos}$$

$$\text{SecSimbolos} \rightarrow \text{SIMBOLO} \\ | \text{SecSimbolos} \text{ SIMBOLO}$$



Eliminemos recursión izquierda

$$\begin{aligned}
 G &\rightarrow \text{Reglas} \\
 \text{Reglas} &\rightarrow \text{Regla} ; \text{Reglas}' \\
 \text{Reglas}' &\rightarrow \text{Regla} ; \text{Reglas}' \mid E \\
 \text{Regla} &\rightarrow \text{LadoIzq FLECHA LadosDerechos} \\
 \text{LadoIzq} &\xrightarrow{\text{string}} \text{SIMBOLO} \\
 \text{LadosDerechos} &\xrightarrow{\text{string}} \text{Lado Derecho} \quad \text{LadosDerechos}' \\
 \text{LadosDerechos} &\rightarrow \text{OR} \text{ Lado Derecho} \text{ LadosDerechos}' \mid \epsilon \\
 \text{Lado Derecho} &\rightarrow \text{SecSimbolos} \\
 \text{SecSimbolos} &\rightarrow \text{SIMBOLO} \quad \text{SecSimbolos}' \\
 \text{SecSimbolos} &\rightarrow \text{SIMBOLO} \quad \text{SecSimbolos}' \mid \epsilon
 \end{aligned}$$

l.Add("Simbolo");
 l.Add("Simbolo");

Analizador Sintáctico

No se deben predefinir los tokens

G	e.r	token
OR		10
CONC	&	20
CerrPOS	+	30
CerrKleen	*	40
Opc	?	50
ParI	(60
ParD)	70
Simb	[a-z] [A-Z] [0-9] \+ \- ...	80
Guion	-	90
CorchI	[100
CorchD]	110

Simbolo	([a-z] [A-Z]) & ([a-z] [A-Z] [0-9] _) * & ?	Token
FLECHA	- & >	10
OR		20
PC	;	30
ESP_NL	\n + (Saltar)	40
		20,000

Gramática posible:

$$\begin{aligned}
 E &\rightarrow T \cup E' ; \\
 E' &\rightarrow \text{mas} \cup T \cup E' \mid \text{menos} \cup T \cup E' \mid \text{epsilon} ; \\
 T &\rightarrow F T' ; \\
 T' &\rightarrow \text{prod} \mid F F' \mid \text{div} \mid F T' \mid \text{epsilon} ; \\
 F &\rightarrow \text{ParI} \mid E \mid \text{ParD}
 \end{aligned}$$

★ 26 y 27 de Noviembre → revisión de 2^{do} parcial

ANÁLISIS SINTÁCTICO LR

Los analizadores LR son del tipo ascendente, es decir se construye el "árbol de derivación" partiendo de las hojas (terminales).

El nombre LR proviene de:

- L: La cadena σ se analiza de izquierda (L) a derecha.
- R: Se construye una derivación inversa por la derecha.

Al ser un proceso de reducción se requiere saber cuando se llega al nodo raíz. El nodo raíz corresponde al símbolo inicial de la gramática G.

Debido a lo anterior el símbolo inicial de G no debe aparecer en ningún lado derecho.

Si tenemos:

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

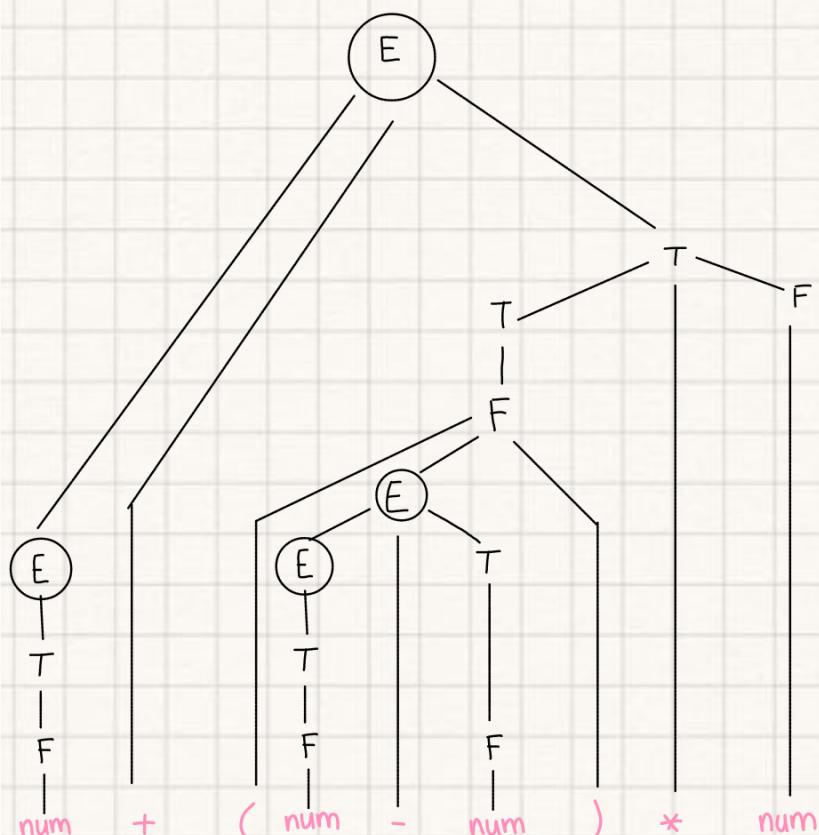
$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid \text{num}$$

el proceso de reducción para:

$$\sigma = \text{num} + (\text{num} - \text{num}) * \text{num}$$

será:



Gramática Aumentada

Sea $G(V_N, V_T, S, \Phi)$ la gramática aumentada de G, denotada por $G^y(V'_N, V_T, S', \Phi')$ se define como:
 $V'_N = V_N \cup \{S'\}$

S' : nuevo símbolo inicial de G'

$$\Phi' = \Phi \cup \{S' \rightarrow S\}$$

Para el ejemplo anterior tendremos
 G' :

$$\begin{aligned} E' &\rightarrow E \\ E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

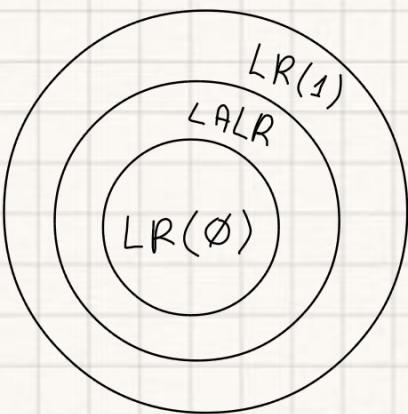
El análisis LR se puede aplicar a gramáticas TT:

- 1) No ambiguas
- 2) Libres de contexto

Hay 3 tipos de analizadores LR.

- 1) SLR ($LR(\emptyset)$)
- 2) LR canónico ($LR(1)$)
- 3) LALR

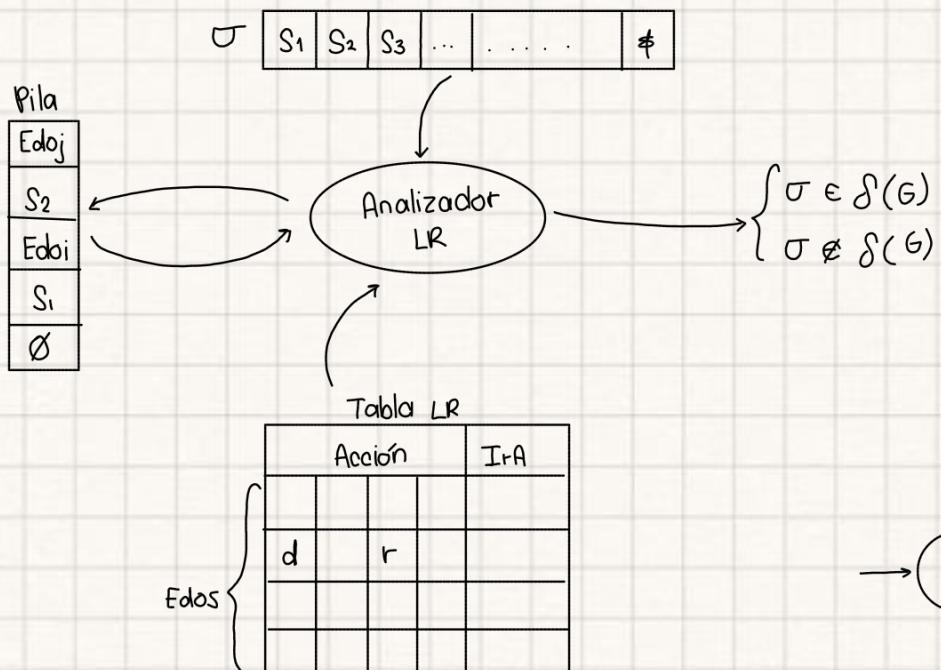
Clases de Gramáticas



Programas para entregar en el segundo parcial:

1. Calculadora Científica
2. Construcción de un AFN a partir de una e.r.
3. Deslenso recursivo para no terminales y terminales
4. Tabla LL(0)

Estructura de los analizadores LR



Ilustraremos con un ejemplo la construcción de la tabla LR(\emptyset).

Consideremos la gramática AUMENTADA:

$$\begin{aligned} E' &\rightarrow E \\ E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow (E) \mid \text{num} \end{aligned}$$

Necesitamos algunas operaciones:

1) Operación Cerradura

Un ítem de G es una regla con un punto al inicio, al final o entre los símbolos del lado derecho.

Ejemplos de ítems son:

$$\begin{aligned} E' &\rightarrow \cdot E \quad , \quad E' \rightarrow E \cdot \\ E &\rightarrow \cdot T * F \quad , \quad E \rightarrow T \cdot * F \quad , \quad E \rightarrow T * \cdot F \quad , \quad E \rightarrow T * F \cdot \end{aligned}$$

Ejemplo de cerradura:

$$d) \quad \mathcal{L}(E \rightarrow \cdot E + T) = \left\{ \begin{array}{l} E \rightarrow \cdot E + T, \quad E \rightarrow \cdot E - T, \quad E \rightarrow \cdot T, \\ T \rightarrow \cdot T * F, \quad T \rightarrow \cdot T / F, \quad T \rightarrow \cdot F, \\ F \rightarrow \cdot (E), \quad F \rightarrow \cdot \text{num} \end{array} \right\}$$

Si después del punto hay un no terminal, se agregan a la cerradura todas las reglas de ese no terminal, con un punto al inicio de su lado derecho.

$$b) \quad \mathcal{L}(\{E \rightarrow E \cdot - T, \quad F \rightarrow (\cdot E)\}) = \left\{ \begin{array}{l} E \rightarrow E \cdot - T, \quad F \rightarrow (\cdot E), \quad E \rightarrow \cdot E + T, \\ E \rightarrow \cdot E - T, \quad E \rightarrow \cdot T, \quad T \rightarrow \cdot T * F, \\ T \rightarrow \cdot T / F, \quad T \rightarrow \cdot F, \quad F \rightarrow \cdot (E), \\ F \rightarrow \cdot \text{num} \end{array} \right\}$$

2) Operación Mover

Si A es un conjunto de ítems, y $b \in V_n \cup V_T$ entonces:

$$\text{Mover}(A, b) = \{ B \rightarrow \alpha \quad b \cdot \gamma^r \mid B \rightarrow \alpha \cdot b \quad \gamma^r \in A \}$$

Ejemplo:

a)

$$\begin{aligned} \text{Mover}(\{E \rightarrow E + \cdot T, \quad T \rightarrow \cdot T * F, \quad F \rightarrow (\cdot E)\}, \quad T) \\ = \{E \rightarrow E + T \cdot, \quad T \rightarrow T \cdot * F\} \end{aligned}$$

3) Operación IrA

Sea A un conjunto de ítems y $b \in V_N \cup V_T$ entonces:

$$IrA(A, b) = \mathcal{L}(\text{Mover}(A, b))$$

Proceso para construir la tabla LR(0)

Se inicia calculando la cerradura de ítem asociado a la regla del símbolo inicial.

Para nuestro ejemplo:

$$S_0 = \mathcal{L}(E' \rightarrow \cdot E) = \{ E' \rightarrow \cdot E, E \rightarrow \cdot E + T, E \rightarrow \cdot E - T, E \rightarrow \cdot T, \\ T \rightarrow \cdot T * F, T \rightarrow \cdot T / F, T \rightarrow \cdot F, F \rightarrow \cdot (E), \\ F \rightarrow \cdot \text{num} \}$$

Analizar S_0 con $\Sigma = V_N \cup V_T$

$$S_1 = IrA(S_0, E) = \mathcal{L}(\{ E' \rightarrow E \cdot, E \rightarrow E \cdot + T, E \rightarrow E \cdot - T \}) \\ = \{ E' \rightarrow E \cdot, E \rightarrow E \cdot + T, E \rightarrow E \cdot - T \}$$

$$IrA(S_0, T) =$$

$$IrA(S_0, F) =$$

$$IrA(S_0, ()) =$$

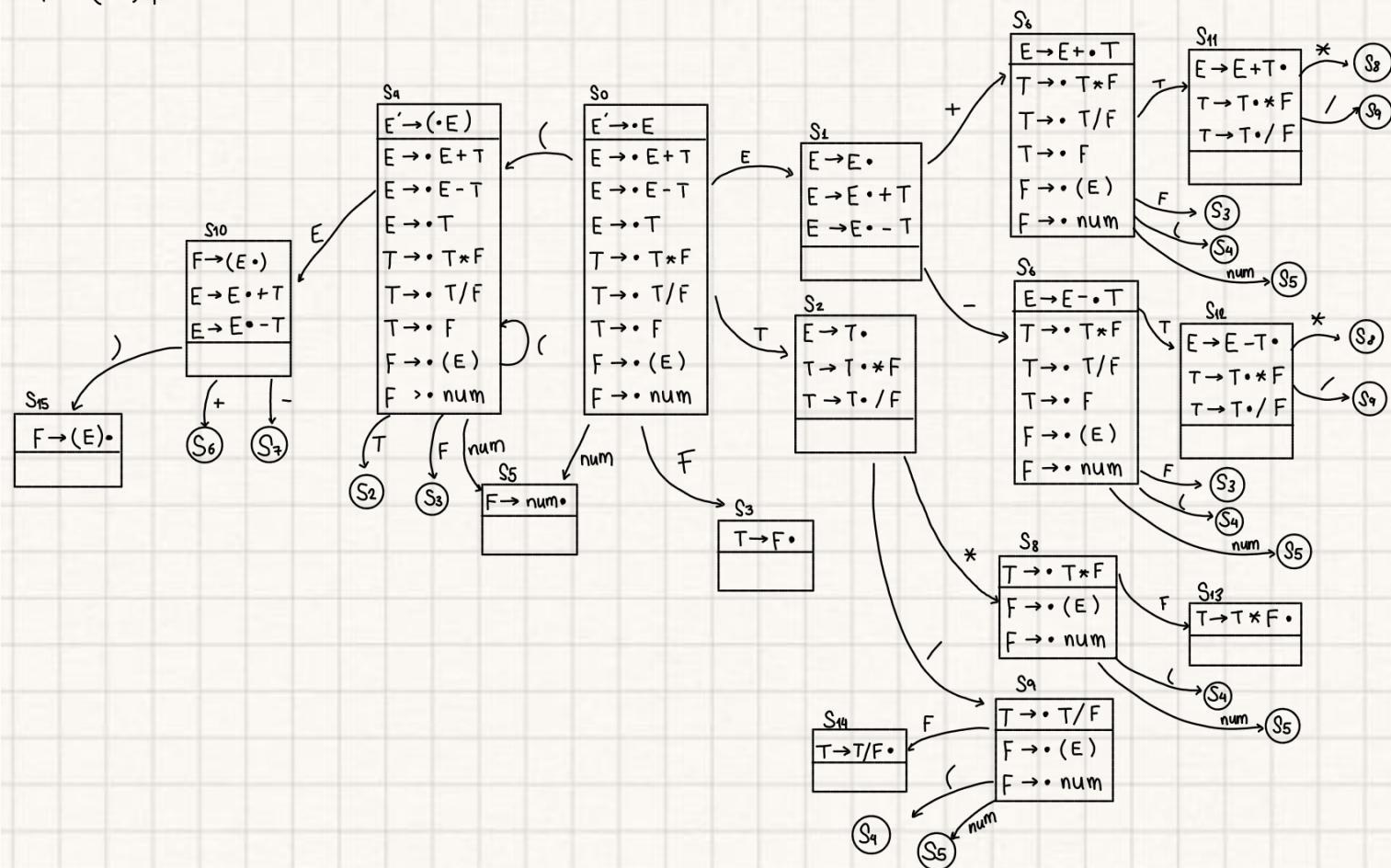
$$IrA(S_0, \text{num}) =$$

jue 30 - oct - 25

Construcción de los conjuntos de los estados del Autómata LR(0)

G:

$$\begin{array}{l} E' \xrightarrow{\circ} E^{\bullet} \\ E \rightarrow E + T \quad | \quad E \rightarrow E - T \quad | \quad E \rightarrow E \cdot \\ T \rightarrow T * F \quad | \quad T \rightarrow T / F \quad | \quad T \rightarrow T \cdot \\ F \rightarrow (E) \quad | \quad F \rightarrow \cdot \text{num} \end{array}$$



Del conjunto de estados S_j hay que identificar aquellos S_j que tienen items con punto final en su lado derecho:

$$E' \rightarrow E \cdot \in S_1 \quad \text{Follow}(E') = \{\$\}$$

regla #0

$$E \rightarrow T \cdot \in S_2 \quad \text{Follow}(E) = \{\$, +, -,)\}$$

regla #3

$$T \rightarrow F \cdot \in S_3 \quad \text{Follow}(T) = \{\ast, /, +, -,), \$\}$$

r6

$$F \rightarrow \text{num} \cdot \in S_5 \quad \text{Follow}(F) = \text{Follow}(T) = \{\ast, /, +, -,), \$\}$$

r8

$$E \rightarrow E + T \cdot \in S_{11} \quad \text{Follow}(E) = \{+, -,), \$\}$$

r1

$$E \rightarrow E - T \cdot \in S_{12} \quad \text{Follow}(E) = \{+, -,), \$\}$$

r2

$$E \rightarrow T * F \cdot \in S_{13} \quad \text{Follow}(T) = \{+, -, \ast, /,), \$\}$$

r4

$$E \rightarrow T / F \cdot \in S_{14} \quad \text{Follow}(T) = \{+, -, \ast, /,), \$\}$$

r5

$$F \rightarrow (E) \cdot \in S_{15} \quad \text{Follow}(F) = \{+, -, \ast, /,), \$\}$$

r7

Usando la tabla LR(0) analizar $\sigma = \text{num} + (\text{num} - \text{num})/\text{num} \$$

Pila	Cadena	Acción
\$0	num + (num - num)/num \$	d5
\$0 num 5	+ (num - num)/num \$	r8, F → num, num * 2 = 2
\$0 F3	+ (num - num)/num \$	r6, T → F, 2 * F = 2
\$0 T2	+ (num - num)/num \$	r3, E → T, 2 * T = 2
\$0 E1	+ (num - num)/num \$	d6
\$0 E1 + 6	(num - num)/num \$	d4
\$0 E1 + 6 (4	num - num)/num \$	d5
\$0 E1 + 6 (4 num	- num)/num \$	r8, F → num, 2 * num = 2
	- num)/num \$	r6, T → F, 2 * F = 2
	- num)/num \$	r3, E → T, 2 * T = 2
\$0 E1 + 6 (4 T2	- num)/num \$	d7
\$0 E1 + 6 (4 E10	- num)/num \$	d7
\$0 E1 + 6 (4 E10 - 7	num)/num \$	d5
\$0 E1 + 6 (4 E10 - 7 num 5)/num \$	r8, F → num, 2 * num = 2
\$0 E1 + 6 (4 E10 - 7 F3)/num \$	r6, T → F, 2 * F = 2
\$0 E1 + 6 (4 E10 - 7 T12)/num \$	r3, E → E-T, 2 * E-T = 6
\$0 E1 + 6 (4 E10)/num \$	d15
\$0 E1 + 6 (4 E10) 15	/num \$	r7, F → (E), 2 * (E) = 6
\$0 E1 + 6 F3	/num \$	r6, T → F, 2 * F = 2
\$0 E1 + 6 T11	/num \$	d9
\$0 E1 + 6 T11/9	num \$	d5
\$0 E1 + 6 T11/9 num 5	\$	r8, F → num, 2 * num = 2
\$0 E1 + 6 T11/9 F14	\$	r5, T → T/F, 2 * T/F = 6
\$0 E1 + 6 T11	\$	r1, E → E+T, 2 * E+T = 6
\$0 E1	\$	r0, aceptar

	+	-	*	/	()	num	\$	E	T	F
0									d4	d5	
1	d6	d7									r0
2	r3	r3	d8	d9					r3	r3	
3	r6	r6	r6	r6					r6	r6	
4					d4	d5				10	2
5	r8	r8	r8	r8			r8	r8			
6					d4	d5				11	3
7					d4	d5				12	3
8					d4	d5					13
9					d4	d5					14
10	d6	d7							d15		
11	r1	r1	d8	d9					r1	r1	
12	r2	r2	d8	d9					r2	r2	
13	r4	r4	r4	r4					r4	r4	
14	r5	r5	r5	r5					r5	r5	
15	r7	r7	r7	r7					r7	r7	