

Apuntes del 19 de mayo

Compiladores e Intérpretes

Marlon Agüero

FRANCISCO JOSÉ TORRES ROJAS

Tabla de contenido

Recordatorios	3
Repaso.....	4
Ejemplo 3	6
Eliminando ambigüedad del if – then -else	9
Resumen.....	12
Parsing LL (1).....	13
Definición.....	14
Propiedades de los LL (1).....	14
Parsers LL (2).....	15
Parsers LL (k).....	16
Descenso Recursivo.....	17
Ejemplo 1 de Descenso recursivo	17
Ejemplo 2 de Descenso Recursivo	18
Ejemplo 3 de Descenso Recursivo	19
Mensajes de Error	20
Errores y LL (1).....	20
Mensajes de Error	21

Recordatorios

- Solo quedan 2 clases, miércoles 24 y viernes 26
- El miércoles 24 es el último quiz.
- El miércoles 24 se entrega la última tarea.
- Miércoles 31 entrega de proyecto.
- Viernes 02 segundo examen.
- Viernes 09 examen final.
- Se definió quienes van a ser apuntadores para el miércoles 24 y el viernes 26.
- Colochos 0 llegó tarde por lo que no se le asignó fecha para apunta (al final hablo con el profe).



Repaso

Construcción de tabla de Parsing

- Tabla M:
 - Una fila por cada No Terminal
 - Una columna por cada terminal y \$
- Primero se debe calcular el FIRST () y luego el FOLLOW ().
- Con el FIRST () y el FOLLOW () se calcula el PREDICT de cada una de las reglas.
- Para toda regla $A \rightarrow \alpha$:
 - ❖ Para cada elemento a en PREDICT ($A \rightarrow \alpha$) agregue regla $A \rightarrow \alpha$ en $M[A][a]$.

Ejemplo 1

Regla	PREDICT ()
1. $S \rightarrow (S)S$	{ (}
2. $S \rightarrow \epsilon$	{ }, \$ }

Tabla de Parsing

	()	\$
S	1	2	2

Ejemplo 2

Regla	Predict ()
1. $E \rightarrow TE'$	{ (, # }
2. $E' \rightarrow OPTE'$	{ +, - }
3. $E' \rightarrow \epsilon$	{ \$, } }
4. $OP \rightarrow +$	{ + }
5. $OP \rightarrow -$	{ - }
6. $T \rightarrow FT'$	{ (, # }
7. $T' \rightarrow MFT'$	{ * }
8. $T' \rightarrow \epsilon$	{ \$, } +, - }
9. $M \rightarrow *$	{ * }
10. $F \rightarrow (E)$	{ (}
11. $F \rightarrow \#$	{ # }

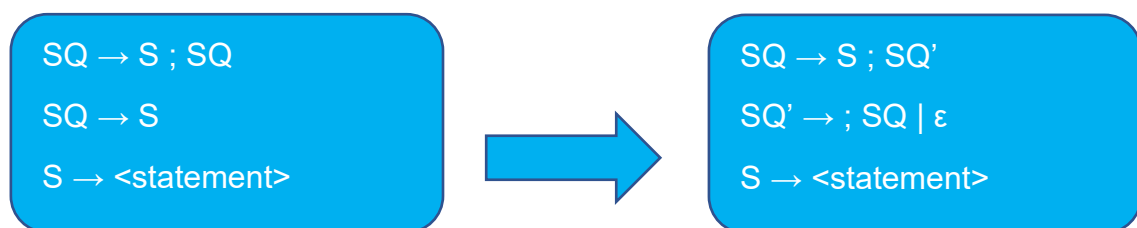
Tabla de Parsing

	(#)	+	-	*	\$
E	1	1					
E'		3	2	2			3
OP					5		
T	6	6					
T'			8	8	8	7	
M						9	
F	10	11					

-----FIN DE REPASO-----

Ejemplo 3

Con la siguiente gramática construir tabla de parsing.



La pasada gramática se transforma porque en la gramática había dos reglas que iniciaban igual, se hace factorización por la izquierda.

Primero hay que calcular el FIRST ().

No terminal	Pasada 1	Pasada 2
SQ	{ }	{ <statement> }
S	{ <statement> }	{ <statement> }
SQ'	{ ; , ϵ }	{ ; , ϵ }

Seguidamente hay que calcular el FOLLOW ().

No Terminal	Pasada 1
SQ	{ \$ }
S	{ ; , \$ }
SQ'	{ \$ }

Ahora con el FIRST () y el FOLLOW () calculamos el PREDICT ().

	FIRST ()	FOLLOW ()
SQ	{ <statement> }	{ \$ }
S	{ <statement> }	{ ; , \$ }
SQ'	{ ; , ϵ }	{ \$ }

Regla	FIRST ()	PREDICT ()
1. $SQ \rightarrow S SQ'$	{ <statement> }	{ <statement> }
2. $SQ' \rightarrow ; SQ$	{ ; }	{ ; }
3. $SQ' \rightarrow \epsilon$	{ ϵ }	{ \$ }
4. $S \rightarrow \text{<statement>}$	{ <statement> }	{ <statement> }

Nota: Si el FIRST () no tiene épsilon, el PREDICT() es igual al FIRST(), sino es el FIRST() – épsilon + FOLLOW ().

Finalmente formamos la tabla de parsing.

	<statement>	;	\$
SQ	1		
S	4		
SQ'		2	3

Eliminando ambigüedad del if – then -else

Gramática

$S \rightarrow \text{IFS} \mid \langle \text{other} \rangle$

$\text{IFS} \rightarrow \text{if}(\text{EX}) \text{ S EL}$

$\text{EL} \rightarrow \text{else S} \mid \varepsilon$

$\text{EX } 0 \mid 1$

Primero calcular el FIRST ()

No terminal	Pasada 1	Pasada 2
S	{ <other> }	{ if, <other> }
IFS	{ if }	{ if }
EX	{ 0,1 }	{ 0,1 }
EL	{ else, ε }	{ else, ε }

Ahora calcular el FOLLOW ()


No terminal	Pasada 1	Pasada 2
S	{ \$ }	{ \$ }
IFS	{ \$ }	{ \$, else }
EX	{ }	{ \$, else }
EL	{) }	{) }

Ahora se calcula el PREDICT() usando el FIRST () y el FOLLOW ().

Regla	PREDICT ()
1. $S \rightarrow IFS$	{ if }
2. $S \rightarrow \langle other \rangle$	{ $\langle other \rangle$ }
3. $IFS \rightarrow \text{if}(EX) S EL$	{ if }
4. $EL \rightarrow \text{else } S$	{ else }
5. $EL \rightarrow \epsilon$	{ \$. else }
6. $EX \rightarrow 0$	{ 0 }
7. $EX \rightarrow 1$	{ 1 }

Finalmente se construye la tabla de parsing.

	If	<other>	Else	0	1	\$
S	1	2				
IFS	3					
EL			4 - 5			5
EX				6	7	

- 
1. Conflicto de reglas
 2. Ambigüedad
 3. ¿Qué hacemos?

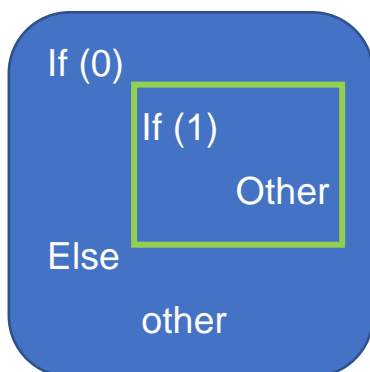


Considere el siguiente código

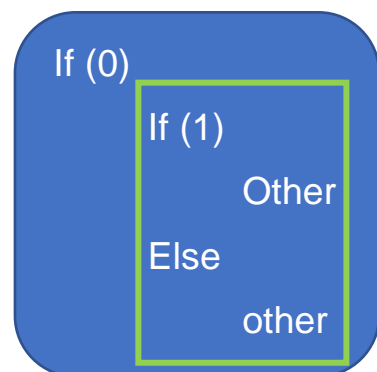
- If(0) if(1)other else other

Con la gramática anterior y el código, se puede hacer de dos formas.

Forma 1



Forma 2



- ¿Cuál de las anteriores es la correcta?
- El problema anterior se le llama “dangling else”

Esto se resuelve forzando que el else se asocie con el if abierto más cercano. Este comportamiento se puede forzar también en la tabla de parsing. En este caso solo se tiene que quitar de la columna del Else, el número 5.

	If	<other>	Else	0	1	\$
S	1	2				
IFS	3					
EL			4			5
EX				6	7	

En principio esta gramática no es LL (1), concepto que se verá más adelante.

Resumen

Hasta ahora esto es lo que se sabe y lo que se puede hacer.

- ❖ Tomar una CFG
- ❖ Corregir la gramática
 - Aplicando factorización por la izquierda.
 - Eliminando recursividad por la izquierda.
- ❖ Calcular FIRST () para todos los No Terminales.
- ❖ Calcular FOLLOW () para todos los No Terminales.
- ❖ Calcular PREDICT () para todas las reglas de la gramática.

- ❖ Construir tabla de parsing
- ❖ No todas las gramáticas permiten este proceso (esto es importante).

Parsing LL (1)

LL(1) PARSING TABLE

	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

- Todo lo que vinimos haciendo es parsing LL (1) aunque no lo sabíamos (si lo sabíamos en la clase anterior el profe se equivocó y puso el primer ejemplo que decía construya la tabla LL (1)).
- Es topdown.
- Es predictivo.
- No hay backtracking.
- La primer L significa que es Left to Right.
- La segunda L es de Leftmost.
- El 1 se refiere a que tiene lookahead de 1.
- Estudiados desde 1968 por Lewis y Stearns.

Definición

- Una CFG es LL (1) si para cualquier par de reglas diferentes, $A \rightarrow \varepsilon$ y $A \rightarrow \beta$ se cumple que el $\text{PREDICT}(A \rightarrow \varepsilon) \cap \text{PREDICT}(A \rightarrow \beta) = \emptyset$.
- Una gramática es LL (1) si se puede construir una tabla de parsing LL (1) para ella.
- Las gramáticas LL (1) son subconjunto propio de CFG.
- No todas las gramáticas son LL (1).

En este punto se armó una pequeña a discusión, que estuvo interesante, básicamente se dijo que un parser que reconozca el lenguaje natural es el cerebro, por el momento no hay nada computacional que lo haga, pero teóricamente es tan posible como la inteligencia artificial. Finalmente, en palabras de Torres, “Si porque usted con toda la gloria de sus colochos y todo eso, usted simplemente es una máquina, química, pero una máquina “(Torres, 2017).

Propiedades de los LL (1)

Sea G una CFG, si G es LL (1), entonces:

- 1- Si una hilera no se rechaza, se garantiza que obtenemos la derivación más izquierda correcta que la genera.
- 2- G no es ambigua.
- 3- La complejidad temporal y espacial del parsing LL (1) es lineal con respecto al largo de la hilera.
- 4- Que sea lineal es una maravilla.

Después de este curso hay que recordar siempre darles gracias a los compiladores.



Todo esto que hemos estado viendo de parsing, John Backus no lo sabía cuándo creó el primer lenguaje de alto nivel. Él lo hizo más probando que funcionaba, como más a lo chanco, aunque es posible que usara algo similar a descenso recursivo que se va a ver más adelante.

Parsers LL (2)

If ($\alpha > 0$)

- Un parser LL (1) puede predecir la regla que debe usarse con solo un símbolo de lookahead.
- Hay gramáticas que, aunque estén totalmente correctas, es decir, aunque no sean recursivas por la izquierda y estén factorizadas, no son LL (1).

- Puede que viendo 2 símbolos por adelantado se pueda predecir la regla que deba usarse.
- Se definen las funciones:
 - $FIRST_2()$
 - $FOLLOW_2()$
 - $PREDICT_2()$
- Los cálculos son más grandes por lo que gasta más memoria.
- La tabla de LL (2) tiene $(n^2 + n)$ columnas, donde n es la cantidad de terminales de la gramática.

Tabla de Parsing LL (2)

1. $S \rightarrow (S) S$

2. $S \rightarrow \epsilon$

	()	()))
S						

Parsers LL (k)

- Un lookahead de k símbolos para decidir que regla usar
- Se definen las siguientes funciones
 - $FIRST_k()$
 - $FOLLOW_k()$
 - $PREDICT_k()$
- Tablas con $(n^k + n^{k-1})$ columnas, donde n es la cantidad de terminales de la gramática.
- Casi todos los lenguajes de programación son LL (1) o, si no, se puede usar otro parser.

Descenso Recursivo

- Fue lo que se usó para el proyecto de MICRO.
- Es equivalente a LL (1).
- Si una gramática es LL (1) se puede hacer por descenso recursivo y viceversa.
- Las dos técnicas tienen los mismos requisitos.
- La gramática se convierte a código.
- Siempre hay un símbolo actual.
- Cada No Terminal tiene un procedimiento asociado.
- Dentro de cada No terminal se escoge una gramática a aplicar, según el símbolo actual.
- El lado derecho de las reglas se convierte en llamadas a procedimientos de cada No Terminal o un match si es un terminal con el símbolo actual.

Ejemplo 1 de Descenso recursivo

Gramática

```
E → iE'  
E' → + iE' | ε
```

Código para procesarlo

```
main (char t)  
{  
    c = getToken();  
    E ();  
}
```

```
match ()  
{  
    If (t == c)  
        c = getToken();  
    else  
        printf ("ERROR");  
}
```

```

E ()
{
    Match("i");
    E' ();
}

```

```

E ()
{
    If (c == "+")
    {
        match ("+" );
        match ("i");
        E' ();
    }
    else
        return;
}

```

Ejemplo 2 de Descenso Recursivo

Gramática

```

SQ → S SQ'
SQ' → ; SQ | ε
S → <statement>

```

Código para procesarlo

```

main (char t)
{
    c = getToken();
    SQ ();
}

```

```

match ()
{
    If (t == c)
        c = getToken();
    else
        printf ("ERROR");
}

```

```
SQ ()
```

```
{
```

```
    S ();
```

```
    SQ' ();
```

```
}
```

```
SQ' ()
```

```
{
```

```
    If (c == “;”)
```

```
    {
```

```
        match (“;”);
```

```
        SQ ();
```

```
    }
```

```
    else
```

```
        return;
```

```
}
```

```
S ()
```

```
{
```

```
    match("<statement>");
```

```
}
```

Ejemplo 3 de Descenso Recursivo

Gramática

$S \rightarrow \text{IFS} \mid \text{<other>}$

$\text{IFS} \rightarrow \text{if}(\text{EX}) \text{ S EL}$

$\text{EL} \rightarrow \text{else S} \mid \epsilon$

$\text{EX} \rightarrow 0 \mid 1$

Código para procesarlo

```
main (char t)
```

```
{
```

```
    c = getToken();
```

```
    S ();
```

```
}
```

```
match ()
```

```
{
```

```
    If (t == c)
```

```
        c = getToken();
```

```
    else
```

```
        printf (“ERROR”);
```

```
}
```

```

S ()
{
    If (c == "if")
    {
        IFS ();
    }
    else
        match("<other>");
}

```

```

IFS ()
{
    Match ("if");
    Match ("{");
    EX ();
    Match ("}");
    S ();
    EL ();
}

```

```

EX ()
{
    If (c == "0")
        match ("0")
    else
        match ("1");
}

```

```

EL ()
{
    If (c == "else")
        match ("else")
        S ();
    else
        return;
}

```

Mensajes de Error

Errores y LL (1)

- El algoritmo detecta hileras invalidad – entrada vacía en la tabla LL (1).
- Estrategia simplista:
 - “Error de sintaxis aquí”.
 - No revisar el resto de la hilera.

- Mejor enfoque:
 - Mensajes de error: dar el mejor diagnóstico posible de la causa del error.
 - Recuperación de error: volver a sincronizar el autómata para reportar otros errores en la hilera.



Mensajes de Error

- ❖ Es casi un arte.
- ❖ Distintos compiladores muestran distintos mensajes para el mismo error.
- ❖ Para un error hay muchas causas posibles.
- ❖ Un error es particular para cada lenguaje y situación.
- ❖ A veces se hace de manera incremental.
 - Dar solo número de error (error N)
 - Ven los ejemplos de error y con esto van mejorando los mensajes de error.

